



Critérios de Teste Funcionais

Prof. Thiago
Departamento de Informática e Matemática
Aplicada - DIMAp

Objetivo

Apresentar critérios de teste
funcionais a partir de exemplos.

Sabemos que...

Os testes exaustivos são caros e
até inviáveis...

Então...

Testar → “escolher” dados de
testes

Critérios de Teste

Critérios de Teste = critérios de escolha.

Guiam na escolha
das entradas com
****maiores chances****
de revelar falhas



Já existem...

... critérios para escolher dados
de teste de forma a aumentar
as chances de revelar as falhas

Tipos de Critérios

Caixa Preta
(Funcional)

Caixa
Branca
(Estrutural)

Baseado
em
Defeitos

Critérios de Cobertura

- Alguns termos ocasionalmente usados no contexto de teste de software “**teste exaustivo**”, “**cobertura total**”.
- Como já vimos o número de entradas para a maioria dos programas é grande o suficiente para considerarmos infinitas.

Critérios de Cobertura

- Considera um compilador Java, o número de entradas em potencial não são somente todos os programas em Java, ou todos os programas corretos, mas também todas as strings.
- Assim o número de entradas não pode ser enumerado explicitamente.

Critérios de Cobertura

- Daí surgem os **critérios de teste**.
- Uma vez que não podemos testar todas as entradas os critérios ajudam a decidir quais entradas usar no teste.

Critérios de Cobertura

- Primeiro precisamos definir
- [Requisito de teste]: um requisito de teste é um elemento específico de um artefato de software que um caso de teste deve satisfazer ou cobrir.

Critérios de Cobertura

- Requisitos de teste podem ser descritos com respeito a uma variedade de artefatos de software.
- Código fonte
- Componente do design
- Elementos da modelagem de especificação
- Descrições do espaço de entrada

Critérios de Cobertura

- Exemplo:
- Cobrir todas as decisões do programa (cobertura de ramos), assim temos dois requisitos de teste, o que a decisão avalia para falso e a que avalia para verdadeiro.

Critérios de Cobertura

- [Critério de Cobertura]: um critério de cobertura é uma regra ou um conjunto de regras que impõem requisitos de teste em um conjunto de teste.
- Isto é, o critério descreve o requisito de teste de maneira completa e não ambígua.

Critérios de Cobertura

- Precisamos de uma maneira de saber o quanto bom uma coleção de teste é.
- Para isso podemos medir um conjunto de teste com relação a um critério em termos de **cobertura**.

Critérios de Cobertura

- [Cobertura]: dado um conjunto de requisitos de teste TR para um critério de cobertura C, um conjunto de testes T satisfaaz C se e somente se para cada requisito de teste tr em TR, existe pelo menos um teste t em T tal que t satisfaaz tr.

Critérios de Cobertura

- Cobertura é importante por dois motivos.
- As vezes satisfazer um critério de cobertura pode ser caro, então procura-se atingir um certo nível de cobertura.

Critérios de Cobertura

- [Nível de Cobertura]: dado um conjunto de requisitos de teste TR e um conjunto de teste T o nível de cobertura é simplesmente a razão entre o número de requisitos satisfeitos pela quantidade de requisitos.

Critérios de Cobertura

- Infelizmente, o uso dessas estratégias podem levar a mal-entendidos.
- Se os nossos testes não alcançam 100% de cobertura?
- 99% é pior que 100% de cobertura, ou 90% ou 75%?

Critérios de Cobertura

- Características de um bom critério de cobertura:
- A dificuldade de computar os requisitos de teste.
- A dificuldade de gerar os testes.
- Quão bem os testes revelam falhas.

Critérios de Teste

- Maneira sistemática e planejada de elaborar os casos de teste
- Podem ser usados de duas formas:
 - * Para seleção dos casos de teste
 - * Para adequação dos casos de teste

Tipos de Critérios

Caixa Preta
(Funcional)

Caixa
Branca
(Estrutural)

Baseado
em
Defeitos

Teste Funcional

- Os requisitos de teste são extraídos da especificação do programa
- Aborda o software de um ponto de vista macroscópico. Por isso, é também chamado Teste Caixa Preta

Teste Funcional

- Problema:

Dificuldade em quantificar a atividade de teste - não se pode garantir que partes essenciais ou críticas do software foram executadas

Teste Funcional

- Critérios:

- * Particionamento em classes de Equivalência
- * Análise do Valor Limite

Particionamento de Equivalência

- Divide o domínio de entrada em classes ou partições de equivalência que podem ser tratadas da mesma maneira.
- Observar também a saída do programa e verificar se, com base na saída, é possível estabelecer classes no domínio de entrada que permitam avaliar se a saída está sendo produzida corretamente.

Particionamento de Equivalência

- As classes podem ser **válidas** ou **inválidas**.



Particionamento de Equivalência

- Diretrizes para definição das classes:
- se a condição de entrada especifica um intervalo.
- se a condição de entrada exige um valor específico.
- se a condição de entrada especifica um membro de um conjunto.
- se a condição de entrada for booleana.

Particionamento de Equivalência

- Observações
- reduz o tamanho do domínio de entrada
- concentra-se em criar casos de teste baseados unicamente na especificação
- e especialmente adequado para aplicações em que as variáveis de entrada podem ser facilmente identificadas e podem ter valores distintos.

Particionamento de Equivalência

- Problemas
- embora a especificação possa sugerir que um grupo de dados seja processado de forma idêntica, isso pode não ocorrer.

Análise do Valor Limite

- Complementa o Particionamento de Equivalência
- Coloca sua atenção em uma fonte propícia a defeitos - os limites de uma classe ou partição de equivalência.

Análise do Valor Limite

- Os valores limites das classes é que devem ser selecionados: 0, 1, 20 e 21
- Os caracteres a serem encontrados devem estar na 1^a e na última posição

Teste Estrutural

- Os requisitos de teste são extraídos de uma implementação em particular
- Teste dos detalhes procedimentais.
Por isso é também chamado Teste Caixa Branca.

Teste Estrutural

- A maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como grafo de programa ou grafo de fluxo de controle.
- Permite uma avaliação de cobertura do código
- Problema: caminhos não executáveis

Grafo de Fluxo de Controle

- Consiste de nós conectados por arcos orientados que mostram sua direção
- os nós representam blocos de comandos.
 - * bloco de comando: é um conjunto de comandos tal que se o primeiro comando for executado, então todos os comandos subsequentes também serão

Grafo de Fluxo de Controle

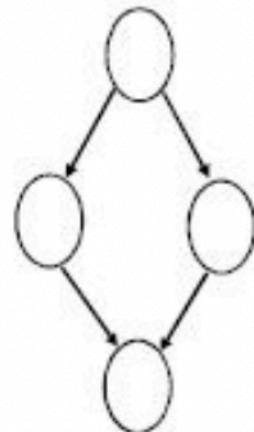
- Os arcos indicam precedência, ou transferência de controle
- essa representação permite que o programa seja examinado independentemente de sua função

Grafo de Fluxo de Controle

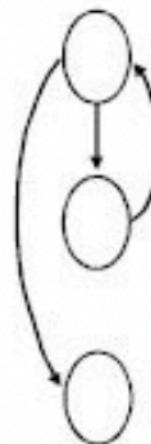
seqüência



if



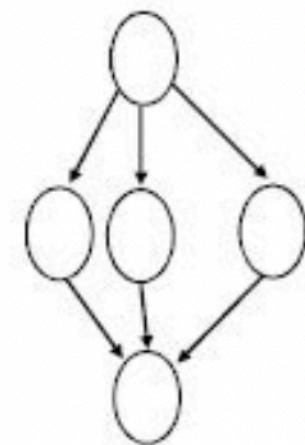
while



repeat

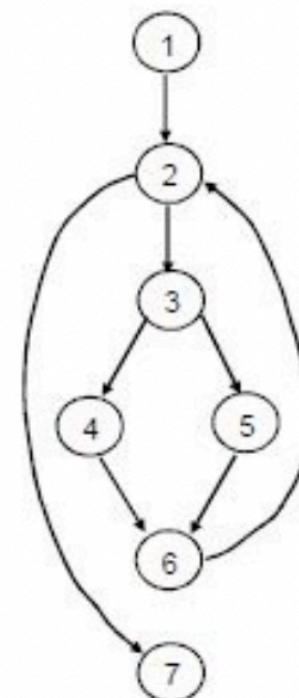


case



Grafo de Fluxo de Controle

```
início
    leia nro
    enquanto nro ≠ 0
        se nro > 0
            raiz = raiz-quadrada(nro)
            escreva raiz
        senão
            escreva mensagem de erro
        fim-se
    leia nro
    fim-enqto
fim }
```



Critérios de Teste Estrutural

- Todos os Nós: Estabelece como requisito de teste que sejam executados todos os comandos do programa **ao menos uma vez**
- Todos os Ramos: Estabelece como requisito de teste que sejam executadas todas as saídas verdadeiro e falso de **todas as decisões**

