

ACH 2028

Qualidade de Software

Aula 07 - Teste de Software

Conceitos Básicos e Teste Funcional

Prof. Marcelo Medeiros Eler
marceloeler@usp.br

Introdução

Como garantir que o código que eu escrevi está correto?

Ou melhor: como encontrar falhas no meu código?

Solução: Teste de Software

Conceito de Teste de Software

*O **Teste de Software** consiste em executar um programa com o objetivo de revelar uma falha (Myers, 1979)*

Uma falha é qualquer evento do sistema que viola um objetivo de qualidade estabelecido

O teste de software não consegue provar a ausência de defeitos

Exemplo bem simples de teste de software

```
resultadoEsperado = 10
resultado = Calculadora.multiplicar(5,2)
if (resultadoEsperado == resultado)
    print ("resultado correto")
else
    print ("resultado incorreto")
```

Exemplo bem simples de teste de software

```
resultadoEsperado = 10
resultado = Calculadora.multiplicar(5,2)
if (resultadoEsperado == resultado)
    print ("resultado correto")
else
    print ("resultado incorreto")
```

```
resultadoEsperado = 10
resultado = Calculadora.multiplicar(5,2)
assertEquals(resultadoEsperado,
resultado)
```

Conceito de Teste de Software

Exemplos típicos de falhas (failures)

- Crash
- *Runtime error*
- Resultado errado
- Tempo de resposta excedido
- Formato de saída fora do padrão

Alguns tipos de falhas

Visuais

- problemas de alinhamento de componentes
- sobreposição de componentes
- texto impossível de ler

Alguns tipos de falhas

Funcionais

- O usuário não consegue fazer login
- O usuário não consegue fazer pagamento com dois cartões
- Não é possível atualizar o número de itens no carrinho de compras
- O usuário não consegue escolher outro endereço de entrega
- ...

Alguns tipos de falhas

Não-funcionais

- Desempenho: a página demora 30 segundos para carregar
- Segurança: a senha digitada não aparece ofuscada

Terminologia

Erro → Defeito → Falha

- **Erro:** item de informação ou estado inconsistente / engano do desenvolvedor
- **Defeito:** deficiência mecânica ou algorítmica que, se ativada, pode levar a uma falha
- **Falha:** evento notável em que o sistema viola suas especificações

Quando começar a testar?

O quanto antes a atividade de teste de software for introduzida no processo, melhor

Uma falha encontrada na etapa de desenvolvimento/teste tem o custo de localizar e corrigir o defeito, e executar novamente os testes

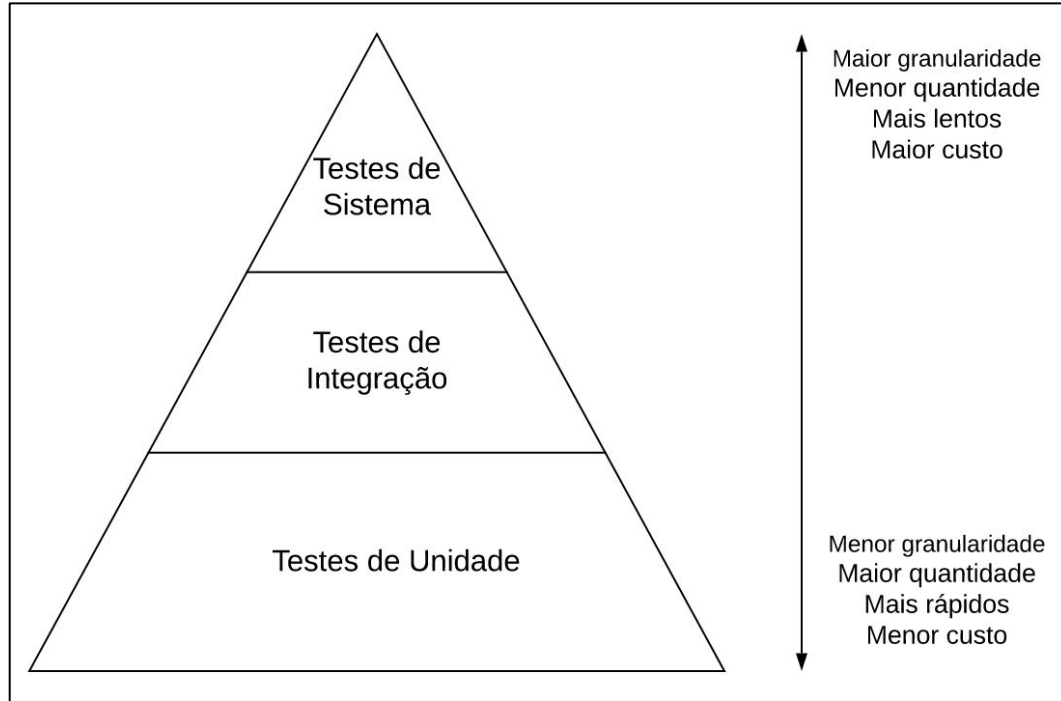
Uma falha encontrada em um software operação tem o custo de *help desk*, escalada dos problemas, alocação de equipe, localizar e corrigir o defeito, executar novos testes e fazer novamente o *deploy* da aplicação corrigida, sem contar a insatisfação do cliente

Quando começar a testar?

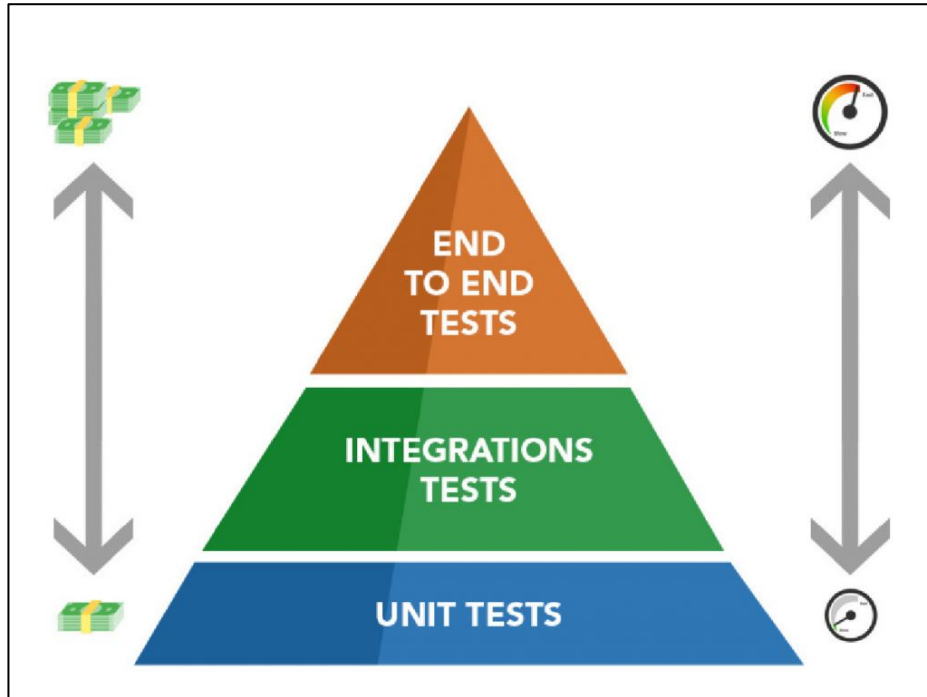
Existem diversos tipos ou fases de teste:

- Teste de Unidade (ou teste de desenvolvedor)
- Teste de Integração
- Teste de Sistema
- Teste de Aceitação (e.g. end to end)
- Teste de Operação
- Teste de Regressão

