Desenvolvimento de Sistemas

Técnicas de teste: caixa-preta, caixa-branca; métodos de testes

Introdução

Utilizar técnicas de teste consistentes e aprovadas pelo mercado de TI (tecnologia da informação) tornará o desenvolvimento do projeto mais ágil e assertivo em seu resultado final. Atualmente, as técnicas são divididas em: funcionais e não funcionais.

Os testes funcionais validam as funções do sistema, ou seja, serão realizados testes nas funcionalidades descritas no documento de requisitos funcionais. Devem ser produzidos testes para contemplar todos os níveis. Geralmente medida em valores percentuais, a cobertura define a eficácia do plano de testes elaborado, tem como técnica conhecida a caixa-preta e baseia-se em experiências.

Um exemplo de teste funcional é em um determinado *software* de controle de estoque em que o operador do sistema deve emitir uma nota fiscal com diferentes produtos e, ao final da operação, o sistema tem como ação esperada a dedução dos itens vendidos da base de estoque.

Testes não funcionais verificarão: segurança, usabilidade, velocidade e demais características envolvidas no escopo do projeto, considerando as regras de negócio e as restrições envolvidas no projeto.

Para testes não funcionais, é possível citar como exemplo a seguinte situação: a empresa está para realizar a inserção de 50 novos operadores para trabalhar simultaneamente aos 20 que já interagem no sistema. Ao total, haverá 70 operadores que emitirão nota fiscal ao mesmo tempo. Dessa forma, será avaliada a *performance* do sistema após a escalabilidade realizada.

Outro exemplo de teste não funcional é a análise de comportamento do sistema ao sofrer uma tentativa de invasão à base de dados por meio de uma técnica de acesso não permitido. Nesse caso, um teste de segurança estará sendo realizado. Ainda, é possível testar a usabilidade do sistema ao avaliar como pessoas de diferentes faixas etárias se portarão ao utilizar o sistema.

Estas são algumas técnicas utilizadas no mercado:

Teste de caixa-branca: consiste na utilização do código-fonte do projeto e também é conhecido como teste estrutural.

Teste de caixa-preta: considera as funcionalidades do sistema, deixando de lado o código-fonte e testando apenas as funções desenvolvidas.

Teste de caixa-cinza: consiste na junção dos testes de caixa-branca e caixa-preta e utiliza a engenharia reversa, sendo possível, assim, analisar erros e problemas gerados no desenvolvimento do projeto.

Nesta unidade curricular, serão abordados com mais ênfase o teste de caixa-branca e o teste de caixa-preta.

Caixa-preta

Essa técnica considera as entradas e saídas esperadas para o **software**, sem levar em consideração a estrutura do código-fonte. Dessa forma, é avaliado se a estrutura está operando da forma especificada na documentação, seguindo os requisitos funcionais, por isso também é conhecida como teste de comportamento. A técnica da caixa-preta não permite a verificação do código desenvolvido para validar se está realizando uma função de forma otimizada ou não.

Geralmente tem um ambiente de teste seguindo as premissas estipuladas em casos de uso, *user stories* e especificações do sistema. Os casos de testes encontrarão possíveis falhas entre o que foi especificado e o que realmente está acontecendo. A cobertura é parametrizada,

considerando os itens testados na base e a técnica aplicada.

Para mais informações sobre o nível de cobertura, busque em seu navegador por “padrão internacional ISO/IEC/IEEE 29119-4”.

Métodos de teste

Agora, você saberá mais a respeito dos métodos de teste.

Particionamento de equivalências

Imagine o seguinte caso: uma nova empresa de tintas está desenvolvendo um *software* para validar o preenchimento de seus tanques de tintas e definiu que um tanque deve ao menos ter 10 l de tintas e no máximo 50 l. Com isso, ao cadastrar o preenchimento do tanque, o campo de litros poderá aceitar valores entre 10 e 50.

Para facilitar o entendimento, analise a linha de valores que podem ou não ser inseridos no momento de realizar a tarefa.

