"Há apenas uma regra no projeto de casos de teste: abranger todas as características; mas não faça muitos casos de teste."

Tsuneo Yamaura

mento interno de um produto, podem ser realizados testes para garantir que "tudo se encaixa", isto é, que as operações internas foram realizadas de acordo com as especificações e que todos os componentes internos foram adequadamente exercitados. A primeira abordagem de teste usa uma visão externa e é chamada de teste caixa-preta. A segunda abordagem requer uma visão interna e é chamada de teste caixa-branca.[2]

O *teste caixa-preta* faz referência a testes realizados na interface do software. Um teste caixa-preta examina alguns aspectos fundamentais de um sistema, com pouca preocupação em relação à estrutura lógica interna do software. O *teste caixa-branca* fundamenta-se em um exame rigoroso do detalhe procedimental. Os caminhos lógicos do software e as colaborações entre componentes são testados exercitando conjuntos específicos de condições e/ou ciclos.

PONTO-CHAVE

Testes caixa-branca só podem ser projetados depois que o projeto no nível de componente (ou código-fonte) existir. Os detalhes lógicos do programa devem estar disponíveis.

À primeira vista poderia parecer que um teste caixa-branca realizado de forma rigorosa resultaria em "programas 100% corretos". Tudo o que seria preciso fazer seria definir todos os caminhos lógicos, desenvolver casos de teste para exercitá-los e avaliar os resultados, ou seja, gerar casos de teste para exercitar a lógica do programa de forma exaustiva. Infelizmente, o teste exaustivo apresenta certos problemas logísticos. Mesmo para pequenos programas, o número de caminhos lógicos possíveis pode ser muito grande. No entanto, o teste caixa-branca não deve ser descartado como impraticável. Um número limitado de caminhos lógicos importantes pode ser selecionado e exercitado. Estruturas de dados importantes podem ser investigadas quando à validade.

INFORMAÇÕES

Teste Exaustivo

Considere um programa de 100 linhas em linguagem C. Após algumas declarações básicas de dados, o programa contém dois laços aninhados que executam de 1 a 20 vezes cada um, dependendo das condições especificadas na entrada. Dentro do ciclo, são necessárias 4 construções if-then-else. Há aproximadamente 10^{14} caminhos possíveis que podem ser executados nesse programa!

Para colocar esse número sob perspectiva, vamos supor que um processador de teste mágico ("mágico" porque não existe tal processador) tenha sido desenvolvido para teste exaustivo. O processador pode desenvolver um caso de teste, executá-lo e avaliar os resultados em um milissegundo. Trabalhando 24 horas por dia, 365 dias por ano, o processador gastaria 3.170 anos para testar o programa. Isso, sem dúvida, tumultuaria qualquer cronograma de desenvolvimento.

Portanto, é razoável afirmar que o teste exaustivo é impossível para grandes sistemas de software.

18.3 TESTE CAIXA-BRANCA

"Os defeitos ficam à espreita nas esquinas e se reúnem nas fronteiras."

Boris Beizer

O *teste caixa-branca, também* chamado de *teste da caixa-de-vidro,* é uma filosofia de projeto de casos de teste que usa a estrutura de controle descrita como parte do projeto no nível de componentes para derivar casos de teste. Usando métodos de teste caixa-branca, o engenheiro de software pode criar casos de teste que (1) garantam que todos os caminhos independentes de um módulo foram exercitados pelo menos uma vez, (2) exercitam todas as decisões lógicas nos seus estados verdadeiro e falso, (3) executam todos os ciclos em seus limites e dentro de suas fronteiras operacionais, e (4) exercitam estruturas de dados internas para assegurar a sua validade.