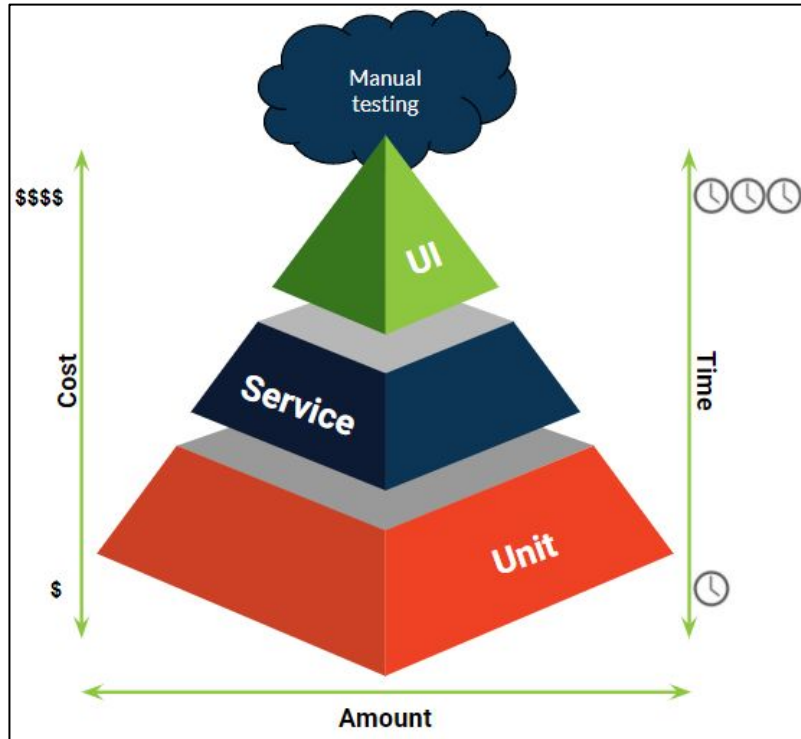
Pirâmide de testes



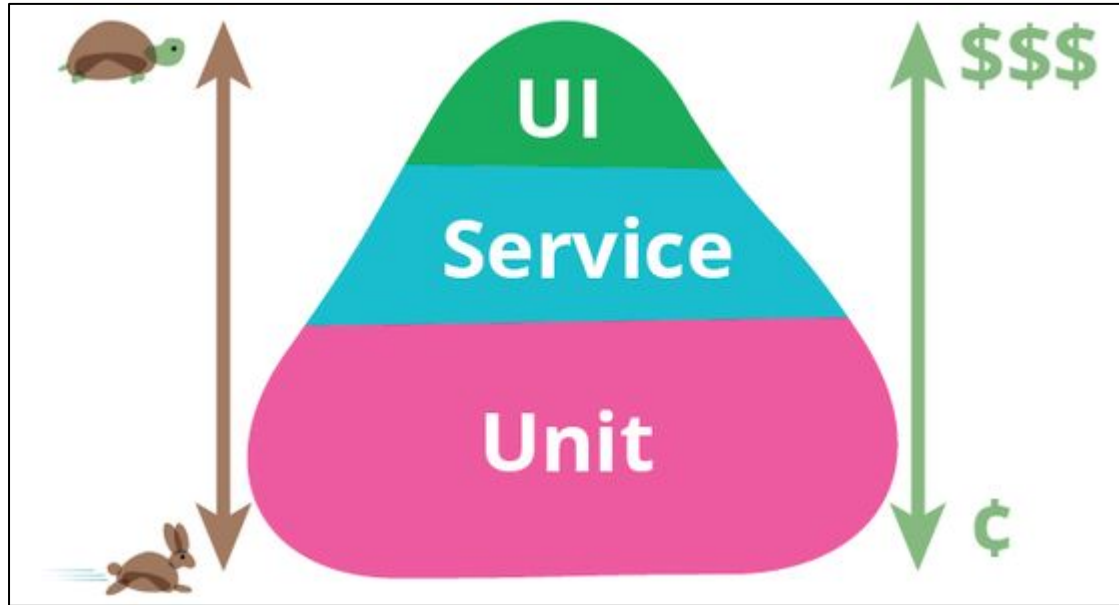
Pirâmide de testes



Pirâmide de testes



Pirâmide de testes



Quando começar a testar?

Existem diversos tipos ou fases de teste:

- **Teste de Unidade (ou teste de desenvolvedor)**
- Teste de Integração
- Teste de Sistema
- Teste de Aceitação (e.g. end to end)
- Teste de Operação
- Teste de Regressão

Teste de Unidade

Também chamados de testes do desenvolvedor, ou micro-testes

São escritos para testar as menores unidades de desenvolvimento (método ou classe, por exemplo)

Teste de Unidade

Também chamados de testes do desenvolvedor, ou micro-testes

São escritos para testar as menores unidades de desenvolvimento (método ou classe, por exemplo)

```
@Test
public void testEmptyStack() {
    Stack<Integer> stack = new Stack<Integer>();
    boolean empty = stack.isEmpty();
    assertTrue(empty);
}
```

Teste de Unidade

Classicamente, os testes considerados de unidade são aqueles que não dependem de nenhuma outra unidade (por implementação, ou por uso de dublês de teste).

Ex: se estou testando um método, ele não pode chamar nenhum outro método (a não ser que seja um dublê)

Entretanto, essa é só uma questão de nomenclatura. No “mundo real”, os testes são classificados de acordo com questões mais práticas

Teste de Unidade

Os testes de unidade devem:

- Ser escritos em grande quantidade (a maior parte dos testes de um software devem ser de unidade - pirâmide de testes)
- Rodar rapidamente (pois são executados muitas vezes durante o desenvolvimento)
- Ajudar a localizar defeitos com grande precisão (afinal, os testes são de partes muito pequenas do código)

Teste de Unidade

É importante que os testes de unidade rodem rapidamente e sejam muitos porque eles são a base para o teste de regressão e refatoração

Após refatorar o código é importante executar casos de testes para verificar se a alteração no código não alterou seu comportamento

Teste de Unidade

Os testes de unidade não devem, em sua execução:

- se comunicar com uma base de dados
- utilizar sistemas de arquivo
- se comunicar com alguma função na rede
- alterar configurações do ambiente

Esses são testes válidos, mas geralmente são testes de integração que vão levar mais tempo para serem executados.

Teste de Unidade

Nos casos em que se deseja testar uma unidade que precisa se comunicar com outros módulos ou sistemas, a alternativa é usar os famosos dublês de teste:

- **Mock:** A ideia por trás dos mocks é de abstrair a lógica da classe dependente, ao mesmo tempo que garante que as interações ocorrem dentro do esperado.
- **Stub:** fornecem respostas prontas para as chamadas feitas durante o teste, geralmente não responde a nada fora do que está programado.
- **Fake:** Objetos que possuem implementação, porém com o objetivo de diminuir a complexidade e/ou tempo de execução de alguns processos para acelerar os testes, porém nunca usado em produção.

Teste de Integração

Testes que envolvem a execução de funções que exigem a integração entre mais de uma unidade/módulo/sistema

Na prática, o teste de integração vai ser aquele que vai testar funções que precisam acessar o banco de dados, escrever/ler dados para/de um arquivo, vão invocar um serviço na rede, etc, sem usar dublês de teste

Esses testes são mais lentos do que os testes de unidade

Teste de Sistema

Envolve o teste do software como um todo (considerando suas operações globais, geralmente acessadas via interface gráfica ou API)

Podem ser chamados também de testes end-to-end

Podem envolver o teste de propriedades como:

- Desempenho
- Segurança
- Disponibilidade
- ...