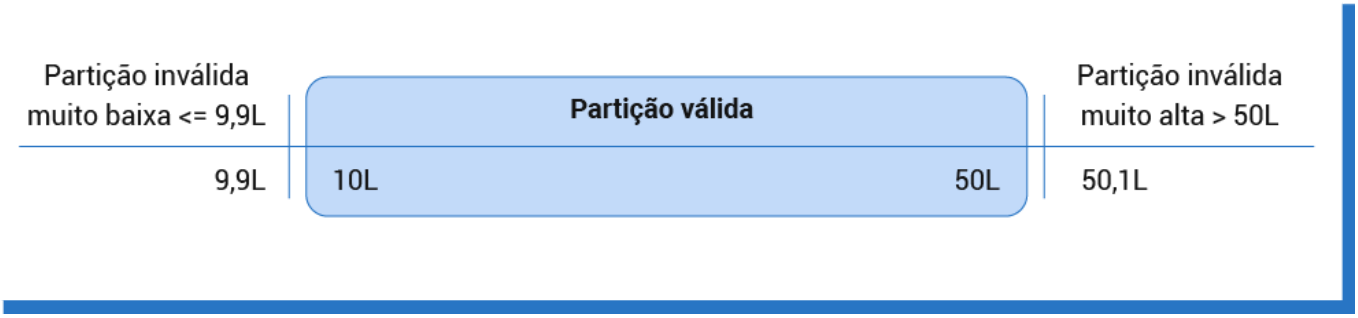

Figura 1 – Método de teste particionamento de equivalência
Fonte: Senac EAD (2023)

Ao analisar a imagem, percebem-se três partições: partição inválida muito baixa, partição válida e partição inválida muito alta. Os valores para a partição muito baixa estão abaixo da quantidade de litros que limita a disponibilidade de reserva, já a partição inválida muito alta limitará os valores acima da quantidade de litros que limita a disponibilidade. A partição válida contém os valores que são aceitos para definir cada quantidade de litros que permitem o preenchimento do tanque.

É o processo de divisão de dados, de modo que todos os membros pertencentes a uma partição serão processados do mesmo modo. Além disso, consiste na criação de partições para valores válidos e inválidos. Dessa forma, cria-se uma divisão para os valores que serão aceitos (válidos) e os que não serão aceitos (inválidos). Muitas vezes é criado de forma “inconsciente”.

A cobertura é feita por meio do seguinte cálculo:

*COBERTURA = (PARTIÇÕES TESTADAS÷TOTAL DE PARTIÇÕES)*100%*

Agora, imaginando os casos de teste que devem ser criados para cobrir essa situação, quantos seriam necessários para atender a todas as partições?

Devem ser criados três casos: um caso de teste com uma quantidade inferior ao mínimo de 10 l, um caso de teste com uma quantidade entre 10 l e 50 l e um caso de teste com uma quantidade superior a 50 l.

As partições inválidas devem ser testadas separadamente, ou seja, em partições diferentes, para evitar que uma falha possa mascarar as demais falhas que possam ocorrer. Desta forma um erro impediria os demais de serem detectados.

Análise de valor-limite

É uma ampliação do particionamento de equivalência e constitui-se de dados numéricos ou sequenciais, em que os valores mínimo e máximo são os valores-limite para uma partição. Pode ser aplicada em qualquer nível de teste e tem uma probabilidade maior de encontrar falhas em relação à partição de equivalência. A cobertura é definida da seguinte maneira:

*COBERTURA = (LIMITES TESTADOS ÷ TOTAL DE LIMITES) * 100%*

Agora, pense no exemplo anterior, em que o limite no campo de litros foi implementado:

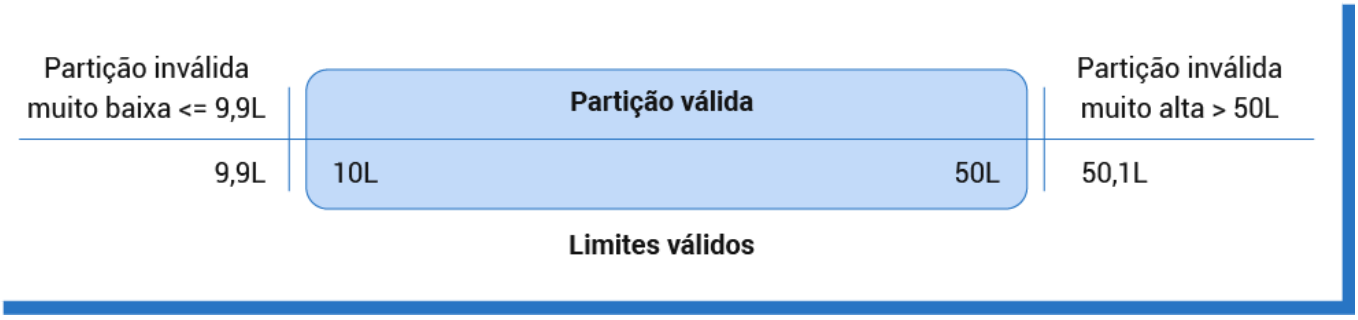

Figura 2 – Método de teste análise de valor-limite

Fonte: Senac EAD (2023)

