

Teste de Software

Estratégias de Teste Funcional de Software

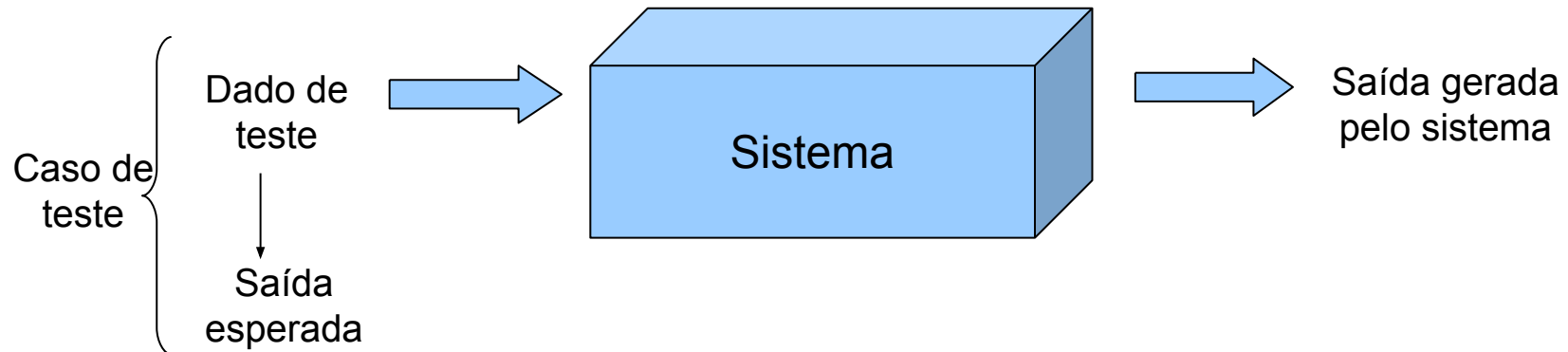
Lesandro Ponciano

Objetivos da Aula

- Contextualizar a técnica de Teste Funcional de software
- Introduzir estratégias de Teste Funcional de software
 - Particionamento de Equivalência
 - Análise do Valor Limite
 - Teste Funcional Sistemico
 - Grafo Causa-Efeito
 - Error-Guessing

A Técnica de Teste Funcional

- O sistema é considerado como uma **caixa preta**
 - Por essa razão, Teste Funcional é muitas vezes chamado de teste “caixa preta”



A “saída gerada pelo sistema” é igual à “Saída Esperada”?

- Sim: **Sucesso** o sistema passou no teste
- Não: **Falha** o sistema não passou no teste

Caso de Teste

- **Pré-condições**
 - Diz o estado obrigatório do software antes do início do teste
 - Se não for atendido o teste pode falhar sem que exista um defeito
- **Entradas**
 - São dados a serem fornecidos ao software para execução
- **Ação**
 - Execução que faz o software e que será objeto do teste
- **Resultados esperados**
 - Saídas conhecidas (correspondentes à entrada) e que espera-se que o software gere
- **Pós-condições**
 - Diz o estado obrigatório do software após a execução da ação

Características do Teste Funcional

- Teste Funcional é independente da implementação
 - Plataforma
 - Framework
 - Linguagem de programação
- Os casos de teste são derivados da especificação do sistema que está sendo testado
 - Idealmente deve haver uma especificação detalhada
- Reflexão: **quantos casos de teste possíveis existem para se testar cada funcionalidade de um sistema?**

Exemplo: Número de Casos de Teste

- Considere o seguinte comando especificado da seguinte forma
 - Comando: *transforma*
 - Entrada: *int*
 - Saída: *int*
- Quantos casos de teste?
 - Considerando valores inteiros de 16 bits
 - Menor valor: -32768
 - Maior valor: 32767
 - 65536 valores diferentes de entrada

Exemplo: Escolha de Casos de Teste

- Considere o seguinte comando especificado da seguinte forma

- Comando: *transforma*
- Entrada: *int*
- Saída: *int*

transforma foi implementado como:
saída=(entrada-1)/30000
Mas deveria ser
saída=(entrada+1)/30000

Entrada	Saída esperada	Saída obtida
1		
42		
32000		
-32000		

Projeto de Plano de Teste

- Formalização (**problema de otimização**)
 - Seja P um programa a ser testado.
 - Seja $D(P)$ o domínio de todos os casos de teste para P .
 - Testar todo o domínio $D(P)$ pode ser inviável
 - A busca é por um conjunto T (*sendo* $T \subset D(P)$), bastante reduzido em relação a $D(P)$ mas que, de certa maneira, representa cada um dos elementos de $D(P)$
- Informalmente
 - Qual o subconjunto de todos os casos de teste possíveis tem a maior probabilidade de detectar a maioria dos defeitos?

