

# Universidade Presbiteriana Mackenzie

## Técnicas de Teste de Software

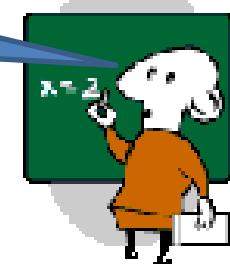
**Ana Claudia Rossi**

Faculdade de Computação e Informática

# Fundamentos de Teste de Software

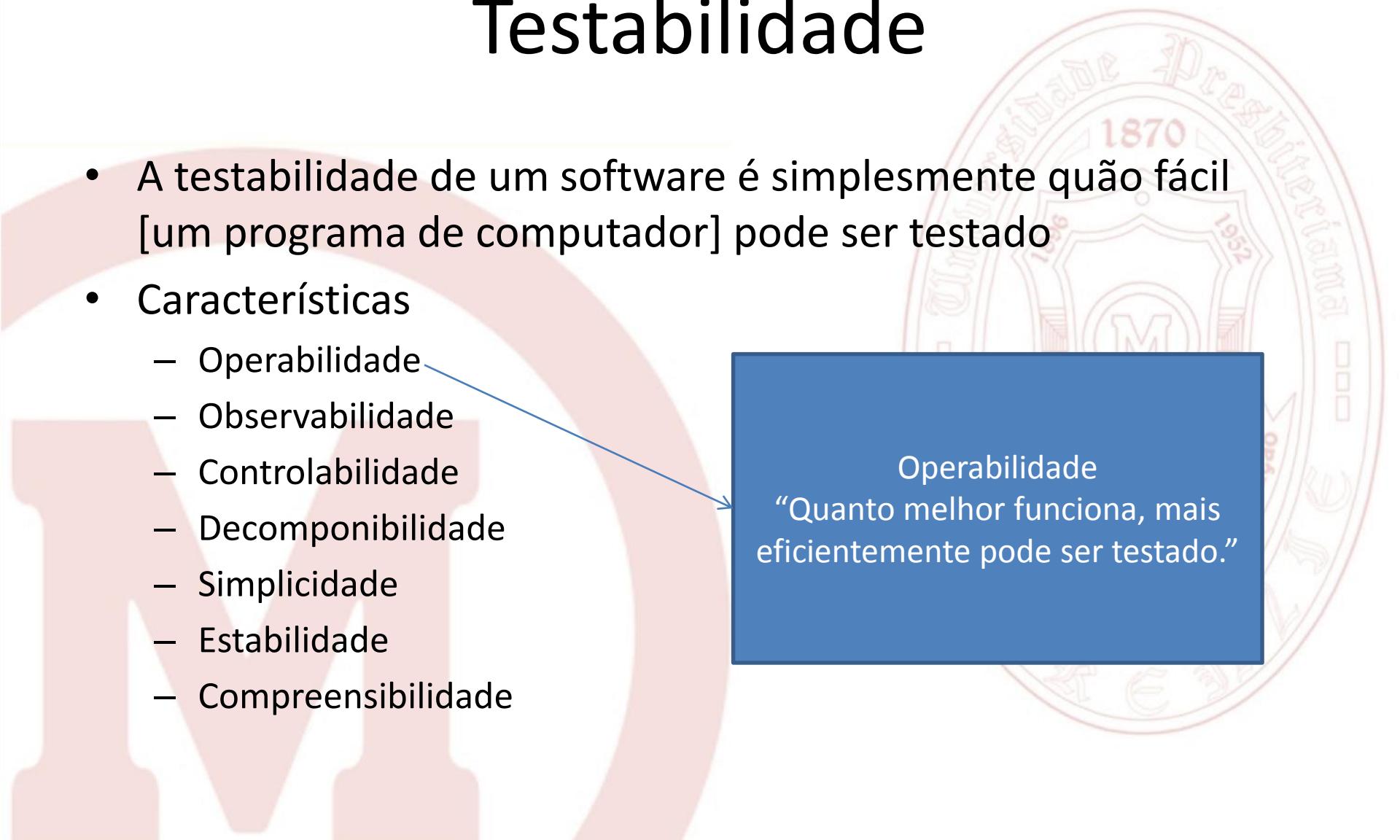
O teste deveria inculcar culpa? O teste realmente é destrutivo?

Não!



# Testabilidade

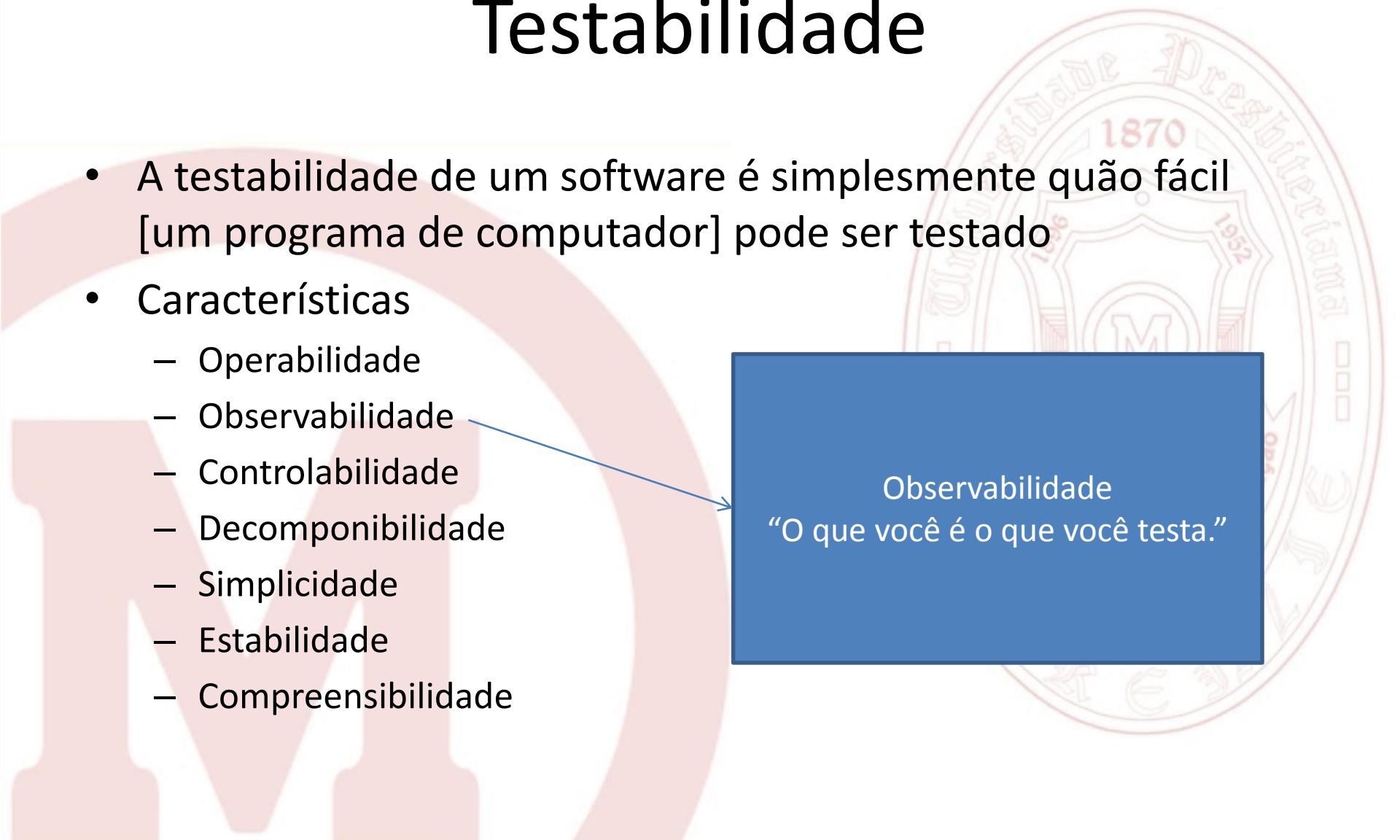
- A testabilidade de um software é simplesmente quanto fácil [um programa de computador] pode ser testado
- Características
  - Operabilidade
  - Observabilidade
  - Controlabilidade
  - Decomponibilidade
  - Simplicidade
  - Estabilidade
  - Compreensibilidade



