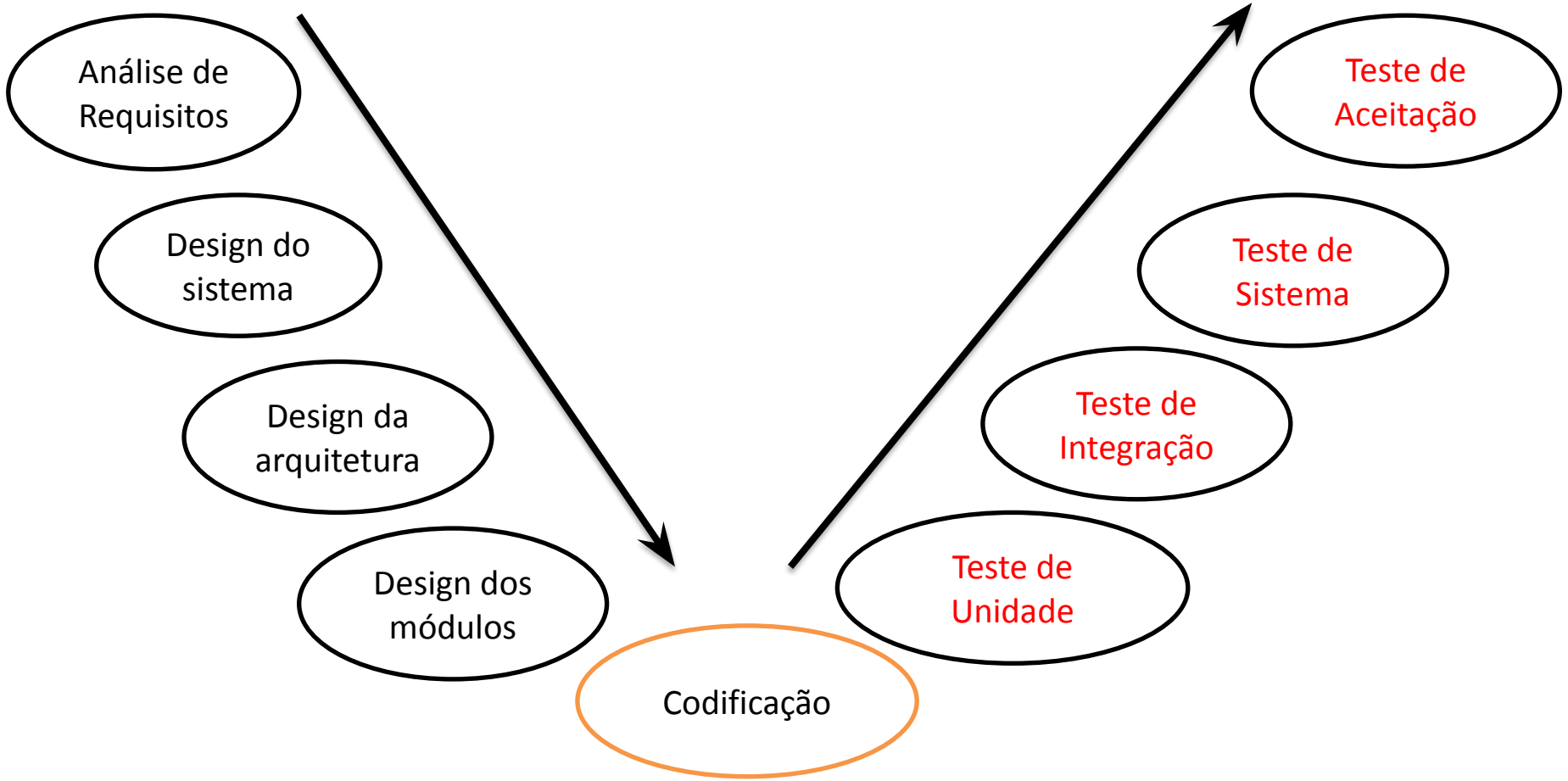
Disciplina Teste de Software

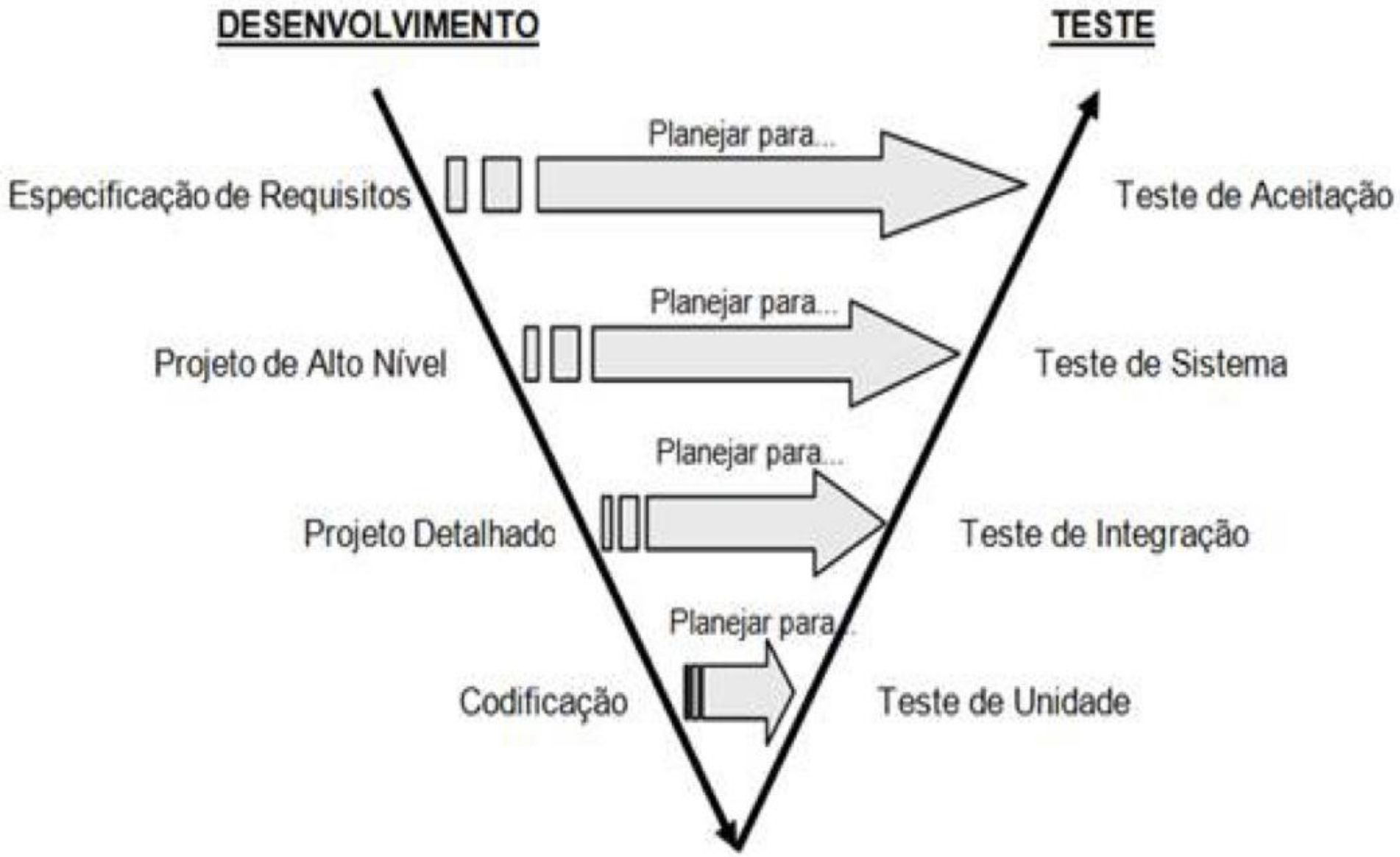
Professora: Iara Carnevale de Almeida
iara.almeida@unicesumar.edu.br

MODELOS DE DESENVOLVIMENTO DE SW
Relembrando ...