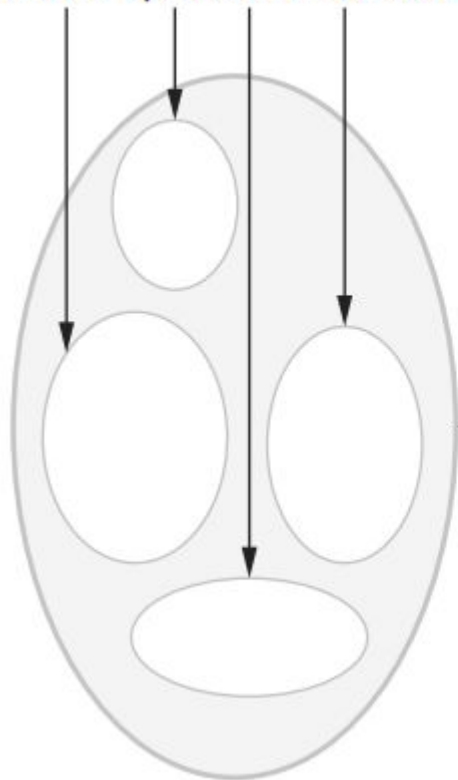
Estratégias de Teste Funcional

- Uma estratégia de teste determina os critérios que devem ser seguidos para se gerar casos de teste
- Estratégias clássicas
 - 1) Particionamento de Equivalência
 - 2) Análise do Valor Limite
 - 3) Teste Funcional Sistêmico
 - 4) Grafo Causa-Efeito
 - 5) *Error-Guessing*

1 Particionamento de Equivalência

- O domínio de entrada do sistema é dividido em **classes de equivalência**
- Dados de entrada em uma mesma classe de equivalência
 - levariam o sistema a se comportar de forma similar
 - revelariam o mesmo defeito
- Apenas um dado de uma classe precisa ser usado para testar o sistema
 - Reduz o domínio de entrada a um tamanho viável