Operabilidade  
“Quanto melhor funciona, mais eficientemente pode ser testado.”

# Testabilidade

- A testabilidade de um software é simplesmente quanto fácil [um programa de computador] pode ser testado
- Características
  - Operabilidade
  - Observabilidade
  - Controlabilidade
  - Decomponibilidade
  - Simplicidade
  - Estabilidade
  - Compreensibilidade



Observabilidade

“O que você é o que você testa.”

# Testabilidade

- A testabilidade de um software é simplesmente quanto fácil [um programa de computador] pode ser testado
- Características
  - Operabilidade
  - Observabilidade
  - Controlabilidade
  - Decomponibilidade
  - Simplicidade
  - Estabilidade
  - Compreensibilidade

## Controlabilidade

“Quanto melhor você pode controlar o software, mais o teste pode ser automatizado e otimizado.”

# Testabilidade

- A testabilidade de um software é simplesmente quanto fácil [um programa de computador] pode ser testado
- Características
  - Operabilidade
  - Observabilidade
  - Controlabilidade
  - Decomponibilidade
  - Simplicidade
  - Estabilidade
  - Compreensibilidade

Decomponibilidade  
“Controlando o escopo do teste,  
pode-se isolar problemas mais  
rapidamente e realizar retestagem  
mais racionalmente.”

# Testabilidade

- A testabilidade de um software é simplesmente quanto fácil [um programa de computador] pode ser testado
- Características
  - Operabilidade
  - Observabilidade
  - Controlabilidade
  - Decomponibilidade
  - Simplicidade
  - Estabilidade
  - Compreensibilidade

Simplicidade  
“Quanto menos houver a testar, mais rapidamente pode-se testá-lo.”

# Testabilidade

- A testabilidade de um software é simplesmente quanto fácil [um programa de computador] pode ser testado
- Características
  - Operabilidade
  - Observabilidade
  - Controlabilidade
  - Decomponibilidade
  - Simplicidade
  - Estabilidade
  - Compreensibilidade

Estabilidade

“Quanto menos modificações,  
menos interrupções no teste.”

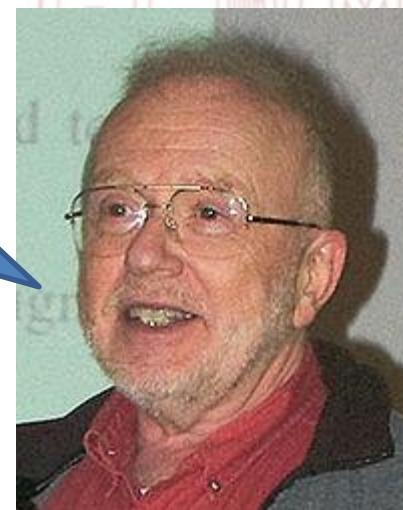
# Testabilidade

- A testabilidade de um software é simplesmente quanto fácil [um programa de computador] pode ser testado
- Características
  - Operabilidade
  - Observabilidade
  - Controlabilidade
  - Decomponibilidade
  - Simplicidade
  - Estabilidade
  - Compreensibilidade

Compreensibilidade

“Quanto mais informações se tem,  
mais racionalmente irá se testar.”

Erros são mais comuns, mais disseminados e mais problemáticos no software do que em outras tecnologias!



David Parnas

# Características do Teste

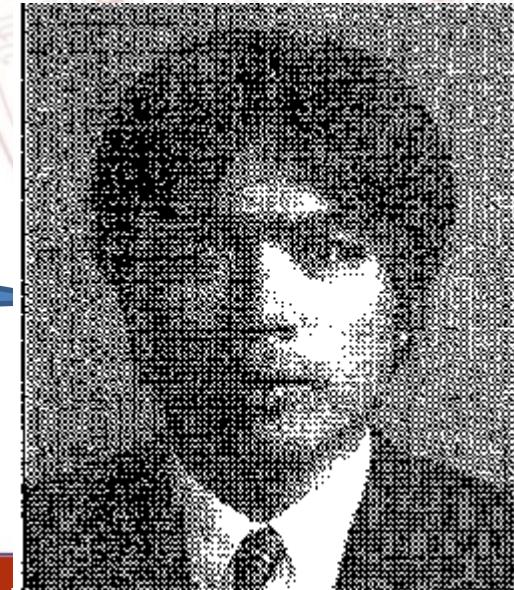
- Atributos para um bom teste
  - Um bom teste tem uma alta probabilidade de encontrar um erro
  - Um bom teste não é redundante
  - Um bom teste deve ser “de boa cepa”\*
  - Um bom teste não deve ser nem muito simples e nem muito complexo

\* de boa índole, pessoa nobre

# Técnicas de Teste de Software

## Caixa Preta/ Caixa Branca

Há apenas uma regra para o projeto de casos de teste:  
abranja todas as características,  
mas não faça muitos casos de teste!