Com base no conhecimento adquirido, você já sabe que valores abaixo de 10 l fazem parte da partição inválida muito baixa, que a partição válida contém os valores entre 10 l e 50 l e que a partição inválida muito alta tem valores acima de 50 l. Porém, agora está sendo feita apenas a análise dos valores-limite, portanto apenas 10 l e 50 l são os limites válidos para a partição válida. O limite da partição inválida muito baixa é somente 9,9 l e o limite de 50,1 l é o correspondente à partição inválida muito alta.

Agora, imaginando os casos de teste que devem ser criados para cobrir os limites dessa situação, quantos seriam necessários para atender a todas as partições?

Nessa situação, você precisa de quatro casos de teste, sendo um teste para cobrir o valor-limite da partição inválida muito baixa, nesse caso 9,9 l; um teste para cobrir o valor-limite mínimo válido da partição válida, nesse caso 10 l; um teste para cobrir o valor-limite máximo válido da partição válida, nesse caso 50 l; e um teste para cobrir o valor-limite da partição inválida muito alta, nesse caso 50,1 l.

Agora é hora praticar criando o código para esse projeto. Primeiro, você desenvolverá a base, criando uma variável para armazenar o nome que referenciará o tanque de tinta e exibirá os dados cadastrados ao final.

```
import java.util.Scanner;

public class TesteTanque {

    public static void main(String[] args) {
        String identificacao;
        double qtdTinta;
        Scanner entrada = new Scanner(System.in);

        System.out.println("Informe a identificação do tanque: ");
        identificacao = entrada.nextLine();
        System.out.println("Informe a quantidade de litros inserido no tanque: ");
        qtdTinta = entrada.nextDouble();

        System.out.println("O nome do tanque é: " + identificacao + " e a quantidade presente
de litros é: " + qtdTinta);
    }
}
```

Agora, você realizará os testes de análise de valor-limite e verificará se está cobrindo os requisitos esperados para o projeto. Então, execute o código e analise o que acontece. Primeiro, informe um valor mínimo aceito de 10 l e, posteriormente, execute o código novamente e informe um valor máximo aceito de 50 l. Em ambos os casos deve ser exibida a mensagem de litros presentes no tanque e a identificação dele.

Então, você realizará os testes para os valores abaixo do mínimo aceito e acima do valor máximo aceito, portanto execute o código novamente. Primeiro, informe 9,9 l e veja o que acontece.

Execute outra vez o código e informe 50,1 l. Em ambos os casos deve ser exibida uma mensagem de valores inválidos informados.

Você percebeu que o código apresentou falhas, pois está faltando algo, não é mesmo? No teste, foram encontradas falhas do projeto; as estruturas de decisão não foram criadas para validar os valores que devem ser aceitos, ou seja, a cobertura do teste é de 50%. Nesse momento, você reportará, em um documento adequado, que essa função do sistema não está adequada ao que era esperado no documento de requisitos e deve ser refeita.

Portanto, refaça e adicione as validações que devem estar presentes.

```
import java.util.Scanner;

public class TesteTanque {

    /**
    * @param args the command line arguments
    */
    Public static void main(String[] args) {
      // TODO code application logic here
          String identificacao;
          Double qtdTinta;
          Scanner entrada = new Scanner(System.in);

          System.out.println("Informe a identificação do tanque: ");
          identificacao = entrada.nextLine();
          System.out.println("Informe a quantidade de litros inserido no tanque: ");
          qtdTinta = entrada.nextDouble();
          if(qtdTinta>=10 && qtdTinta<=50){
              System.out.println("O nome do tanque é: " + identificacao + " e a quantidade pre
sente de litros é: "+ qtdTinta);
          } else {
              System.out.println("A quantidade de tinta informada é inadequada para a capacida
de do tanque");
          }
    }
```

```
}
```

Agora, você deve refazer os testes de análise de valor-limite e verificar se o projeto cobre os requisitos esperados.