Critérios de Teste Estrutural

- Todos-Caminhos: - Esse critério determina um conjunto básico de caminhos linearmente independentes, de modo que executando-os se garante a execução de todos os ramos ao menos uma vez.
- Um caminho linearmente independente é aquele que contém ao menos um novo nó

Critérios de Teste Estrutural

- O número de caminhos é determinado pela fórmula da Complexidade Ciclomática de Mc'Cabe:

$$V(G) = a - n + p$$

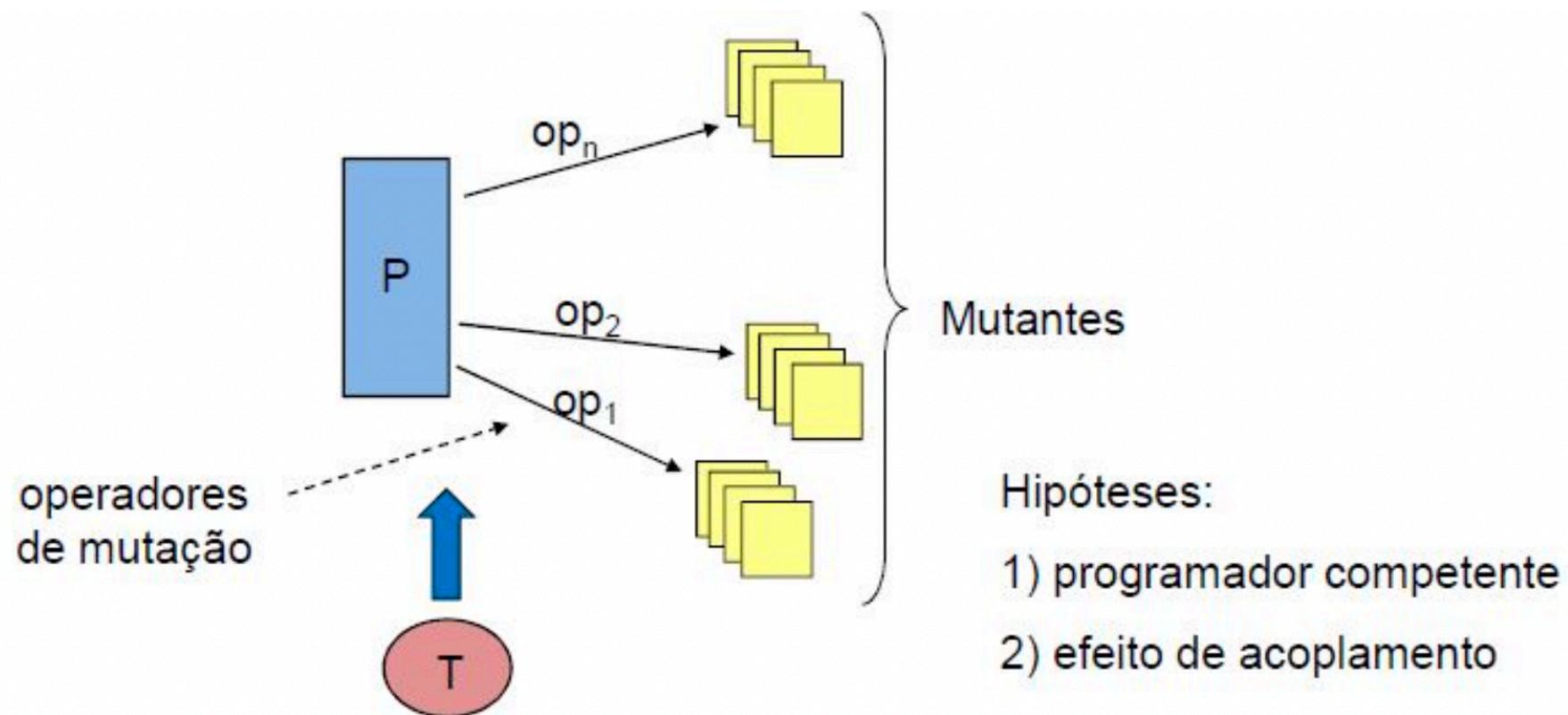
Teste Baseado em Defeitos

- Os requisitos de teste são estabelecidos com base nos defeitos típicos e comuns cometidos durante o desenvolvimento do software.

Teste Baseado em Defeitos

- Hipótese do programador competente:
Programadores experientes escrevem
programas corretos ou muito próximos do
correto.
- Efeito do acoplamento: Casos de teste
capazes de revelar erros simples são tão
sensíveis que, implicitamente também são
capazes de revelar erros mais complexos.

Teste Baseado em Defeitos



$$\text{escore de mutação} = \frac{\# \text{ mutantes mortos}}{\# \text{ mutantes gerados não equivalentes}}$$

Operadores de Mutação

- Retira um comando de cada vez do programa.
- Troca um operador relacional por outro tipo de operador relacional.
- Troca o comando while por do-while.
- Interrompe a execução do laço após duas execuções.
- Troca uma constante por outra constante.

Análise de Mutantes

- Dados P (programa) e T (testes)
- Passos para a aplicação da Análise de Mutantes
 - P é executado com os casos de teste de T
 - Mutantes são gerados
 - Mutantes são executados com os casos de teste de T
 - Mutantes são analisados

Análise de Mutantes

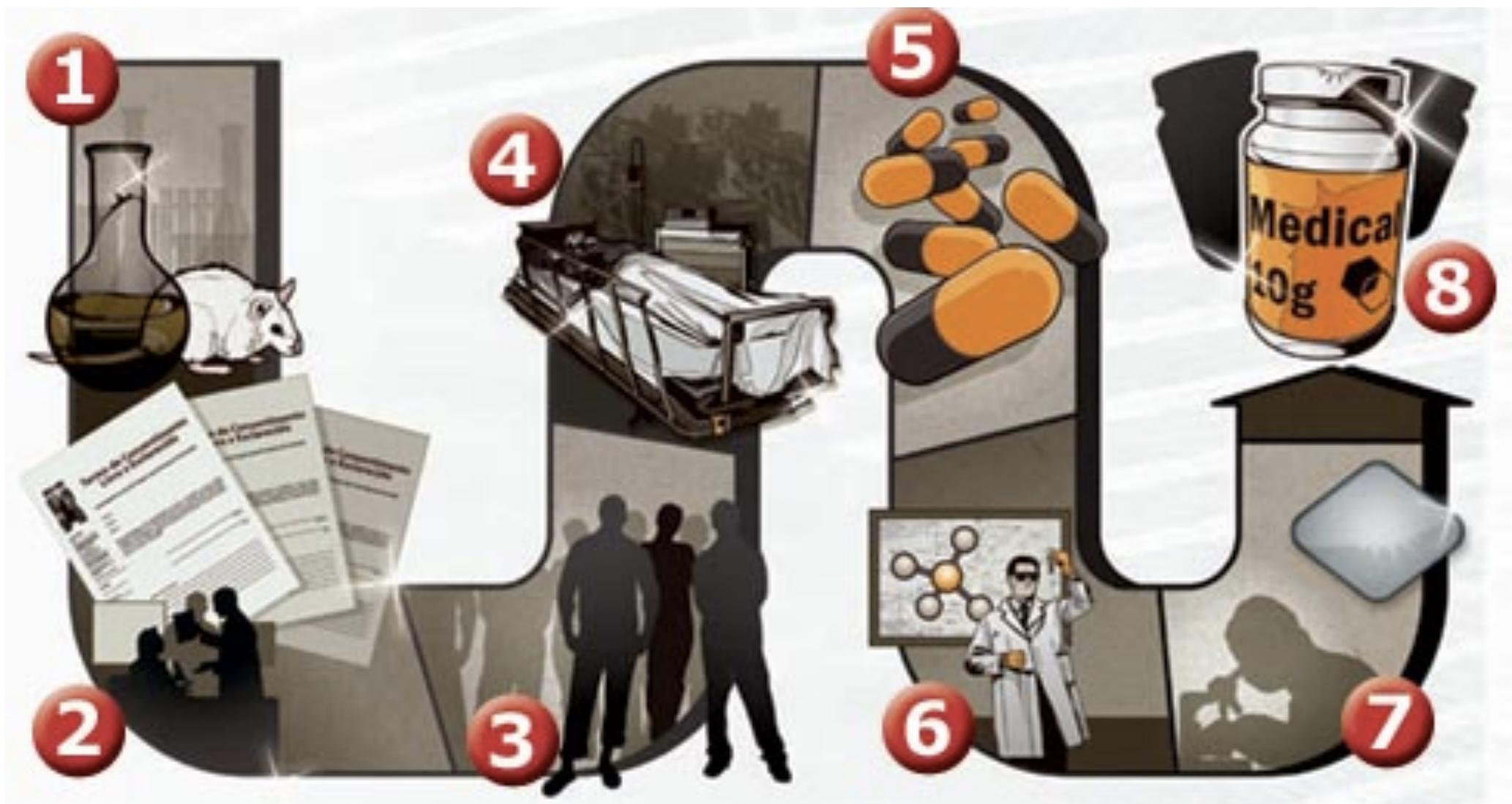
- Esse critério força o testador a analisar cuidadosamente o programa, uma vez que ele precisa criar casos de teste que exponham os defeitos introduzidos.
- Desvantagem: é computacionalmente caro devido ao grande número de mutantes gerados, o tempo e recursos usados para compilar e executar todos os eles

Critérios Funcionais

- Partição em Classes de Equivalência
- Análise do Valor Limite

Particionamento em Classes de Equivalência

Fazendo um Paralelo...