18.4 TESTE DO CAMINHO BÁSICO

O *teste de caminho básico* é uma técnica de teste caixa-branca proposta por Tom McCabe [McC76]. O teste de caminho básico permite ao projetista de casos de teste derivar uma medida da complexidade lógica de um projeto procedimental e usar essa medida como guia para

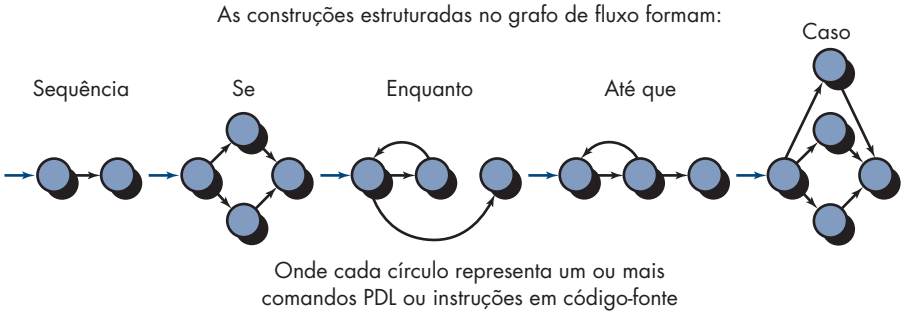
2 Os termos *teste funcional* e *teste estrutural* são às vezes usados em lugar de teste caixa-preta e teste caixa-branca, respectivamente.

definir um conjunto base de caminhos de execução. Casos de teste criados para exercitar o conjunto básico executam com certeza todas as instruções de um programa pelo menos uma vez durante o teste.

18.4.1 Notação de grafo de fluxo

Antes de considerarmos o teste de caminho básico, deve ser introduzida uma notação simples para a representação do fluxo de controle, chamada de *grafo de fluxo* (ou *grafo de programa*).[3] O grafo de fluxo representa o fluxo de controle lógico usando a notação ilustrada na Figura 18.1. Cada construção estruturada (Capítulo 10) tem um símbolo correspondente no grafo de fluxo.

FIGURA 18.1
Notação de grafo de fluxo

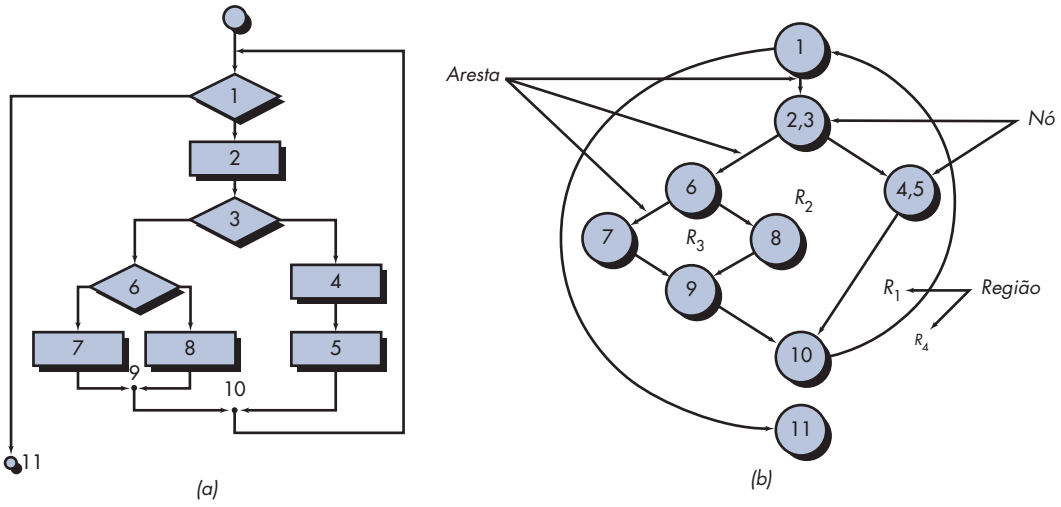


AVISO

Um grafo de fluxo somente deve ser desenhado quando a estrutura lógica de um componente for complexa. O grafo de fluxo permite seguir os caminhos de programa mais facilmente.