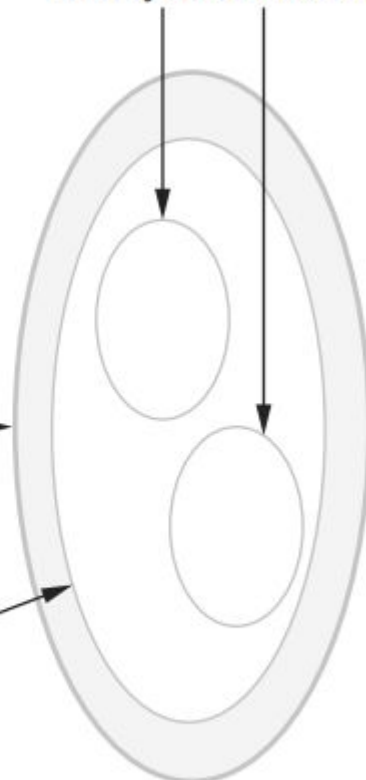
Partições de equivalência de entrada



Entradas possíveis

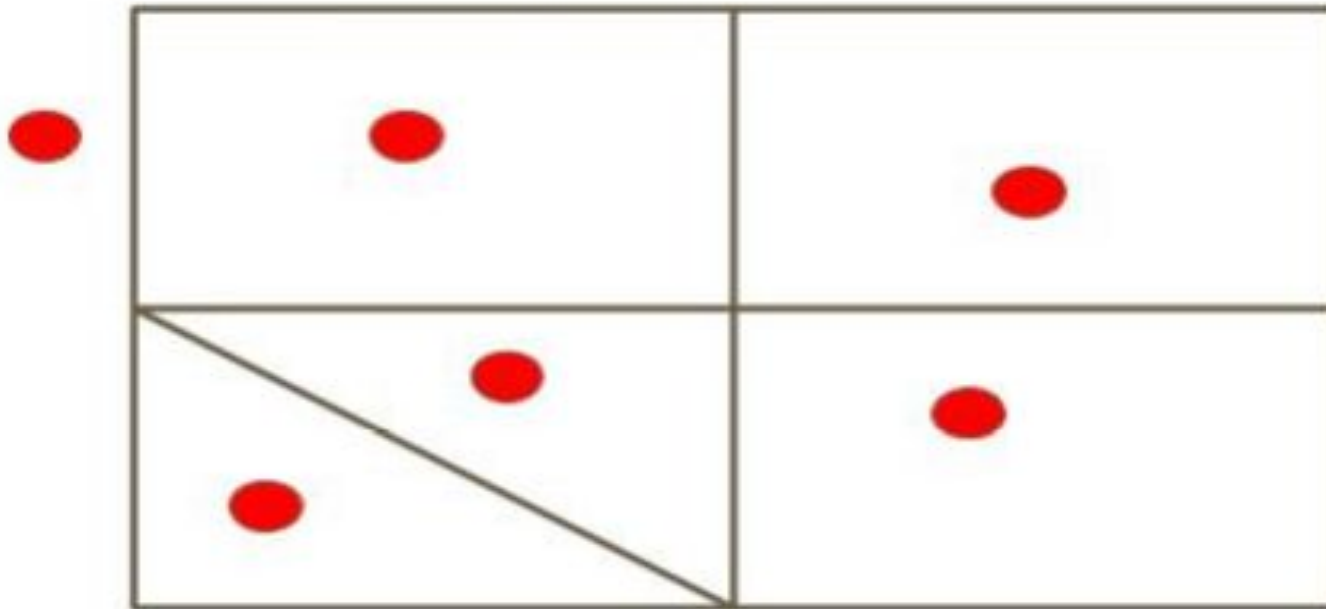


Partições de saída



Saídas corretas

Saídas possíveis



- Divide o domínio de entrada do software em classes de dados (classes de equivalências)
- Casos de teste são derivados a partir das classes de equivalência (válidas e inválidas)

Exemplo

- Digamos que uma especificação de programa define que o programa
 - aceita entradas de 4 a 10
 - que sejam valores inteiros
 - de cinco dígitos superiores a 10.000.
- Podemos usar essas informações para identificar as partições de entrada e de possíveis valores de entrada de teste

Menor do que 4	Entre 4 e 10	Mais do que 10
----------------	--------------	----------------

Número de valores de entrada

Menor do que 10000	Entre 10000 e 99999	Mais do que 99999
--------------------	---------------------	-------------------

Valores de entrada

Diretrizes

- As classes de equivalência podem ser definidas seguindo as seguintes diretrizes
 - 1) Se há intervalo de valores: 1 classe válida e 2 inválidas
 - 2) Se há número de valores: 1 classe válida e 2 inválidas
 - 3) Se há conjunto de valores cada um deles processado de uma forma: 1 classe válida para cada valor e 1 inválida
 - 4) Se a condição de entrada especifica uma situação do tipo "deve ser de tal forma"
 - 1 classe válida
 - 1 classe inválida

Exemplos

1) Intervalo de valores

- *calculaPagamento t l*
- *t* é taxa de participação no lucro (fica entre 0 e 1) e *l* é o lucro

2) Número (quantidade) de valores

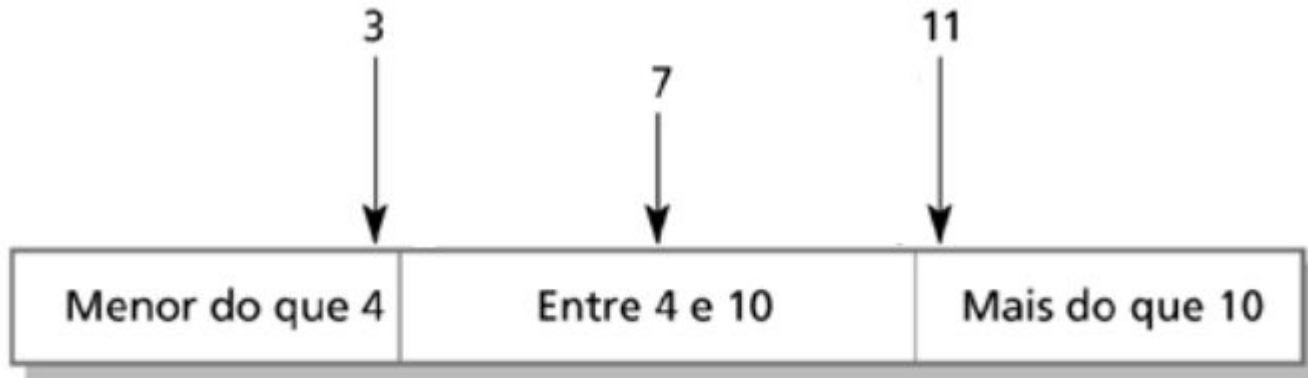
- *formaEquipe nalunos*
- Equipe de alunos pode ter no mínimo 2 e no máximo 4 alunos