Test end-to-end

Como o próprio nome diz, tem o objetivo de testar uma interação de ponta a ponta em um software

Ex: testar todo o processo de um aluguel de carro:

- Usuário faz login
- Usuário faz uma busca pela categoria de carro desejado
- Usuário seleciona o carro desejado e adiciona no carrinho
- Usuário finaliza a compra
- Usuário faz o pagamento e recebe notificação

Outros tipos de teste

Teste de aceitação:

- Às vezes também chamado de teste de sistema (tudo questão de nomenclatura)
- Teste que geralmente envolve a participação de clientes/usuários para “aceitar” o sistema (ou seja, implementa o que foi proposto)
- Baseado em regras de negócio/requisitos
- Pode também se referir a teste end-to-end

Outros tipos de teste

Testes de regressão:

- Conjunto de testes executados para verificar se mudanças feitas no software introduziram algum defeito
- Dão segurança para a refatoração, correção e evolução do software
- Podem incluir testes de unidade, de integração e de sistema
- Quanto mais rápido executar, melhor

Outros tipos de teste

Testes de operação:

- Testes realizados pelos próprios usuários
- Alfa: realizados no ambiente da organização que desenvolve
- Beta: realizados no ambiente dos usuários

Forma de execução de testes

Testes manuais

Testes automatizados

Forma de execução de testes

Testes manuais

Testes automatizados:

- Facilitam a reexecução dos testes

Automação de testes

Ferramentas e Frameworks de teste

- Atendem a diversos tipos de teste (unidade, integração, sistema, end-to-end)
- Dependem da linguagem de programação, plataforma e tipo de teste

Teste de unidade/integração

Framework JUnit (Java)

Método: `classificaTriangulo(int LA, int LB, int LC)` throws `LadoInvalidoException`

Teste de unidade/integração

```
import org.junit.jupiter.api. Test;  
import static org.junit.jupiter.api.Assertions. assertEquals;  
import static org.junit.jupiter.api.Assertions. assertThrows;  
  
public class TrianguloTest {  
    ...  
}
```

Teste de unidade/integração

```
@Test
public void testEquilatero () throws LadoInvalidoException{
    int LA = 5;
    int LB= 5;
    int LC = 5;
    String result = Triangulo.classificaTriangulo(LA, LB, LC);
    assertEquals("EQUILATERO",result);
}
```

Teste de unidade/integração

```
@Test
public void testEquilatero () throws LadoInvalidoException{
    int LA = 5;
    int LB= 5;
    int LC = 5;
    String result = Triangulo.classificaTriangulo(LA, LB, LC);
    assertEquals("EQUILATERO",result);
}
```

▼ ✓ Test Results	13 ms
▼ ✓ TrianguloTest	13 ms
✓ testEquilatero()	13 ms

Teste de unidade/integração

```
@Test
public void testEquilatero() throws LadoInvalidoException{
    int LA = 5;
    int LB= 5;
    int LC = 5;
    String result = Triangulo.classificaTriangulo(LA, LB, LC);
    assertEquals("EQUILATERO", result);
}
```

Test Results	10 ms	/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
TrianguloTest	10 ms	
testEquilatero()	10 ms	<pre>org.opentest4j.AssertionFailedError: Expected :EQUILATERO Actual :ISOSCELES <Click to see difference> <5 internal lines> at geometria.TrianguloTest.testEquilatero(TrianguloTest.java:18) <19 internal lines> at java.util.ArrayList.forEach(ArrayList.java:1259) <9 internal lines> at java.util.ArrayList.forEach(ArrayList.java:1259) <21 internal lines></pre>

Teste de unidade/integração

```
@Test
public void testIsosceles1 () throws LadoInvalidoException{
    int LA = 5;
    int LB= 5;
    int LC = 6;
    String result = Triangulo.classificaTriangulo(LA, LB, LC);
    assertEquals("ISOSCELES",result);
}
```

Teste de unidade/integração

```
@Test
public void testLAINvalido () throws LadoInvalidoException{
    int LA = -2;
    int LB= 4;
    int LC = 5;
    assertThrows(LadoInvalidoException.class, () -> Triangulo.classificaTriangulo(LA, LB, LC));
}
```


Teste de unidade/integração

Test Results	23 ms	
TrianguloTest	23 ms	org.opentest4j.AssertionFailedError:
testEquilatero()	11 ms	Expected :NAO FORMA TRIANGULO
testEscaleno()	1 ms	Actual :ESCALENO
testLAINvalido()	3 ms	<Click to see difference>
testLCInvalido()	5 ms	
testNaoTriangulo1()	1 ms	<5 internal lines>
testNaoTriangulo2()		at geometria.TrianguloTest.testNaoTriangulo3(TrianguloTest.java:116) <19 internal lines>
testNaoTriangulo3()	1 ms	at java.util.ArrayList.forEach(ArrayList.java:1259) <9 internal lines>
testIsosceles1()		at java.util.ArrayList.forEach(ArrayList.java:1259) <21 internal lines>
testIsosceles2()		
testIsosceles3()		
testLBInvalido()	1 ms	

Teste end-to-end

```
import org.openqa.selenium.By ;
import org.openqa.selenium.WebDriver ;
import org.openqa.selenium.chrome.ChromeDriver ;

public class SeleniumTest {

    public static void main(String[] args) {
        // declaration and instantiation of objects/variables
        System.setProperty("webdriver.chrome.driver" , "../ChromeDriver/chromedriver" );
        WebDriver driver=new ChromeDriver() ;
        // Launch website
        driver.navigate().to( "http://www.google.com/" );
        // Click on the search text box and send value
        driver.findElement(By.id( "lst-ib" )).sendKeys( "javatpoint tutorials" );
        // Click on the search button
        driver.findElement(By.name( "btnK" )).click() ;
    }
}
```

Teste end-to-end (Robot Framework)

```
*** Test Case ***
```

```
Scenario: Buy product
```

```
Wait Until Page Contains Element  edit_text_username
```

```
Input Text  edit_text_username  "User12345"
```

```
Input Text  edit_text_passwd  "Mypassword12345"
```

```
Wait Until Page Contains Element  btn_login
```

```
Click Element  btn_login
```

```
Wait Until Page Contains Element  edit_text_search
```

```
Input Text  edit_text_search "Micro USB Charger Cable"
```

```
Wait Until Page Contains Element  btn_search
```

```
Click Element  btn_search
```

```
Page Should Contain Text  "Micro USB Charger Cable"
```

```
Click Text  "Micro USB Charger Cable"
```

```
Wait Until Page Contains Element  btn_buy
```

```
Click Element  btn_buy
```

```
Wait Until Page Contains Element  btn_checkout
```

```
Click Element  btn_checkout
```

```
...
```

Teste end-to-end (Robot Framework)

```
*** Test Case ***
```

```
Scenario: Buy product
```

```
Log User
```

```
Search for product
```

```
Buy product
```

```
*** Keywords ***
```

```
Log User
```

```
Wait Until Page Contains Element edit_text_username
```

```
Input Text edit_text_username "User12345"
```

```
Input Text edit_text_passwd "Mypassword12345"
```

```
Wait Until Page Contains Element btn_login
```

```
Click Element btn_login
```

```
Search for product
```

```
Wait Until Page Contains Element edit_text_search
```

```
Input Text edit_text_search "Micro USB Charger Cable"
```

```
Wait Until Page Contains Element btn_search
```

```
Click Element btn_search
```

```
Page Should Contain Text "Micro USB Charger Cable"
```

```
By product
```

```
Click Text "Micro USB Charger Cable"
```

```
Wait Until Page Contains Element btn_buy
```

```
Click Element btn_buy
```

```
Wait Until Page Contains Element btn_checkout
```

```
Click Element btn_checkout
```

Teste end-to-end (Robot Framework)

```
*** Test Case ***
```

```
Scenario: Buy product
```

```
Log User "User12345" "Mypassword12345"
```

```
Search for product "Micro USB Charger Cable"
```

```
Buy product "Micro USB Charger Cable"
```

```
*** Keywords ***
```

```
Log User {${username}} {${password}}
```

```
Wait Until Page Contains Element edit_text_username
```

```
Input Text edit_text_username {${username}}
```

```
Input Text edit_text_passwd {${password}}
```

```
Wait Until Page Contains Element btn_login
```

```
Click Element btn_login
```

```
Search for product {${product}}
```

```
Wait Until Page Contains Element edit_text_search
```

```
Input Text edit_text_search {${product}}
```

```
Wait Until Page Contains Element btn_search
```

```
Click Element btn_search
```

```
Page Should Contain Text {${product}}
```

```
By product {${product}}
```

```
Click Text {${product}}
```

```
Wait Until Page Contains Element btn_buy
```

```
Click Element btn_buy
```

```
Wait Until Page Contains Element btn_checkout
```

```
Click Element btn_checkout
```

Automação de testes

É preciso estudar cada framework/ferramenta de testes para entender os detalhes sobre como automatizar os testes que se pretende criar

Para as próximas aulas sobre teste

Como escrever bons casos de teste?

