Tópicos Especiais em Engenharia de Software

Aula 01

Técnicas e critérios de teste

- Segundo Pressman (1997) técnicas de teste fornecem diretrizes para projetar testes que exercitam a lógica interna do componente de software, bem como os domínios de entrada e saída.

Tipos de teste

- Existem basicamente duas categorias de teste, teste baseado na especificação e teste baseado no programa. Com base nestas categorias, os testes podem ser classificados em:
  - Funcional (baseado na especificação) e;
  - Estrutural (baseado no programa).

Teste Funcional

Teste Funcional

- Também conhecido como Técnica de caixa-preta, pois não considera a estrutura interna do produto a ser testado, leva em conta apenas a entrada e a saída.
- Baseia-se apenas na especificação do sistema para derivar requisitos de teste.
- Não se preocupa com detalhes de implementação.

Passos da Técnica Funcional

- Passos básicos para aplicar um critério de teste funcional:
  - A especificação de requisitos é analisada.
  - Entradas válidas são escolhidas para determinar se o produto em teste comporta-se corretamente.
  - Entradas inválidas são escolhidas para verificar se estas são detectadas e manipuladas adequadamente.
  - Os casos de testes são construídos (saídas são determinadas para cada entrada).
  - O conjunto de teste é executado e as saídas obtidas são comparadas com as saídas esperadas.
  - Um relatório é gerado para avaliar o resultado dos testes.

Critérios de Teste Funcional

- Diversos critérios são definidos para técnica funcional de teste, dentre eles os mais utilizados são:

- Particionamento em Classes de Equivalência
  - Divide o domínio de entrada (e de saída) de um programa em classes de equivalência, a partir das quais derivam-se os casos de teste.

- Análise do Valor Limite
  - Complementa o critério Particionamento de Equivalência, exigindo casos de teste nos limites (fronteiras) de cada classe de equivalência.

- Grafo de Causa-Efeito
  - Verifica o efeito combinado de dados de entrada.
  - Causas (condições de entrada) e efeitos (ações) são identificados e combinados em um grafo.
  - Tabela de Decisão -> Casos de Teste

Particionamento em classes de equivalência

Particionamento em classes de equivalência

- Critério utilizado para reduzir o número de casos de teste, visando garantir uma boa cobertura dos casos de teste.

- A partir das condições de entrada de dados identificadas na especificação, divide-se o domínio de entrada do programa em classes de equivalência válidas de inválidas.

Exemplo

- O programa solicita do usuário um inteiro positivo no intervalo entre 1 e 20 e então lê uma cadeia de caracteres desse comprimento. Após isso, o programa solicita um caractere e retorna a posição na cadeia em que o caractere é encontrado pela primeira vez ou uma mensagem indicando que o caractere não está presente na cadeia. O usuário tem a opção de procurar vários caracteres.

Exemplo

Variável de entrada	Classes de equivalência válidas	Classes de equivalência inválidas
Tamanho da cadeia (T)	$1 \leq T \geq 20$ (1)	T < 1 (2) e T > 20 (3)
Opção de procurar mais caracteres (O)	S (4) N (5)	Outro (6)
Caractere procurado (C)	Pertence (7) Não pertence (8)	

Exemplo

- Computando casos de teste
- As classes obtidas são usadas para computar casos de teste:
  - Definem-se casos de teste para cobrir o maior número de classes válidas possíveis
  - Para cada classe não válida é criado um caso de teste específico.

Exemplo

- Casos de teste:

- T = 3 CC = abc C = c O = s C = k O = n
  - Saída: Encontrado na posição 3; Não encontrado
  - Classes cobertas: 1, 4, 5, 7, 8 (todas as válidas)

- T = 3 CC = abc C = c O = w
  - Saída: Responda com s ou n
  - Classe coberta: 6 (inválida)

*CC (cadeia de caracteres)

Vantagens

- Reduz significativamente o número de casos de teste em relação ao teste exaustivo.
- Mais adequado para o teste de produtos com domínios de entrada divididos em intervalos ou conjuntos.
- Assume que os valores dentro da mesma classe são equivalentes (isso nem sempre é verdade!).
- Importante empregar outros critérios de teste!!
- Aplicável em todas as fases de teste: unidade, integração e sistema.

Análise do Valor Limite

Análise do Valor Limite

- Complementa o critério anterior, os casos de teste são escolhidos nas fronteiras das classes de equivalência, pois nesses pontos se concentram um grande número de erros.