3) Conjunto de valores

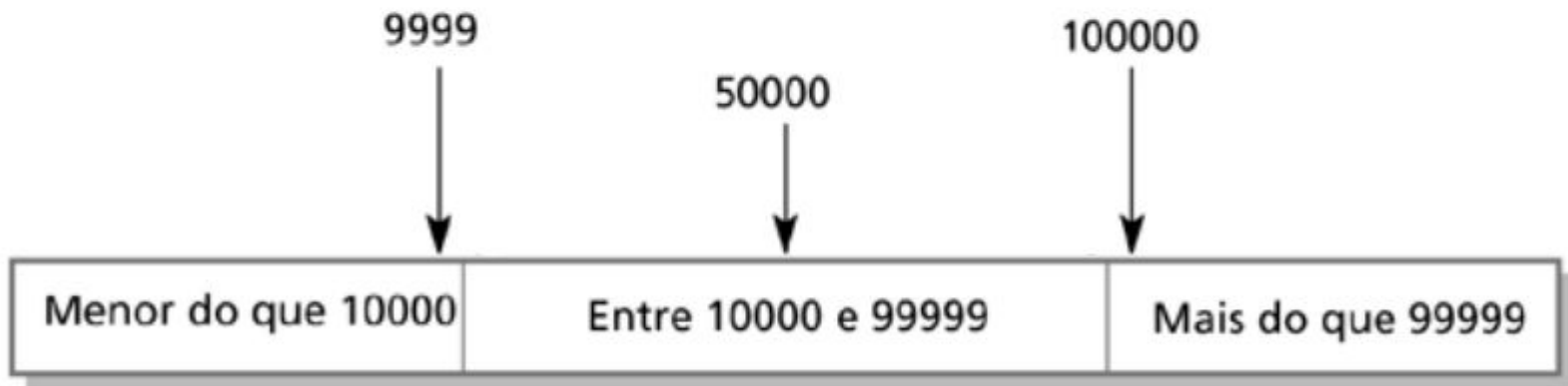
- *dosaMedicacao paciente*
- Paciente pode ser "criança", "adulto", "idoso"

4) "Deve ser de tal forma"

