Testes e Qualidade em Jogos

5º período

Professora: Michelle Hanne

Sumário

PUC Minas

- Técnicas de Testes de Software

Exemplo:
https://fases.ifrn.edu.br/process/fases_plug-in/guidances/templates/resources/casos-de-teste.pdf
Apresentação:
https://s3-sa-east-1.amazonaws.com/thedevconf/presentations/TDC2019FLP/testes/JOE-9589_2019-04-29T021706_Game_Test_Strategy.pdf

PUC Minas

Caso de Teste

Um item de teste que contém:

- um conjunto de entradas de teste;
- o procedimento de teste a ser executado;
- as saídas esperadas.

PUC Minas

Criando Casos de Testes

Testadores precisam encontrar um número finito de casos de teste;

- selecionados de um domínio frequentemente muito grande.

O objetivo é:

- maximizar a cobertura:
  - os testes devem explorar o maior número possível de possibilidades diferentes;
- minimizar o tempo:
  - deve-se buscar executar o menor número possível de testes.

Existem algumas técnicas diferentes que ajudam na seleção dos casos de teste.

PUC Minas

Técnicas de Teste de Software

Testes caixa-branca

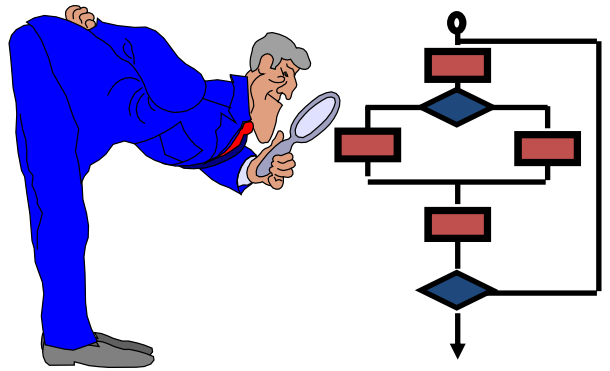
Testes caixa-preta

PUC Minas

Testes Caixa-Branca (*White-Box*)

Os testes são desenvolvidos considerando-se a estrutura interna do código;

- de maneira a exercitá-lo suficientemente.



PUC Minas

Testes Caixa-Preta (*Black-Box*)

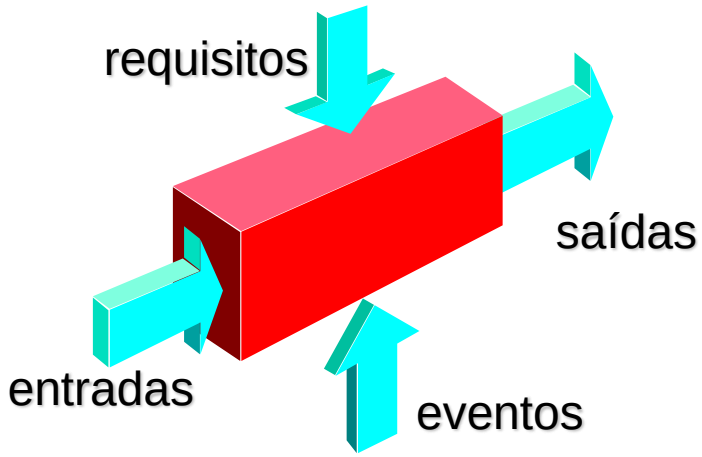
Os testes são desenvolvidos considerando-se somente:

- as entradas aceitas pelo componente;
- e as saídas esperadas.

Não verificam como ocorre o processamento;

- apenas validam os resultados produzidos.

Têm por objetivo determinar se os requisitos foram total ou parcialmente satisfeitos pelo produto.