Passos

- Identificar as classes de equivalência (requisitos de teste do critério).
- Identificar os limites de cada classe.
- Criar casos de teste para os limites escolhendo:
  - Um ponto abaixo do limite.
  - O limite.
  - Um ponto acima do limite.
- Observe que acima e abaixo são termos relativos e dependentes do valor dos dados.
- Casos de teste adicionais podem ser criados dependendo dos recursos disponíveis.

Exemplo – Programa Identifier

```
/*****************************************************************************************
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
*****************************************************************************************/

            #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 &&
                  (length >= 1) && (length < 6))
 /*  9 */       {
 /*  9 */           printf ("Valido\n");
 /*  9 */       }
 /* 10 */       else
 /* 10 */       {
 /* 10 */           printf ("Invalid\n");
 /* 10 */       }
 /* 11 */   }

            int valid_s(char ch)
 /* 1 */    {
 /* 1 */        if(((ch >= 'A') &&
                   (ch <= 'Z')) ||
                  ((ch >= 'a') &&
                   (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') &&
                   (ch <= 'Z')) ||
                  ((ch >= 'a') &&
                   (ch <= 'z')) ||
                  ((ch >= '0') &&
                   (ch <= '9')))
 /* 2 */        {
 /* 2 */            return (1);
 /* 2 */        }
 /* 3 */        else
 /* 3 */        {
 /* 3 */            return (0);
 /* 3 */        }
 /* 4 */    }
```

Exemplo

- Classes Válidas e Inválidas:

Condições de entrada	Classes válidas	Classes inválidas
Tamanho t do identificador	$1 \leq t \geq 6$ (1)	1 > t (2) t > 6 (3)
Primeiro caractere c é uma letra	Sim (4)	Não (5)
Só contém caracteres válidos	Sim (6)	Não (7)

Exemplo

- Casos de teste:

Identificador	Resultado	Descrição
a	Válido	Primeiro minúsculo válido, tam. mínimo
b3	Válido	Segundo minúsculo válido, tam. mínimo
Xkl	Válido	Penúltimo maiúsculo válido, tam. mínimo
Z9	Válido	Último maiúsculo válido, tam. acima mínimo
xkl	Válido	Penúltimo minúsculo válido, tam. mínimo
zAaZ1	Válido	Último minúsculo válido, tam. abaixo máximo
AaZz91	Válido	Primeiro maiúsculo válido, tam. máximo
BaZz91	Válido	Segundo maiúsculo válido, tam. máximo
abcdefg	Inválido	Caracteres válidos, tam. acima máximo
@	Inválido	Primeiro minúsculo abaixo, tam. mínimo
[	Inválido	Primeiro minúsculo acima, tam. mínimo
`	Inválido	Primeiro maiúsculo abaixo, tam. mínimo
{	Inválido	Primeiro maiúsculo acima, tam. mínimo
A-&$#	Inválido	Caracteres inválidos, tam. máximo

Vantagens

- Reduz significativamente o número de casos de teste em relação ao teste exaustivo.

- Mais adequado para o teste de produtos com domínios de entrada divididos em intervalos ou conjuntos.

- Aplicável em todas as fases de teste: unidade, integração e sistema.

Grafo causa-efeito

Grafo causa-efeito

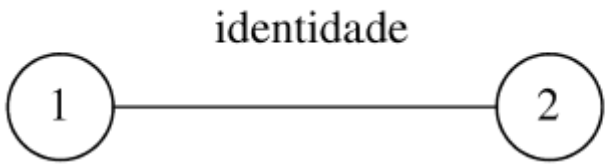
- Uma das limitações dos critérios anteriores é que eles não exploram combinações dos dados de entrada.
- O critério causa-efeito procura suprir essa deficiência.
- O grafo é uma linguagem formal na qual a especificação é traduzida.