- *validaNovaSenha senha*
- O primeiro caractere da senha deve ser um número



Número de valores de entrada



Valores de entrada

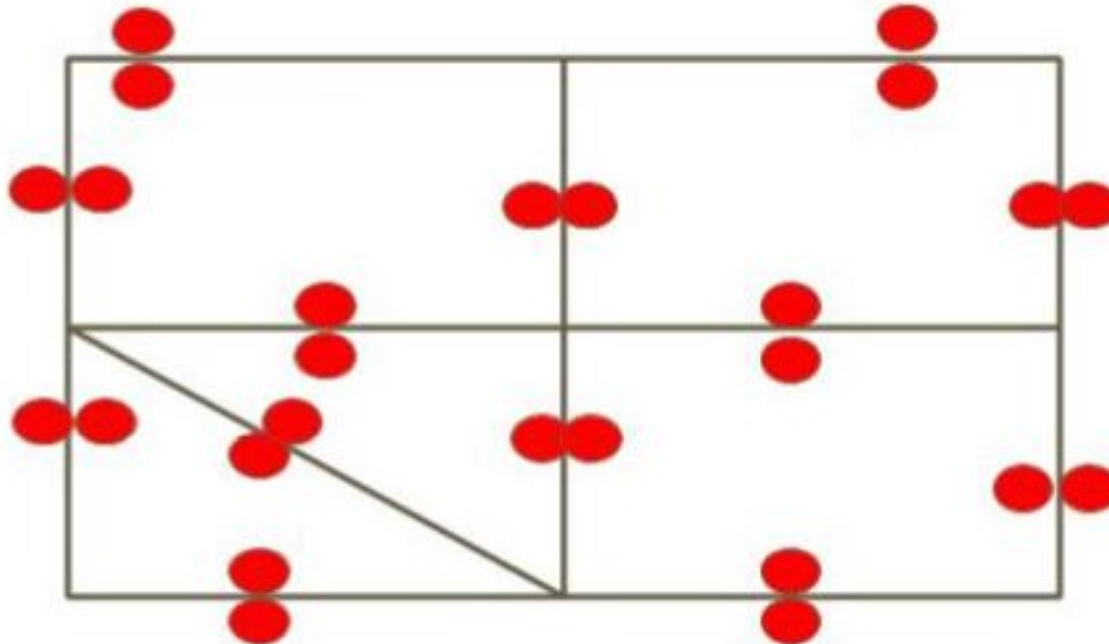
Elemento Distinto

- Elemento distinto
 - É uma valor ou elemento que tem tratamento diferenciado
- Se em uma classe houver um elemento distinto, isto implica em uma partição em classes menores
- Exemplo:
 - Tratamento diferenciado para o valor zero (0) em alguns cálculos matemáticos

2

Análise do Valor Limite

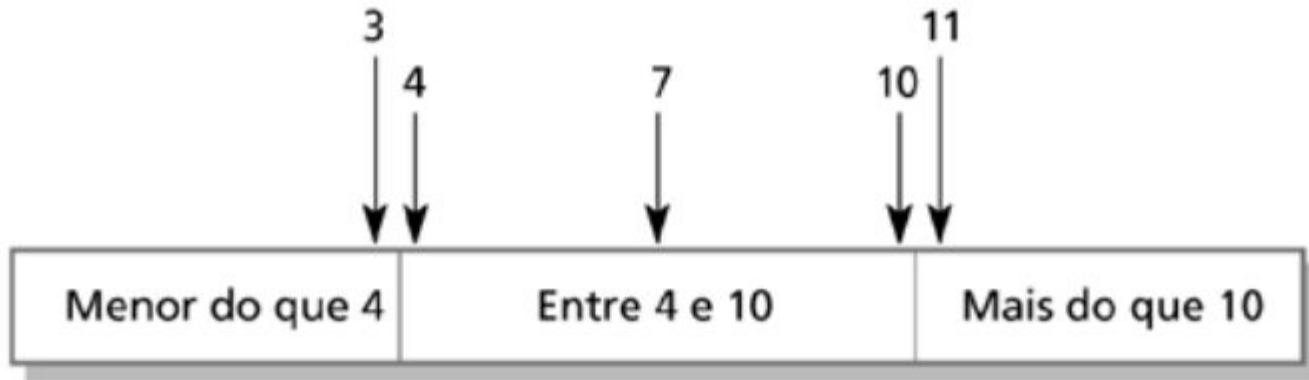
- Conjectura-se que casos de teste que exploram condições limites têm maior probabilidade de encontrarem defeitos
- Valor limite
 - Valor imediatamente acima do limite da classe de equivalência
 - Valor imediatamente abaixo do limite da classe de equivalência



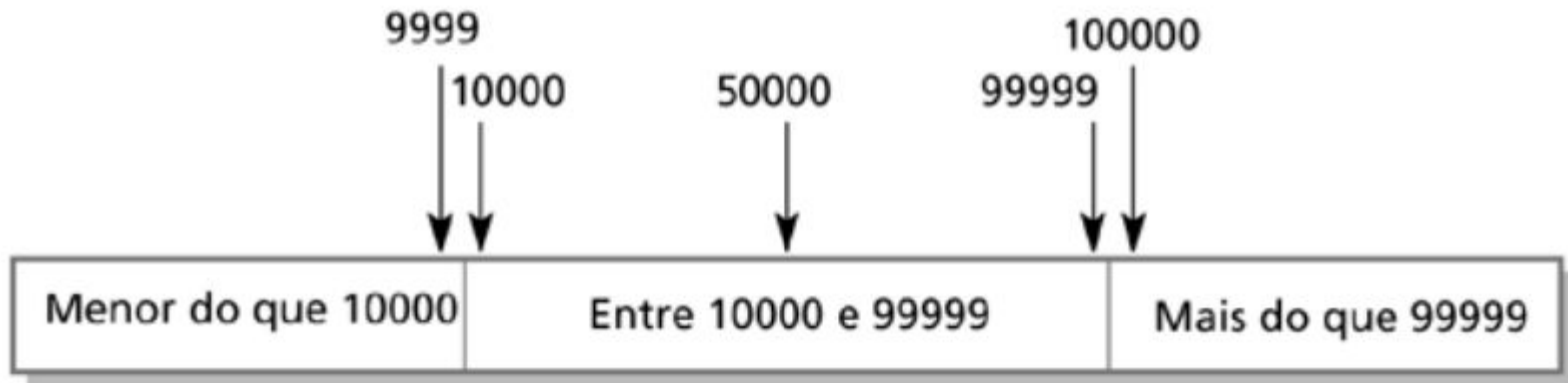
- Complementa o Particionamento de Equivalência
- Considera que os limites de uma classe de equivalência são fontes propícias a defeitos
- Casos de teste nos limites de classes de equivalência