- Casos de teste que tem grande potencial de revelar uma falha e revelar defeitos no código
- É preciso aprender sobre técnicas e critérios de teste (funcional, estrutural, etc)

Para as próximas aulas sobre teste

Casos de teste também são uma parte do software compartilhada entre o time de desenvolvimento

Os casos de testes precisam ser corrigidos e evoluídos constantemente

Portanto, existem:

- Boas práticas/padrões para escrever testes
- Test smells (design de teste ruim)

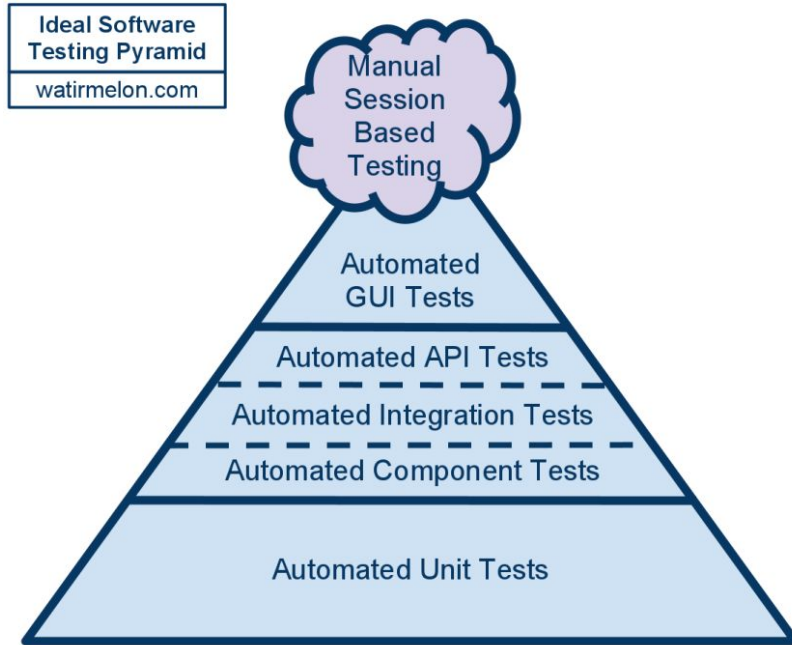
Recado desta aula

Escrevam testes antes (TDD - próximas aulas) ou depois de escrever cada pedaço de código

Escrevam muitos casos de teste de unidade para detectar defeitos inseridos ao longo do desenvolvimento, principalmente nas atividades de refatoração e evolução do produto

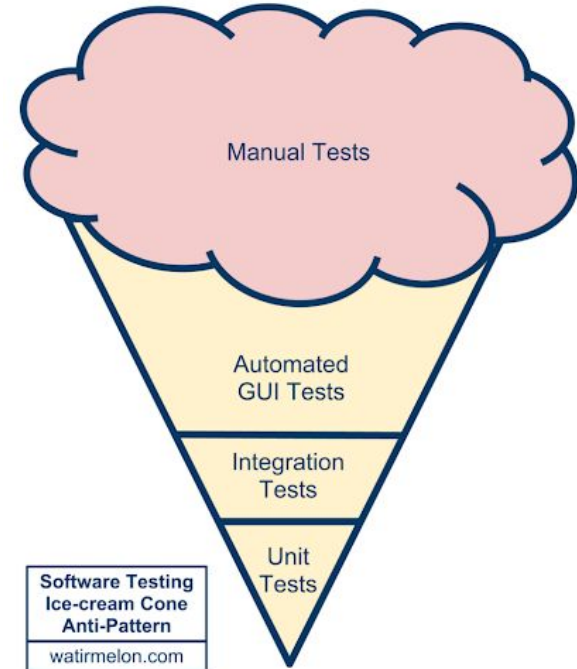
Automatizem a execução dos casos de teste que criarem (achem as ferramentas ou frameworks adequados para a linguagem/plataforma/tipo de teste)

Cuidado com o cone de teste (é comum, mas perigoso)



IDEAL

VS



MUITO COMUM

Cuidado com o cone de teste (é comum, mas perigoso)

Testes de unidade ajudam a achar o defeito mais rapidamente (teste localizado)

Os testes end-to-end revelam falhas cujos defeitos associados são mais difíceis de localizar a origem

Portanto, invistam em muitos testes de unidade

Leituras recomendadas

Leitura recomendada:

- **Autor:** Ham Vocke
- The Practical Test Pyramid
(<https://martinfowler.com/articles/practical-test-pyramid.html>)

Leituras recomendadas

Leitura recomendada:

- **Livro:** Engenharia de Software Moderna - Princípios e Práticas para Desenvolvimento de Software com Produtividade
- **Autor:** Marco Tulio Valente
- Capítulo 8 - Testes (<https://engsoftmoderna.info/cap8.html>)

Como encontrar todas as falhas de um software?

Um conjunto de casos de teste com **qualidade ótima** é capaz de revelar todas as falhas de um software

Para encontrar todas as falhas e garantir que o software está livre de defeitos é necessário **testá-lo em todas as condições** e com **todos os dados de entrada possíveis**

Como encontrar todas as falhas de um software?

Entretanto, as condições, os cenários, os dados de entrada e suas combinações tendem a ser **infinitos ou muito grandes**

Portanto, criar um conjunto de casos de teste ótimo é geralmente **impossível ou impraticável**

Uma pergunta importante neste cenário

Como criar um conjunto de casos de teste que seja:

- Finito e factível?
- Capaz de revelar o maior número de falhas perceptíveis?
- Capaz de revelar as falhas mais críticas ou relevantes do software?

Esta pergunta esconde algumas outras:

- Como seleccionar cenários/dados para criar bons casos de teste?
- Quantos testes são necessários para testar um software ou parte dele?
- Como saber se os testes criados são suficientes?

Exemplo: IRPF 2021 (uma simplificação)

Entrada	Valor
Rend. tributáveis recebidos PJ:	
Rend. isentos e não-tributáveis:	
Imposto pago (retido na fonte):	
Pagamentos efetuados (educação):	
Pagamentos efetuados (saúde):	
Número de dependentes:	
Total a pagar ou a receber:	

IRPF 2021 (uma simplificação) - CT1

Entrada	Valor
Rend. tributáveis recebidos PJ:	120000
Rend. isentos e não-tributáveis:	20000
Imposto pago (retido na fonte):	30000
Pagamentos efetuados (educação):	9000
Pagamentos efetuados (saúde):	9000
Número de dependentes:	2
Total a pagar ou a receber:	

IRPF 2021 (uma simplificação) - CT2

Entrada	Valor
Rend. tributáveis recebidos PJ:	150000
Rend. isentos e não-tributáveis:	30000
Imposto pago (retido na fonte):	40000
Pagamentos efetuados (educação):	12000
Pagamentos efetuados (saúde):	15000
Número de dependentes:	3
Total a pagar ou a receber:	

IRPF 2021 (uma simplificação) - CT3

Entrada	Valor
Rend. tributáveis recebidos PJ:	200000
Rend. isentos e não-tributáveis:	30000
Imposto pago (retido na fonte):	60000
Pagamentos efetuados (educação):	15000
Pagamentos efetuados (saúde):	18000
Número de dependentes:	3
Total a pagar ou a receber:	

IRPF 2021 (uma simplificação) - CT4

Entrada	Valor
Rend. tributáveis recebidos PJ:	1000000
Rend. isentos e não-tributáveis:	30000
Imposto pago (retido na fonte):	200000
Pagamentos efetuados (educação):	35000
Pagamentos efetuados (saúde):	28000
Número de dependentes:	2
Total a pagar ou a receber:	

Perguntas

Esses casos de teste são capazes de:

- revelar o maior número de falhas perceptíveis?
- revelar as falhas mais críticas ou relevantes do software?
- são suficientes?