PUC Minas

Testes Caixa-Branca

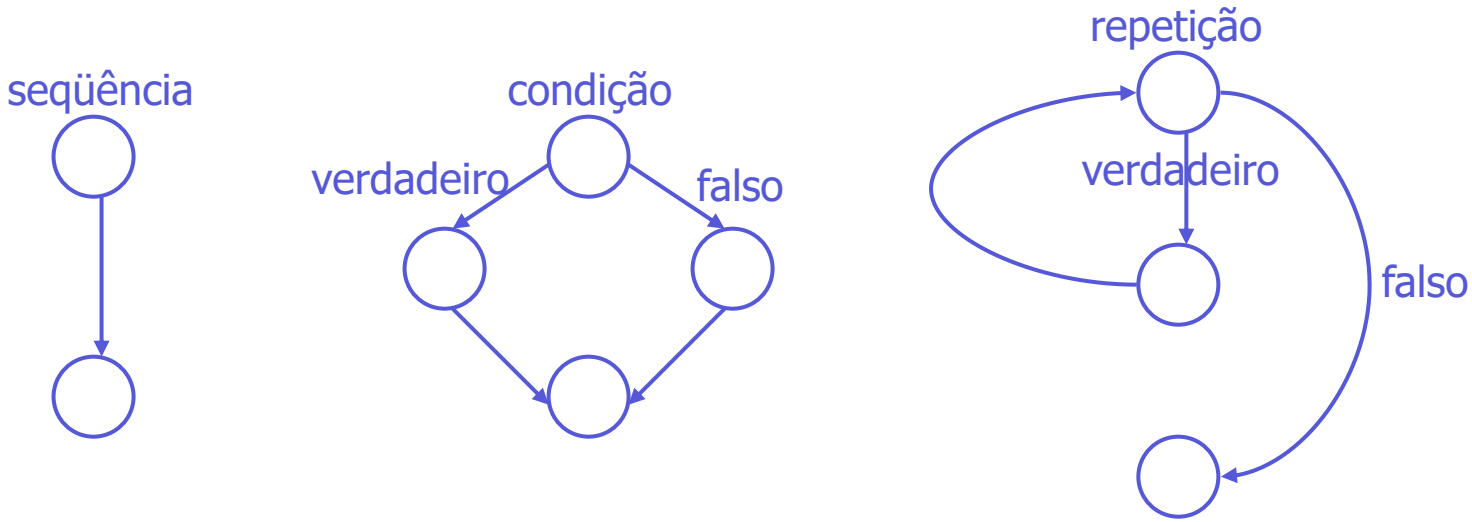
Testador tem que possuir conhecimento da estrutura interna do *software* a ser testado.

Útil para testar componentes pequenos.

Nível de detalhes é grande.

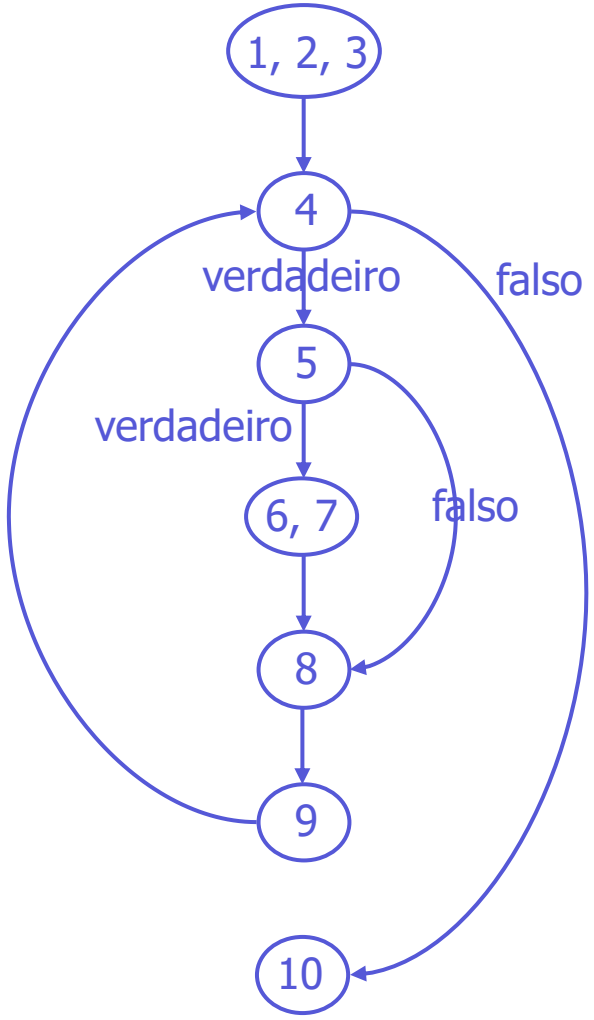
Grafo de Programa/Fluxo de Controle

- Auxilia na criação de casos de teste caixa-branca;
  - nodos representam comandos;
  - arestas representam fluxo de controle.