Para ilustrar o uso de um grafo de fluxo, considere a representação do projeto procedimental na Figura 18.2a. É usado um fluxograma para mostrar a estrutura de controle do programa. A Figura 18.2b mapeia o fluxograma em um grafo de fluxo correspondente (considerando que os losangos de decisão do fluxograma não contêm nenhuma condição composta). Na Figura 18.2b, cada círculo, chamado de *nó do grafo de fluxo,* representa um ou mais comandos procedurais. Uma sequência de retângulos de processamento e um losango de decisão podem ser mapeados em um único nó. As setas no grafo de fluxo, chamadas de *arestas* ou *ligações,* representam fluxo de controle e são análogas às setas do fluxograma. Uma aresta deve terminar em um nó, mesmo que esse nó não represente qualquer comando procedural (por exemplo, veja o símbolo do diagrama de fluxo para a construção se-então-senão - if-then-else). As áreas limitadas por arestas e nós são chamadas de *regiões.* Ao contarmos as regiões, incluímos a área fora do grafo como uma região.[4]

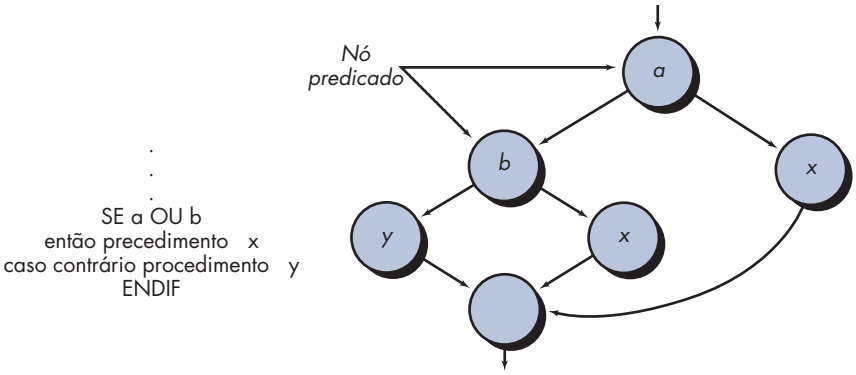
FIGURA 18.2
(a) Fluxograma e (b) grafo de fluxo



3 Na realidade, o método do caminho básico pode ser executado sem o uso de grafos de fluxo. No entanto, eles servem como uma notação útil para entender o fluxo de controle e ilustrar a abordagem.

4 Uma discussão mais detalhada sobre grafos e seus usos é apresentada na Seção 18.6.1.

FIGURA 18.3
Lógica composta



Quando condições compostas são encontradas em um projeto procedimental, a geração de um grafo de fluxo torna-se ligeiramente mais complicada. Uma condição composta ocorre quando um ou mais operadores booleanos (OR, AND, NAND, NOR lógicos) estão presentes em um comando condicional. De acordo com a Figura 18.3, o trecho de PDL (*program design language*) é traduzido no grafo de fluxo mostrado. Note que é criado um nó separado para cada uma das condições *a* e *b* no comando SE *a* OU *b*. Cada nó contendo uma condição é chamado de *nó predicado* (*predicate node*) e é caracterizado por duas ou mais arestas saindo dele.

18.4.2 Caminhos de programa independentes

Um *caminho independente* é qualquer caminho através do programa que introduz pelo menos um novo conjunto de comandos de processamento ou uma nova condição. Quando definido em termos de um grafo de fluxo, um caminho independente deve incluir pelo menos uma aresta que não tenha sido atravessada antes de o caminho ser definido. Por exemplo, um conjunto de caminhos independentes para o grafo de fluxo ilustrado na Figura 18.2b é

Caminho 1: 1-11

Caminho 2: 1-2-3-4-5-10-1-11

Caminho 3: 1-2-3-6-8-9-10-1-11

Caminho 4: 1-2-3-6-7-9-10-1-11

Note que cada novo caminho introduz uma nova aresta. O caminho

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

não é considerado um caminho independente porque ele é simplesmente uma combinação dos caminhos já especificados e não atravessa nenhuma nova aresta.