Exemplo: IRPF 2021 (uma simplificação)

Entrada	Valor
Rend. tributáveis recebidos PJ:	
Rend. isentos e não-tributáveis:	
Imposto pago (retido na fonte):	
Pagamentos efetuados (educação):	
Pagamentos efetuados (saúde):	
Número de dependentes:	
Total a pagar ou a receber:	

Exemplo: IRPF 2021 (com qualificação)

Entrada	
Rend.	
Rend.	
Impos	
Pagament	
Pagamentos efetu	
Número de dependentes:	
Total a pagar ou a receber:	

Como selecionar cenários/dados para criar bons casos de teste?

Exemplo: IRPF 2021 (com qualificação)

Entrada	
Rend.	
Rend.	
Impos	
Pagament	
Pagamentos efetu	
Número de dependentes:	
Total a pagar ou a receber:	

Como selecionar
cenários/dados para criar
bons casos de teste?

Exemplo: IRPF 2021 (com isenção e qualificação)

Entrada	
Rend.	
Rend.	
Impos	
Pagament	
Pagamentos efet	
Número de dependentes:	
Total a pagar ou a receber:	

Aleatório?

Exemplo: IRPF 2021 (com isenção de qualificação)

Entrada	
Rend.	
Rend.	
Impos	
Pagament	
Pagamentos efet	
Número de dependentes:	
Total a pagar ou a receber:	

Aleatório?
Viciado?

Exemplo: IRPF 2021 (Declaração de Imposto de Renda)

Aleatório?

Viciado?

**Estudar cada campo e
entender as categorias
específicas?**

Entrada	
Rend.	
Rend.	
Impos	
Pagament	
Pagamentos efetu	
Número de dependentes:	
Total a pagar ou a receber:	

Técnicas e critérios de teste

Um bom caso de teste é aquele que tem uma alta probabilidade de revelar uma falha ainda não descoberta

Portanto, técnicas e critérios de teste foram propostos para orientar o testador na tarefa de derivar os casos de teste com base nos artefatos do projeto (especificação, código, modelos, etc):

- Teste Adhoc
- Teste Exploratório
- Teste Funcional ou Caixa-Preta
- Teste Estrutural ou Caixa-Branca
- Teste Baseado em Erros

Técnicas de teste

Um bom caso de teste é aquele que tem uma alta probabilidade de revelar uma falha ainda não descoberta

Portanto, técnicas de teste foram propostas para orientar o testador na tarefa de derivar os casos de teste com base nos artefatos do projeto (especificação, código, modelos, etc):

- Teste Adhoc
- Teste Exploratório
- **Teste Funcional ou Caixa-Preta**
- Teste Estrutural ou Caixa-Branca
- Teste Baseado em Erros

Cr terios de teste

Definem requisitos de teste que devem ser satisfeitos pelos casos de teste

Um crit rio   satisfeito somente quando todos os requisitos que ele define s o satisfeitos

Ajudam a responder  s quest es:

- Como selecionar valores de entrada para criar bons casos de teste?
- Quantos casos de teste devem ser criados?
- Quando parar de testar?

Teste funcional ou caixa-preta

Tem o objetivo de verificar se uma funcionalidade, operação, função, método, classe, programa está de acordo com os requisitos especificados

Os casos de teste são definidos com base na especificação do software (descrições, casos de uso, requisitos, diagramas, etc)

Exemplos de critérios desta técnica:

- Classes de equivalência
- Análise de valor-limite
- Tabelas de decisão

Teste funcional ou caixa-preta



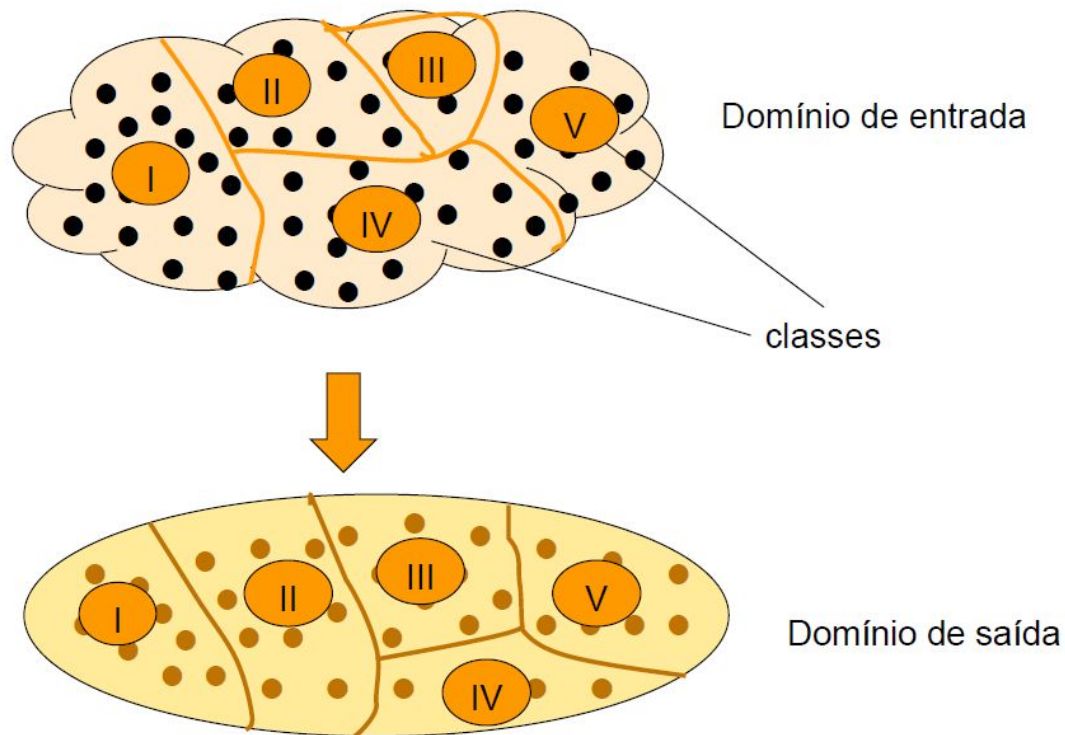
Critérios da técnica de teste funcional

Classe de Equivalência

Análise de valor-limite

Tabela de decisão

Classes de Equivalência



Classes de Equivalência

A partir das condições de entrada de dados identificadas na especificação, divide-se o domínio de entrada e de saída em classes de equivalência

As classes de equivalência podem ser caracterizadas como válidas ou inválidas, mas nem sempre existem os dois tipos de classe.

Classes de Equivalência

Em seguida, seleciona-se o menor número possível de casos de teste, baseando-se na hipótese que um elemento de uma dada classe seria representativo da classe toda, sendo que para cada uma das classes inválidas deve ser gerado um caso de teste distinto.