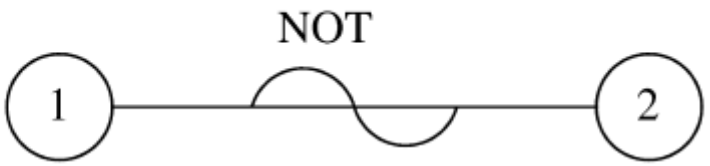
Passos

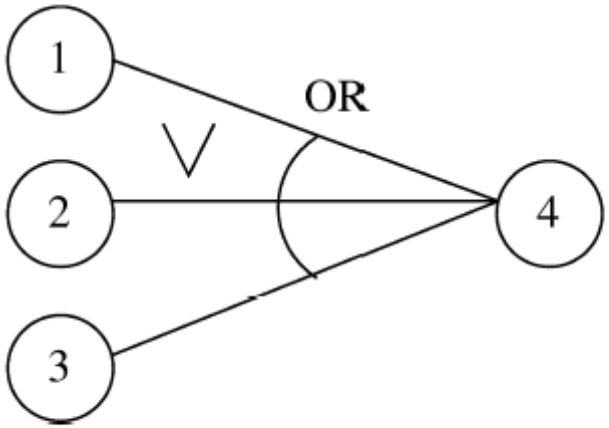
- Primeiramente, são levantadas as possíveis condições de entrada (causas) e as possíveis ações (efeitos) do programa.
- Em seguida é construído um grafo relacionando as causas e efeitos levantados.
- O grafo é convertido em uma tabela de decisão, a partir da qual são derivados os casos de teste.

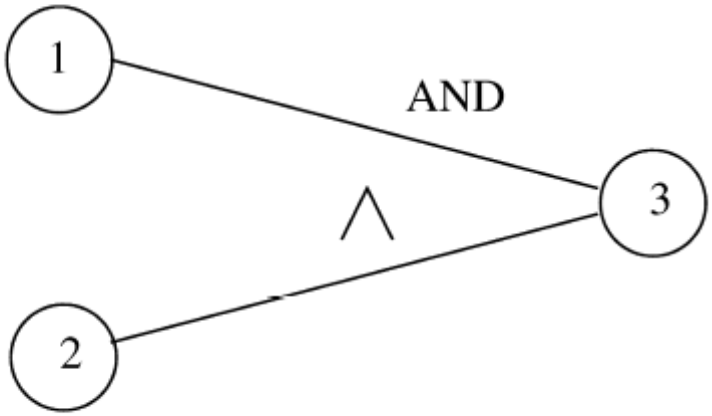
Notação

- Cada nó tem o valor 0 ou 1
  - Valor 1 indica que aquela determinado estado existe (é verdadeiro)
  - Valor 0 indica que estado não é verdadeiro
- Função identidade: se nó “1” é 1, então nó “2” é 1; senão nó “2” é 0.
- Função not: se nó “1” é 1, então nó “2” é 0; senão nó “2” é 1.
- Função or: se nó “1” ou “2” ou “3” é 1, então nó “4” é 1; senão nó “4” é 0.
- Função and: se ambos nós “1” e “2” são 1, então nó “3” é 1; senão nó “3” é 0.


identidade
1
2


NOT
1
2


1
OR
2
4
3


1
AND
3
2

Exemplo

- “Imprime Mensagens”: o programa lê dois caracteres e, de acordo com eles, mensagens serão impressas.
  - O primeiro caractere deve ser um “A” ou um “B”
  - O segundo caractere deve ser um dígito.
  - Nessa situação, o arquivo deve ser atualizado
  - Se o primeiro caractere é incorreto, enviar a mensagem X.
  - Se o segundo caractere é incorreto, enviar a mensagem Y