Objetivo: Testar um novo medicamento





População

Como testar??

Opcão 1: Amostra aleatória (Random)



Opcão 1: Amostra aleatória (Random)



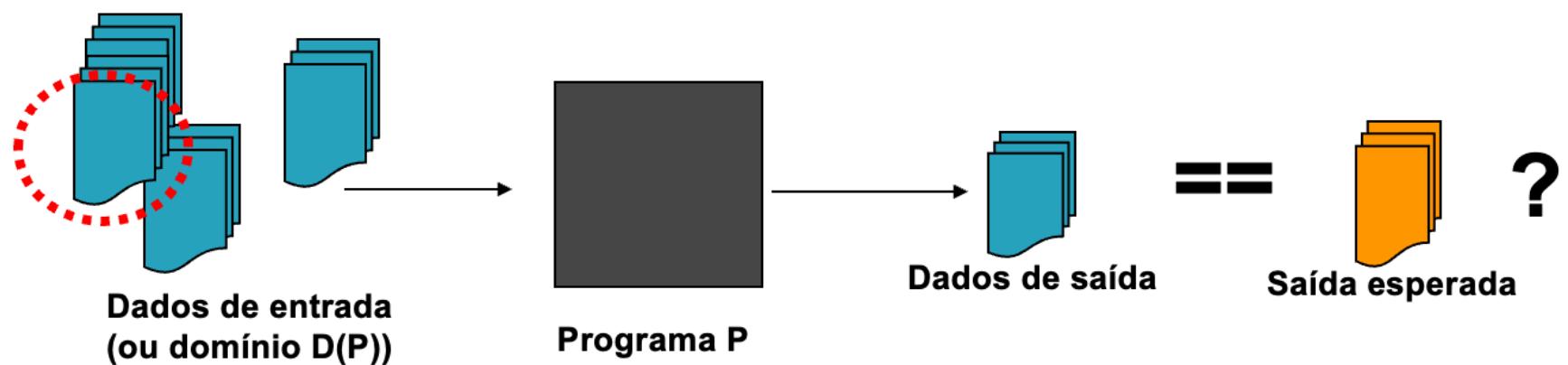
Qual
Limitação???

Opcão 1: Amostra aleatória (Random)

Um efeito colateral pode se manifestar só numa faixa etária.

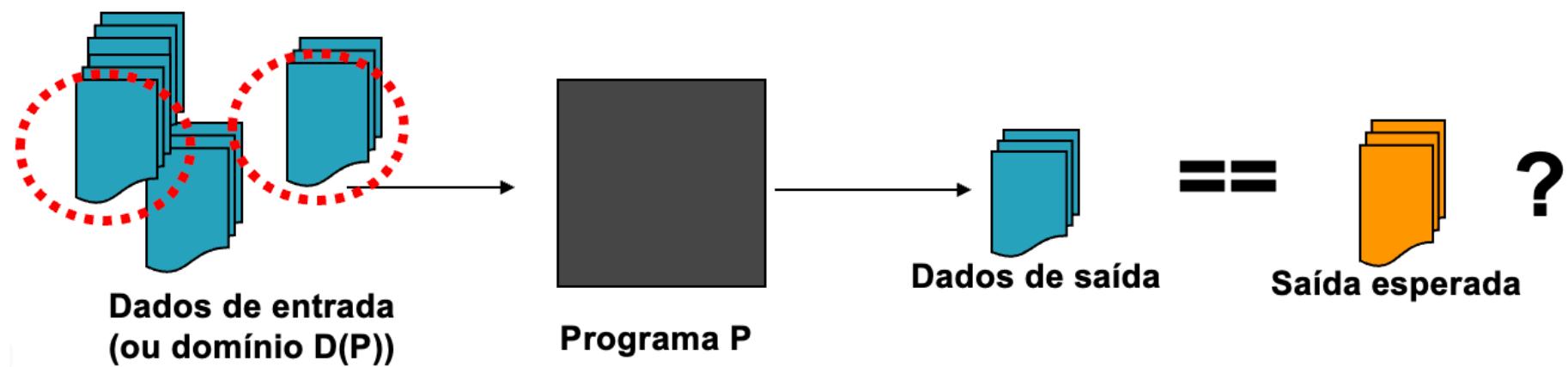


Random Testing



- Estratégia 1: Usar valores quaisquer aleatórios (Random Testing)

Problema do Random Testing



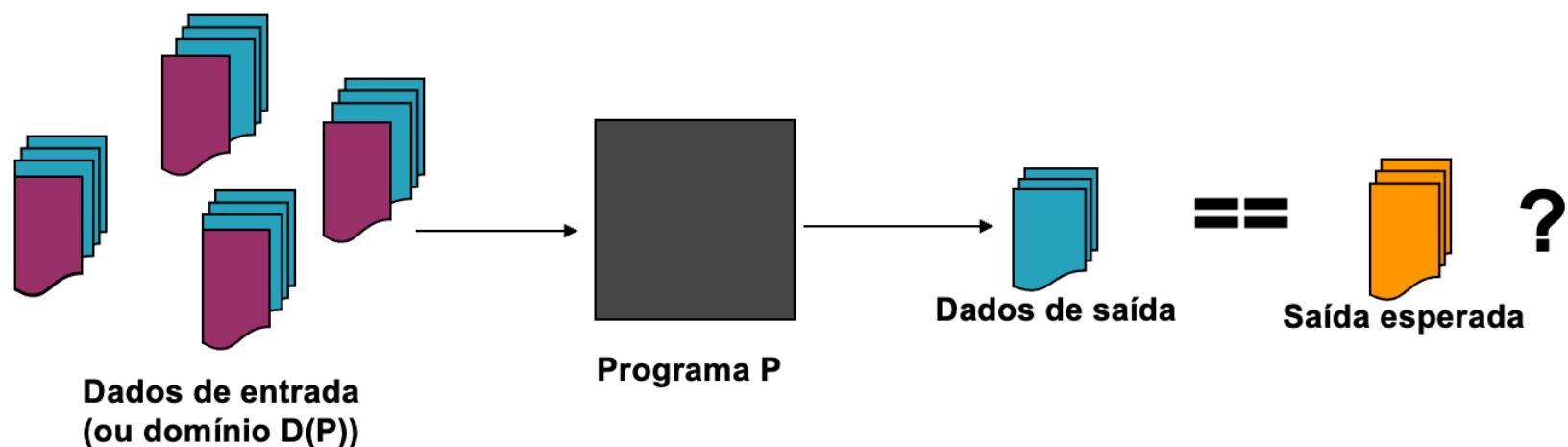
- Estudos tem mostrado que esta é a forma menos efetiva.
- Não é sistemática.

