# Testes de Unidade

# Testes de Unidade

- Uma unidade de programa é um pedaço de código, como uma função ou método de classe, que pode ser invocado por ou invocar outras unidades.

- Os **testes de unidade** são os testes das unidades de programa que são realizados isoladamente.

# Testes de Unidade

O programador é responsável em executar testes de unidade com o objetivo de verificar se o seu código funciona corretamente ou não, pois já conhece bem o código.

# Testes de Unidade

O teste de unidade é realizado em duas fases complementares:

- **Teste Estático** (baseado na não-execução)
- **Teste Dinâmico** (baseado na execução)

Após a revisão do código (teste estático), o teste dinâmico é realizado. Eles não são alternativos.

# Testes de Unidade Estáticos

- O código é examinado/revisado por um programador sem ser executado.
- O teste estático é mais barato do que o teste dinâmico.

# Testes de Unidade Dinâmicos

## O que é um Caso de Teste?

- Um caso de teste é um par de <entrada, saída esperada>.

Em sistemas *stateless*, onde a saída depende somente da entrada corrente, os casos de testes são bastante simples.

Exemplo: classificar um triângulo (escaleno, isosceles ou equilátero).

Em sistemas *state-oriented*, onde a saída depende do estado corrente do sistema e da entrada corrente, um caso de teste pode consistir de uma sequência de pares <entrada, saída esperada>.

Exemplo: ATM (Automated Teller Machine).

# Testes de Unidade Dinâmicos

1. Selecionar as entradas
2. Calcular a saída esperada
3. Configurar o ambiente de execução do programa
4. Executar o programa
5. Analisar o resultado do teste

# Testes de Unidade

- Um caso de teste é significativo somente se for possível decidir sobre a aceitabilidade do resultado produzido pelo programa em teste.

# Testes Automatizados

- Independente do processo, os testes podem ser automatizados.
  - Os testes podem ser rodados mais frequentemente pois não precisam de intervenção manual.

# Testes Automatizados: Características

- Eles podem rodar sem nenhum esforço.
- Eles detectam e reportam erros sem inspeção manual.
- Podem ser rodados várias vezes com o mesmo resultado.
- Cada teste é independente.

# Conceitos relacionados (Meszaros)

- **SUT (System Under Test)**

É qualquer coisa que está sendo testada.

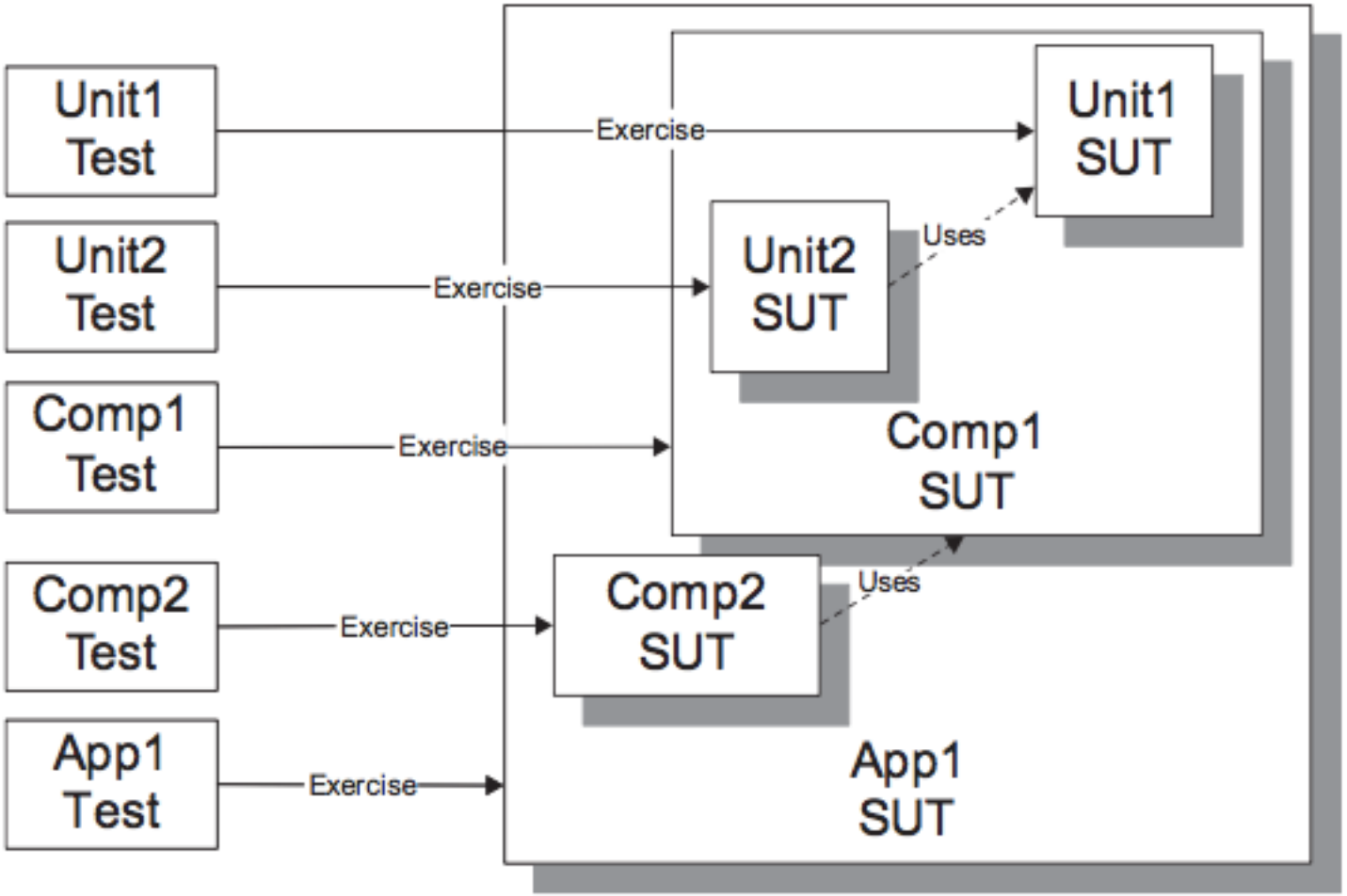
Nos testes de unidade o SUT é uma classe ou método.