Como era esperado, dessa vez os valores-limite inválidos, muito baixo e muito alto, não são aceitos. Já os limites válidos são aceitos, portanto o seu teste tem 100% de cobertura.

Tabela de decisão

A técnica de tabela de decisão testa combinações que resultarão em dados diferentes, sendo considerada interessante para cobrir regras de negócio complexas que um sistema tem. A tabela é composta de condições, que são as entradas, e ações (os resultados gerados na saída), que são as linhas. Para as colunas, há regras que indicarão quais condições têm as ações esperadas. O número de colunas é baseado no número de casos de testes.

As condições são expressas da seguinte forma:

“S”: especifica que a condição é atendida (em algumas literaturas, pode ser exibido como “V” ou “1”).
“N”: especifica que a condição não é atendida (em algumas literaturas, pode ser exibido como “F” ou “0”).
“-“: especifica que o valor não é importante para a condição (em algumas literaturas, pode ser exibido como “N/A”).

Tabela 1 – Condições

Fonte: Senac EAD (2023)

Para as ações, há as seguintes expressões:

“X”: especifica que a ação deve ocorrer (em algumas literaturas, pode ser exibido como “S”, “V” ou “1”).
Em branco: especifica que a ação não deve ocorrer (em algumas literaturas, pode ser exibido como “-“, “N”, “F” ou “0”).

Tabela 2 – Ações

Fonte: Senac EAD (2023)

Agora, pense na seguinte situação: uma *startup* de recursos humanos está desenvolvendo uma plataforma para facilitar a contratação de colaboradores e deseja disponibilizar para os clientes uma ferramenta que verifica se o candidato tem as qualificações esperadas para a vaga. Dessa forma, alguns requisitos devem ser expressos: ter formação esperada para a área, ter nível de experiência adequado, conforme especificação do cliente, e ter domínio no idioma solicitado.

Os candidatos que tiverem os requisitos necessários, ou seja, atenderem às condições, participarão da próxima etapa do processo seletivo. Os que não atenderem a um ou mais requisitos irão para o banco de talentos da empresa, sendo esta a ação esperada.

Veja a tabela de decisão:

CONDIÇÕES	Regra 1 (caso de teste 1)	Regra 2 (caso de teste 2)	Regra 3 (caso de teste 3)	Regra 4 (caso de teste 4)	Regra 5 (caso de teste 5)	Regra 6 (caso de teste 6)	Regra 7 (caso de teste 7)	Regra 8 (caso de teste 8)
Formação esperada para a área?	S	S	S	S	N	N	N	N
Nível de experiência adequado?	S	S	N	N	S	S	N	N
Domínio no idioma solicitado?	S	N	S	N	S	N	S	N
AÇÕES								
Participar da próxima etapa do processo seletivo	X							
Banco de talentos da empresa		X	X	X	X	X	X	X
LEGENDA: S = sim; N = não; X = ação a ser executada								

Tabela 3 – Tabela de decisão

Fonte: Senac EAD (2023)

Dessa forma, é possível verificar diferentes casos de teste e analisar o resultado de cada um, sendo, assim, uma técnica adequada para cobrir os requisitos mais complexos. No exemplo, apenas o primeiro caso de teste cobre por completo os requisitos e seguiria no processo seletivo; os demais iram para o banco de talentos.

Transição de estado

O método de transição de estados mostrará, por meio de um diagrama, os possíveis estados que o *software* pode ter, exibindo o modo como este alterna entre os estados, passando por entrada, saída e transição.

As transições são iniciadas com base em um evento, que, por exemplo, pode ser a inserção de dado em um campo. Com isso, uma mudança de estado pode gerar a execução de uma ação, que pode ser, por exemplo, a exibição de uma mensagem de confirmação na tela ou um erro em razão de inconsistências.

Há uma tabela de transição de estado que exibirá as transições válidas e inválidas entre estados. Os eventos também fazem parte da tabela. Já os diagramas mostram apenas transições válidas, deixando de fora as transições inválidas.

Pense em uma situação de abastecimento do tanque de tintas e elabore uma tabela de transição de estados. Há uma bomba de abastecimento que encherá o tanque ou ele poderá receber uma determinada quantidade de litros de tinta.