Opção 2: Definir Classes de Equivalência

1. Define Características
2. Classifica a população de acordo com elas
3. Seleciona representantes de cada "classe".

Consequência

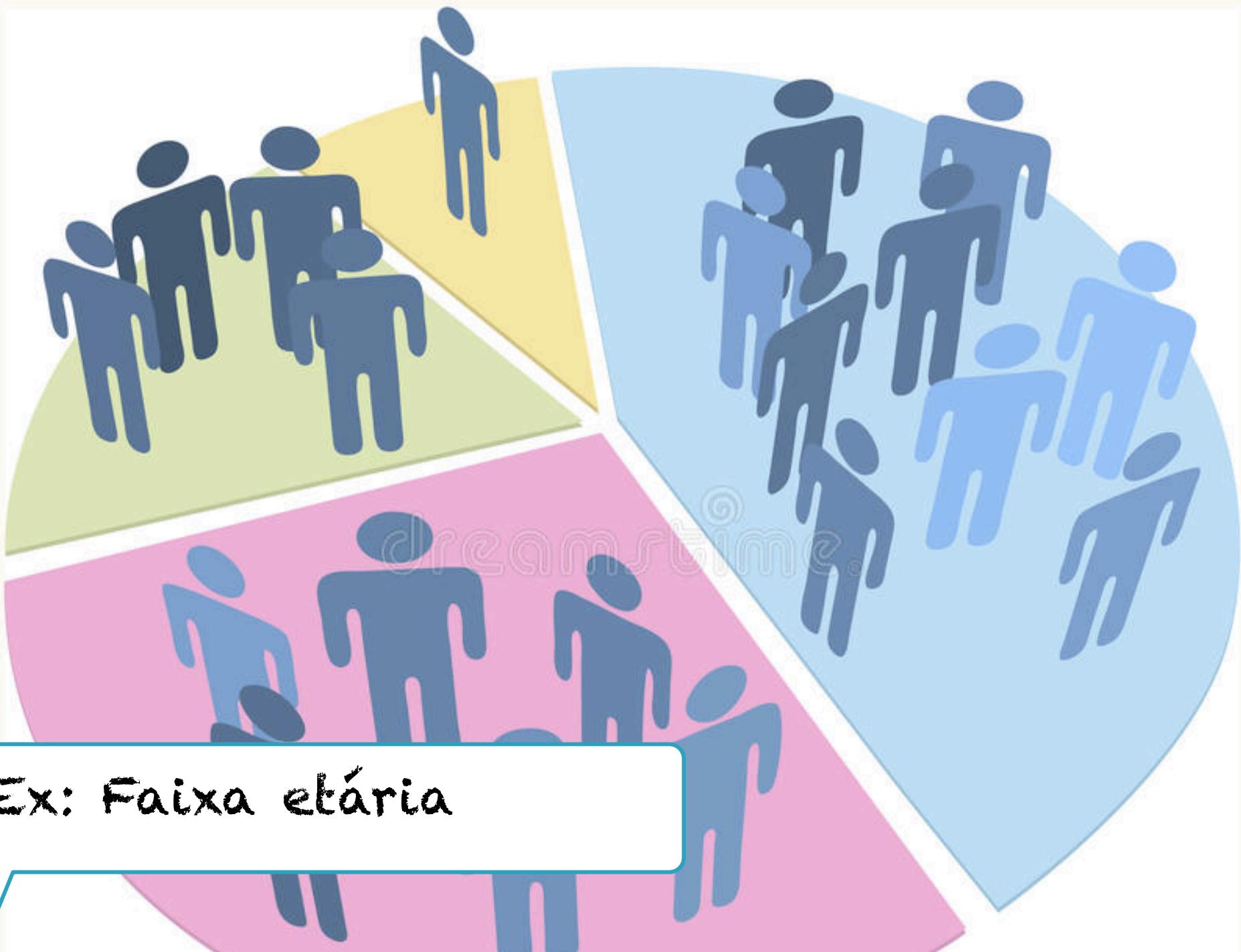
- Durante os testes
 - * selecionamos um ou mais dados de cada partição.



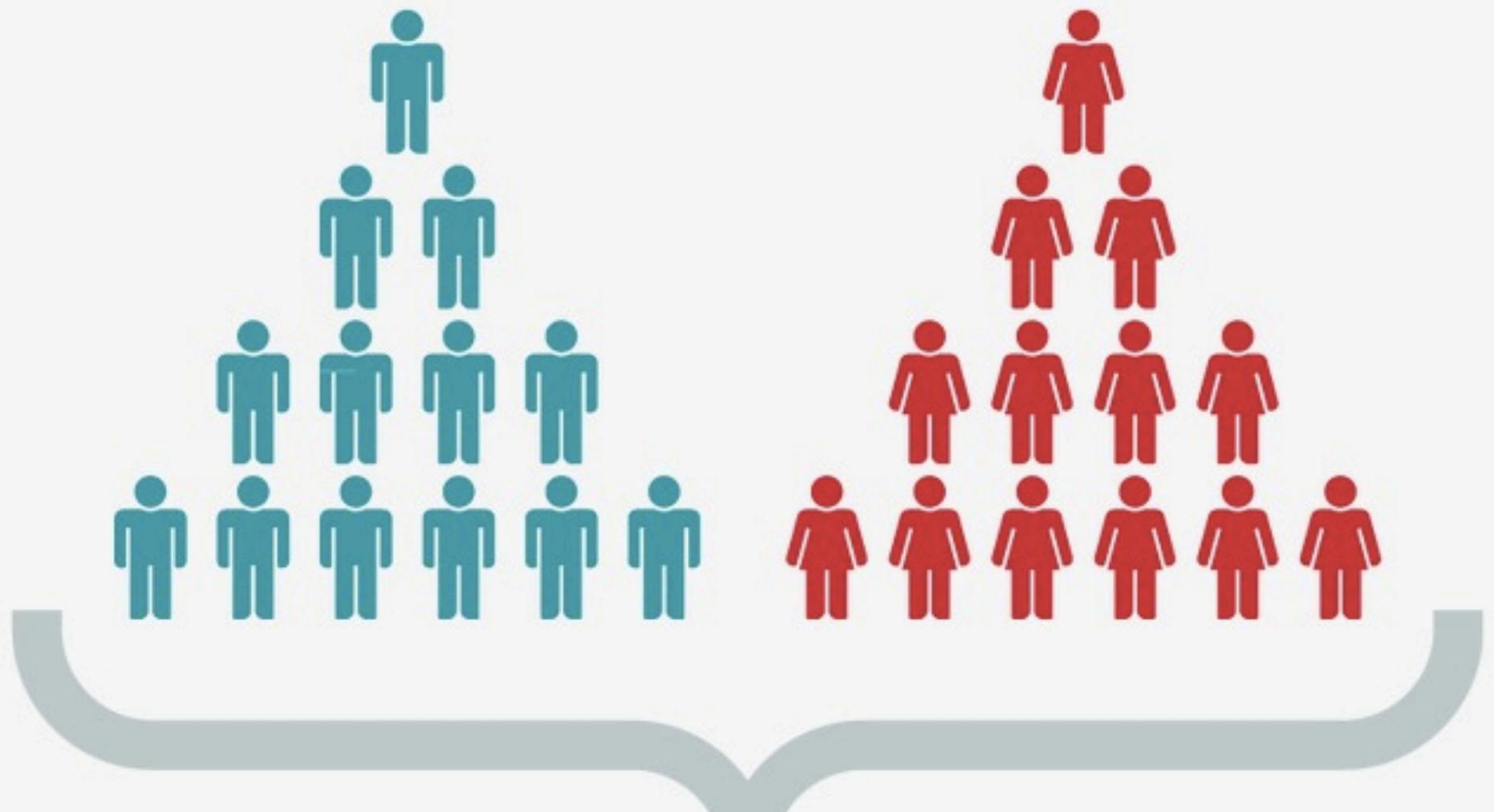
- Critério de parada: cada partição deve ser considerada ao menos 1 vez.

Opção 2: Definir Classes de Equivalência

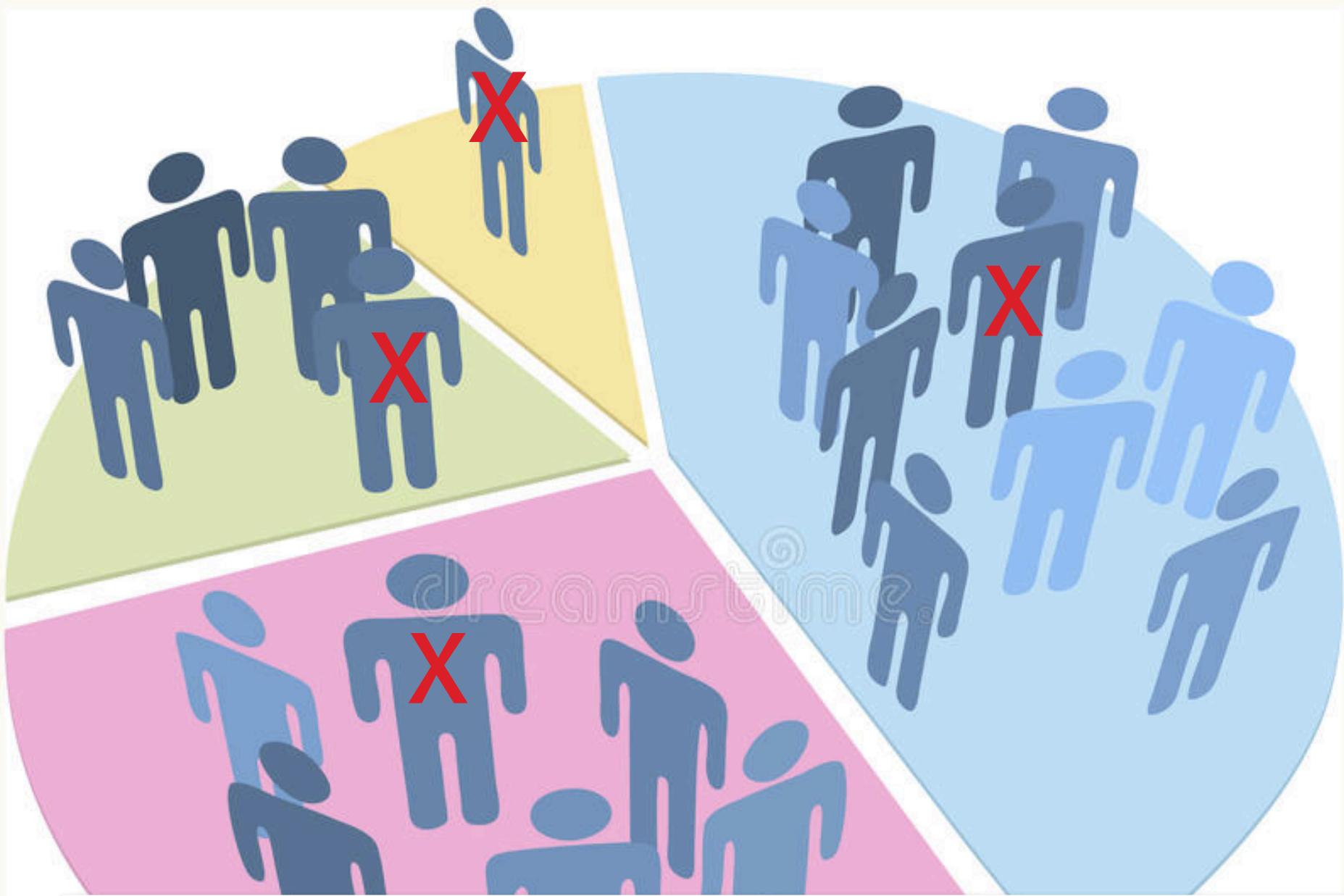
Defino classes de pessoas equivalentes (de acordo com alguma característica).