Diretrizes

- 1) Definir classes vizinhas no intervalo. Exemplo, para intervalo válido em -1.0 e +1.0, testar
 - -1.1;-1.0;-0.9; +0.9;+1.0;+1.1
- 2) Definir limites vizinhos à quantidade de valores. Exemplo, para testar de 1 a 255 valores, testar:
 - 0;1;2;254;255;256
- 3) Usar a diretriz 1 para condições de saída
- 4) Usar a diretriz 2 para condições de saída
- 5) Para entrada e saída de conjunto ordenado, maior atenção ao primeiro e último elemento
- 6) Usar a intuição para definir outras condições limite



Número de valores de entrada



Valores de entrada

3

Teste Funcional Sistêmico

- Combina Particionamento de Equivalência e Análise do Valor Limite
- Requer
 - Dois casos de teste de cada partição
 - Avaliação nos limites de cada partição

Diretrizes

- Para valores numéricos
 - de **entrada**: testar todos os valores discretos, testar os extremos e um valor interior do intervalo de valores
 - de **saída**: gerar cada um deles e gerar cada um dos extremos e um valor interior do intervalo de valores
- Tipos de valores diferentes e casos especiais, exemplo 0, valores em branco, etc.
 - Explorar na entrada e na saída
- Valores ilegais
 - Incluir casos de teste com valores que são entradas ilegais

Diretrizes

- Valores reais (que incluem casas decimais)
 - Limites de números reais podem não ser exatos, mas deve haver caso de testes aproximados em uma margem de erro
 - Exemplo: Números reais bem pequenos e também 0
- Intervalos variáveis, que dependem de um valor de entrada.
Exemplo, para x , que varia de 0 a y , testar:
 - Legais: $x=y=0$; $x=0<y$; $0<x=y$; $0<x<y$
 - Ilegais: $y=0<x$; $0<y<x$; $x<0$; $y<0$

Diretrizes

- Arranjos
 - Testar como única estrutura
 - Testar como coleção de estruturas
 - Testar subestruturas independentes
- Dados do tipo texto ou *string*
 - Validar comprimento
 - Validar tipo de caracteres
 - Alfabéticos
 - Alfanuméricos
 - Caracteres especiais

