1 - Grafo de Fluxo de Controle

Os testes de Caixa Branca são testes estruturais, ou seja, os testes devem checar as instruções do programa a procura de falhas. O conhecimento do código é usado para identificar os casos de teste. Nesse tipo de teste todos os caminhos possíveis no código precisam ser testados, para tanto um Grafo de Fluxo de Controle ajuda na visualização da estrutura do programa e assim identificar os casos de teste.

1.1 Grafo de Fluxo de Controle

É uma representação que usa notação de grafo para descrever todos os caminhos que podem ser executados por um programa, ou seja, mostra o fluxo de controle no código. O grafo é formado por nós e arestas (ramos) com os seguintes papeis:

- Nó: representam instruções;
- Aresta: representam o fluxo de execução, ou seja, transferência de controle.

A Figura 1 mostra a representação das instruções de um programa na forma de grafos. Observações:

- Existe um arco de um nó ***1*** para um nó ***2***, se e somente se, ***2*** pode ser executado imediatamente após ***1***;
- Não se usa representar declarações de variáveis e outras instruções que não transformam valores ou alteram o fluxo de execução do código.

A Figura 2 mostra um exemplo de grafo de um programa, os números nos nós representam os números das linhas.

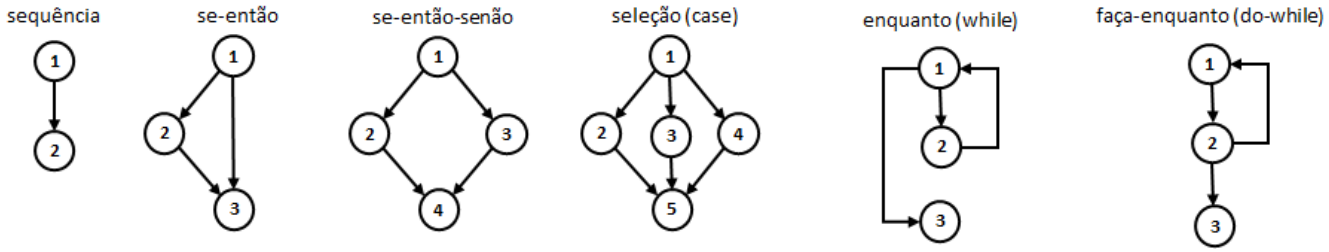

Figura 1 – Representação das instruções no grafo.

```
1.   int x = Integer.parseInt("5");
2.   int y = Integer.parseInt("18");
3.   while( x < y ){
4.      if( x%2 == 0 ){
5.         x += 1;
       }
       else{
6.         x += 3;
7.      }
     }
8.   System.out.println(x);
```

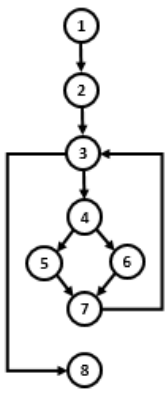

Figura 2 – Representação do grafo de um programa.

1.2 Análise de Cobertura

Para fazer o teste de Caixa Branca é necessário percorrer o código a procura por falhas nas instruções. Tome como exemplo um método, falhas em instruções internas do método podem não ser detectadas por testes que verificam apenas entradas e saídas do método. Acontece que uma verificação minuciosa no código precisa checar (cobrir) todas as instruções que serão executadas em cada caso de teste, pois cada caso de teste corresponde a um caminho no grafo.

O fluxo de execução de um programa pode ser alterado pelas estruturas de decisão, seleção e repetição, então é necessário ter testes que percorrem (cobrem) todos os caminhos possíveis no código. Além disso, estruturas de decisão, seleção e repetição são formadas por expressões booleanas, como exemplo, if(a < b && b < c), então um teste também precisa testar cada uma das condições que fazem parte da expressão, ou seja, ter pelo menos dois testes para a < b que resultam em true e false, e ter outros dois testes para de b < c que também resultem em true e false.