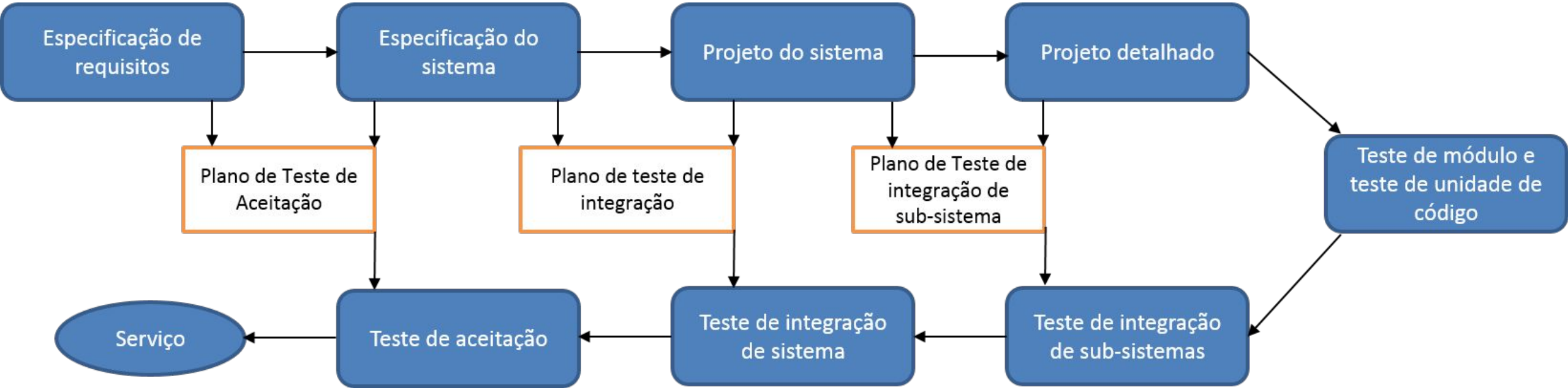
Modelos de Desenvolvimento de Software - Modelo V



Modelo V (Figura 2)



Modelo V (Figura 3)



TESTE UNITÁRIO

Bibliografia básica: Sommerville. 9º ed. Capítulo 8 (pág.148 –149)

Teste Unitário

Procuram por defeitos e verificam o funcionamento de programas, objetos, classes, etc

Feitos de forma isolada do resto do sistema

Podem ser preparados e automatizados antes da codificação (*Test Driven Development – parte 8*)