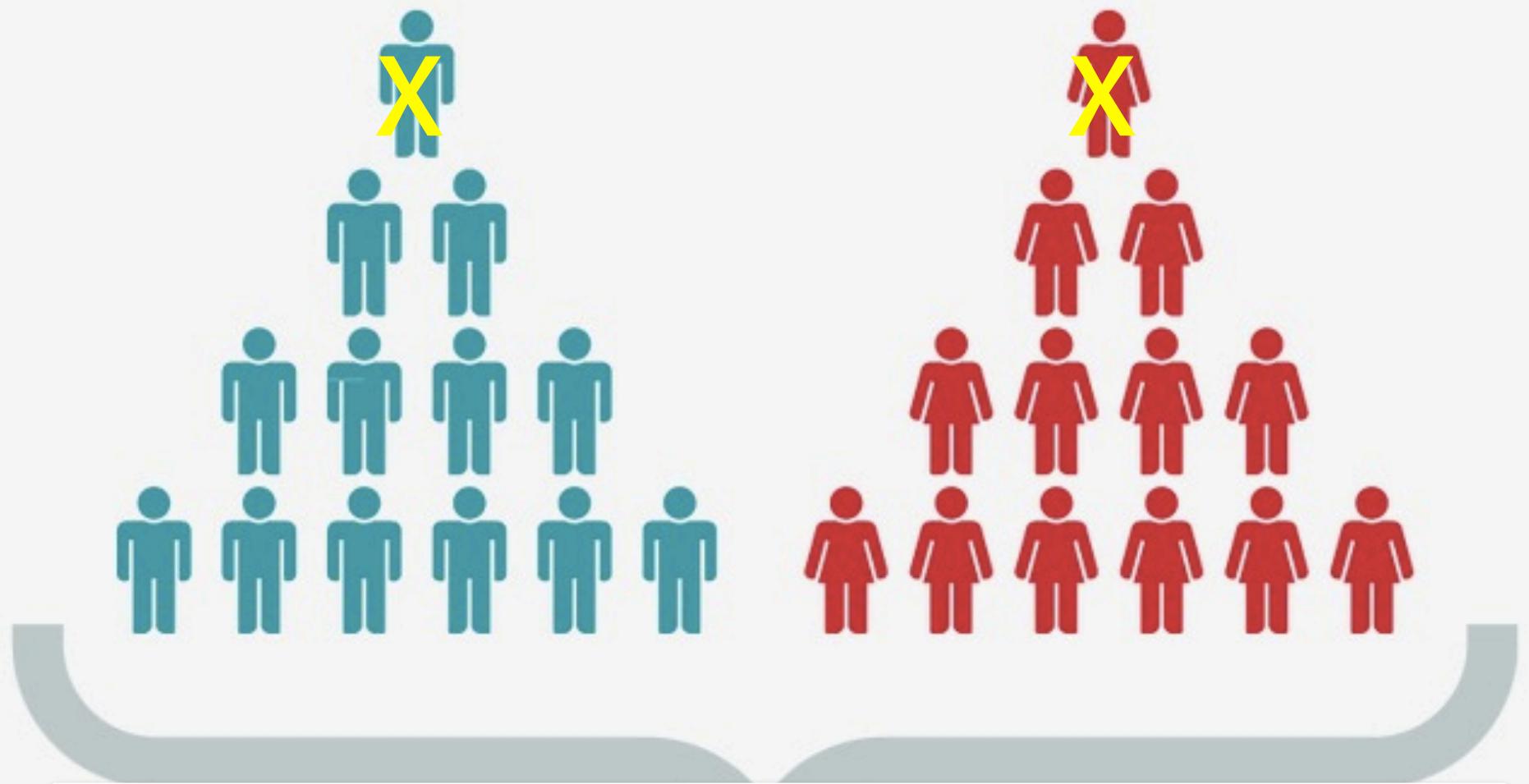
Ex: Faixa etária



Ex: Feminino e Masculino



E testar 1 representante de cada classe

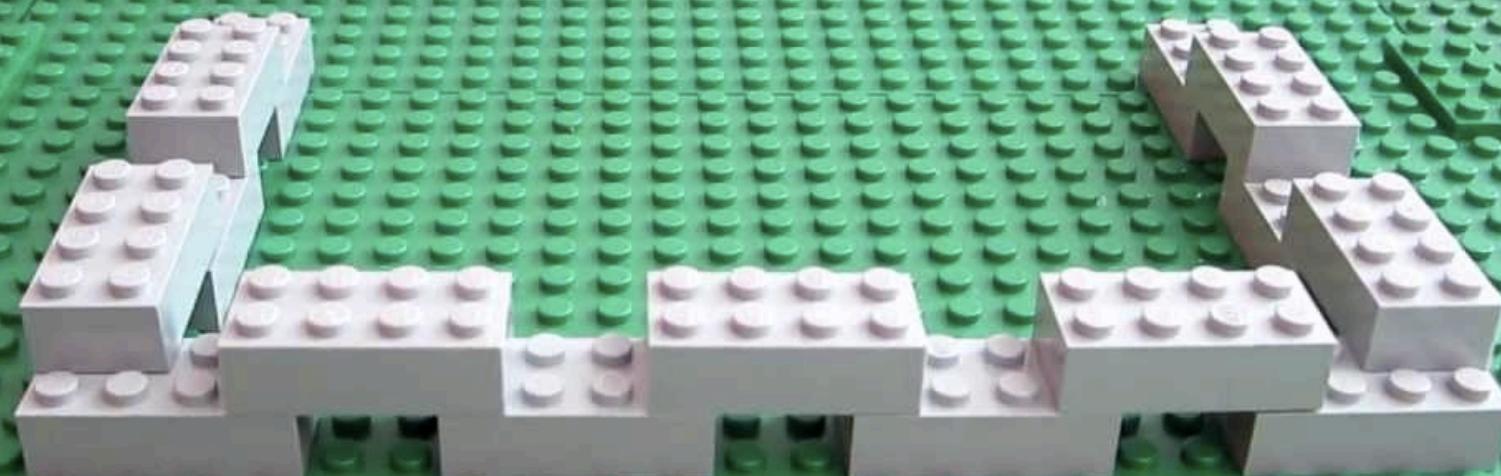


E testar 1 representante de cada classe



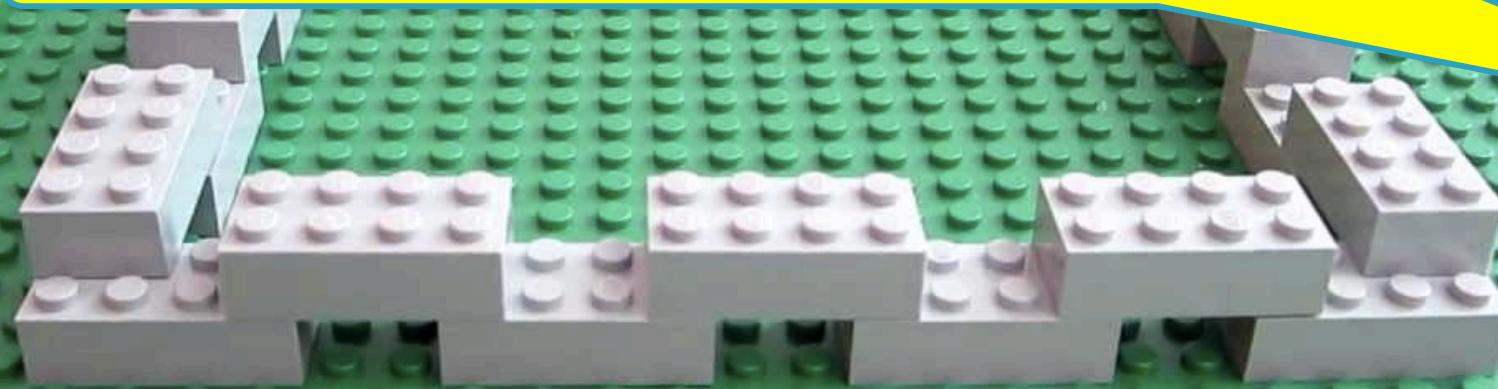
Vantagens??