Exemplo – Grafo de Programa

```
1 somaVetor (a, numElementos, soma)
2     soma = 0;
3     i = 1;
4     while (i <= numElementos)
5         if (a[i] > 0)
6                 soma = soma + a[i];
7         endif
8         i = i + 1;
9     end while
10 end somaVetor
```



PUC Minas

Caminhos no Grafo de Programa

- Um caminho é uma seqüência de nodos;
  - começando no nodo de entrada no grafo;
  - e chegando ao nodo de saída.
- Exercício:
  - Encontre caminhos no grafo de programa do exemplo anterior.
- Exercício:
  - A sequência 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 é um caminho factível?

Critérios de Cobertura de Testes

- Critérios de cobertura de testes:
  - comandos (*statements*);
  - desvios ou decisões (*branches*);
  - condições;
  - múltiplas condições;
  - caminhos;
  - fluxo de dados.
- **Alguns critérios de cobertura possuem uma habilidade para detectar defeitos maior do que outros;**
  - isto implica em análise de cobertura.

Cobertura de Comandos (Statements)

Seu objetivo é executar todas as linhas de código do componente a ser testado;

- pelo menos uma vez.

É a abordagem mais simples.

Em termos do grafo de programa;

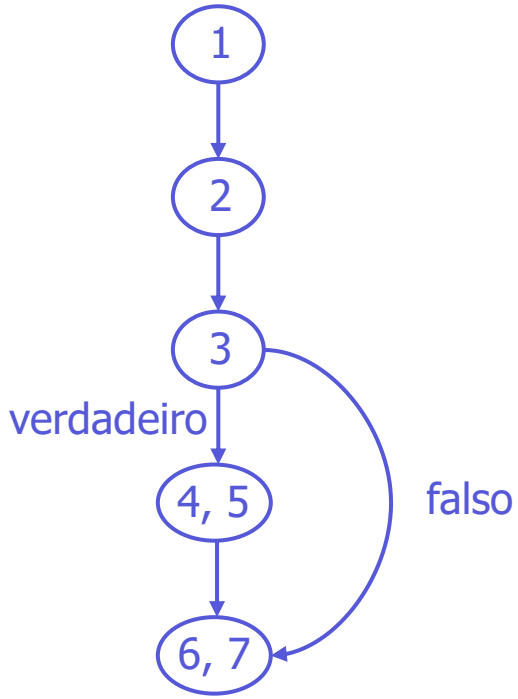
- consiste em garantir que todos os nodos serão exercitados;
  - pelo menos uma vez pelo conjunto de casos de teste.

Cobertura de Comandos (Statements)

- Não é capaz de detectar diversos tipos de defeito:
  - *if (a > 0)*;
    - onde deveria ser *if (a >= 0)*;
  - basta executar cada *loop* uma única vez;
  - condições compostas não são testadas;
  - *ifs* sem *elses* são particularmente vulneráveis.

Cobertura de Comandos (Statements)

```
1 public int thisHasAnError (int x, int y ) {
2         String c;
3         if (x == y) {
4                 c = new String();
5                 }
6         return c.length();
7 }
```