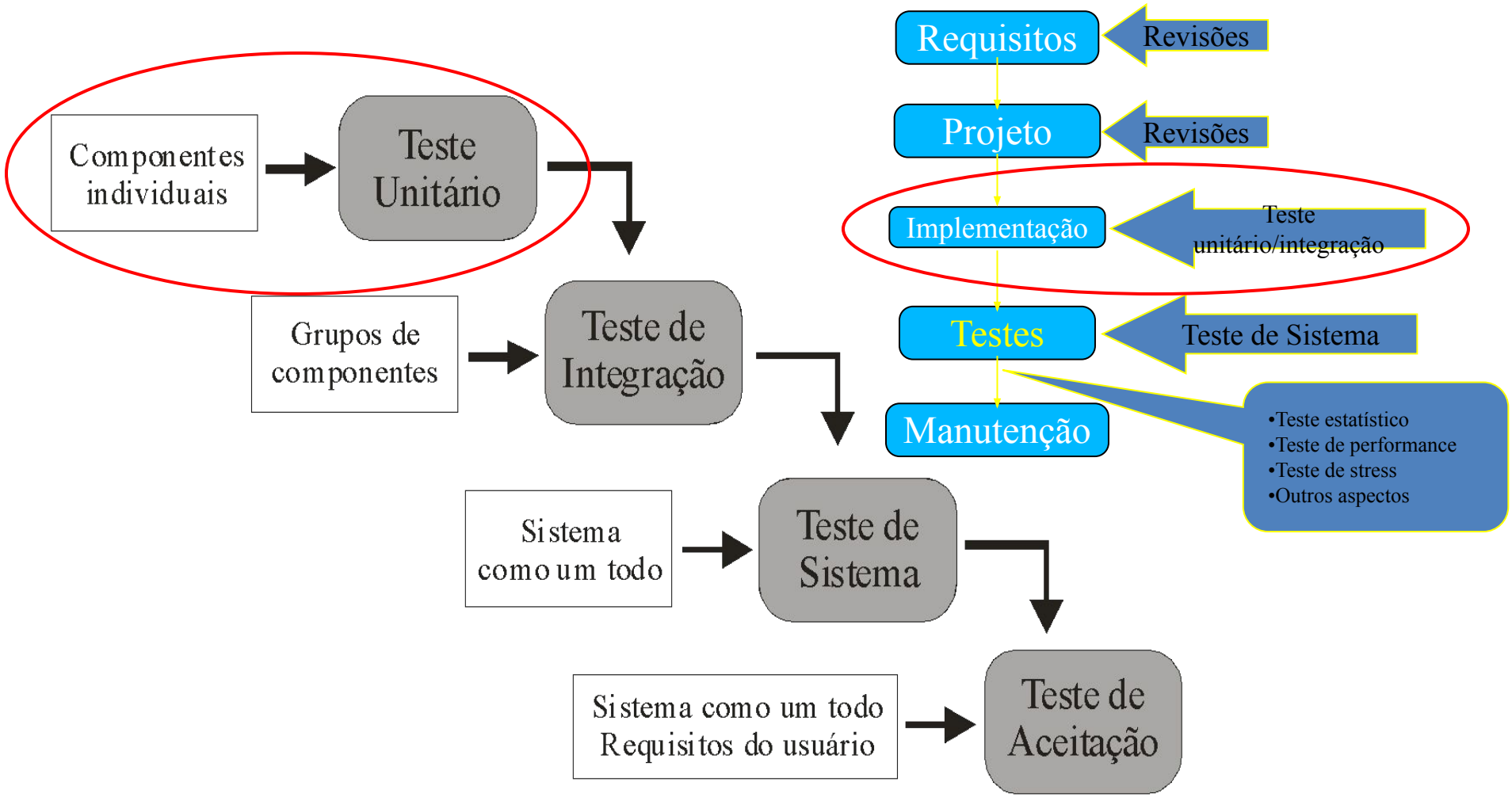
Teste Unitário - Objetivos

Assegurar que cada unidade está funcionando de acordo com sua especificação funcional.

Projetam-se testes para revelar defeitos relativos

- a descrição das funcionalidades
- aos algoritmos
- aos dados
- a lógica de controle

Teste Unitário - Ciclo de desenvolvimento



Teste Unitário - Definição de Unidade

Uma unidade é o menor componente de software que se pode testar em

- um sistema procedural, isto é uma Função ou um Procedimento;
- um sistema orientado a objetos, isto é uma Classe;
- qualquer um dos casos, onde têm-se um componente
    - comprado de um terceiro, que está sob avaliação;
    - que será reusado a partir de uma biblioteca desenvolvida pela própria organização.

Teste unitário

Os testes unitários devem ser chamadas por funções individuais ou métodos, com diferentes parâmetros de entrada.

Deve-se projetar os testes para fornecer uma cobertura de todas as características do objeto:

- testar todas as operações associadas ao objeto;
- definir e verificar o valor em todos os atributos associados aos objeto;
- colocar o objeto em todos os estados possíveis.

Lembrar das técnicas do teste de caixa branca!!

Teste unitário - exemplo

Considere a classe estação meteorológica:

- Testar se o atributo *identificador* foi configurado corretamente;
- Casos de teste para todos os métodos associados ao objeto: *relatarClima()*, *relatarStatus()*,etc..

Deve-se procurar testar os métodos de forma isolada mas, em alguns casos, algumas sequências são necessárias, por exemplo: primeiro *desligar(instrumentos)* e depois *reiniciar(instrumentos)*

Teste Unitário - papéis x processo

Quem testa, desenvolvedor ou Equipe de testes?

- os erros (*bugs*) devem ser registrados como parte da história do módulo;
- a informalidade nesta etapa faz com que um número maior de *bugs* seja detectado nas etapas seguintes, teste de integração e teste de sistema, onde o custo de localização e correção é maior;
- a generalização ou herança faz testes de classes de objeto mais complicados. É necessário testar a operação herdada em todos os contextos de uso.

Artefatos gerados pelo Teste Unitário

Dependente do tipo de linguagem de programação, isto é

- Linguagens imperativas
  - Trechos de código que exercitam as funções ou procedures que se deseja testar
- Linguagens orientadas a objetos
  - Classes *drivers*
- Em ambos pode ocorrer a necessidade do desenvolvimento de *dublês*

