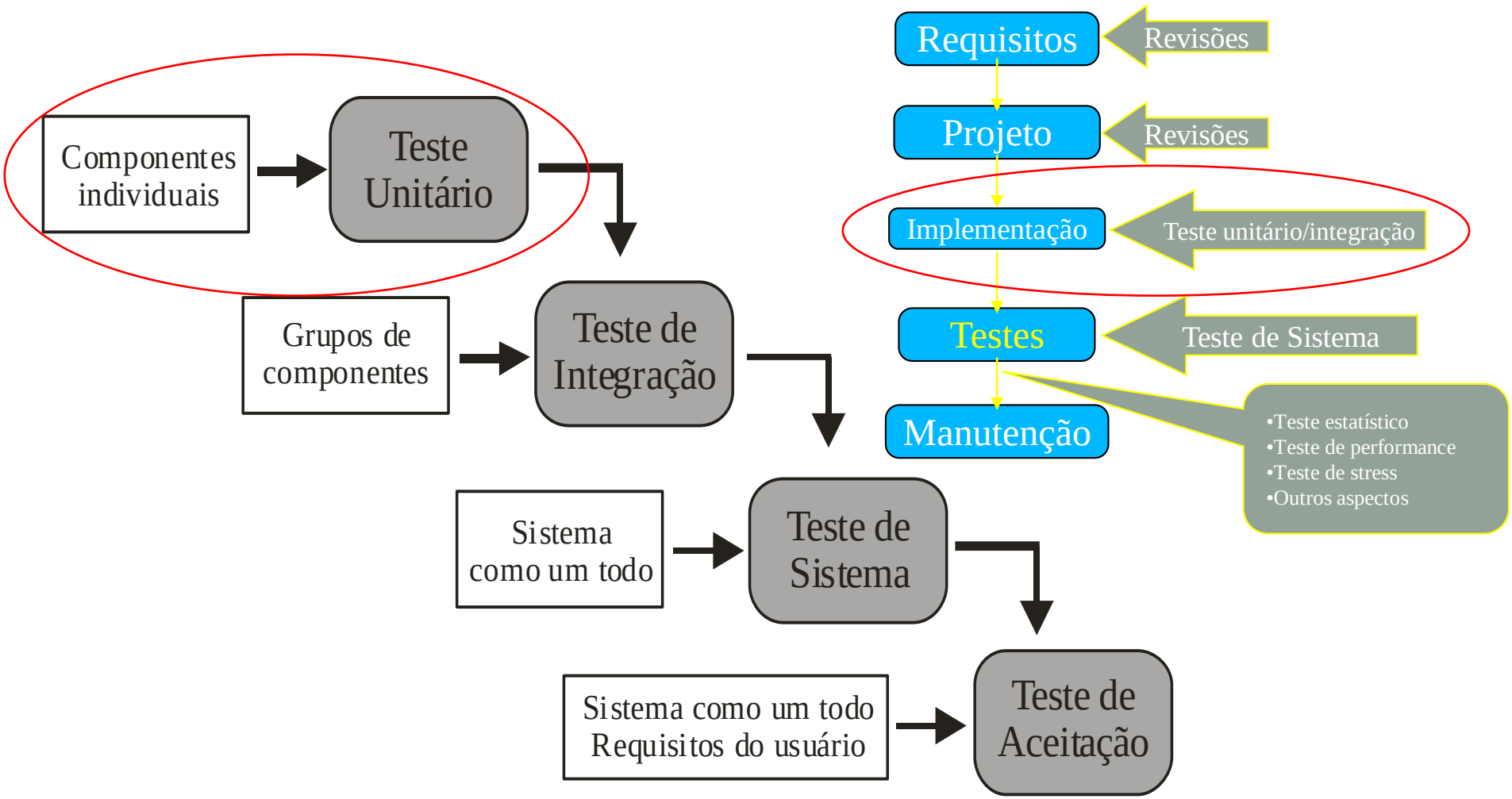
ENGENHARIA DE SOFTWARE (ENGS)

Teste Unitário

INTRODUÇÃO

Teste unitário no ciclo de vida de desenvolvimento de software



Entradas para o teste unitário

- Especificação do módulo antes da sua implementação:
  - Fornece subsídios para o desenvolvimento de casos de teste.
  - Fundamental como oráculo.

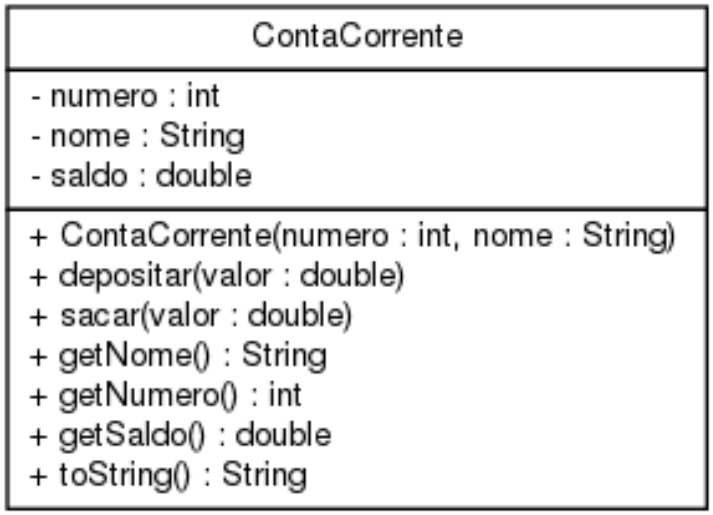
- Código fonte do módulo:
  - Desenvolvimento de casos de teste complementares após a implementação do módulo.
  - Não pode ser usado como oráculo.