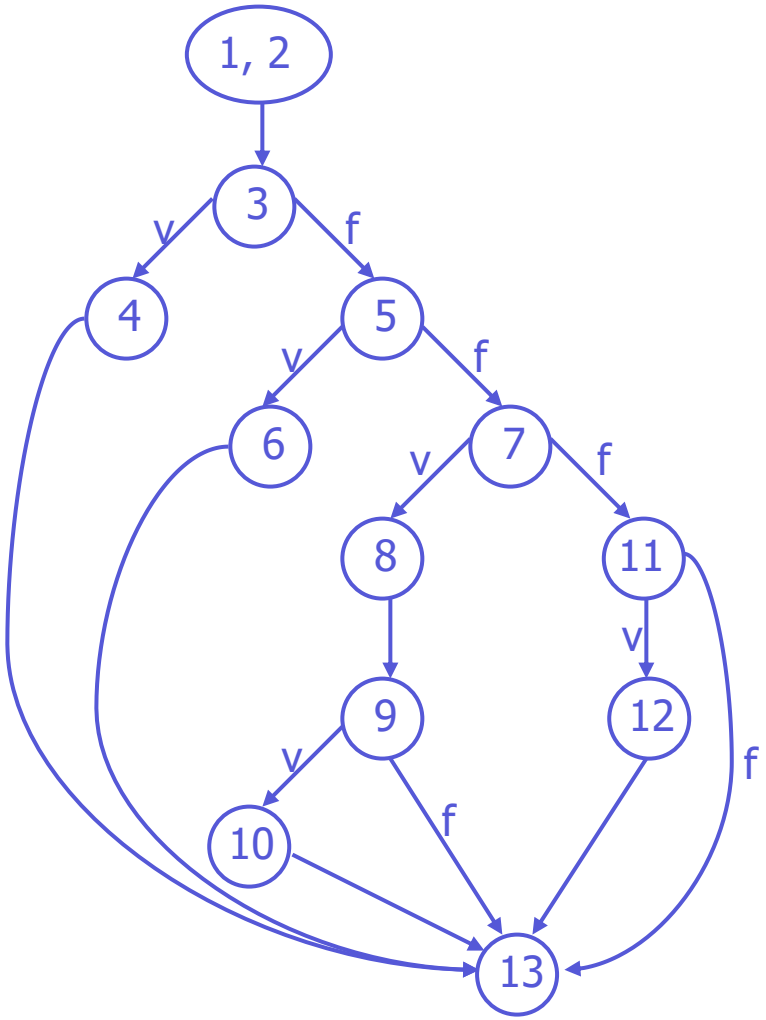


Exercício – Cobertura de Comandos

Determine um conjunto de casos de teste que garanta cobertura de comandos para o programa ao lado.
Use um grafo de programa.

```
1  public int determinaTipo (int a,b,c ) {
2        int tipo = ESCALENO;
3        if ((a <= 0) || (b <= 0) || (c <= 0)) {
4              tipo = INEXISTENTE;
         } else
5        if (! ( ((a + b) > c) && ((a + c) > b) && ((b + c) > a) )) {
6              tipo = INEXISTENTE;
         } else
7        if (a == b){
8              tipo = ISOSCELES;
9              if (b == c) {
10                          tipo = EQUILATERO;
               }
         } else
11       if ((b == c) || (a == c)) {
12             tipo = ISOSCELES;
         }
13       return tipo;
   }
```

Exercício – Cobertura de Comandos



- Todos os nodos do grafo devem ser exercitados.
- Casos de teste:
  - caminho: 1,2,3,4,13
    - entrada: 0,0,0
    - saída: INEXISTENTE
  - caminho: 1,2,3,5,6,13
    - entrada: 1,2,3
    - saída: INEXISTENTE
  - caminho: 1,2,3,5,7,8,9,10,13
    - entrada: 1,1,1
    - saída: EQUILÁTERO
  - caminho: 1,2,3,5,7,11,12,13
    - entrada: 1,2,2
    - saída: ISÓSCELES

PUC Minas

Cobertura de Desvios ou Decisões (*Branches*)

Cada condição (simples ou composta) deve ser avaliada;

- pelo menos uma vez como verdadeira;
- e outra como falsa.

Para *n* condições;

- exige no máximo *n+1* casos de teste.

Em termos do grafo de programa;

- consiste em garantir que todas as arestas serão exercitadas;
  - pelo menos uma vez pelo conjunto de casos de teste.