Classes de Equivalência

Em teoria, todos os elementos pertencentes à mesma classe de equivalência terão o mesmo efeito no teste de software

Classes de Equivalência

O uso de particionamento permite examinar os requisitos mais sistematicamente e restringir o número de casos de teste existentes.

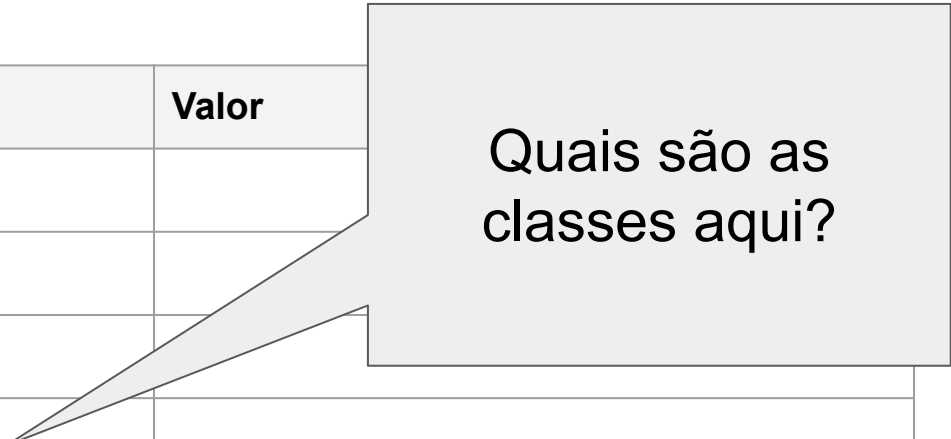
Uma classe de equivalência representa um conjunto de estados válidos e inválidos para uma dada condição de entrada

Classes de equivalência são disjuntas

A união das classes de equivalência = domínio completo

Exemplo: IRPF 2021 (uma simplificação)

Entrada	Valor
Rend. tributáveis recebidos PJ:	
Rend. isentos e não-tributáveis:	
Imposto pago (retido na fonte):	
Pagamentos efetuados (educação):	
Pagamentos efetuados (saúde):	
Número de dependentes:	
Total a pagar ou a receber:	



Quais são as classes aqui?

Exemplo: IRPF 2021 (uma simplificação)

Entrada	Valor
Rend. tributáveis recebidos PJ:	
Rend. isentos e não-tributáveis:	
Imposto pago (retido na fonte):	
Pagamentos efetuados (educação):	
Pagamentos efetuados (saúde):	
Número de dependentes:	
Total a pagar ou a receber:	

Nas despesas com educação (ensino infantil, fundamental, médio, técnico e superior, o que engloba graduação e pós-graduação), o limite de dedução permaneceu em R\$ 3.561,50 por dependente.

IRPF 2021 (uma simplificação) - CT1

Entrada	Valor
Rend. tributáveis recebidos PJ:	120000
Rend. isentos e não-tributáveis:	20000
Imposto pago (retido na fonte):	30000
Pagamentos efetuados (educação):	9000
Pagamentos efetuados (saúde):	9000
Número de dependentes:	2
Total a pagar ou a receber:	

IRPF 2021 (uma simplificação) - CT2

Entrada	Valor
Rend. tributáveis recebidos PJ:	120000
Rend. isentos e não-tributáveis:	20000
Imposto pago (retido na fonte):	30000
Pagamentos efetuados (educação):	12000
Pagamentos efetuados (saúde):	9000
Número de dependentes:	2
Total a pagar ou a receber:	

IRPF 2021 (uma simplificação) - CT3

Entrada	Valor
Rend. tributáveis recebidos PJ:	120000
Rend. isentos e não-tributáveis:	20000
Imposto pago (retido na fonte):	30000
Pagamentos efetuados (educação):	15000
Pagamentos efetuados (saúde):	9000
Número de dependentes:	2
Total a pagar ou a receber:	

IRPF 2021 (uma simplificação) - CT3

Entrada	Valor
Rend. tributáveis recebidos PJ:	12000
Rend. isentos e não-tributáveis:	
Imposto pago (retido na fonte):	3000
Pagamentos efetuados (educação):	15000
Pagamentos efetuados (saúde):	9000
Número de dependentes:	2
Total a pagar ou a receber:	

9000 vs 12000 vs 15000
(todos pertencem à mesma classe)

4500 / 6000 / 7500 por dependente
(acima do limite)

IRPF 2021 (uma simplificação) - CT3

Entrada	Valor
Rend. tributáveis recebidos PJ:	120000
Rend. isentos e não-tributáveis:	20000
Imposto pago (retido na fonte):	30000
Pagamentos efetuados (educação):	4000
Pagamentos efetuados (saúde):	9000
Número de dependentes:	2
Total a pagar ou a receber:	

IRPF 2021 (uma simplificação)

Entrada	Valor
Rend. tributáveis recebidos PJ:	12000
Rend. isentos e não-tributáveis:	2000
Imposto pago (retido na fonte):	3000
Pagamentos efetuados (educação):	4000
Pagamentos efetuados (saúde):	9000
Número de dependentes:	2
Total a pagar ou a receber:	

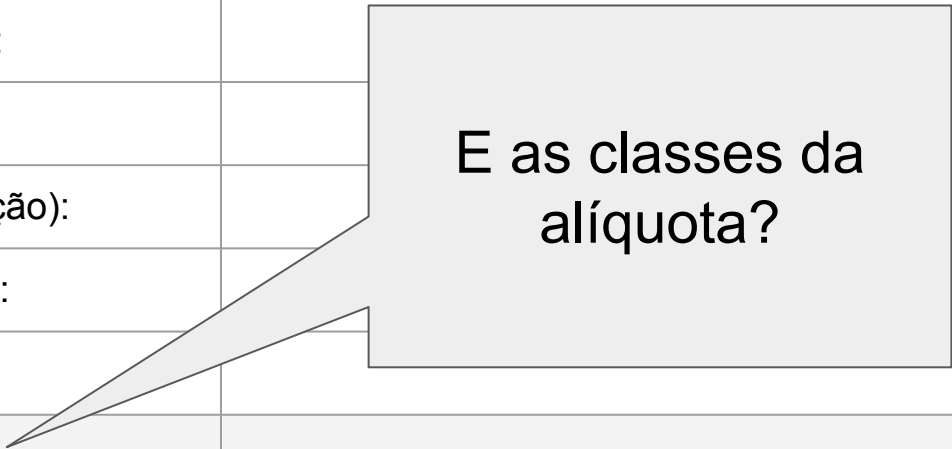
9000 vs 12000 vs 15000
(todos pertencem à mesma classe)

4500 / 6000 / 7500 por dependente
(acima do limite)

4000 ou 2000 por dependente
(outra classe)

Exemplo: IRPF 2021 (uma simplificação)

Entrada	Valor
Rend. tributáveis recebidos PJ:	
Rend. isentos e não-tributáveis:	
Imposto pago (retido na fonte):	
Pagamentos efetuados (educação):	
Pagamentos efetuados (saúde):	
Número de dependentes:	
Total a pagar ou a receber:	



E as classes da alíquota?