# Teste exaustivo

- (1) cenário de uso tem no mínimo 1 entrada e no máximo 10, (2) 20 entradas diferentes são possíveis e (3) entradas podem ser repetidas
- Se cada cenário pode ser testado em 1s, o sistema levará 300.000 anos para ser testado
- Se 100 cenários podem ser testados em 1s, o tempo de teste se reduz a 3000 anos
- **Teste Exaustivo é impossível na prática**

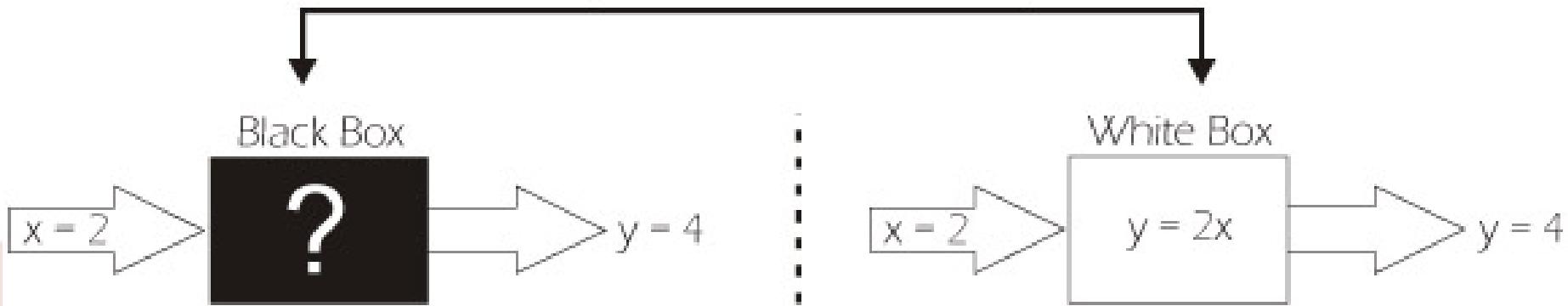
# Formas de se testar

- Conhecendo-se a função especificada que o produto foi projetado para realizar, podem ser realizados testes que demonstrem que o programa que cada função está plenamente operacional e, ao mesmo tempo procurem erros em cada função
- Sabendo-se que como é o trabalho interno de um produto, podem ser realizados testes para garantir que “todas as engrenagens combinem”, isto é, que as operações sejam realizadas de acordo com as especificações e que todos os componentes internos formam exercitados

# Técnicas de Teste

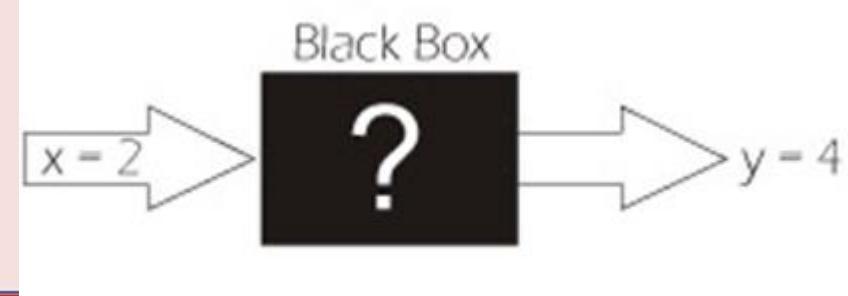
- Testes Funcionais
  - Teste de Caixa-Preta
- Testes Estruturais
  - Teste de Caixa-Branca

Unknown Equation versus Known Equation



# Teste de Caixa-Preta

- Também conhecido como comportamental
- É uma abordagem complementar ao teste caixa branca
- Não tem importância como os resultados foram alcançados
  - Será fornecido uma entrada
  - E o resultado deverá “sair” correto
- Aumento de 40% na qualidade de um software
- Utilizado, principalmente, nos estágios finais da fase de testes
  - A execução, mas o projeto dos casos de teste baseados nesta técnica podem ocorrer antes



# Teste de Caixa-Preta

- Focaliza os requisitos funcionais
  - As entradas especificadas testarão esses requisitos
  - Baseado nas entradas, quais são as saídas esperadas
- Categorias
  - Funções incorretas ou omitidas
  - Erros de interface
  - Erros de estruturas de dados ou de acesso a dados
  - Erros de comportamento ou desempenho
  - Erros de iniciação ou término

M



# Teste de Caixa-Preta

- Situação Ideal
  - Testar o software com todas as combinações possíveis de entradas
  - Demanda muito tempo e dinheiro
- Solução
  - Fazer uma seleção das possibilidades de casos de teste
  - Existem métodos que ajudam fazer esta seleção

# Teste de Caixa-Preta

- Questões
  - Como a validade funcional é testada?
  - Como o comportamento e o desempenho é testado?
  - Que classes de entrada constituirão um bom caso de teste?
  - O sistema é sensível a certos valores de entrada?
  - Como são isolados os limites de uma classe de dados?
  - Que taxa e volume de dados o sistema pode agüentar?
  - Que efeitos e combinações específicas de dados terão no sistema?

# Teste de Caixa-Preta

- Teste Baseado em Grafo/Estado
- Particionamento de Equivalência
- Análise de Valor-Limite
- Tabela de Decisão
- Teste de Matriz Ortogonal
- Teste de Caso de Uso

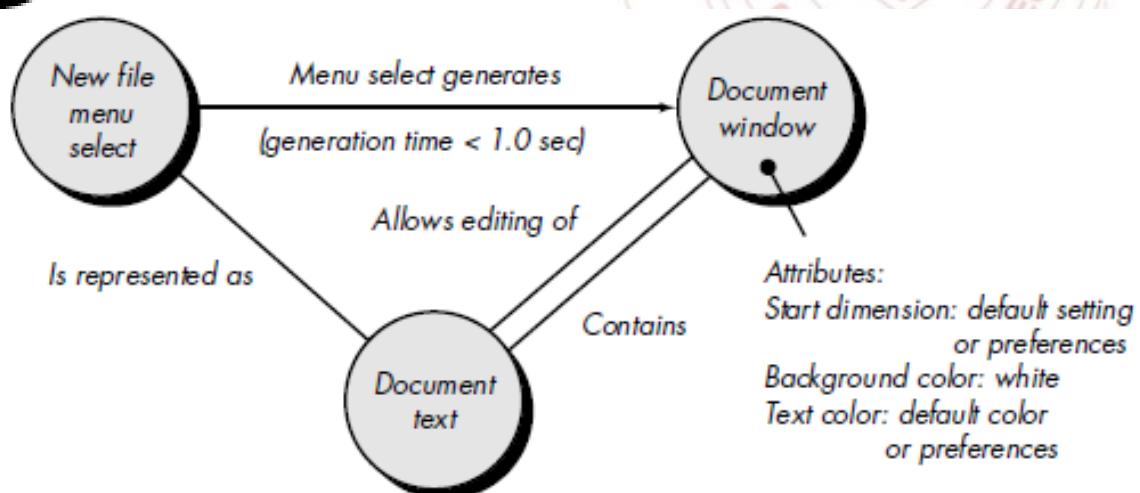
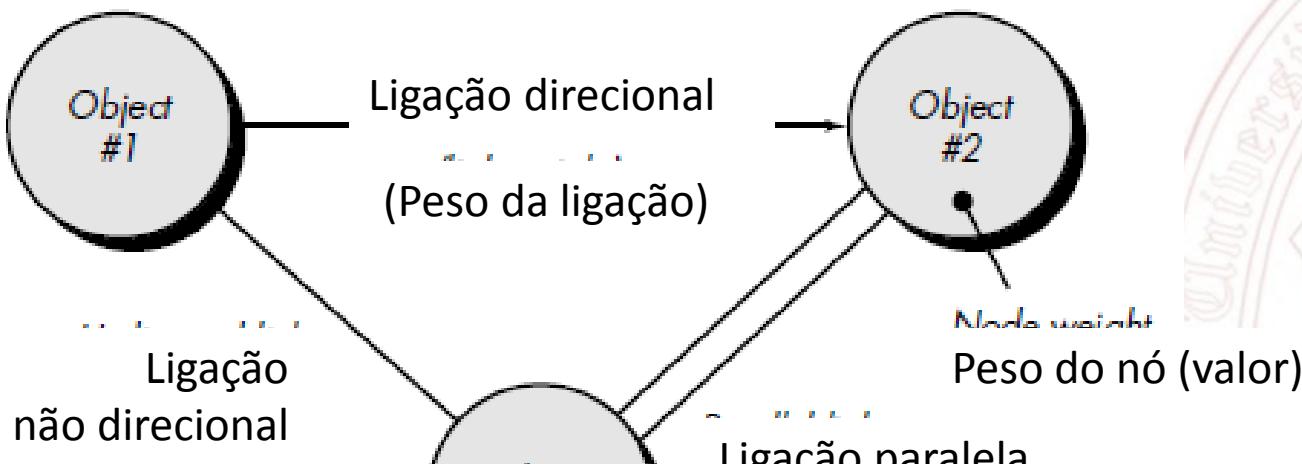
# Teste Baseado em Grafo