Nos testes de aceitação o SUT é todo o sistema ou um subsistema.

- **Test Fixture (Acessório de Teste)**

Qualquer coisa necessária para exercitar o SUT.

# Teste de Software



# Fases dos Testes Automatizados

Cada teste automatizado é projetado em 4 fases:

1. Fixture Setup (Configuração das Fixtures)
2. Exercise SUT (Exercício do SUT)
3. Result Verification (Verificação do Resultado)
4. Fixture Teardown (Destruição das Fixtures)

# Exemplo de Teste Automatizado

```
@Test
public void criaAgenciaCaixaEconomicaTrindade() throws Exception {
    // Fixture Setup
    SistemaBancario sistemaBancario = new SistemaBancario();
    Banco caixaEconomica = sistemaBancario.criarBanco("Caixa Econômica");
    // Exercise SUT
    Agencia caixaEconomicaTrindade = caixaEconomica.criarAgencia("Trindade");
    // Result Verification
    assertEquals("001", caixaEconomicaTrindade.obterIdentificador());
    assertEquals("Trindade", caixaEconomicaTrindade.obterNome());
    assertEquals(caixaEconomica, caixaEconomicaTrindade.obterBanco());
    // Fixture Teardown
}
```

# Fases dos Testes Automatizados

1. Fixture Setup (Configuração das Fixtures)

   O SUT é criado e colocado no estado requerido para ser executado, ou seja, as fixtures necessárias para que o SUT exiba o comportamento esperado são configuradas.
2. Exercise SUT (Exercício do SUT)
3. Result Verification (Verificação do Resultado)
4. Fixture Teardown (Destruição das Fixtures)

# Fixtures de Teste (Acessórios de Teste)

- O **fixture setup** (configuração das fixtures) inclui:
  - código necessário para instanciar o SUT;
  - código para colocar o SUT em estado inicial apropriado;
  - código para criar e inicializar o estado de qualquer coisa da qual o SUT depende ou que será passado para ele como parâmetro.

# Fases dos Testes Automatizados

1. Fixture Setup (Configuração das Fixtures)
2. Exercise SUT (Exercício do SUT)

   É executado um único comportamento do SUT que precisa ser verificado.
3. Result Verification (Verificação do Resultado)
4. Fixture Teardown (Destruição das Fixtures)

# Fases dos Testes Automatizados

1. Fixture Setup (Configuração das Fixtures)
2. Exercise SUT (Exercício do SUT)
3. Result Verification (Verificação do Resultado)

   Verifica se a saída esperada foi obtida ou se o teste falhou.
4. Fixture Teardown (Destruição das Fixtures)

# Fases dos Testes Automatizados

1. Fixture Setup (Configuração das Fixtures)
2. Exercise SUT (Exercício do SUT)
3. Result Verification (Verificação do Resultado)
4. Fixture Teardown (Destruição das Fixtures)

   Teardown as fixtures e coloca o sistema no estado que estava antes do teste.

# Testes Automatizados

Como organizamos as **fixtures**?

- Colocamos junto com o teste (*Test Methods*)?
- Colocamos dentro do método setup da classe de teste (*Testcase Class*)?
- Fatoramos dentro de outros métodos (*Utility Methods*) que são chamados de dentro do teste?

**Fixture Setup**

# Teste de Software

Problemas que aumentam o custo de manutenção dos testes:

- O teste não comunica a sua intenção, ou seja, o que ele está tentando fazer.
- O teste tem vários caminhos de execução, ao invés de ter uma sequência linear de comandos.
- Duplicação de código em diferentes testes.

# Teste de Software

- Independente do processo, os testes devem ser bem escritos.
  - Padrões de Testes
  - Anti-padrões de testes (test smells)

# Test Smells (Anti-Patterns)

- Descrevem problemas recorrentes que os padrões ajudam a resolver.