Exemplo: IRPF 2021 (uma simplificação)

Entrada	Valor		
	BASE DE CÁLCULO (R\$)	ALÍQUOTA (%)	PARCELA A DEDUZIR DO IRPF
Rend. tributável	Até 1.903,98	isento	isento
Rend. isento	De 1.903,99 até 2.826,65	7,5%	R\$142,80
Imposto pago	De 2.826,66 até 3.751,05	15%	R\$354,80
Pagamentos	De 3.751,06 até 4.664,68	22,5%	R\$636,13
Pagamentos	Acima de 4.664,68	27,5%	R\$869,36
Número de dependentes:			
Total a pagar ou a receber:			

Classes de Equivalência: algumas diretrizes

Quando há um intervalo de valores

- Uma classe válida para valores pertencentes ao intervalo
- Uma classe inválida para valores menores que o limite inferior
- Uma classe inválida para valores maiores que o limite superior

Classes de Equivalência: algumas diretrizes

Quando há uma lista de valores válidos:

- Uma classe válida para cada valor incluído na lista
- Uma classe inválida para os valores foras da lista

Classes de Equivalência: algumas diretrizes

Quando há restrições (expressão lógica; sintaxe; valor específico; compatibilidade com outras variáveis)

- Uma classe válida para os valores que satisfazem às restrições
- Uma classe inválida para os outros valores

Classes de Equivalência: algumas diretrizes

Quando a saída é uma lista de valores distintos:

- Uma classe válida para cada entrada que provoca cada uma das saídas da lista
- Uma classe inválida para valores que não provocam cada uma das saídas da lista

Análise de Valor-Limite

É um complemento ao critério Particionamento em Classes de Equivalência, sendo que os limites associados às condições de entrada são exercitados de forma mais rigorosa.

Observações da prática profissional mostram que grande parte dos erros ocorre nas fronteiras do domínio de entrada

Análise de Valor-Limite

Portanto, ao invés de selecionar qualquer elemento de uma classe, os casos de teste são escolhidos nas fronteiras das classes, pois nesses pontos se concentra um grande número de erros.

O espaço de saída do programa também é particionado e são exigidos casos de teste que produzam resultados nos limites dessas classes de saída.

Análise de Valor-Limite

Diretrizes:

- Se os limites da condição de entrada forem a e b, projetar casos de teste para os valores de e imediatamente acima e abaixo de a e b.
- Se uma condição de entrada especifica vários valores, projetar casos de teste para os valores imediatamente acima e abaixo do valor mínimo e do valor máximo.
- Se as estruturas de dados internas do programa têm limites definidos na especificação do programa, deve-se criar casos de teste para exercitar a estrutura de dados no seu limite.

Exemplo 1 - Triângulo

Escreva casos de teste para testar um algoritmo que recebe como entrada três números inteiros positivos que representam os tamanhos dos três lados de um triângulo, e retorna o tipo de triângulo formado (Equilátero, isósceles ou escaleno).

Se algum dos lados for negativo, o algoritmo deve informar que houve um erro porque um dos lados não tem valor válido

Se um dos lados tiver tamanho maior do que a soma dos outros dois lados, então o algoritmo deve informar que esses valores não são suficientes para formar um triângulo

Exemplo 1 - Triângulo

Classes de Equivalência

Entrada	Classes válidas	Classes inválidas
L1	$L1 > 0$ (C1)	$L1 \leq 0$ (C7)

Exemplo 1 - Triângulo

Classes de Equivalência

Entrada	Classes válidas	Classes inválidas
L1	$L1 > 0$ (C1)	$L1 \leq 0$ (C7)
L2	$L2 > 0$ (C2)	$L2 \leq 0$ (C8)

Exemplo 1 - Triângulo

Classes de Equivalência

Entrada	Classes válidas	Classes inválidas
L1	$L1 > 0$ (C1)	$L1 \leq 0$ (C7)
L2	$L2 > 0$ (C2)	$L2 \leq 0$ (C8)
L3	$L3 > 0$ (C3)	$L3 \leq 0$ (C9)

Exemplo 1 - Triângulo

Classes de Equivalência

Entrada	Classes válidas	Classes inválidas
L1	$L1 > 0$ (C1)	$L1 \leq 0$ (C7)
L2	$L2 > 0$ (C2)	$L2 \leq 0$ (C8)
L3	$L3 > 0$ (C3)	$L3 \leq 0$ (C9)
L1,L2,L3	Equilátero (C4): $L1 == L2 == L3$ Isósceles (C5): $L1 == L2 \neq L3$, $L2 == L3 \neq L1$, $L1 == L3 \neq L2$ Escaleno (C6): $L1 \neq L2 \neq L3 \neq L1$	

Exemplo 1 - Triângulo

Classes de Equivalência

Entrada	Classes válidas	Classes inválidas
L1	$L1 > 0$ (C1)	$L1 \leq 0$ (C7)
L2	$L2 > 0$ (C2)	$L2 \leq 0$ (C8)
L3	$L3 > 0$ (C3)	$L3 \leq 0$ (C9)
L1,L2,L3	Equilátero (C4): $L1 == L2 == L3$ Isósceles (C5): $L1 == L2 \neq L3$, $L2 == L3 \neq L1$, $L1 == L3 \neq L2$ Escaleno (C6): $L1 \neq L2 \neq L3 \neq L1$	Não forma triângulo (C10): $L1 \geq L2 + L3$ $L2 \geq L1 + L3$ $L3 \geq L1 + L2$

Exemplo 1 - Triângulo

Entrada	Classes válidas	Classes inválidas
L1	$L1 > 0$ (C1)	$L1 \leq 0$ (C7)
L2	$L2 > 0$ (C2)	$L2 \leq 0$ (C8)
L3	$L3 > 0$ (C3)	$L3 \leq 0$ (C9)
L1,L2,L3	$L1 < L2 + L3$ (C11) $L2 < L1 + L3$ $L3 < L1 + L2$	Não forma triângulo (C10): $L1 \geq L2 + L3$ $L2 \geq L1 + L3$ $L3 \geq L1 + L2$
Equilátero	$L1 = L2$ e $L2 = L3$ (C4)	
Isósceles	$L1 = L2 \neq L3$, $L2 = L3 \neq L1$, $L1 = L3 \neq L2$ (C5)	
Escaleno	$L1 \neq L2 \neq L3 \neq L1$ (C6)	

Exemplo 1 - Triângulo

Classes de Equivalência

Entrada	Classes válidas	Classes inválidas
L1	$L1 > 0$ (C1)	$L1 \leq 0$ (C7)
L2	$L2 > 0$ (C2)	$L2 \leq 0$ (C8)
L3	$L3 > 0$ (C3)	$L3 \leq 0$ (C9)
L1,L2,L3	Equilátero (C4): $L1 == L2 == L3$ Isósceles (C5): $L1 == L2 \neq L3$, $L2 == L3 \neq L1$, $L1 == L3 \neq L2$ Escaleno (C6): $L1 \neq L2 \neq L3 \neq L1$	Não forma triângulo (C10): $L1 \geq L2 + L3$ $L2 \geq L1 + L3$ $L3 \geq L1 + L2$

Exemplo 1 - Triângulo

[illegible]

Exemplo 1 - Triângulo

[illegible]

Exemplo 1 - Triângulo

[illegible]

Exemplo 1 - Triângulo

[illegible]

Exemplo 1 - Triângulo

[illegible]

Exemplo 1 - Triângulo

ID	Entrada	Saída esperada	Classe de equivalência
1	5,5,5	Equilátero	C1,C2,C3,C4
2	5,5,6	Isósceles	C1,C2,C3,C5.1
3	6,5,5	Isósceles	C1,C2,C3,C5.2
4	6,5,6	Isósceles	C1,C2,C3,C5.3
5	4,5,6	Escaleno	C1,C2,C3,C6
6	-2,4,5	ERRO – valor inválido	C7

Exemplo 1 - Triângulo

ID	Entrada	Saída esperada	Classe de equivalência
1	5,5,5	Equilátero	C1,C2,C3,C4
2	5,5,6	Isósceles	C1,C2,C3,C5.1
3	6,5,5	Isósceles	C1,C2,C3,C5.2
4	6,5,6	Isósceles	C1,C2,C3,C5.3
5	4,5,6	Escaleno	C1,C2,C3,C6
6	-2,4,5	ERRO – valor inválido	C7
7	4,-2,5	ERRO – valor inválido	C8