- Teste de Caixa-Preta
- Define-se os objetos do sistema
  - Funções, componentes, módulos, dados
- Descobre a relação entre os objetos
- Define uma série de casos de teste
  - Verifica se todos os objetos têm a relação esperada uns com os outros

# Métodos de Teste baseados em Grafo

1. Começa criando um grafo dos objetos importantes e de suas relações.
2. Depois estabelece uma série de testes que irão cobrir o grafo de modo que cada objeto e relação sejam exercitados e que os erros sejam descobertos.
3. O grafo representa os relacionamentos entre os objetos de dados e de programas.
4. Uma coleção de nós representa os objetos, os pesos descrevem as propriedades de um nó, os links representam o relacionamento entre os objetos e o peso do link descreve alguma característica do link.

# Teste Baseado em Grafo



# Teste Baseado em Grafo

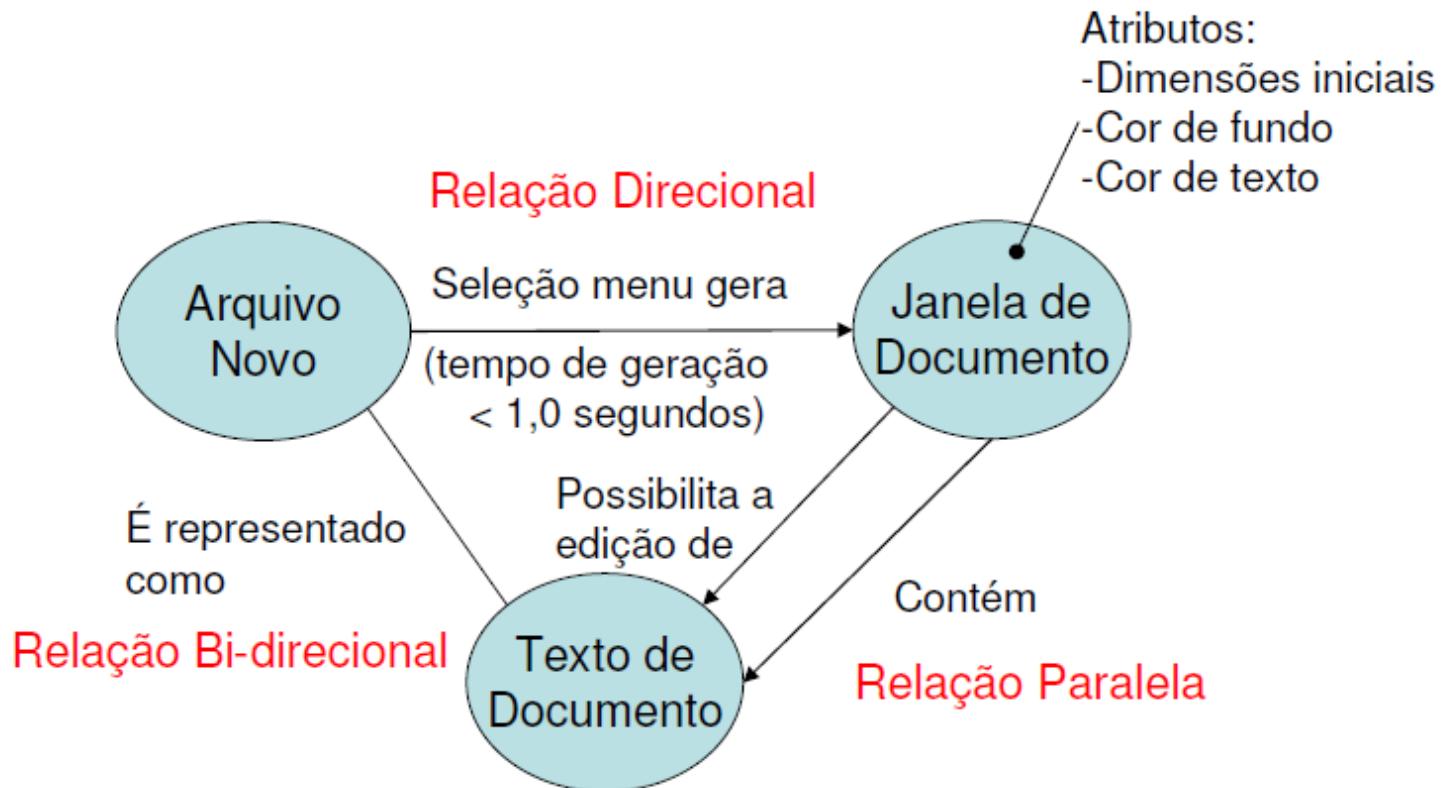
- Fluxo de Transação
  - Passos entre transação e suas ligações
- Estados finitos
  - Representam estados e as ligações as transições
- Fluxo de Dados
  - Os nós são os dados e as ligações são as transformações
- Tempo
  - Os nós são os objetos (programas) e as ligações são as ligações. Elas possuem pesos que representam sua execução

# Teste Baseado em Grafo

- Modelagem de fluxo de transação
  - Nós representam passos em alguma transação.
  - Arestras representam as conexões lógicas entre os passos.
- Modelagem de estado finito
  - Nós representam diferentes estados do software observáveis pelo usuário.
  - Arestras representam transições de um estado para outro.
- Modelagem de fluxo de dados
  - Nós representam os objetos de dados.
  - Arestras representam as transformações que ocorrem para traduzir um objeto em outro.
    - Eg: FTW = 0,62 x GW
- Modelagem de tempo
  - Nós representam objetos do programa.
  - Arestras representam ligações seqüenciais entre esses objetos.
    - Pesos são usados para especificar os tempos de execução.