Idéia BASE: Todos os representantes da classe tem a mesma chance de revelar a problema no medicamento



Idéia BASE: Todos os representantes da classe tem a mesma chance de revelar a problema no medicamento

ISSO É SEMPRE VERDADE??



É uma suposição...
pois pessoas (de
um mesma classe
de eq) podem ter
características
diferentes



Pessoas não
selecionadas
podem ter
intolerância
a fórmula



Particionamento em Classes de Equivalência

- Técnica de teste caixa preta (ou funcional).
- Pode ser aplicada: *unidade, integração, sistema...*

Particionamento em Classes de Equivalência

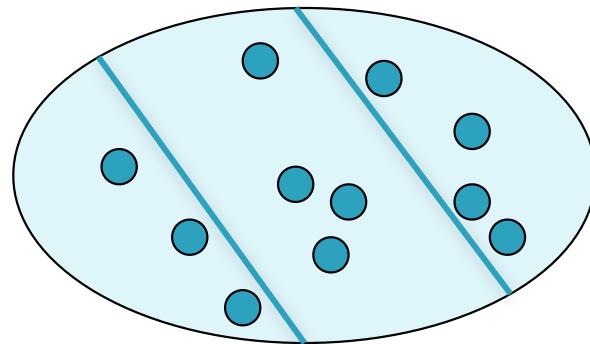
Divide o domínio de entrada em
“classes de dados equivalentes”

Particionamento em Classes de Equivalência

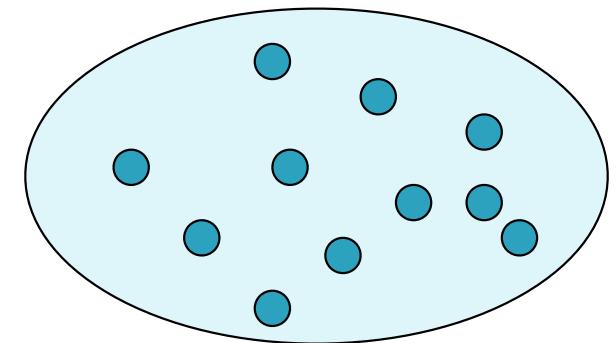
Geralmente adotada quando
temos intervalos de valores de
entrada.

Idéia central

Dominio de Entrada



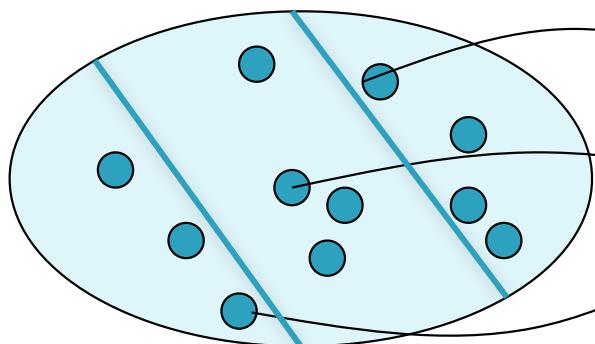
Dominio de Saída



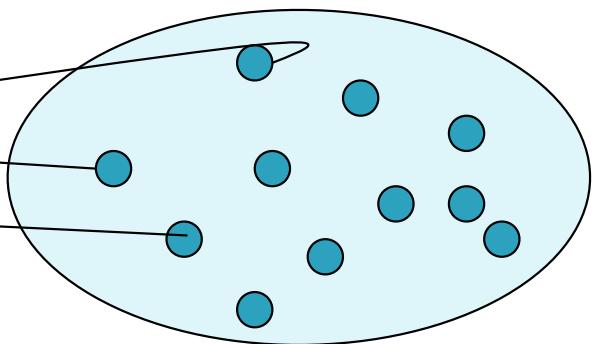
1. Define propriedade e fatia o domínio

Idéia central

Dominio de Entrada



Dominio de Saída



2. Seleciona representantes

Hipótese

- ▶ Supõe-se que elementos de uma mesma classe/bloco têm capacidade de revelar as mesmas falhas

Classes de Equivalência

- **classes válidas:** valores válidos do domínio de entrada
- **classes inválidas:** valores inválidos do domínio de entrada.

Exemplo - Site Avianca

COMPRE AQUI CHECK-IN UPGRADE MINHAS RESERVAS

Ida e Volta Só Ida

Origem
Natal (NAT) ▾

Data de Ida
08/10/2018

Adultos 1 ▾ **Crianças** 0 ▾ **Bebês** 0 ▾
2 a 11 anos 0 a 23 meses

Utilizar pontos

Para mais de 9 passageiros, [clique aqui.](#)

Buscas Recentes

Destino
Selecionar...

Data de Volta
dd/mm/aaaa

Cabine
Selecionar...

BUSCAR VOO

Exemplo - Site Avianca

The screenshot shows the Avianca website's flight search interface. At the top, there are navigation links: 'COMPRE AQUI' (Buy Here) in red, and 'CHECK-IN', 'UPGRADE', and 'MINHAS RESERVAS' in grey. Below this, there are two tabs: 'Ida e Volta' (Round trip) and 'Só Ida' (One-way), with 'Só Ida' currently selected. To the right of the tabs is a 'Buscas Recentes' (Recent searches) dropdown menu. The main search area is outlined in red and contains fields for 'Adultos' (1), 'Crianças' (0), and 'Bebês' (0). Below these are dropdown menus for age groups ('2 a 11 anos' for children, '0 a 23 meses' for babies) and travel dates ('aaaa' for year and 'mmaa' for month). There is also a checkbox for 'Utilizar pontos' (Use points). A green link at the bottom encourages users to click for more passengers.

COMPRE AQUI CHECK-IN UPGRADE MINHAS RESERVAS

Ida e Volta Só Ida Buscas Recentes

Adultos Crianças Bebês

1 0 0

2 a 11 anos 0 a 23 meses

Utilizar pontos

Para mais de 9 passageiros, [clique aqui.](#)

Exemplo - Site Avianca

The screenshot shows the search interface for the Avianca website. At the top, there are four navigation links: 'COMPRE AQUI' (in red), 'CHECK-IN', 'UPGRADE', and 'MINHAS RESERVAS'. Below these are two tabs: 'Ida e Volta' (highlighted in blue) and 'Só Ida'. To the right is a dropdown menu labeled 'Buscas Recentes' with a close button ('X').

The search form includes fields for 'Origem' (set to 'Natal (NAT)') and 'Destino' (set to 'Selecionar...'). Below these are date selection fields: 'Data de Ida' (set to '08/10/2018') and 'Data de Volta' (set to 'dd/mm/aaaa').

A red box highlights the passenger selection area, which includes dropdowns for 'Adultos' (set to '1'), 'Crianças' (set to '0'), 'Bebês' (set to '0'), and 'Cabine' (set to 'Selecionar...'). Below this are age filters: '2 a 11 anos' under 'Crianças' and '0 a 23 meses' under 'Bebês'. There is also a checkbox for 'Utilizar pontos' (Check points) and a link for 'Para mais de 9 passageiros, clique aqui.' (For more than 9 passengers, click here). A large red button at the bottom right says 'BUSCAR VOO' (Search flight).