A complexidade ciclomática é uma métrica útil para previsão dos módulos que têm a tendência de apresentar erros. Ela pode ser usada tanto para o planejamento de teste quanto para projeto de casos de teste.

Os caminhos de 1 a 4 constituem um *conjunto base* para o grafo de fluxo da Figura 18.2b. Isto é, se testes podem ser projetados para forçar a execução desses caminhos (conjunto base), cada comando do programa terá sido executado com certeza pelo menos uma vez e cada condição terá sido executada em seus lados verdadeiro e falso. Deve-se notar que o conjunto base não é único. De fato, vários conjuntos base diferentes podem ser derivados para um dado projeto procedimental.

Como sabemos quantos caminhos procurar? O cálculo da complexidade ciclomática fornece a resposta. *Complexidade ciclomática* é uma métrica de software que fornece uma medida quantitativa da complexidade lógica de um programa. Quando usada no contexto do método de teste de caminho básico, o valor computado para a complexidade ciclomática define o número de caminhos independentes no conjunto base de um programa, fornecendo um limite superior para a quantidade de testes que devem ser realizados para garantir que todos os comandos tenham sido executados pelo menos uma vez.

A complexidade ciclomática tem um fundamento na teoria dos grafos e fornece uma métrica de software extremamente útil. A complexidade é calculada por uma de três maneiras:

Como se calcula a complexidade ciclomática?

1. O número de regiões do grafo de fluxo corresponde à complexidade ciclomática.
2. A complexidade ciclomática $V(G)$ para um grafo de fluxo G é definida como
   $V(G) = E - N + 2$
   em que E é o número de arestas do grafo de fluxo e N é o número de nós do grafo de fluxo.
3. A complexidade ciclomática $V(G)$ para um grafo de fluxo G também é definida como
   $V(G) = P + 1$
   em que P é o número de nós predicados contidos no grafo de fluxo G.

PONTO-CHAVE

A complexidade ciclomática fornece o limite superior no número de casos de teste que precisam ser executados para garantir que cada comando do programa tenha sido executado pelo menos uma vez.

Examinando mais uma vez o diagrama de fluxo da Figura 18.2b, a complexidade ciclomática pode ser calculada usando cada um dos algoritmos citados anteriormente:

1. O grafo de fluxo tem quatro regiões.
2. $V(G) = 11$ arestas $- 9$ nós $+ 2 = 4$.
3. $V(G) = 3$ nós predicados $+ 1 = 4$.

Portanto, a complexidade ciclomática para o grafo de fluxo da Figura 18.2b é 4.

E o mais importante, o valor para $V(G)$ fornece um limite superior para o número de caminhos independentes que podem formar o conjunto base e, como consequência, um limite superior sobre o número de testes que devem ser projetados e executados para garantir a abrangência de todos os comandos do programa.

CasaSegura

Usando a complexidade ciclomática

Cena: Sala da Shakira.

Participantes: Vinod e Shakira — membros da equipe de engenharia de software do CasaSegura que estão trabalhando no planejamento de teste para as funções de segurança.

Conversa:

Shakira: Olha... Eu sei que deveríamos fazer o teste de unidade em todos os componentes da função de segurança, mas eles são muitos, e se você considerar o número de operações que precisam ser exercitadas, eu não sei... Talvez devêssemos esquecer o teste caixa-branca, integrar tudo e começar a aplicar os testes caixa-preta.

Vinod: Você acha que não temos tempo suficiente para fazer o teste dos componentes, realizar as operações e então integrar?

Shakira: O prazo final para o primeiro incremento está se esgotando e eu gostaria de... Oh, estou preocupada.