# Teste Baseado em Grafo

## Exemplo



# Particionamento de Equivalência

- Teste de Caixa-Preta
- Domínio da entrada
  - Dividido em classes de dados
  - Os casos de testes utilizam essas informações
- Sua objetivo é descobrir classes de teste que apresentam erros
  - Diminui a quantidade de casos de testes a serem definidos

# Particionamento de Equivalência

- Diretrizes

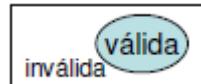
- Se a condição de entrada especifica um intervalo, são definidas uma classe válida e duas inválidas



- Se a condição de entrada exige um valor específico, são definidas uma classe válida e duas inválidas



- Se a condição de entrada especifica um membro de um conjunto, são definidas uma classe válida e uma inválida



- Se a condição de entrada for booleana, são definidas uma classe válida e uma inválida

# Particionamento de Equivalência

Devemos seguir tais passos para geração dos testes usando este critério:

- 1. Identificar classes de equivalência (é um processo heurístico)
  - condição de entrada
  - válidas e inválidas
- 2. Definir os casos de teste
  - enumeram-se as classes de equivalência
  - casos de teste para as classes válidas
  - casos de teste para as classes inválidas

# Particionamento de Equivalência

## EXEMPLOS

“Um identificador válido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo 1 caractere e no máximo 6 caracteres de comprimento. Exemplo: “abc12” (válido), “cont\*1” (inválido), “1soma” (inválido) e “a123456” (inválido).”

# Particionamento de Equivalência

## EXEMPLOS

O primeiro passo é a identificação das classes de equivalência. Isso está descrito na **Tabela 1**.

Condições de Entrada	Classes	Classes
Tamanho $t$ do identificador	(1) 1 ??t???6	(2) $t > 6$
Primeiro caractere $c$ é uma letra	(3) Sim	(4) Não
Só contém caracteres válidos	(5) Sim	(6) Não

**Tabela 1.** Classes de Equivalência do programa identifier.c.

# Particionamento de Equivalência

## EXEMPLOS

- A partir disso, conseguimos especificar quais serão os casos de teste necessários. Para ser válido, um identificador deve atender às condições (1), (3) e (5), logo é necessário um caso de teste válido que cubra todas essas condições. Além disso, será necessário um caso de teste para cada classe inválida: (2), (4) e (6). Assim, o conjunto mínimo é composto por quatro casos de teste, sendo uma das opções:  $T_0 = \{(a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido})\}$

Condições de Entrada	Classes	Classes
Tamanho $t$ do identificador	(1) $1 \leq t \leq 6$	(2) $t > 6$
Primeiro caractere $c$ é uma letra	(3) Sim	(4) Não
Só contém caracteres válidos	(5) Sim	(6) Não

**Tabela 1.** Classes de Equivalência do programa identifier.c.

# Particionamento de Equivalência

## EXEMPLOS

- A partir disso, conseguimos especificar quais serão os casos de teste necessários. Para ser válido, um identificador deve atender às condições (1), (3) e (5), logo é necessário um caso de teste válido que cubra todas essas condições. Além disso, será necessário um caso de teste para cada classe inválida: (2), (4) e (6). Assim, o conjunto mínimo é composto por quatro casos de teste, sendo uma das opções:  $T_0 = \{(a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido})\}$

Condições de Entrada	Classes	Classes
Tamanho $t$ do identificador	(1) $1 \leq t \leq 6$	(2) $t > 6$
Primeiro caractere $c$ é uma letra	(3) Sim	(4) Não
Só contém caracteres válidos	(5) Sim	(6) Não

**Tabela 1.** Classes de Equivalência do programa identifier.c.

## Exemplos



Veja as diretrizes:

- 1 Se uma condição de entrada especifica um intervalo, uma classe de equivalência válida e duas inválidas devem ser definidas;
- 2 Se uma condição de entrada exige um valor específico, uma classe de equivalência válida e duas inválidas são definidas;
- 3 Se uma condição de entrada especifica o membro de um conjunto, uma classe de equivalência válida e uma inválida são definidas;
- 4 Se uma condição de entrada é booleana, uma classe de equivalência válida e uma inválida são definidas.

Veja um exemplo:

Dividir todas as combinações possíveis em classes. Em um sistema de gestão de contratos, a idade dos clientes varia de 18 a 120 anos.

Entrada	Valores Permitidos	Classes	Casos de Teste
Idade	Idade entre 18 e 120	18 a 120	Idade = 20
		< 18	Idade = 10
		> 120	Idade = 150

## A form for enumerating equivalence classes.

External condition	Valid equivalence classes	Invalid equivalence classes

# Particionamento de Equivalência

- O programa aceita entre 4 e 10 entradas numéricas.
- Cada número é formado com cinco dígitos, compreendidos no intervalo entre 10.000 e 99.999 .



Number of input values



# Particionamento de Equivalência

- Etapas da Técnica
  1. Identificar as classes de equivalência de Entrada
  2. Refinar as classes de Entrada
  3. Identificar as classes de Saída
  4. Refinar as Classes de Saída
  5. Relacionar as Classes de Entrada e de Saída
  6. Definir os casos de teste

# Etapas da Técnica

- **Etapa 1: Identificar as Classes de Equivalência**
  - Identificar, através da especificação dos dados de entrada, as classes de equivalência
  - Caracterizar pelo menos 2 grupos de valores de entrada para cada classe:
    - Válidas: Valores Esperados pelo programa
    - Inválidas: Entradas Inválidas ou inesperadas
- **Etapa 2: Refinamento das Classes de Entrada**
  - Se há razão para acreditar que elementos de uma mesma classe são tratados de forma diferente pelo programa, divida esta classe em menores
  - Particionar classes cujos atributos são considerados elementos chave para o item em teste.
- **Etapa 3: Identificar Classes de Saída**
  - Mapear todas as possíveis classes de saídas geradas pelo item em teste

# Etapas da Técnica

- **Etapa 4: Refinar as Classes de Saída**
  - Verificar se alguma classe de saída merece ser subdividida em classes menores
    - Ex: Código Genérico de Erro pode ser desmembrado em outros códigos.
- **Etapa 5: Relacionar Classes de Entrada e Classes de Saída**
  - Criar um mapeamento entre as classes de entrada e as classe de saída esperadas
- **Etapa 6: Derivar os Casos de Teste**
  - Atribuir um valor de entrada para cada classe de entrada (válidas e inválidas)
  - Elaborar casos de teste para as combinações das classes válidas
  - Elaborar casos de teste para todas as classes inválidas
  - Derivar casos de teste que cubram, de uma só vez, o maior número de classes válidas/inválidas

# Exemplo

- Considere dados mantidos como parte de uma aplicação de automação bancária. O usuário pode ter acesso ao banco usando um computador pessoal, fornecer uma senha de seis dígitos e a seguir dar uma série de comandos digitados que disparam as várias funções bancárias. Durante a seqüência de admissão, o software fornecido para aplicação bancária aceita dados de forma:
- **Código de área:** vazio ou três dígitos numéricos que iniciem por “zero”;  
**Prefixo:** três dígitos numéricos que não iniciem por “zero”;  
**Sufixo:** quatro dígitos numéricos;  
**Senha:** seis dígitos alfanuméricicos;  
**Operação:** c=cheque, d=depósito, e=extrato.



# Exemplo

- **Código de área:** vazio ou três dígitos numéricos que iniciem por “zero”;
- Prefixo:** três dígitos numéricos que não iniciem por “zero”;
- Sufixo:** quatro dígitos numéricos;
- Senha:** seis dígitos alfanuméricicos;
- Operação:** c=cheque, d=depósito, e=extrato.