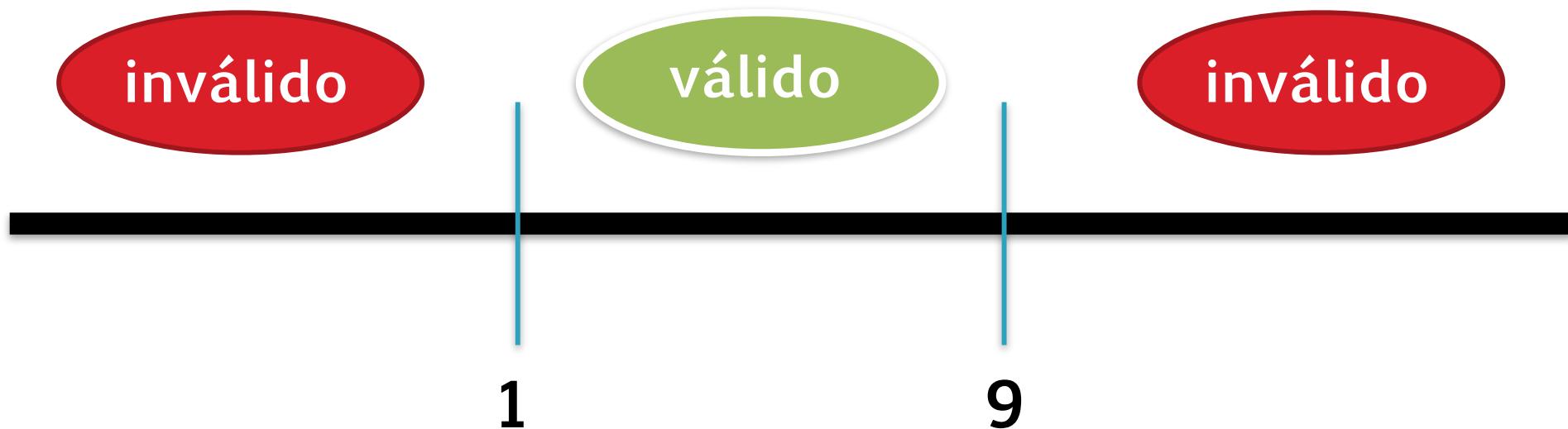
Exemplo - Site Avianca

The screenshot shows the Avianca website's search form. At the top, there are navigation links: 'COMPRE AQUI' (Buy Now), 'CHECK-IN', 'UPGRADE', and 'MINHAS RESERVAS' (My Reservations). Below the search form, there are date input fields ('de, mm/aaaa' and 'até, mm/aaaa') with a calendar icon. The search form itself has four dropdowns for passenger selection: 'Adultos' (1 selected), 'Crianças' (0 selected), 'Bebês' (0 selected), and 'Cabine' (labeled 'Selecionar...'). Below these dropdowns are age filters: '2 a 11 anos' under 'Crianças' and '0 a 23 meses' under 'Bebês'. There is also a checkbox for 'Utilizar pontos' (Use points) and a link 'Para mais de 9 passageiros, clique aqui.' (For more than 9 passengers, click here). A large red button at the bottom right says 'BUSCAR VOO' (Search Flight). Handwritten red text on the left side of the form reads:
Campo: Adultos
Valida: entre 1 e 9
Invalida: Superiores a 9

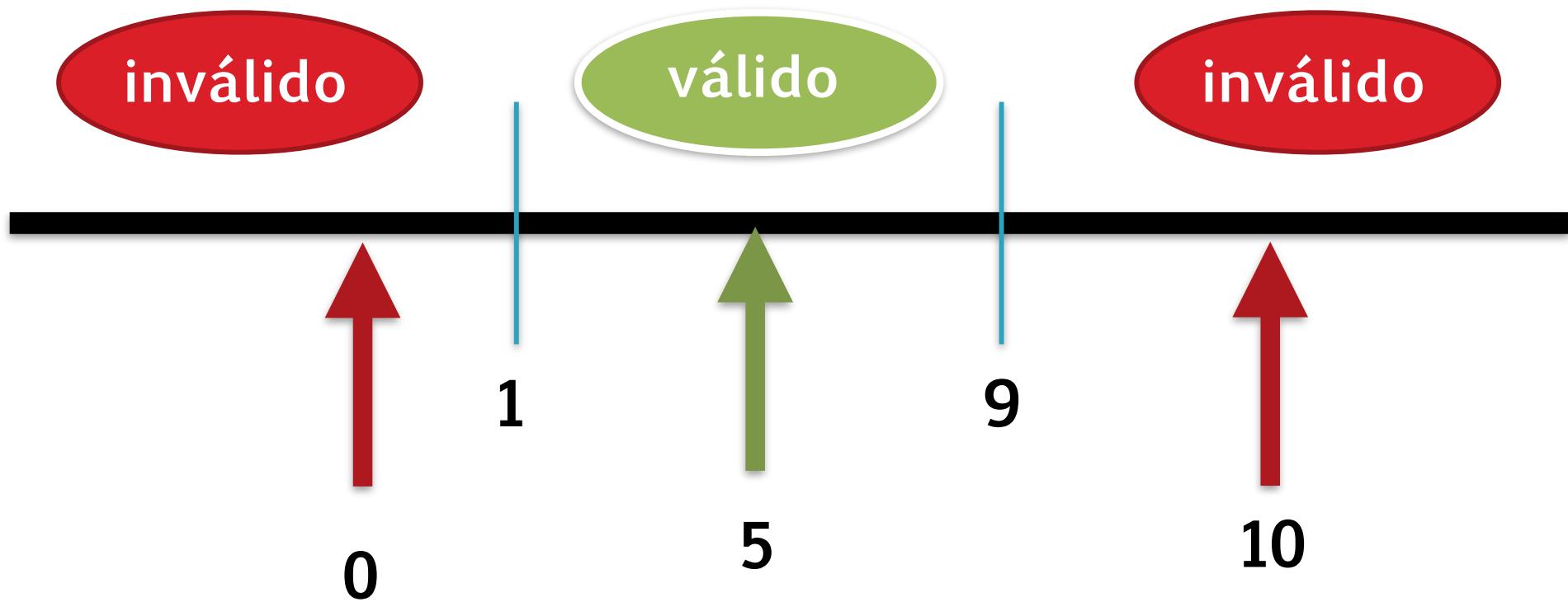
Domínio de entrara Inteiros



Domínio de entrara Inteiros



Domínio de entrada Inteiros



3 casos de testes!!!!



Exemplo 2 – Identificador

O identificador de uma variável em uma linguagem de programação que deve iniciar com 1 letra e conter apenas letras e dígitos, e deve conter de 1 a 6 caracteres.



```

*****
Identifier.c
ESPECIFICACAO: O programa deve determinar se um identificador eh ou nao valido em 'Silly
Pascal' (uma estranha variante do Pascal). Um identificador valido deve comecar com uma
letra e conter apenas letras ou digitos. Alem disso, deve ter no minimo 1 caractere e no
maximo 6 caracteres de comprimento
*****
```

<pre> #include <stdio.h> main () /* 1 */ { /* 1 */ char achar; /* 1 */ int length, valid_id; /* 1 */ length = 0; /* 1 */ valid_id = 1; /* 1 */ printf ("Identificador: "); /* 1 */ achar = fgetc (stdin); /* 1 */ valid_id = valid_s(achar); /* 1 */ if(valid_id) /* 2 */ { /* 2 */ length = 1; /* 2 */ } /* 3 */ achar = fgetc (stdin); /* 4 */ while(achar != '\n') /* 5 */ { /* 5 */ if(! (valid_f(achar))) /* 6 */ { /* 6 */ valid_id = 0; /* 6 */ } /* 7 */ length++; /* 7 */ achar = fgetc (stdin); /* 7 */ } /* 8 */ if(valid_id && /* 8 */ (length >= 1) && (length < 6)) /* 9 */ { /* 9 */ printf ("Valido\n"); /* 9 */ } /* 10 */ else /* 10 */ { /* 10 */ printf ("Invalido\n"); /* 10 */ } /* 11 */ }</pre>	<pre> int valid_s(char ch) /* 1 */ { /* 1 */ if(((ch >= 'A') && /* 1 */ (ch <= 'Z')) /* 1 */ ((ch >= 'a') && /* 1 */ (ch <= 'z'))) /* 2 */ { /* 2 */ return (1); /* 2 */ } /* 3 */ else /* 3 */ { /* 3 */ return (0); /* 3 */ } /* 4 */ } int valid_f(char ch) /* 1 */ { /* 1 */ if(((ch >= 'A') && /* 1 */ (ch <= 'Z')) /* 1 */ ((ch >= 'a') && /* 1 */ (ch <= 'z')) /* 1 */ ((ch >= '0') && /* 1 */ (ch <= '9'))) /* 2 */ { /* 2 */ return (1); /* 2 */ } /* 3 */ else /* 3 */ { /* 3 */ return (0); /* 3 */ } /* 4 */ }</pre>
---	--

Resposta

Restrições de Entrada	Classes Válidas	Classes Inválidas
Tamanho (t) do identificador	$1 \leq t \leq 6$ (1)	$t > 6$ (2)
Primeiro caracter (c) é uma letra	Sim (3)	Não (4)
Contém somente caracteres válidos	Sim (5)	Não (6)

Quais seriam as classes de equivalência validas e inválidas?



Restrições de Entrada	Classes Válidas	Classes Inválidas
Tamanho (t) do identificador	$1 \leq t \leq 6$ (1)	$t > 6$ (2)
Primeiro caracter (c) é uma letra	Sim (3)	Não (4)
Contém somente caracteres válidos	Sim (5)	Não (6)

Resposta:

Caso de Testes 1:

Caso de Testes2:

Caso de Testes 3:

Caso de Testes 4:

Quais os casos de testes para
“cobrir” as classes de
equivalência??



Restrições de Entrada	Classes Válidas	Classes Inválidas
Tamanho (t) do identificador	$1 \leq t \leq 6$ (1)	$t > 6$ (2)
Primeiro caracter (c) é uma letra	Sim (3)	Não (4)
Contém somente caracteres válidos	Sim (5)	Não (6)

Resposta:

Caso de Testes 1: testeteste

Caso de Testes 2: teste

Caso de Testes 3: 2teste

Caso de Testes 4: teste!!

4 casos de testes!!!!



Exemplo 3 - Calendário



Este programa determina a próxima data de acordo com o calendário gregoriano.

1/1/1582 <= Data <= 31/12/3000.