PUC Minas

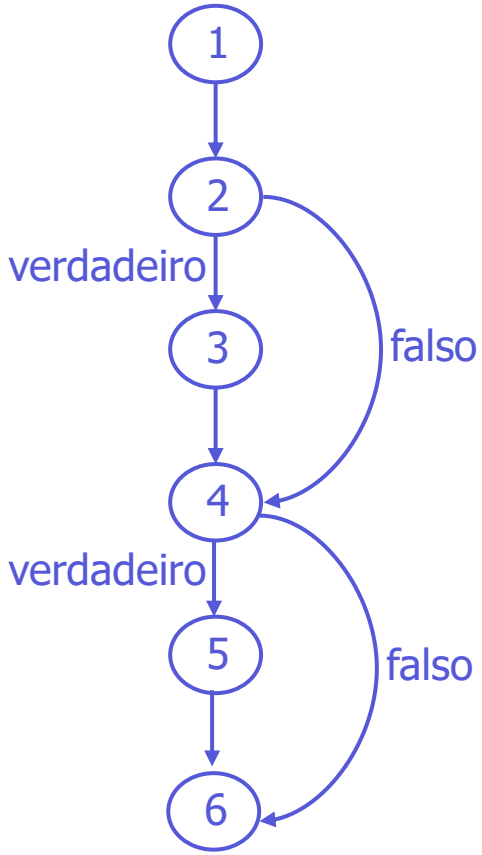
Cobertura de Desvios ou Decisões (*Branches*)

Resolve alguns problemas da cobertura de comandos;

- mas não todos...
  - condições compostas podem não ser testadas;
  - comandos de repetição só necessitam ser executados uma única vez;
    - apesar da condição de repetição ter de ser testada duas vezes;
    - ou seja, apenas o caminho de comprimento um pode ter sido testado.

Cobertura de Desvios ou Decisões (*Branches*)

```
1 public int testThis (int i) {
2  if (a == b)
3       ++i;
4  if (x == y)
5       --i;
6  return i;
}
```



4 caminhos possíveis;
apenas 2 são testados.

PUC Minas

Cobertura de Desvios ou Decisões (*Branches*)

Determine um conjunto de casos de teste que garanta cobertura de desvios para o programa ao lado. Use um grafo de programa.

```
1  public int determinaTipo (int a,b,c ) {
2        int tipo = ESCALENO;
3        if ((a <= 0) || (b <= 0) || (c <= 0)) {
4              tipo = INEXISTENTE;
         } else
5        if (! ( ((a + b) > c) && ((a + c) > b) && ((b + c) > a) )) {
6              tipo = INEXISTENTE;
         } else
7        if (a == b){
8              tipo = ISOSCELES;
9              if (b == c) {
10                          tipo = EQUILATERO;
               }
         } else
11       if ((b == c) || (a == c)) {
12             tipo = ISOSCELES;
         }
13       return tipo;
   }
```

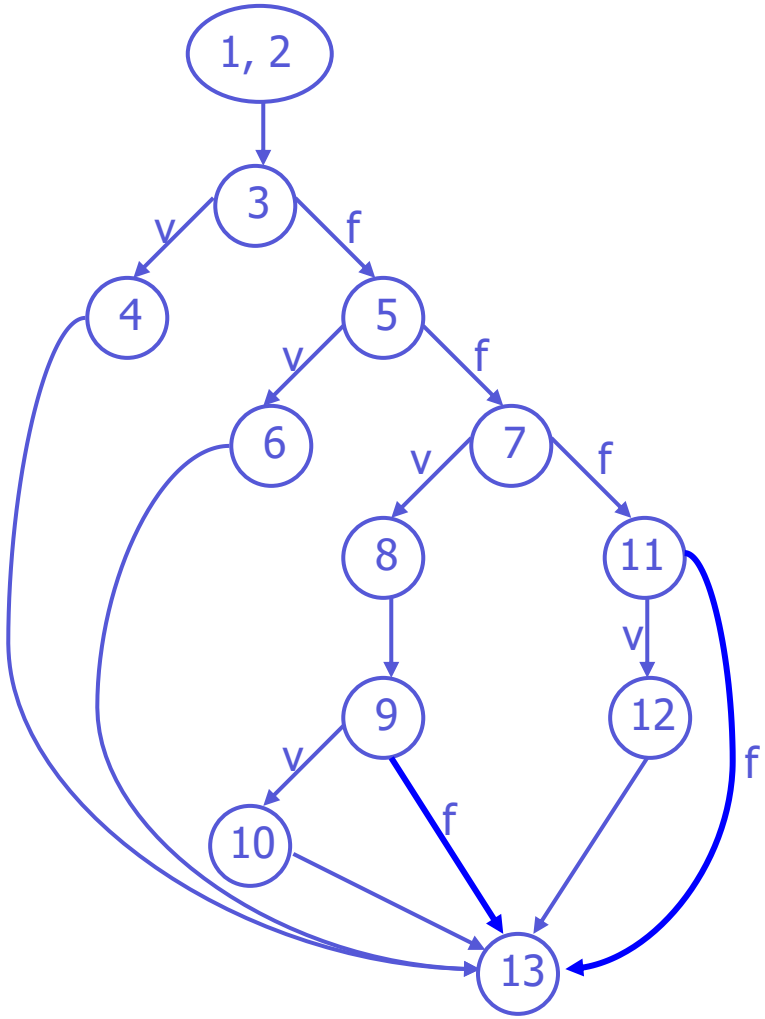
Solução do Exercício – Cobertura de Desvios ou Decisões