Os casos de teste devem cobrir todas as instruções do código, para isso tem-se as seguintes coberturas:

- Cobertura de instruções;
- Cobertura de decisões;
- Cobertura de condições;
- Cobertura de repetições.

Cobertura de Instruções

Neste critério os casos de teste devem ser criados de forma que, ao serem executados, todos os caminhos possíveis no programa sejam percorridos (cobertos) no mínimo uma vez. Considere como exemplo o método da Figura 3, para fazer o teste será necessário:

1) Criar o grafo do código a ser testado, assim como na Figura 3;
2) Determinar os caminhos factíveis no grafo. No exemplo da Figura 3 podemos identificar os seguintes caminhos factíveis:
    - Para x = 0, y = 0 tem-se o caminho: 1, 3, 4, 5, 7, 9
    - Para x = 2, y = 3 tem-se o caminho: 1, 3, 4, 5, 6, 5, 7, 9
    - Para x = -2, y = 3 tem-se o caminho: 1, 3, 4, 5, 6, 5, 7, 9
    - Para x = 2, y = -3 tem-se o caminho: 1, 2, 4, 5, 6, 5, 7, 8, 9
    - Para x = -2, y = -3 tem-se o caminho: 1, 2, 4, 5, 6, 5, 7, 8, 9
3) Selecionar o menor conjunto de caminhos factíveis que testam todas as instruções:
    - Para este caso bastaria os testes com (x = 0, y = 0), (x = 2, y = 3) e (x = 2, y = -3).

```
   public void potencia(int x, int y){
     long p = 0;
1.   if( y < 0 ){
2.      p = 0 - y;
     }
     else{
3.      p = y;
     }
4.   float res = 1;
5.   while( p != 0 ){
6.      res = res * x;
6.      p = p - 1;
     }
7.   if( y < 0 ){
8.      res = 1/res;
     }
9.   System.out.println( res );
   }
```

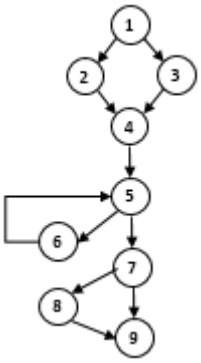

Figura 3 – Representação do grafo do método potencia.

Cobertura de Decisões

Neste critério os testes devem cobrir cada saída possível de um nó que possui condição (estrutura de decisão), ou seja, cada ramo da decisão deve ser percorrido pelo menos uma vez.

Para os seguintes valores de x e y todas as arestas são percorridas pelo menos uma vez:

- Para x = 0, y = 0 tem-se as arestas: (1,3), (3,4), (4,5), (5,7), (7,9)
- Para x = 2, y = -3 tem-se as arestas: (1,2), (2,4), (4,5), (5,6), (5,7), (7,8), (8,9)

Os testes de decisão são usados para identificar se todos os desvios possíveis no código estão sendo percorridos. Esse tipo de teste é capaz de apontar erros de lógica.

Cobertura de Condições

Neste critério os testes devem cobrir todas as combinações de condições nas estruturas de decisão e repetição. Tome como exemplo a decisão n2 > n1 && n1 > 2 no método da Figura 4, os valores de n1 e n2 devem cobrir as 4 combinações possíveis assim como na tabela verdade ao lado. A seguir tem-se um conjunto de caminhos factíveis para o teste:

n1	n2	n2 > n1	n1 > 2
5	8	T	T
2	4	T	F
4	2	F	V
1	1	F	F

- Para n1 = 5, n2 = 8 tem-se o caminho: 1, 3, 4, 5, 6, 7, 9
- Para n1 = 2, n2 = 4 tem-se o caminho: 1, 3, 5, 6, 8, 9
- Para n1 = 4, n2 = 2 tem-se o caminho: 1, 3, 5, 6, 8, 9
- Para n1 = 1, n2 = 1 tem-se o caminho: 1, 3, 5, 6, 8, 9
- Para n1 = 0, n2 = 8 tem-se o caminho: 1, 2, 9
- Para n1 = 8, n2 = 0 tem-se o caminho: 1, 2, 9