Exemplo 1 - Triângulo

ID	Entrada	Saída esperada	Classe de equivalência
1	5,5,5	Equilátero	C1,C2,C3,C4
2	5,5,6	Isósceles	C1,C2,C3,C5.1
3	6,5,5	Isósceles	C1,C2,C3,C5.2
4	6,5,6	Isósceles	C1,C2,C3,C5.3
5	4,5,6	Escaleno	C1,C2,C3,C6
6	-2,4,5	ERRO – valor inválido	C7
7	4,-2,5	ERRO – valor inválido	C8
8	4,5,-2	ERRO – valor inválido	C9
9	10, 2, 3	Não forma triângulo	C10.1
10	2, 10, 3	Não forma triângulo	C10.2
11	2, 3, 10	Não forma triângulo	C10.3

Exemplo 1 - Triângulo

Análise de Valor-Limite

Entrada	Classes válidas	Classes inválidas
L1	L1=1 (v1)	L1=0 (v5), L1=-1 (v6)
L2	L2=1 (v2)	L2=0 (v7), L2=-1 (v8)
L3	L3=1 (v3)	L3=0 (v9), L3=-1 (v10)

Exemplo 1 - Triângulo

Análise de Valor-Limite

Entrada	Classes válidas	Classes inválidas
L1	L1=1 (v1)	L1=0 (v5), L1=-1 (v6)
L2	L2=1 (v2)	L2=0 (v7), L2=-1 (v8)
L3	L3=1 (v3)	L3=0 (v9), L3=-1 (v10)
L1,L2,L3	Equilátero (1,1,1) (v4)	Não forma triângulo: L1 = L2 + L3 (v11) L2 = L1 + L3 (v12) L3 = L1 + L2 (v13)

Exemplo 1 - Triângulo

ID	Entrada	Saída esperada	Valor-limite
12	0, 3, 4	ERRO - valor inválido	v5
13	-1, 3, 4	ERRO - valor inválido	v6
14	1, 2, 2	Isósceles	v1
15	3, 0, 4	ERRO - valor inválido	v7
16	3, -1, 4	ERRO - valor inválido	v8
17	2, 1, 2	Isósceles	v2
18	2, 3, 0	ERRO - valor inválido	v9
19	2, 3, -1	ERRO - valor inválido	v10
20	2, 2, 1	Isósceles	v3
21	1, 1, 1	Equilátero	v4
22	5, 2, 3	Não forma triângulo	v11

Exemplo 1 - Triângulo

[illegible]

Análise de causa-efeito

Partição em classes de equivalência e Análise de Valores-Limite:

- Não levam em conta combinações de valores difícil testar situações em que diferentes combinações levam a diferentes saídas do sistema

Análise de causa-efeito

Útil quando especificações são representadas como estruturas de decisão:

- Conjuntos de condições sobre valores de entradas e as ações correspondentes do sistema
- Causas: condições de entrada (valor lógico)
- Efeitos: ações realizadas em resposta às diferentes condições de entrada

Modelos de teste:

- tabelas de decisão

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Condições

$L1 < L2 + L3$	F	V	V	V	V	V	V	V	V	V	V
$L2 < L1 + L3$		F	V	V	V	V	V	V	V	V	V
$L3 < L2 + L3$			F	V	V	V	V	V	V	V	V
$L1 = L2$				V	V	V	V	F	F	F	F
$L2 = L3$				V	V	F	F	V	V	F	F
$L1 = L3$				V	F	V	F	V	F	V	F

Saídas

Não é triângulo	X	X	X								
Isósceles											
Equilátero				X							
Escaleno											
Impossível					X						

Condições

$L1 < L2 + L3$	F	V	V	V	V	V	V	V	V	V	V
$L2 < L1 + L3$		F	V	V	V	V	V	V	V	V	V
$L3 < L2 + L3$			F	V	V	V	V	V	V	V	V
$L1 = L2$				V	V	V	V	F	F	F	F
$L2 = L3$				V	V	F	F	V	V	F	F
$L1 = L3$				V	F	V	F	V	F	V	F

Saídas

Não é triângulo	X	X	X								
Isósceles											
Equilátero				X							
Escaleno											
Impossível					X	X					

Condições

$L1 < L2 + L3$	F	V	V	V	V	V	V	V	V	V	V
$L2 < L1 + L3$		F	V	V	V	V	V	V	V	V	V
$L3 < L2 + L3$			F	V	V	V	V	V	V	V	V
$L1 = L2$				V	V	V	V	F	F	F	F
$L2 = L3$				V	V	F	F	V	V	F	F
$L1 = L3$				V	F	V	F	V	F	V	F

Saídas

Não é triângulo	X	X	X								
Isósceles							X				
Equilátero				X							
Escaleno											
Impossível					X	X					

Condições

$L1 < L2 + L3$	F	V	V	V	V	V	V	V	V	V	V
$L2 < L1 + L3$		F	V	V	V	V	V	V	V	V	V
$L3 < L2 + L3$			F	V	V	V	V	V	V	V	V
$L1 = L2$				V	V	V	V	F	F	F	F
$L2 = L3$				V	V	F	F	V	V	F	F
$L1 = L3$				V	F	V	F	V	F	V	F

Saídas

Não é triângulo	X	X	X								
Isósceles							X				
Equilátero				X							
Escaleno											
Impossível					X	X		X			

Condições

$L1 < L2 + L3$	F	V	V	V	V	V	V	V	V	V	V
$L2 < L1 + L3$		F	V	V	V	V	V	V	V	V	V
$L3 < L2 + L3$			F	V	V	V	V	V	V	V	V
$L1 = L2$				V	V	V	V	F	F	F	F
$L2 = L3$				V	V	F	F	V	V	F	F
$L1 = L3$				V	F	V	F	V	F	V	F

Saídas

Não é triângulo	X	X	X								
Isósceles							X		X		
Equilátero				X							
Escaleno											
Impossível					X	X		X			

Condições

$L1 < L2 + L3$	F	V	V	V	V	V	V	V	V	V	V
$L2 < L1 + L3$		F	V	V	V	V	V	V	V	V	V
$L3 < L2 + L3$			F	V	V	V	V	V	V	V	V
$L1 = L2$				V	V	V	V	F	F	F	F
$L2 = L3$				V	V	F	F	V	V	F	F
$L1 = L3$				V	F	V	F	V	F	V	F

Saídas

Não é triângulo	X	X	X								
Isósceles							X		X	X	
Equilátero				X							
Escaleno											
Impossível					X	X		X			

Condições

$L1 < L2 + L3$	F	V	V	V	V	V	V	V	V	V	V
$L2 < L1 + L3$		F	V	V	V	V	V	V	V	V	V
$L3 < L2 + L3$			F	V	V	V	V	V	V	V	V
$L1 = L2$				V	V	V	V	F	F	F	F
$L2 = L3$				V	V	F	F	V	V	F	F
$L1 = L3$				V	F	V	F	V	F	V	F

Saídas

Não é triângulo	X	X	X								
Isósceles							X		X	X	
Equilátero				X							
Escaleno											X
Impossível					X	X		X			

Benefícios da tabela de decisão

Facilita a determinação de quais testes aplicar.

Permite que se analise a especificação para determinar:

- Redundâncias: duas regras iguais, i.e, mesmas causas levando aos mesmos efeitos
- Contradições: duas regras com as mesmas causas levando a efeitos diferentes
- Omissões: não há regras para todas as combinações de causas.
- Redundâncias e contradições não são necessariamente erros: podem indicar concorrência.
- Omissões podem indicar situações irrelevantes ou até mesmo impossíveis

Próxima parte da aula

Atividades práticas

ACH 2028

Qualidade de Software

Aula 07 - Teste de Software

Conceitos Básicos e Teste Funcional

Prof. Marcelo Medeiros Eler
marceloeler@usp.br