Exemplo 3 – Calendário

Campo	Classes Validas	Classes Invalidas
Dia		
Mês		
Ano		



Exemplo 3 - Calendário

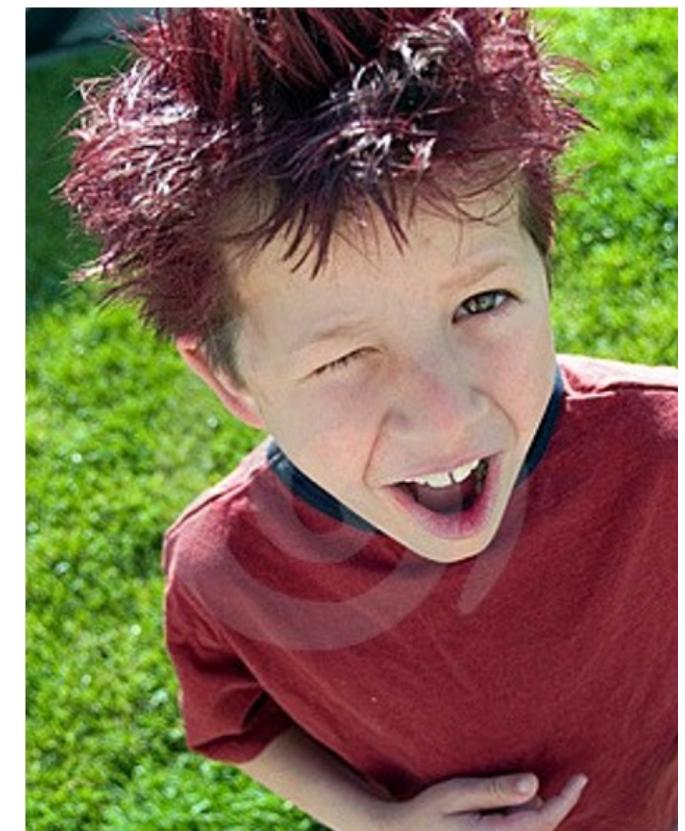
Campo	Classes Validas	Classes Invalidas
Dia	<p>c1: $1 \leqslant \text{dia} \leqslant 30 \ \& \&$ mes pertence a {1,3,5,7,8,12}</p> <p>c2: $1 \leqslant \text{dia} \leqslant 31 \ \& \&$ mes pertence a {2,4,6,9,11}</p> <p>c3: $1 \leqslant \text{dia} \leqslant 29 \ \& \&$ mes = 2 e ano bissexto</p> <p>c4: $1 \leqslant \text{dia} \leqslant 28 \ \& \&$ mês = 2 e ano não bissexto</p>	i1: dia < 1 i2 :dia – não inteiro i3 :dia > 30 & mes pertence a {1,3,5,7,8,12} i4 :dia > 31 & mes pertence a {2,4,6,9,11} i5 :dia > 29 & mes = 2 e ano bissexto i6 :dia > 28 & mes = 2 e ano nao bissexto
Mês	v4C5 $1 \leqslant \text{mes} \leqslant 12$	Ii7 :3 mes < 1 Ii8: mes > 12
Ano	vC65 $1582 \leqslant \text{Ano} \leqslant 3000$	I5i9 Ano < 1582 Ii106 Ano > 3000
Restricao relacionada a Data	C7 Data <= 31/12/3000	i11 Data > 31/12/3000

Quais os casos de testes para
“cobrir” as classes de
equivalência??



Passo 1

Selecionando representantes de cada uma das classes validas e invalidas e montando os casos de teste



Campo	Classes Validas	Classes Invalidas
Dia	v1 $1 \leq dia \leq 30$ v2 $1 \leq dia \leq 31$ v3 $1 \leq dia \leq 29$	i1 dia < 1 i2 dia - não inteiro
Mês	v4 $1 \leq mes \leq 12$	i3 mes < 1 i4 mes > 12
Ano	v5 $1582 \leq Ano \leq 3000$	i5 Ano < 1582 i6 Ano > 3000

Caso de Testes 1: 30 / 1 / 1582

Classes: v1, v4, v5

Caso de Testes 2: 29 / 2 / 1583

Classes: v3, v4, v5

Caso de Testes 3: 31 / 12 / 1997

Classes: v2, v4, v5

Caso de Testes 4: 0 / 1 / 1582

Classes: i1, v4, v5

Caso de Testes 5: A / 2 / 1583

Classes: i2, v4, v5

Caso de Testes 6: 31 / 0 / 1997

Classes: v2, i3, v5

Caso de Testes 7: 30 / 13 / 1582

Classes: v1, i4, v5

Caso de Testes 8: 29 / 2 / 1581

Classes: v3, v4, i5

Caso de Testes 9: 31 / 12 / 3001

Classes: v2, v4, i6

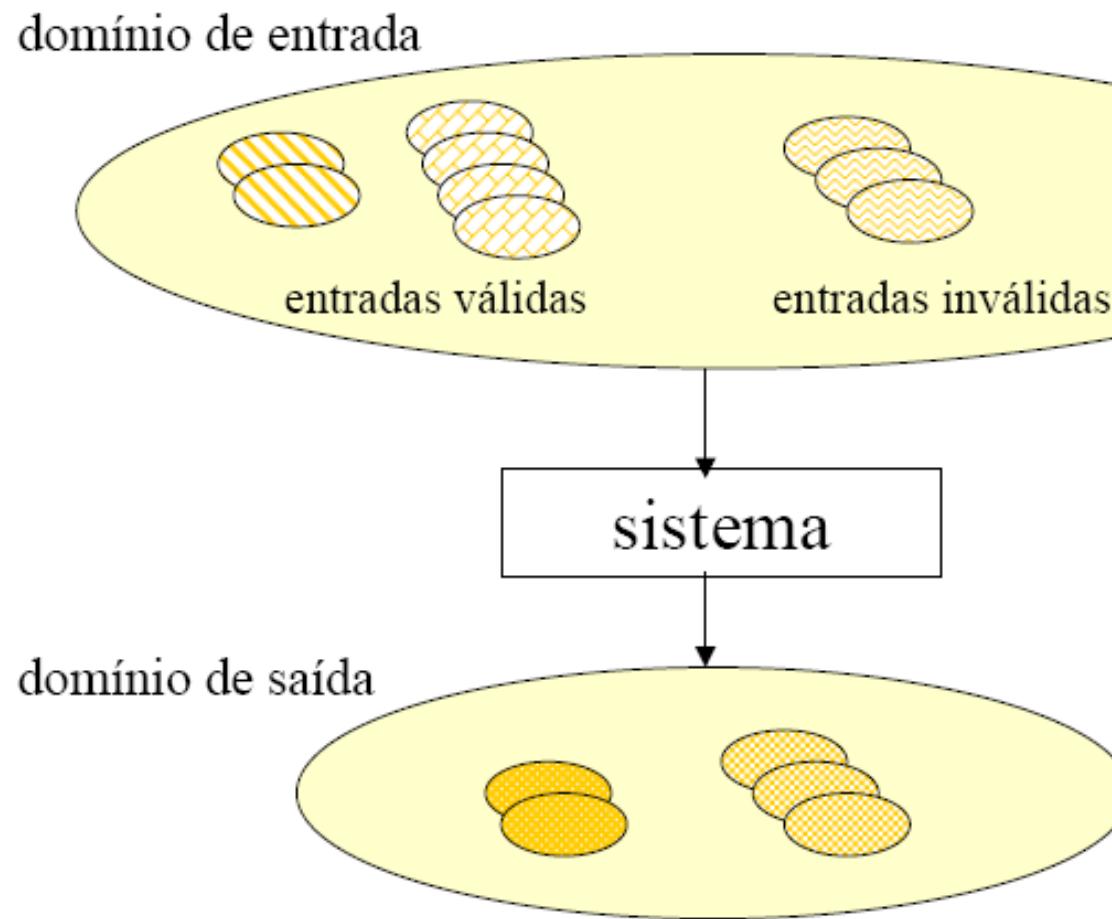
9 casos de testes!!!!



Idéia Central

- "se um elemento de determinada classe revela um erro, todos os demais elementos daquela mesma classe em a mesma chance de revelar o mesmo erro!!!""

Resumo



Dicas

Definição da variável de entrada	Classes de equivalência
Intervalo	<ul style="list-style-type: none">• 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
Lista de valores válidos	<ul style="list-style-type: none">• Uma classe válida para os valores incluídos na lista• Uma classe inválida para todos os outros valores

Dicas

Número de valores válidos	<ul style="list-style-type: none">• Uma classe válida para número de valores igual ao número previsto• Uma classe inválida para número de valores = 0• Uma classe inválida para número de valores maior ou menor que o valor previsto
Restrições (expressão lógica; sintaxe; valor específico; compatibilidade com outras variáveis)	<ul style="list-style-type: none">• Uma classe válida para os valores que satisfazem às restrições• Uma classe inválida para os outros valores

Mão na Massa



Mão na Massa

Defina as classes de equivalência para o método **calculalImposto** da classe param.CalculolImpostoRenda do projeto da disciplina.

E a partir das classes de equivalência selecione casos de testes de forma a “cobrir” as classes e construa testes JUnit para exercitar estas entradas.