Exemplo

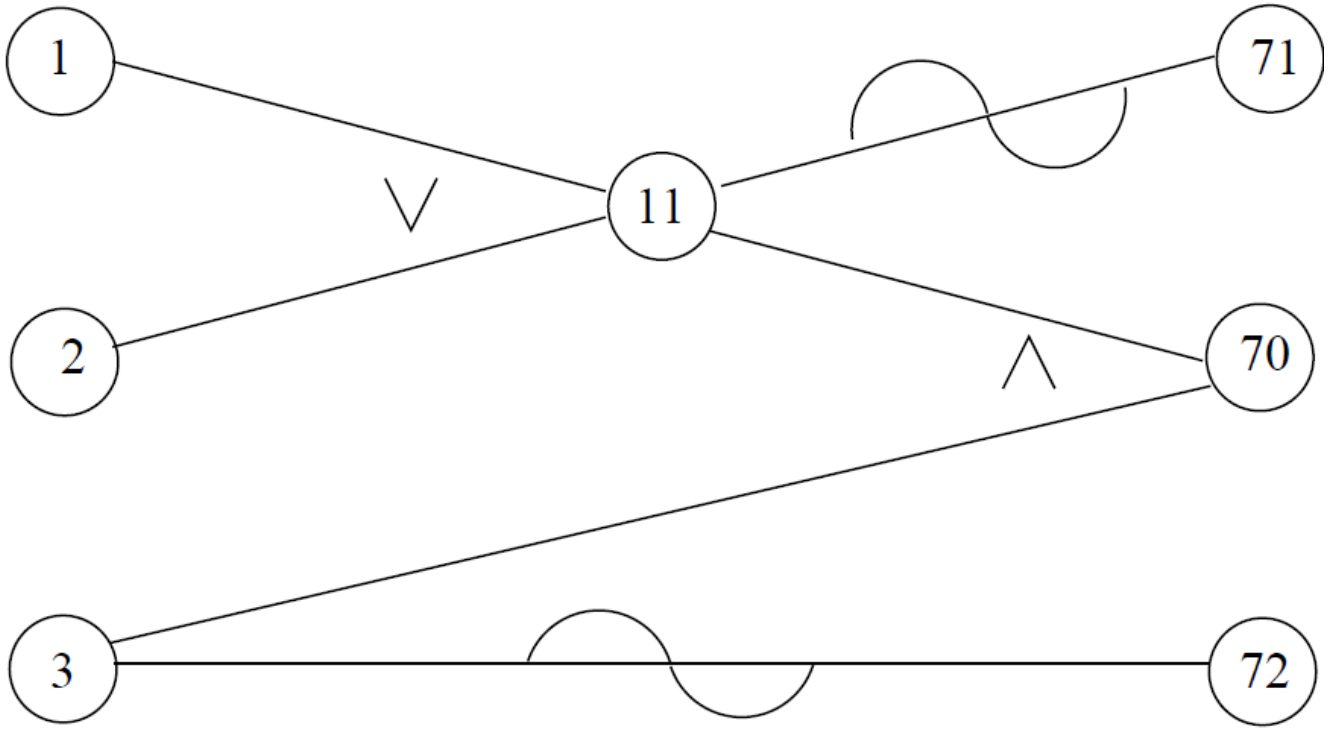
- Causas
  - 1 - caractere na coluna 1 é “A”
  - 2 - caractere na coluna 1 é “B”
  - 3 - caractere na coluna 2 é um dígito

Exemplo

- Efeitos
  - 70 - a atualização é realizada
  - 71 - a mensagem X é enviada
  - 72 - a mensagem Y é enviada

Exemplo

- O grafo

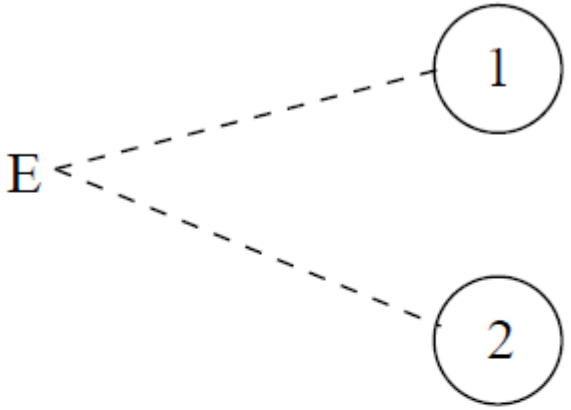


Exemplo

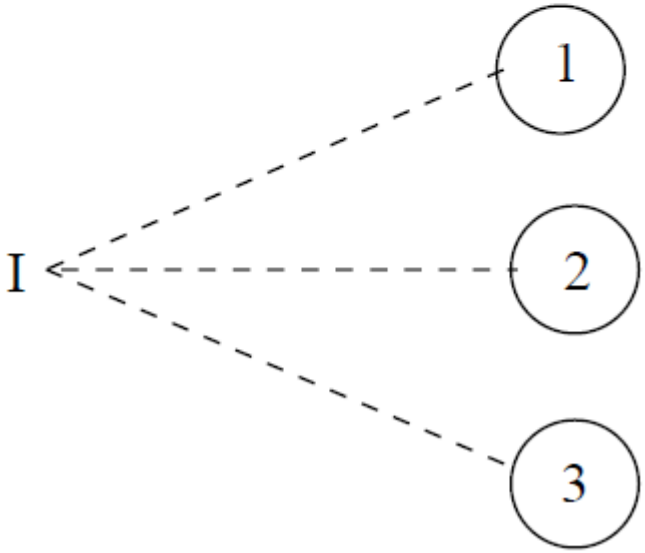
- Restrições:
- Para verificar se o grafo está correto, deve-se atribuir valores 0 e 1 para as causas e verificar se os efeitos assumem o valor correto.
- Existem combinações que não são possíveis.
  - Essas restrições devem ser anotadas no grafo.
    - Por exemplo, causas 1 e 2 não podem ser verdadeiras ao mesmo tempo

Exemplo

- Restrição E: no máximo um entre “1” e “2” pode ser igual a 1 (ou seja, “1” e “2” não podem ser 1 simultaneamente).