A condições de entrada associadas a cada elemento da aplicação bancária podem ser especificadas como:

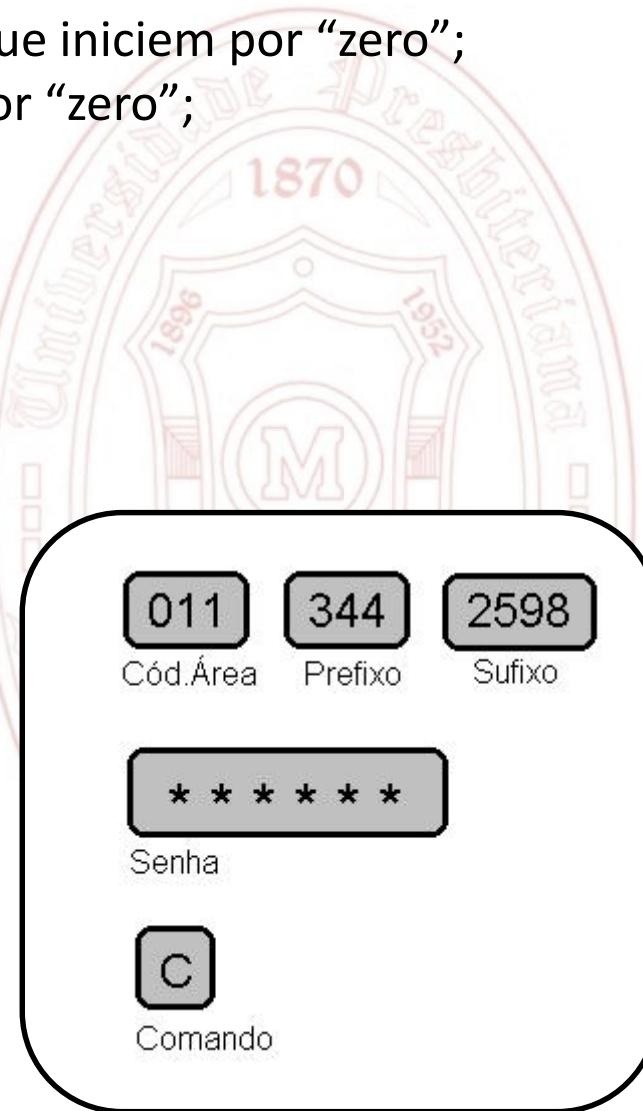
**Código de área:** Condição booleana (estar ou não presente) / Conjunto de valores de 000 a 099

**Prefixo:** Conjunto de valores de 100 a 999

**Sufixo:** 9 Conjunto de valores de 0000 a 9999

**Senha:** Condição booleana (estar ou não presente) / Conjunto de 6 caracteres alfanuméricicos (caracteres especiais presentes ?) / Conjunto de 1 a 5 caracteres / Conjunto vazio

**Operação:** Entrada de 1 caractere válido (c,d,e) / Entrada de 1 caractere inválido (demais, inclusive ESC e F1)



# Análise de Valor-Limite

- Grande quantidade de erros ocorrem nas fronteiras do domínio da entrada
  - Análise de valor-limite ou de fronteira
- Esta técnica complementa a de Particionamento de Equivalência
  - Considera as “bordas” das classes envolvidas

# Análise de Valor-Limite

- Diretrizes
  - Se uma condição de entrada especifica um intervalo limitado por valores dois valores, casos de testes devem ser projetados com esses dois valores e imediatamente acima e abaixo
  - Se uma condição de entrada especifica vários valores, casos de testes devem ser desenvolvidos para exercitar os números mínimos e máximos e valores imediatamente acima e abaixo
  - Aplicar as diretrizes 1 e 2 para as condições de saídas
  - Se as estruturas internas têm limites prescritos, certifique-se de projetar casos de teste para exercitar a estrutura de dados no seu limite

# Exemplo

- Considere as seguintes regras organizacionais, sobre contratação baseada em idade, em um programa de uma empresa de recursos humanos:
- 0-16 Não Contrata
- 16-18 Contrata meio-expediente
- 18-55 Contrata tempo integral
- 55-99 Não Contrata

**Uma regra correta seria:**

Se (idadeCandidato >= 0 && idadeCandidato <= 15) estadoContratação =  
“Não”;

Se (idadeCandidato >= 16 && idadeCandidato <= 17) estadoContratação =  
“Meio”;

Se (idadeCandidato >= 18 && idadeCandidato <= 54) estadoContratação =  
“Integral”;

Se (idadeCandidato >= 55 && idadeCandidato <= 99) estadoContratação =  
“Não”;

# Exemplo

- Considere as seguintes regras organizacionais, sobre contratação baseada em idade, em um programa de uma empresa de recursos humanos:
- 0-16 Não Contrata
- 16-18 Contrata meio-expediente
- 18-55 Contrata tempo integral
- 55-99 Não Contrata

Conjuntos de valores interessantes para serem testados são {-1,0,1},{15,16,17},{17,18,19},{54,55,56} e {98,99,100}.

Dependendo das pré-condições, valores como {-42, 1001, FRED, %\$#@} também devem ser testados!

**Uma regra correta seria:**

```
Se (idadeCandidato >= 0 && idadeCandidato <= 15) estadoContratação =  
"Não";  
Se (idadeCandidato >= 16 && idadeCandidato <= 17) estadoContratação =  
"Meio";  
Se (idadeCandidato >= 18 && idadeCandidato <= 54) estadoContratação =  
"Integral";  
Se (idadeCandidato >= 55 && idadeCandidato <= 99) estadoContratação =  
"Não";
```

# Exemplo

- Considere as seguintes regras organizacionais, sobre contratação baseada em idade, em um programa de uma empresa de recursos humanos:
  - 0-16 Não Contrata
  - 16-18 Contrata meio-expediente
  - 18-55 Contrata tempo integral
  - 55-99 Não Contrata

## Passos:

Identifique as classes de equivalência.

Identifique as fronteiras de cada classe.

Crie casos de teste para cada valor de fronteira, escolhendo um ponto abaixo, um acima e outro ponto na fronteira.

Obs.: “Acima” e “abaixo” são termos relativos, dependendo da unidade, exemplo: Inteiros, Moeda, horas, etc.

## Uma regra correta seria:

```
Se (idadeCandidato >= 0 && idadeCandidato <= 15) estadoContratação =  
"Não";
```

```
Se (idadeCandidato >= 16 && idadeCandidato <= 17) estadoContratação =  
"Meio";
```

```
Se (idadeCandidato >= 18 && idadeCandidato <= 54) estadoContratação =  
"Integral";
```

```
Se (idadeCandidato >= 55 && idadeCandidato <= 99) estadoContratação =  
"Não";
```

# Exemplo

- Se uma condição de entrada especifica uma faixa de valores limitada em  $a$  e  $b$ , casos de teste devem ser projetados com valores  $a$  e  $b$  e imediatamente acima e abaixo de  $a$  e  $b$ . Exemplo: Intervalo = {1..10}; Casos de Teste à {1, 10, 0,11}.