Os testes de condições são usados para identificar os desvios possíveis no código e testar cada condição que faz os desvios serem percorridos. Esse tipo de teste é capaz de apontar:

- Erros de operador booleano;

- Erros de variáveis booleana;
- Erros de parênteses booleano;
- Erros de operadores relacionais;
- Erros de expressão aritmética.

```
   public void aprovado(float n1, float n2){
      String resp = null;
1.      if( n1 == 0 || n2 == 0 ){
2.         resp = "reprovado";
        }
        else{
3.         if( n2 > n1 && n1 > 2 ){
4.            n1 = n2;
           }
5.         float media = n1 * 0.4f + n2 * 0.6f;
6.         if( media >= 6 ){
7.            resp = "aprovado";
           }
           else{
8.            resp = "reprovado";
           }
        }
9.      System.out.println(resp);
   }
```

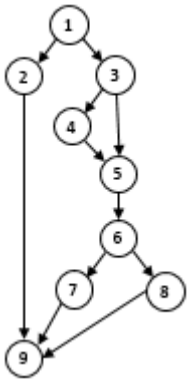

Figura 4 – Representação do grafo do método aprovado.

Cobertura de Repetições

Neste critério devem ser criados casos de teste que fazem com que as estruturas de repetição sejam executadas *n* vezes, onde *n* são valores no limite. Tenha como exemplo um método de busca de um valor no array e considere que este array tenha 100 elementos. Então deve-se escolher valores a serem buscados de tal forma que o laço seja executado somente 1 vez (que é o melhor caso) e o maior número possível de vez (que é o pior caso).

Complexidade Ciclomática

O número de caminhos independentes no código é igual à complexidade ciclomática. O cálculo da Complexidade ciclomática é dado por:

CC = Número de ramos (arestas) - Número de nós + 2

No exemplo da Figura 4 tem que a Complexidade Ciclomática é

Cc = 11 – 9 + 2 = 4

Desta forma, tem-se que 4 é o número mínimo de casos de teste para testar adequadamente todos os caminhos independentes do método da Figura 4.

2 - Teste de Caixa Preta

Nesta metodologia os casos de teste são gerados sem o conhecimento da estrutura interna do programa. É necessário apenas o conhecimento das entradas e saídas possíveis para o programa. São também denominados testes funcionais, ou seja, enquanto os testes de Caixa Branca são considerados testes estruturais, por testarem as estruturas internas do

programa, os testes de Caixa Preta são testes que se preocupam apenas com as funcionalidades do programa, sem se preocupar com as estruturas internas.

Um preocupação do teste Caixa Preta é verificar se todas as funcionalidades previstas foram implementadas e se respondem como esperado. Este tipo de teste é incapaz de verificar funcionalidades ou comportamentos extras não especificados.

A única maneira de mostrar que um programa está correto é o teste exaustivo. Contudo, teste exaustivo é impraticável, então usa-se algumas técnicas para obter os casos de teste:

- Criar conjuntos de valores de entrada e selecionar apenas alguns valores nestes conjuntos. Por exemplo, para o teste com números inteiros bastaria escolher um valor muito pequeno, um muito grande, um positivo, um negativo, zero e um;
- Testes com os valores de fronteira (limite);
- Testes usando os requisitos do sistema.

O teste de Caixa Branca é executado pelo próprio programador já o teste de Caixa Preta é executado por um testador de software.

3 - Teste de Caixa Cinza

O teste de Caixa Cinza é a combinação do teste de Caixa Branca e de Caixa Preta. Nesta metodologia o testador faz o teste apenas de entrada e saída em alguns componentes, mas em outros ele pode optar por também verificar detalhes internos do componente, como exemplo, para averiguar a causa da demora no processamento de um componente, neste caso, ele teria de checar laços de repetições, instruções que fazem cálculos, acesso a bases de dados etc.