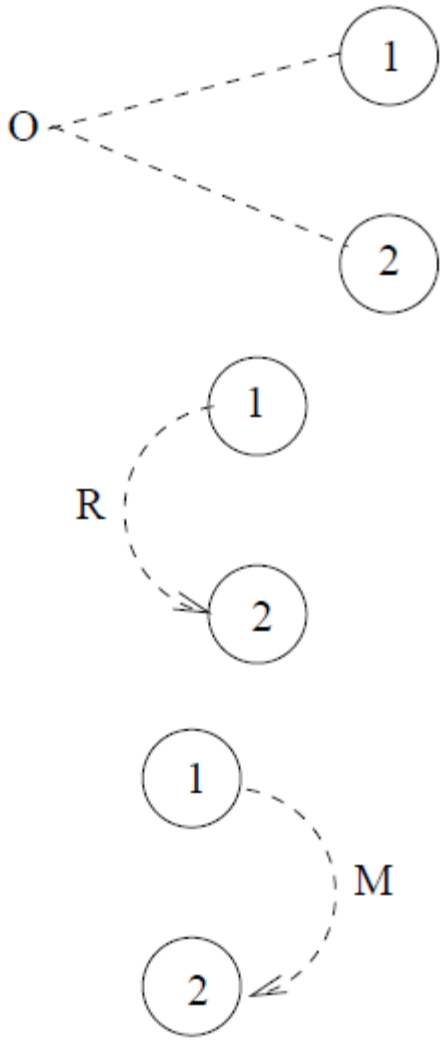


- Restrição I: no mínimo um entre “1”, “2” e “3” deve ser igual a 1 (ou seja, “1”, “2” e “3” não podem ser 0 simultaneamente)