# Exemplo

Como exemplo, extraído de (BARBOSA *et al.*, 2000), iremos considerar a seguinte situação:

"... o cálculo do desconto por dependente é feito da seguinte forma: a entrada é a idade do dependente que deve estar restrita ao intervalo [0..24]. Para dependentes até 12 anos (inclusive) o desconto é de 15%. Entre 13 e 18 (inclusive) o desconto é de 12%. Dos 19 aos 21 (inclusive) o desconto é de 5% e dos 22 aos 24 de 3%..."

# Exemplo

Aplicando o teste de valor limite convencional serão obtidos casos de teste semelhantes a este: {-1,0,12,13,18,19,21,22,24,25}.

# Mas, e a Combinação de Valores de Entrada?

- Os dois métodos anteriores consideram dados de entrada e saída como independentes
- As dependências entre as entradas e os efeitos nas saídas não são considerados ao gerar os casos de teste

# Mas, e a Combinação de Valores de Entrada?

- Domínio de entrada ilimitado
  - Possibilidade de gerar casos de testes para todas elas ?
  - Teste exaustivo ?
- Entrada
  - X, Y, Z
  - Cada um assume 3 valores distintos
  - Total de 27 possibilidades ( $3^3$ )

# Mas, e a Combinação de Valores de Entrada?

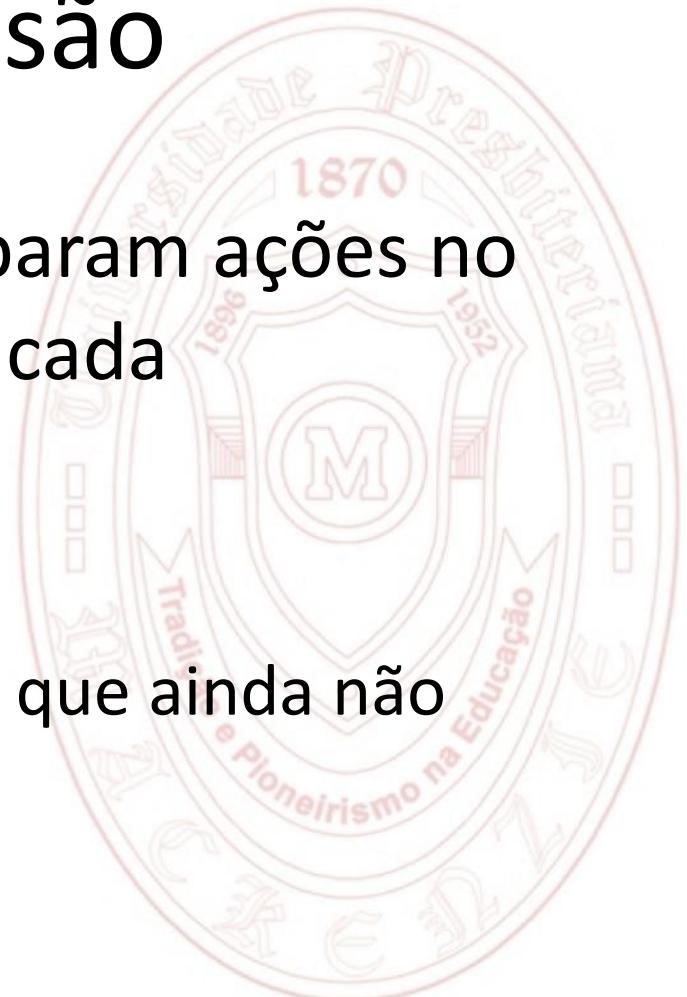
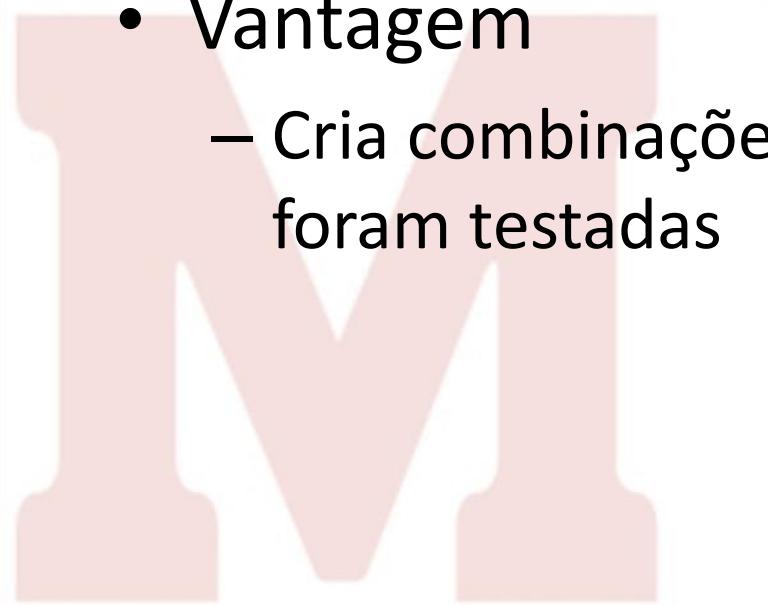
- Consideremos uma função chamada *envie* de uma aplicação de fax
  - Ela possui 4 parâmetros: P1, P2, P3 e P4
  - Cada parâmetro assume os valores discretos 1, 2 ou 3
  - Abordagem “um de cada vez”
    - $(1,1,1,1), (2,1,1,1), (3,1,1,1), (1,2,1,1), (1,3,1,1), (1,1,2,1), (1,1,3,1), (1,1,1,2), (1,1,1,3)$
  - E, a combinação?  $3^4 = 81$

# Tabela de Decisão

- Existem requisitos de sistema que possuem condições lógicas (regras de negócio complexas)
- Tabela de Decisão permite que as regras de negócio sejam especificadas de forma compacta
- Devem ser usadas em programas que tenham muitas decisões lógicas

# Tabela de Decisão

- Contem as condições que disparam ações no software e os resultados para cada combinação de condições
- Vantagem
  - Cria combinações de condições que ainda não foram testadas



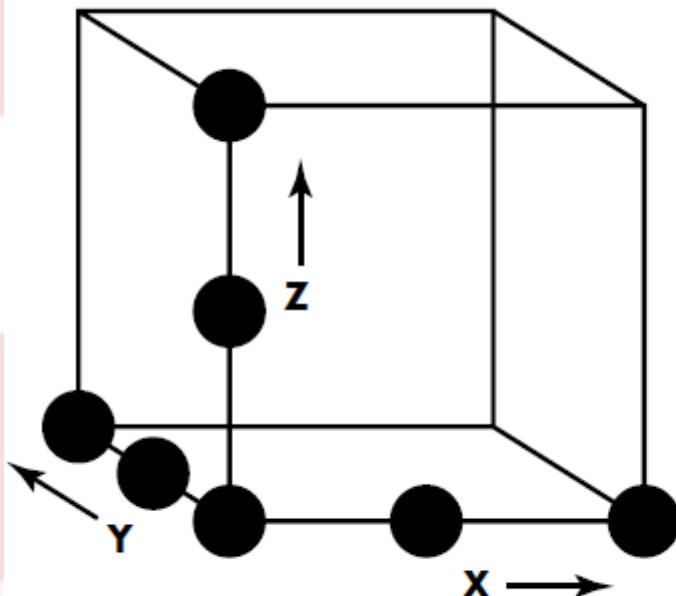
# Tabela de Decisão

Condição	1	2	3	4	5
Cartão válido	N	S	S	S	S
Senha válida	-	N	N	S	S
Senha inválida pela terceira vez	-	N	S	N	N
Saldo disponível	-	-	-	N	S
Ação					
Rejeita cartão	S	N	N	N	N
Informar senha novamente	N	S	N	N	N
Apreende cartão	N	N	S	N	N
Escolhe outra opção	N	N	N	S	N
Libera dinheiro	N	N	N	N	S