Artefatos gerados pelo teste unitário

- Classes ou Módulos “drivers”
  - São as classes/módulos que contêm os casos de teste.
  - Procuram exercitar os métodos/funções da classe “alvo” buscando detectar falhas.
  - Normalmente: uma classe “driver” para cada classe do sistema.
- Dublês (“mockups”)
  - Simulam o comportamento de classes necessárias ao funcionamento da classe “alvo” e que ainda não foram desenvolvidas.
  - Quando a classe correspondente ao “dublê” estiver pronta, será necessário re-executar o “driver” que foi executado usando-se o “dublê”.

Exemplo de criação de classe “driver”

- A partir do projeto de uma classe pode-se especificar os casos de teste.
- Deve-se criar um conjunto de casos de teste capaz de cobrir as funcionalidades básicas da classe.



Exemplo de conjunto de casos de teste

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.

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 cumpre um conjunto de requisitos mínimos (os garantidos pelos testes).
- Facilita a detecção de efeitos colaterais no caso de manutenção ou refactoring.

Dificuldades no uso das classes drivers

- Necessidade de construção do “cenário” em cada método.
- Necessidade de construir um programa para execução 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.

Solucionando as dificuldades

- Uso de ferramentas de automação de teste unitário:
  - JUnit
  - NUnit
  - CppUnit
  - TestNG
  - Catch/Catch2
  - etc

FRAMEWORKS XUNIT E O CATCH2

Teste unitário

- O nivel de teste unitário é todo baseado na construção de classes/módulo “driver”.
- Para cada classe/móduo que se deseja testar deve-se construir uma classe/driver “driver” de teste que exercita a classe original procurando identificar defeitos.
- A execução do teste unitário implica na execução dos métodos de teste que compõem as classes/módulos drivers.
- O uso de “dubles” pode ser necessário quando:
  - Uma classe depende de outra que ainda não está disponível
  - Uma classe depende de outra que não é adequada para uso nos testes que se pretende executar.

Vantagens do uso de classes/módulos “drivers”

- 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 refactoring

Desvantagens do uso de classes/módulos “drivers”

- Necessidade de construção do “cenário” em cada método.
- Necessidade de construir um programa para execução 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

O framework XUnit

- Foi criado no contexto do surgimento do eXtreme Programming em 1998;
- Permite a criação de testes unitários:
  - Estruturados
  - Eficientes
  - Automatizados
- Sua concepção adapta-se facilmente aos IDEs de desenvolvimento
- JUnit: versão para Java do framework
- Para C++/C
  - Diversos: Cgreen, CppUnit, Catch/Catch2, ...
  - Nem todos na linha dos xUnit

Recomendações

- Projete casos de teste independentes uns dos outros;
- Não teste apenas o “positivo”. Garanta que seu código responde adequadamente em todos os cenários;
- Crie um driver para cada classe;
- Inclua o nome do método em cada teste. Ex:
  - PositiveLoadTest
  - NegativeLoadTest
  - PositiveScalarLoadtest
- Depure os testes quando for o caso. Não se esqueça de que os testes também são código !!

Limites

- O teste unitário não deve cruzar certos limites !!!
- Um teste não é um teste unitário se:
  - “Conversa” com o banco de dados
  - “Comunica-se” através da rede
  - “Interage” com o sistema de arquivos
  - Não pode ser executado ao mesmo tempo que os demais testes unitários
  - Necessita de ajustes na configuração do ambiente (edição de arquivos de configuração) para poder ser executado.

Limites

- Testes que não respeitam os limites não são “de todo maus” ...
- Respeitando os limites teremos:
  - Testes que executam rapidamente
  - Testes com alto grau de acoplamento apenas com as classes que testam
  - Testes sem acoplamento com a camada de persistência ou de interface
  - Testes que “resistem” melhor a manutenção do código !!

Catch2

- Framework de testes (multiparadigma) para C++/C
- Single Head File
- Headers adicionais para extensões adicionais
- Como usar?
  - Catch2 é “header only”
    - Não possui libs, .o, DLLs externas
    - Tudo que é necessário está nos headers (.h / .hpp)
  - Coloca-se o header file onde pode ser encontrado
    - Local central
    - Pastas do projeto

Catch2

- Um exemplo

```
unsigned int Factorial( unsigned int number ) {
      return number <= 1 ? number : Factorial(number-1)*number;
}
```

- Testes com tudo dentro

```
#define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this in one cpp file
#include "catch.hpp"

unsigned int Factorial( unsigned int number ) {
    return number <= 1 ? number : Factorial(number-1)*number;
}

TEST_CASE( "Factorials are computed", "[factorial]" ) {
          REQUIRE( Factorial(1) == 1 );
          REQUIRE( Factorial(2) == 2 );
          REQUIRE( Factorial(3) == 6 );
          REQUIRE( Factorial(10) == 3628800 );
}
```

Catch2

- Este código
  - Gera um executável completo
  - Responde à argumentos na linha de comando
  - Sem argumentos
    - Roda todos os casos de teste
    - Gera relatório de execução
  - Exercício
    - Criar um projeto Fatorial
    - Baixar o Catch2
    - Instalar (colocar o header no seu projeto)
    - Compilar e rodar