PUC Minas

- Todas as arestas do grafo devem ser exercitadas.
- Casos de teste:
  - caminho: 1,2,3,4,13
    - entrada: 0,0,0 saída: INEXISTENTE
  - caminho: 1,2,3,5,6,13
    - entrada: 1,2,3 saída: INEXISTENTE
  - caminho: 1,2,3,5,7,8,9,10,13
    - entrada: 1,1,1 saída: EQUILÁTERO
  - caminho: 1,2,3,5,7,11,12,13
    - entrada: 1,2,2 saída: ISÓSCELES
  - caminho: 1,2,3,5,7,8,9,13
    - entrada: 2,2,1 saída: ISÓSCELES
  - caminho: 1,2,3,5,7,11,13
    - entrada: 2,3,4 saída: ESCALENO

Comparação – Cobertura de Comandos x Cobertura de Desvios ou Decisões

- As arestas em destaque;
  - não foram exercitadas na cobertura de comandos;
  - mas foram exercitadas na cobertura de desvios.
- Isto sugere que a cobertura de desvios;
  - é melhor que a cobertura de comandos.



Cobertura de Condições

PUC Minas

Cada parte de uma condição deve ser avaliada;

- pelo menos uma vez como verdadeira;
- e outra como falsa.

Não requer que todos os desvios sejam cobertos.

- O desvio pode ser sempre falso, por exemplo;
  - mas cada parte da condição deve ser testada como verdadeira e como falsa.

Pode-se combiná-la com a cobertura de desvios.

Cobertura de Múltiplas Condições

Todas as combinações de verdadeiro e falso, de cada condição simples, devem ser testadas.

Uma condição composta deverá ter 2^n combinações;

- onde n é o número de condições simples.

PUC Minas

Cobertura de Múltiplas Condições

Cobre todos os comandos, desvios e condições.

Não cobre todos os caminhos possíveis.

Cobertura de Caminhos

Deve-se cobrir todos os caminhos possíveis;

- da entrada até a saída do componente que está sendo testado.

O número de caminhos pode chegar a 2^n;

- para n condições.

No caso de comandos de repetição;

- o número de caminhos pode ser infinito.

Em geral, o custo deste tipo de cobertura é proibitivo.

PUC Minas

Cobertura de Fluxo de Dados

- Uma análise do padrão de fluxo de dados de variáveis específicas;
  - pode ser útil na detecção de defeitos.
  - **Exemplo:**
    - uso de variáveis que não foram inicializadas;
      - indica um defeito no código.

Cobertura de Fluxo de Dados

Todas as definições de variáveis;

- devem ser exercitadas pelo menos uma vez.
- Uma variável é definida em um comando que:
  - atribui a ela um valor;
  - ou altera seu valor.

Todos os usos de variáveis;

- devem ser exercitados pelo menos uma vez.
- Uma variável é usada em um comando que:
  - utiliza seu valor;
  - mas não o altera.

Orientação a Objetos

Ao testar programas orientados a objetos;

- é preciso prestar especial atenção nas chamadas polimórficas.

Cada chamada polimórfica deve ser considerada como um desvio condicional;

- com tantos pontos de entrada quantos forem os possíveis métodos executados.

PUC Minas

Testes Caixa-Preta

Baseados na descrição do comportamento do *software*.