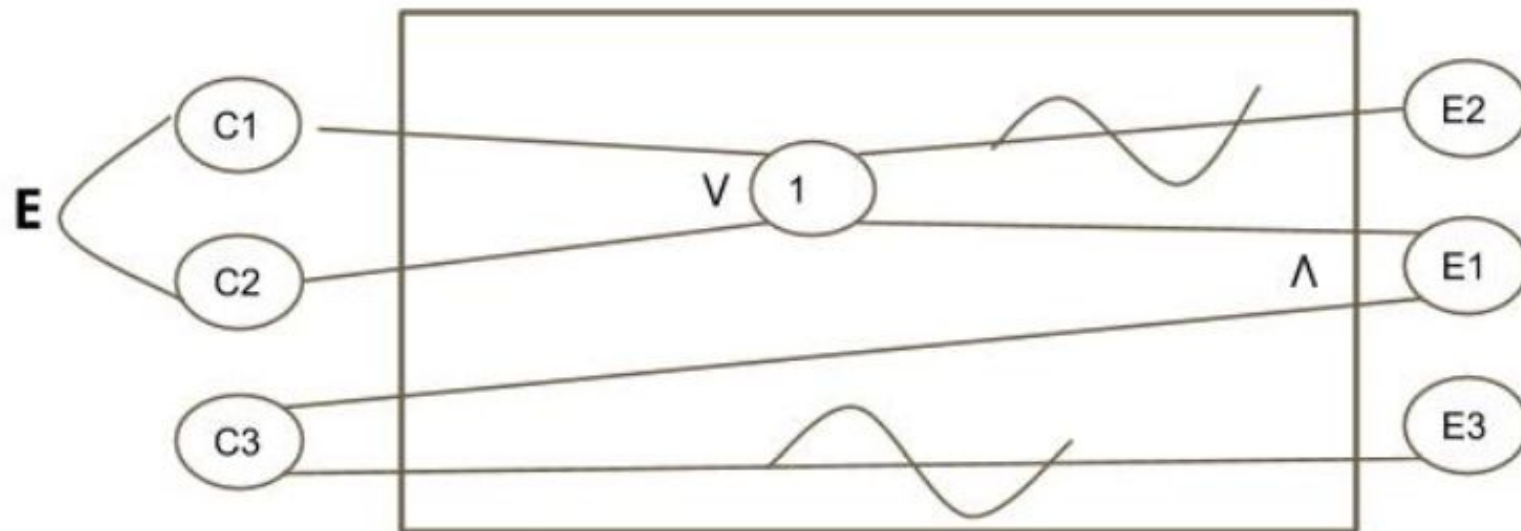
4

Grafo Causa-Efeito

- Explora combinações de entrada de dados
- **Causas**
 - condições de entrada, estímulo ou qualquer coisa que provoque uma reação do sistema (valor lógico)
- **Efeitos**
 - ações realizadas em resposta às diferentes condições de entrada
 - saídas, mudanças de estado ou qualquer resposta observável

Exemplo

- O primeiro caractere deve ser 'A' ou 'B'. O segundo caractere deve ser um número. Assim sendo, o arquivo deve ser atualizado. Se o primeiro caracter for incorreto, escrever X12. Se o segundo caractere não for número, escrever X13.



C1= o primeiro caractere é 'A'
 C2= o primeiro caractere é 'B'
 C3= o segundo caractere é um número

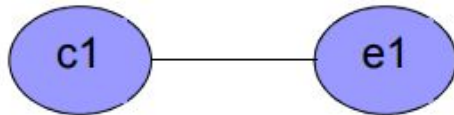
E1=Atualiza o arquivo
 E2= Escreve X12
 E3=Escreve X13

- Grafo que representa conjuntos de condições sobre entradas (causas) e as ações correspondentes do sistema (efeitos).
- A partir do grafo, monta-se uma tabela de decisão, e a partir desta, os casos de teste

Diretrizes

- 1) Dividir a especificação do software em partes
- 2) Identificar as causas e efeitos na especificação
- 3) Criar o grafo de causa-efeito, ligando as causas aos efeitos
- 4) Adicionar anotações ao grafo, as quais descrevem combinações das causas e efeitos devido às restrições sintáticas ou de ambiente
- 5) Converter o grafo em tabela de decisão
- 6) Converter as colunas da tabela em casos de teste

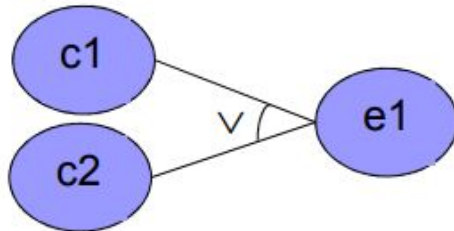
Operadores



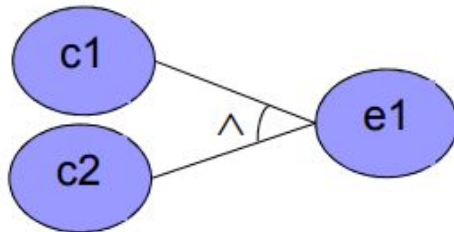
Identidade: Se $c1=1$, então $e1=1$, senão $e1=0$