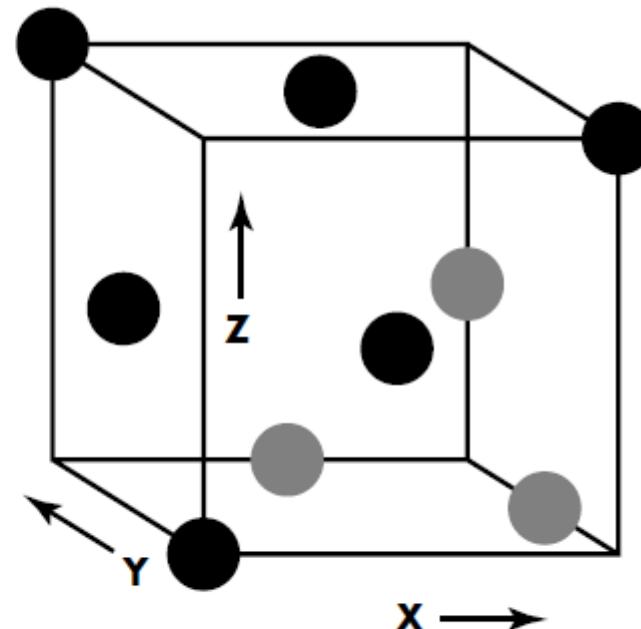
# Teste de Matriz Ortogonal

- Também utilizado para teste de regiões ou componentes
- Utiliza-se uma visão geométrica das entradas
  - Um cubo
  - Realiza-se um distribuição dos casos nesse espaço
  - Esta distribuição encontra-se balanceada
- Isso é a *Matriz Ortogonal L9*

# Matriz Ortogonal



**One input item at a time**



**L9 orthogonal array**

# Matriz Ortogonal – L9

## Exemplo

- Considere a função envie de uma aplicação de fax. Quatro parâmetros são passados P1, P2, P3, P4, são passados para a função envie. Cada um assume três valores específicos:
  - P1 = 1, envie já
  - P1 = 2, envie uma hora mais tarde
  - P1 = 3, envie depois da meia noite
- P2, P3, P4 também assumiram valores de 1, 2, 3, significando outras funções de envie
- Quantidade de Casos de Teste
  - $3^4 = 81$

# Teste de Matriz Ortogonal

Teste	P1	P2	P3	P4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

# Teste de Matriz Ortogonal

- Resultado
  - Falha singular é detectada
  - Falhas duplas também
  - Encontra-se algumas no “multimodo”
- Como construir uma matriz ortogonal
  - A escolha de 2 colunas aleatoriamente produz todas as combinações de valores para os pares envolvidos

# Teste de Caso de Uso

- Casos de teste podem ser derivados de casos de uso
- Um caso de uso descreve interações entre os atores e o sistema
- Para gerar os casos de testes a partir dos casos de uso considera-se:
  - Pré-condições
  - Outras possíveis condições
  - Resultados esperados
  - Pós-condições

# Teste de Caso de Uso

- Descobrir defeitos no fluxo do processamento durante a utilização do software
- Construir testes de aceite junto aos usuários finais
- Descobrir defeitos de integração de componentes

# Notícia

O foguete Ariane 5 explodiu na subida devido tão-somente a um defeito de software envolvendo a conversão de um valor de ponto flutuante com 64 bits em um inteiro de 16 bits.

O foguete e seus 4 satélites estavam sem seguro e valiam US\$ 500 milhões. Um teste de sistema abrangente teria encontrado o erro, mas foi vetado por razões orçamentárias!

# Exercícios

- Um programa recebe campos numéricos como entrada. Os valores menores que 50 devem ser rejeitados. Valores entre 50 e 101 são considerado como válidos. Valores maiores ou iguais a 102 devem ser rejeitados.
  - Usando partição de equivalência, gere valores de entrada que validem todas estas condições
  - Quais os valores de entrada abrangem a maioria dos limites deste programa?

# Exercícios

Código do produto:	<input type="text"/>
Quantidade:	<input type="text"/>
Valor Unitário:	<input type="text"/>
Valor Total:	<input type="text"/>
<input type="button" value="Confirma"/>	<input type="button" value="Cancela"/>

- Especificações do sistema:
  - O código do produto dever ser um número entre 0 e 99999
  - A quantidade deve ser um valor entre 1 e 100
  - O valor total máximo de um pedido é de R\$ 99999,99

# Exercícios

- Caso se queira cobrir todas as possibilidades de combinação de partições possíveis qual o número de testes que precisam ser realizados?

Condição	1	2	3	4
Cartão válido	N	S	S	S
Senha válida	-	N	N	S
Senha inválida pela terceira vez	-	N	S	N
Ação				
Rejeita cartão	S	N	N	N
Informar senha novamente	N	S	N	N
Apreende cartão	N	N	S	N

# Exercícios

- Quais as características esperadas de uma pessoa que vai realizar os teste da caixa-preta?
- Que tipo de conhecimento ela deve ter?
- Ela deve ser um profundo conhecedor de ferramentas de programação?

# Exercícios

- Considere a especificação de requisitos:
  - A rotina recebe como parâmetros o tipo de livro (HARDCOVER, SOFTCOVER, ou LOOSE) e sua quantidade (1-9999).
- Os casos de teste abaixo são alguns dos casos de teste para a especificação acima.

Caso de Teste Número	Dados de Entrada
1	(XYZ, 0)
2	(HARDCOVER, 9999)
3	(SOFTCOVER, 50)
4	(LOOSE, 100)

- Considerando os testes caixa preta vistos em aula, estes casos de testes estão completos? Justifique.

# Exercícios

- O programa aceita três valores inteiros
  - Os três valores ( $a$ ,  $b$ ,  $c$ ) representam as dimensões dos lados de um triângulo. O programa imprime uma mensagem que informa se o triângulo é escaleno, isósceles ou eqüilátero.
  - Os lados do triângulo devem ser inteiros positivos cujos valores não excedam 10.
  - Identifique quatro casos de teste usando apenas análise de valores limite.

# Obrigado

Ana Claudia Rossi

[ana.rossi@mackenzie.br](mailto:ana.rossi@mackenzie.br)