- *Software* a ser testado é visto como uma “caixa-preta”;
  - não se sabe como funciona sua estrutura interna;
  - conhece-se apenas o que ele faz.
- A “caixa preta” pode variar de uma simples função, objeto ou módulo, até um sistema completo.

PUC Minas

Testes Caixa-Preta

A descrição do comportamento do *software* pode vir de:

- um conjunto de pré e pós-condições;
- uma especificação de requisitos;
- um conjunto de entradas válidas e saídas esperadas.

O testador:

- submete as entradas ao *software*;
- executa o *software*;
- determina se as saídas produzidas são iguais às especificadas.

PUC Minas

Técnicas de Testes Caixa-Preta

Testes aleatórios;

Partição em classes de equivalência;

Análise dos valores de borda.

PUC Minas

Testes Aleatórios

As entradas dos casos de teste são escolhidas aleatoriamente do conjunto-domínio.

- Exemplo:
  - conjunto-domínio:
    - inteiros de 1 a 100;
  - possíveis casos de teste:
    - 24, 55, 3.

PUC Minas

Testes Aleatórios

Problemas com esta técnica:

- Pequena chance de produzir um conjunto de casos de teste eficaz.
- No caso do exemplo anterior:
  - os três casos de teste escolhidos são suficientes para indicar se o *software* está em conformidade com sua especificação?
  - devemos escolher mais ou menos valores?
  - existem outros valores que revelariam defeitos?
    - como inteiros positivos perto do início ou do fim do domínio?
  - devemos escolher valores fora do domínio?
    - como números reais, inteiros negativos ou inteiros maiores do que 100?

PUC Minas

Partição em Classes de Equivalência

Naturalmente, é impossível testar todas as combinações possíveis de entradas.

- É preciso achar um subconjunto significativo.

A técnica fundamental envolve achar partições;

- para cada tipo de dado.

Partições

Uma partição divide o domínio do tipo;

- em subconjuntos de valores do tipo;
  - que afetam as condições presentes no código.

Por exemplo:

- if (x > 15) { ... }
- O domínio da variável x pode ser particionado em dois subconjuntos:
  - o dos valores de x que são maiores que 15;
  - e o dos que não são.

Partição em Classes de Equivalência – Procedimento

PUC Minas

Particionar o domínio de entrada;

- em classes de equivalência.
- Resultado:
  - número finito de partições ou classes de equivalência.

Selecionar um dos membros da classe de equivalência;

- como o representante desta classe.
- Premissa:
  - assume-se que todos os membros de uma classe de equivalência são processados de maneira equivalente pelo *software*.

PUC Minas

Vantagens das Classes de Equivalência

Elimina a necessidade de testes exaustivos;

- que não são viáveis, de qualquer forma.

Guia o testador na seleção de um subconjunto de entradas para os testes;

- com alta probabilidade de detectar um defeito.

Permite cobrir um domínio maior da entrada;

- selecionando um subconjunto menor de valores, dadas as classes de equivalência.

Como Identificar as Classes de Equivalência?

Identificar nas especificações, condições que dividem o domínio de entrada:

- intervalo de valores válidos;
- conjunto de valores válidos;
- valor ou característica mandatória;
- elemento distinto.

PUC Minas

Intervalo de Valores Válidos

abaixo do intervalo válido	intervalo válido	acima do intervalo válido

- **Exemplo:**
  - o comprimento de um objeto pertence ao intervalo de 1 a 499 milímetros.
  - Criar uma classe de equivalência:
    - para todos os valores dentro do intervalo válido;
    - para todos os valores menores que 1;
    - para todos os valores maiores que 499.

PUC Minas

Conjunto de Valores Válidos

- Exemplo:
  - a especificação de um módulo de pintura de objetos diz que as cores vermelho, verde e amarelo são permitidas como valores de entrada.
  - Criar uma classe de equivalência:
    - para todos os valores válidos:
      - vermelho, verde e amarelo;
    - para todos os valores inválidos:
      - todas as outras entradas.

vermelho
verde
amarelo

lilás
azul
rosa

Valor ou Característica Mandatória

- Exemplo:
  - o primeiro caracter do identificador de um item de pedido tem de ser uma letra.
  - Criar uma classe de equivalência:
    - válida:
      - onde os identificadores começam com uma letra;
    - inválida:
      - onde o primeiro caracter dos identificadores não é uma letra.

ff23
g567
w34

9fggg
4t*
1F345
*987

PUC Minas

Elemento Distinto

- Elemento de uma classe de equivalência que não é tratado pelo *software* da mesma forma que outros elementos da classe.
  - Implica em nova divisão da classe de equivalência;
    - em partições menores.