Negação (NOT): Se $c1=1$, então $e1=0$, senão $e1=1$

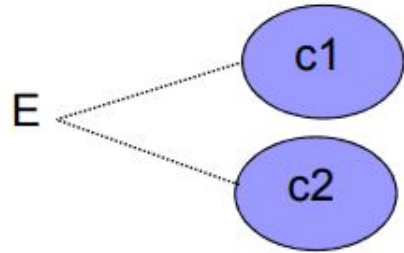


Ou (OR): Se $c1=1$ ou $c2=1$, então $e1=1$, senão $e1=0$

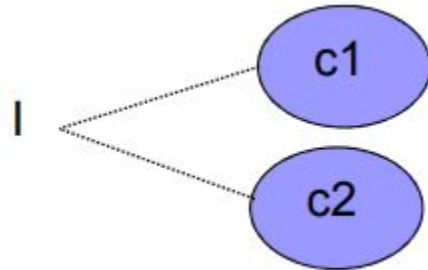


E (AND): Se $c1=1$ e $c2=1$, então $e1=1$, senão $e1=0$

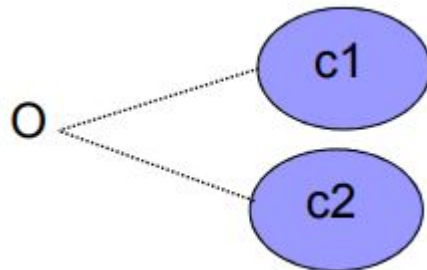
Restrições



Restrição E: c1 e c2 não podem ser 1 simultaneamente

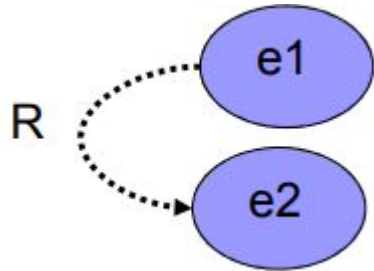


Restrição I: c1 e c2 não podem ser 0 simultaneamente

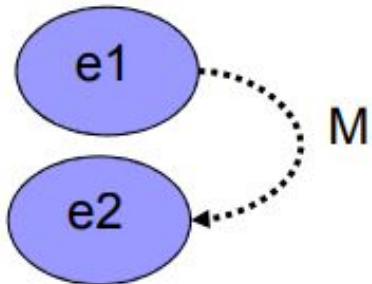


Restrição O: um e somente um entre c1 e c2 deve ser igual a 1

Restrições



Restrição R: é impossível $e1=1$ se $e2=0$



Restrição M: se $e1=1$, então $e2=0$

Tabela de Decisão

- Mostra os efeitos que ocorrem para todas as combinações de causas
- Se podem ocorrer n causas, então a tabela contém 2^n entradas
- Passos para elaborar a tabela
 - 1) Selecionar um efeito com valor 1
 - 2) Rastrear todas as combinações de causas (sujeitas a restrições) que fazem com que esse efeito seja 1
 - 3) Criar uma coluna na tabela para cada combinação de causa
 - 4) Determinar, para cada combinação, os estados de todos os outros efeitos, anotando na tabela

Tabela de Decisão

Entradas	O primeiro caractere é 'A'	V	V	V	V	F	F	F	F
	O primeiro caractere é 'B'	V	F	F	V	F	V	F	
	O segundo caractere é um número	V	V	F	F	V	V	F	F
Saídas	Atualiza o arquivo	-	V		-	V			
	Escreve X12	-			-		V		V
	Escreve X13	-		V	-			V	V

Sobre o Passo 2

- Quando o nó for do tipo *OR* e a saída deva ser 1
 - Nunca atribuir mais de uma entrada com valor 1 simultaneamente
- Quando o nó for do tipo *AND* e a saída deva ser 0
 - Todas as combinações de entrada que levem à saída 0 devem ser enumeradas
 - Porém, se uma das entradas é 0 e uma ou mais das outras entradas é 1, não é necessário enumerar todas as condições em que as outras entradas sejam iguais 1
- Quando o nó for do tipo *AND* e a saída deva ser 0
 - Somente uma das condições em que todas as entradas sejam 0 precisa ser enumerada

Criação dos Casos de Teste

- Converter cada coluna da tabela de decisão em um caso de teste

5

Error Guessing

- Abordagem *ad hoc*
 - Supõe-se, por intuição e experiência, alguns tipos prováveis de erros e, a partir disso, define os casos de teste para detectá-los
- Exemplo, para um módulo de ordenação, testar
 - Lista de entrada vazia
 - Lista com apenas uma entrada
 - Todas as entradas com o mesmo valor
 - Lista de entrada já ordenada

Atividade de Fixação

- 1) Qual o principal problema que o projeto de casos de teste tenta resolver?
- 2) Descreva a principal característica de cada uma das seguintes estratégias de projeto de casos de teste para Teste Funcional
 - 1) Particionamento de Equivalência
 - 2) Análise do Valor Limite
 - 3) Teste Funcional Sistemico
 - 4) Grafo Causa-Efeito
 - 5) *Error-Guessing*

Referências

- Delamaro, M. E., Maldonado, J. C., & Jino, M. (2016). Introdução ao teste de software. Rio de Janeiro: Elsevier. (Capítulo 2)
- Myers, Glenford J. et al (2004) "The Art of Software Testing." 2ed. New York, NY, USA: John Wiley & Sons. (Capítulo 4)

Teste de Software

Prof. Dr. Lesandro Ponciano

<https://orcid.org/0000-0002-5724-0094>