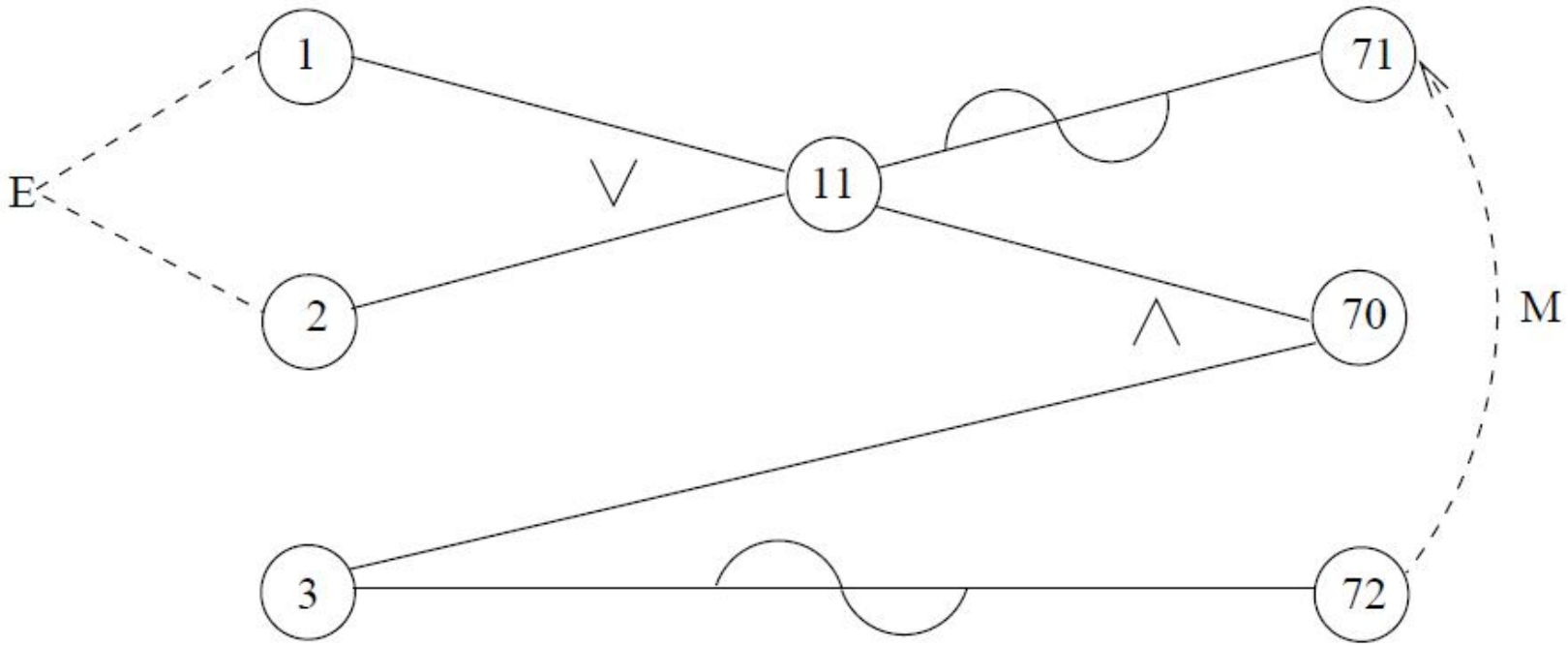
Exemplo

- Restrição O: um e somente um entre “1” e “2” deve ser igual a 1.
- Restrição R: para que “1” seja igual a 1, “2” deve ser igual a 1 (ou seja, é impossível que “1” seja 1 se “2” for 0).
- Restrição M: se o efeito “1” é 1 o efeito “2” é forçado a ser 0.



Exemplo

- Grafo com restrições



Exemplo

- Tabela de decisão
  - Forma de representar condições e ações
  - O próximo passo é estudar sistematicamente o grafo e construir uma tabela de decisão.
    - 1 Selecionar um efeito para estar com valor 1.
    - 2 Rastrear o grafo para trás, encontrando todas as combinações de causas (sujeitas a restrições) que fazem com que esse efeito seja 1.
    - 3 Criar uma coluna na tabela de decisão para cada combinação de causa.
    - 4 Determinar, para cada combinação, os estados de todos os outros efeitos, anotando na tabela.

Considerações

- Ao executar o passo 2, fazer as seguintes considerações:

  - 1 - Quando o nó for do tipo OR e a saída deva ser 1, nunca atribuir mais de uma entrada com valor 1 simultaneamente. O objetivo disso é evitar que alguns erros não sejam detectados pelo fato de uma causa mascarar outra.

  - 2 - 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. No entanto, se a situação é tal que uma entrada 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 a 1.

  - 3 - Quando o nó for do tipo AND e a saída deva ser 0, somente uma condição em que todas as entradas sejam 0 precisa ser enumerada. (Se esse AND estiver no meio do grafo, de forma que suas entradas estejam vindo de outros nós intermediários, pode ocorrer um número excessivamente grande de situações nas quais todas as entradas sejam 0.)

Exemplo

- Tabela de decisão

	Nós	Casos de teste
	1	0 1 0 0 1
Causas	2	0 0 1 1 0
	3	- 1 1 0 0
	70	0 1 1 0 0
Efeitos	71	1 0 0 0 0
	72	- 0 0 1 1

Conclusões

- A técnica funcional pode ser utilizada em todas as fases de teste.
- Independe do paradigma de programação utilizado.
- Eficaz em detectar determinados tipos de erros.
  - Por exemplo: Funcionalidade ausente.
- Dependente de uma boa especificação de requisitos.
  - Especificações descritivas e não formais.
  - Requisitos imprecisos e informais.
- Dificuldade em quantificar a atividade de teste.
- Não é possível garantir que partes essenciais ou críticas do software sejam executadas.
- Dificuldade de automatização: em geral, a aplicação é manual.

Referências

- Notas de aula: Professora Simone do Rocio Senger de Souza.
- Notas de aula: Professor Marcio Eduardo Delamaro.
- E. F. Barbosa.; J. C. Maldonado; A. M. R. Vicenzi; M. E. Delamaro; S. R. S. Souza; & M. Jino. Introdução ao teste de software. In: XIV Simpósio Brasileiro de Engenharia de Software. João Pessoa, PB.
- R. S. Pressman. Software Engineering - A Practitioner's Approach. McGraw-Hill, 4 edition, 1997.
- G. J. Myers. The Art of Software Testing. Wiley, New Jersey, 2004.