PUC Minas

Exercício – Classes de Equivalência

- Identifique as classes de equivalência para os valores de entrada de uma função que calcula a raiz quadrada de um número.
  - Condições de entrada:
    - x é o valor de entrada;
    - x é real;
    - $x \geq 0$.
  - Condições de saída:
    - y é o valor de saída;
    - y é real;
    - $y \geq 0$;
    - $y*y = x$.

PUC Minas

Solução do Exercício – Classes de Equivalência

- Classe de equivalência 1:
  - x real;
    - válida.
- Classe de equivalência 2:
  - x não é real;
    - inválida.
- Classe de equivalência 3:
  - $x \geq 0$;
    - válida.
- Classe de equivalência 4:
  - $x < 0$;
    - inválida.

PUC Minas

Análise dos Valores de Borda

Muitos defeitos acontecem justamente nas fronteiras das classes de equivalência:

- exatamente na fronteira;
- logo abaixo da fronteira;
- logo acima da fronteira.

Casos de teste que consideram estas bordas (fronteiras);

- em geral revelam defeitos.

Casos de teste devem incluir valores das fronteiras inferior e superior de uma classe de equivalência.

PUC Minas

Análise dos Valores de Borda

- Especificação dos valores de entrada:
  - intervalo de valores:
    - criar casos de teste com valores:
      - válidos nas fronteiras do intervalo;
      - inválidos:
        - logo acima da fronteira do intervalo;
        - e logo abaixo da fronteira do intervalo.
    - Exemplo:
      - entrada é um valor pertencente ao intervalo de -1.0 a +1.0.
      - Casos de teste:
        - entrada: -1.0; saída: válido;
        - entrada: +1.0; saída: válido;
        - entrada: +1.1; saída: inválido;
        - entrada: -1.1; saída: inválido.

PUC Minas

Análise dos Valores de Borda

- Especificação dos valores de entrada:
  - conjunto ordenado (listas, tabelas, etc):
    - criar casos de teste focando:
      - no primeiro elemento do conjunto;
      - no último elemento do conjunto;
      - no elemento anterior ao primeiro do conjunto (inválido);
      - no elemento posterior ao último do conjunto (inválido).
    - Exemplo:
      - entrada é um elemento da lista ordenada: {B, K, M, R, S}
      - Casos de teste:
        - entrada: B; saída: válido;
        - entrada: S; saída: válido;
        - entrada: A; saída: inválido;
        - entrada: T; saída: inválido.

PUC Minas

Listas de Conferência Pessoais

Manter uma *checklist* com seus *bugs* pessoais.

Testar seus erros mais freqüentes.

- O uso de *checklists* personalizadas;
  - reduz o número de *bugs* em até 75%.

Referências

PAULA-FILHO, Wilson de Pádua. *Engenharia de Software: Fundamentos, Métodos e Padrões*. 2ª edição, Rio de Janeiro: LTC – Livros Técnicos e Científicos, 2003. Capítulo 8.

PRESSMAN, Roger S.. *Engenharia de Software*. 5ª edição, Rio de Janeiro: McGraw Hill, 2017. Capítulo 17.

MYERS, Glenford J.. *The Art of Software Testing*. Willey Interscience, 1979.

ROSE, Laura. *Myths and Realities of Iterative Testing*. http://www-128.ibm.com/developerworks/rational, Abril, 2006.

BOEHM, B. W.. *Software Engineering*. IEEE Transactions on Computers, C-25(12):1226–1241, 1976.

BAZIUK, W.. *Path to Improve Product Quality, Reliability, and Customer Satisfaction*. 6th International Symposium on Software Reliability Engineering: Toulouse, France, Outubro, 1995.