Vinod: Porque você não aplica testes caixa-branca pelo menos nas operações que têm maior probabilidade de apresentar erros?

Shakira (desesperada): E como eu posso saber exatamente quais são as que têm maior possibilidade de erro?

Vinod: V de G.

Shakira: Hã?

Vinod: Complexidade ciclomática — V de G. Basta calcular V(G) para cada uma das operações dentro de cada um dos componentes e ver quais têm os maiores valores para V(G). São essas que têm maior tendência a apresentar erro.

Shakira: E como eu calculo V de G?

Vinod: É muito fácil. Aqui está um livro que descreve como fazer.

Shakira (folheando o livro): OK, não parece difícil. Vou tentar. As operações que tiverem os maiores V(G) serão as candidatas aos testes caixa-branca.

Vinod: Mas lembre-se de que não há nenhuma garantia. Um componente com baixo valor V(G) pode ainda ser sensível a erro.

Shakira: Tudo bem. Isso pelo menos me ajuda a limitar o número de componentes que precisam passar pelo teste caixa-branca.

"O foguete Ariane 5 explodiu no lançamento simplesmente por um defeito de software (uma falha) envolvendo a conversão de um valor em ponto flutuante de 64 bits em um inteiro de 16 bits. O foguete e seus quatro satélites não estavam segurados e valiam $500 milhões. Testes de caminho [que exercitam o caminho de conversão] teriam descoberto o defeito, mas foram vetados por razões de orçamento."

Notícia de um jornal

18.4.3 Derivação de casos de teste

O método de teste de caminho base pode ser aplicado a um projeto procedimental ou ao código-fonte. Nesta seção, apresento o teste de caminho básico como uma série de passos. O procedimento *média* (*average*), mostrado em PDL na Figura 18.4, será usado como exemplo para ilustrar cada passo no método de projeto de casos de teste. Note que *média,* embora sendo um algoritmo extremamente simples, contém condições compostas e ciclos. Os seguintes passos podem ser aplicados para derivar o conjunto base:

1. **Usando o projeto ou o código como base, desenhe o grafo de fluxo correspondente.** Um grafo de fluxo é criado usando os símbolos e regras de construção apresentados na Seção 18.4.1. De acordo com o PDL para *média* na Figura 18.4, é criado um grafo de fluxo enumerando-se os comandos PDL que serão mapeados por nós correspondentes do grafo de fluxo. O grafo de fluxo correspondente é mostrado na Figura 18.5.
2. **Determine a complexidade ciclomática do diagrama de fluxo resultante. A complexidade ciclomática** $V(G)$ é determinada aplicando-se os algoritmos descritos na Seção 18.4.2. Deve-se notar que $V(G)$ pode ser determinado sem desenvolver um grafo de fluxo contando todos os comandos condicionais no PDL (para o procedimento *média,* o total de condições compostas é igual a dois) e somando 1. De acordo com a Figura 18.5,

   $V(G) = 6$ regiões

   $V(G) = 17$ arestas $- 13$ nós $+ 2 = 6$

   $V(G) = 5$ nós predicados $+ 1 = 6$
3. **Determine um conjunto base de caminhos linearmente independentes.** O valor de $V(G)$ fornece o limite superior no número de caminhos linearmente independentes através da estrutura de controle do programa. No caso do procedimento *média,* esperamos especificar seis caminhos:

   Caminho 1: 1-2-10-11-13

   Caminho 2: 1-2-10-12-13

   Caminho 3: 1-2-3-10-11-13

FIGURA 18.4
PDL com nós identificados

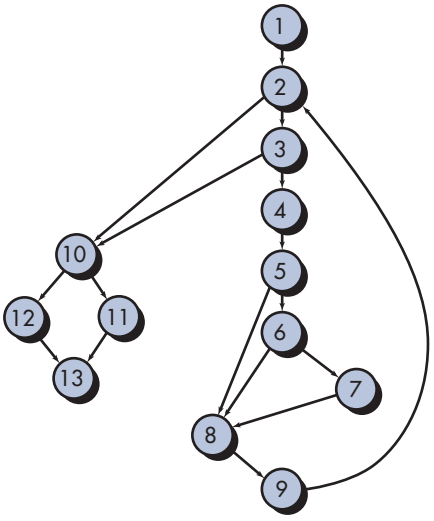
```
PROCEDURE average;

    *  Este procedimento calcula a média de 100 ou menos números
       situados entre valores limites; também calcula a soma e o
       total de números válidos.

    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid;
       minimum, maximum, sum IS SCALAR;
    TYPE i IS INTEGER;
1   i = 1;
    total.input = total.valid = 0;
    sum = 0;
2   DO WHILE value[i] <> -999 AND total.input < 100   3
4      increment total.input by 1;
       IF value[i] > = minimum AND value[i] < = maximum   6
5         THEN increment total.valid by 1;
7                sum = s sum + value[i]
          ELSE skip
8      ENDIF
       increment i by 1;
9   ENDDO
    IF total.valid > 0   10
11     THEN average = sum / total.valid;
12        ELSE average = -999;
13  ENDIF
END average
```

FIGURA 18.5
Grafo de fluxo para o procedimento *médio*



Caminho 4: 1-2-3-4-5-8-9-2-...

Caminho 5: 1-2-3-4-5-6-8-9-2-...

Caminho 6: 1-2-3-4-5-6-7-8-9-2-...

A reticência (...) após os caminhos 4, 5 e 6 indica que qualquer caminho através do resto da estrutura de controle é aceitável. Muitas vezes compensa identificar nós predicados como um auxílio na dedução de casos de teste. Nesse caso, os nós 2, 3, 5, 6 e 10 são nós predicados.

4. **Prepare casos de teste que vão forçar a execução de cada caminho do conjunto base.** Os dados devem ser escolhidos de forma que as condições nos nós predicados sejam definidas de forma apropriada à medida que cada caminho é testado. Cada caso de teste é executado e comparado com os resultados esperados. Depois que todos os casos de teste tiverem sido completados, o testador pode ter a certeza de que todos os comandos do programa foram executados pelo menos uma vez.

É importante notar que alguns caminhos independentes (por exemplo, caminho 1 em nosso exemplo) não podem ser testados de forma individual. Isso significa que a combinação de dados necessária para percorrer o caminho não pode ser conseguida no fluxo normal do programa. Nesses casos, esses caminhos são testados como parte de outro teste de caminho.

18.4.4 Matrizes de grafos

O procedimento para derivar o grafo de fluxo e até determinar um conjunto de caminhos base é passível de mecanização. Uma estrutura de dados, chamada de *matriz de grafos,* pode ser muito útil para o desenvolvimento de uma ferramenta de software que ajuda no teste do caminho base.

Uma matriz de grafo é uma matriz quadrada cujo tamanho (número de linhas e colunas) é igual ao número de nós no grafo de fluxo. Cada linha e coluna corresponde a um nó identificado, e as entradas da matriz correspondem a conexões (arestas) entre nós. Um exemplo simples de um grafo de fluxo e sua correspondente matriz de grafo [Bei90] é mostrado na Figura 18.6.

Observando a figura, cada nó do grafo de fluxo é identificado por números, enquanto cada aresta é identificada por letras. Uma letra na matriz corresponde à conexão entre dois nós. Por exemplo, o nó 3 está conectado ao nó 4 pela aresta *b*.

? O que é uma matriz de grafo e como posso ampliá-la para uso em teste?

Até aqui, a matriz de grafo é nada mais do que uma representação tabular de um grafo de fluxo. No entanto, acrescentando um *peso de ligação* a cada entrada da matriz, a matriz de grafo pode se tornar uma poderosa ferramenta para avaliar a estrutura de controle de programa