Artefatos gerados pelo teste unitário

Classe *Driver*

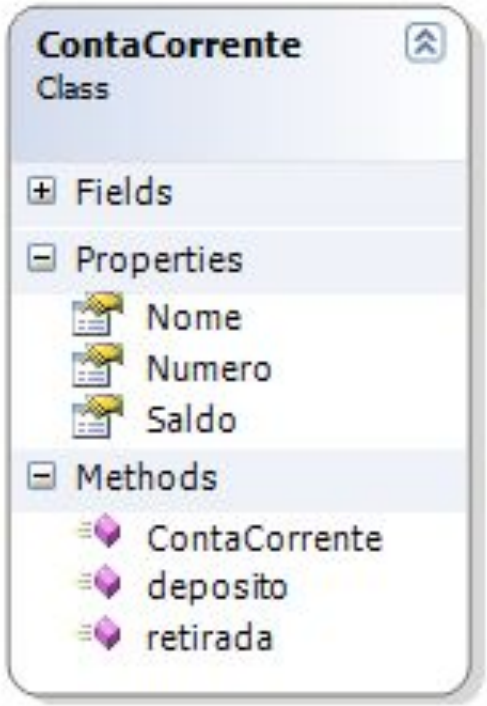
- são as classes que contém os casos de teste;
- procuram exercitar os métodos da “classe alvo” buscando detectar erros;
- normalmente, uma classe “driver” para cada classe do sistema.

Dublês

- simulam o comportamento das classes necessárias ao funcionamento da “classe alvo” e que ainda não foram desenvolvidas.
- quando a classe correspondente ao “dublê” estiver pronta, deve-se re-executar a “classe *driver*” que foi executada anteriormente.

Exemplo de criação de classe "*Driver*"

GRADUAÇÃO
UniCesumar



Configuração	**Conta C1:** **Nome: Fulano** **Número: 100** **Saldo inicial: R$ 0,00**
Casos de teste	Efetuar um depósito de R$ 1000,00. Conferir saldo.
	Efetuar depósito negativo. Verificar lançamento de exceção.
	Efetuar uma retirada de R$ 1000,00. Conferir saldo.
	Efetuar uma retirada de R$ 6000,00. Verificar lançamento de exceção.
	Efetuar uma retirada negativa. Verificar lançamento de exceção.

Legenda:
Teste Positivo
Teste Negativo

Exemplo de classe driver


```
public class ContaCorrenteTest{
        public void TestaDeposito(){
            ContaCorrente target = new ContaCorrente(100,"Fulano");
            target.depositar(1000.0M);
            if (target.Saldo == 1000.0M)
               Console.WriteLine("TestaDeposito: Pass");
            else Console.WriteLine("TestaDeposito: Fail");
        }

        public void TestaRetirada(){
            ContaCorrente target = new ContaCorrente(100,"Fulano");
            target.depositar(6000.0M);
            target.retirar(1000.0M);
            if (target.Saldo == 5000.0M)
               Console.WriteLine("TestaRetirada: Pass");
            else Console.WriteLine("TestaRetirada: Fail");

        }

        public void TestaDepositoNegativo{… }
…
```

Passos para testar unidades, uma sugestão

Se a unidade usa outras, que ainda não estão completas ou que não serão testadas em conjunto, criar classes *Dublês*.

Criar uma ou mais classes *Drivers* para a unidade, incluindo:

- Definir casos de teste e, com apoio do oráculo, definir passos claros e dados de entrada/esperados
- Executar a unidade, usando os casos de teste
- Registrar os resultados obtidos:
  - exibição na tela ou
  - armazenamento em um log.

Automatizar teste unitário

Pode-se utilizar um framework de automação de testes (jUnit).

Framework de testes unitários fornecem classes de testes genéricas que você pode estender para criar casos de teste específicos.

Pode-se executar todos os testes implementados informando, por meio de alguma interface gráfica, do sucesso ou do fracasso deste.

Automatizar teste unitário, em três partes

Configuração: permite definir os valores de entradas e de saídas esperadas.

Execução: chamada do objeto ou do método a ser testado.

Comparação: compara-se o resultado da chamada com o valor de saída esperado. Se a comparação falhar, o teste falou; caso contrário, o teste foi bem-sucedido.

Vantagens no uso de Classes Drivers

- Exige que se reflita sobre as funcionalidades da classe e sua implementação **antes** de seu desenvolvimento.
- Permite a identificação rápida de bugs mais simples.
- Permite garantir que a classe cumpra um conjunto de requisitos mínimos (definidos nos casos de testes).
- Facilita a detecção de efeitos colaterais no caso de manutenção ou *refactoring* -> Aplicação dos Testes de Regressão

Dificuldades no uso das Classes Drivers

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

Evitar de realizar em Teste Unitário

- “Conversar” com banco de dados.
- “Comunicar-se” através da rede.
- “Interagir” com sistema de arquivos.
- Executar junto com outros testes unitários, dos quais existe alguma dependência.
- Realizar ajustes na configuração do ambiente (edição de arquivos de configuração) para poder executar o teste.