A tabela de testes de transição de estados ficará da seguinte forma:

Número do teste	Estado inicial	Ação	Estado final	Resultado esperado
1	Tanque vazio	Abastecer tanque	Tanque cheio	Tanque cheio
2	Vazio	Abastecimento com uma quantidade X	Tanque com valor X	Tanque com X litros de tinta
3	Abastecimento em andamento	Cancelar abastecimento	Vazio	Ao cancelar a ação, o tanque não realiza o abastecimento
4	Tanque cheio	Abastecimento com uma quantidade X de litros	Tanque com valor X de litros	Tanque já está cheio e não é possível abastecer
5	Tanque com valor X de tinta	Abastecimento com uma quantidade X de litros	Tanque com valor X de litros	Não é possível abastecer, pois ultrapassará a quantidade máxima do tanque
6	Falha do sistema	Cancelar abastecimento	Vazio	Em falha, o tanque não deve abastecer

Tabela 4 – Tabela de testes de transição de estados

Fonte: Senac EAD (2023)

A tabela de testes de transição de estados para a bomba de abastecimento mostra os diferentes estados que a bomba pode estar e as ações que o usuário pode realizar, levando a um estado final esperado. A tabela começa com o estado vazio da bomba e as ações de informar um valor específico para abastecimento ou escolher encher o tanque completamente. Quando o usuário escolhe informar um valor X, a bomba passa para o estado de abastecimento em andamento e começa a realizar o abastecimento. Se o usuário cancelar, a bomba volta ao estado vazio. Se a bomba estiver cheia, o usuário não poderá adicionar mais combustível, independentemente da ação escolhida. Em ambos os casos, o sistema informará ao usuário que o tanque já está cheio. No caso de falha do sistema, a bomba volta ao estado vazio.

Geralmente os testes são criados de forma a percorrer todos os estados, passar por cada transição ou cobrir uma sequência predefinida de transições ou transições inválidas.

Teste de caso de uso

O método de teste de caso de uso é baseado nos casos de uso. Desse modo, especificará o comportamento que uma funcionalidade terá ao interagir com um ou mais atores do sistema.

Os testes de caso de uso ajudam a garantir a qualidade e a confiabilidade do sistema, validando situações de uso e verificando se situações excepcionais têm o tratamento de falhas de sistema adequados. A medida de cobertura é expressa pela quantidade de comportamentos de casos de uso que serão testados dividida pelo total de comportamentos de caso de uso.

Caixa-branca

A técnica de teste de caixa-branca analisa a arquitetura do projeto, avaliando de modo interno a estrutura, sendo, assim, possível resolver questões de otimização do projeto, gerando um processamento mais adequado. Dessa forma, é possível criar testes que percorrerão mais trechos do código, evitando que ocorram redundâncias ou testes em duplicidade por meio de *scripts* criados para executar os testes de forma automática.

Ainda é possível realizar uma análise de cobertura de código. Dessa forma, é possível mensurar qual percentual do código é verificado e, com base nesse dado, criar testes complementares para validar o percentual faltante. A cobertura é medida com base nos itens testados de uma estrutura, de acordo com a técnica utilizada.

Por exemplo, ao criar um teste para cobrir todas as estruturas de decisão, os **ifs/elses** criados serão testados para verificar se o comportamento do sistema é o adequado ao passar por aquele trecho. Essa técnica tem um cálculo específico para verificar o percentual dos testes que obtiveram êxito ao serem executados.

Como se baseia na estrutura interna do objeto, as classes, as funções e os pacotes são verificados para validar a cobertura. Pode ser realizado em todos os níveis de teste, porém é mais utilizado para teste de componentes ou de integração de componentes.

Teste de cobertura de instruções

Verifica as instruções executáveis de um código. Tem como cálculo para percentual de cobertura a seguinte formula:

(Instruções Executadas / Total de Instruções Existentes) * 100

É a técnica de cobertura menos eficiente, pois verifica apenas se as instruções estão sendo executadas, desconsiderando outros tipos de cobertura, como situações condicionais. Em algumas literaturas, pode ser encontrada como cobertura de declaração, comando ou até mesmo sentença.

Agora, pense em um exemplo prático: em uma competição de salto em altura, um atleta tem três oportunidades para registrar a maior marcar de seu salto. Você foi convidado a desenvolver um *software* para aferir a maior marca de um atleta, portanto construirá o seguinte código:

```
import java.util.Scanner;

public class Competicao {

public static void main(String[] args) {
  // TODO code application logic here
  Scanner entrada = new Scanner(System.in);
  double salto1, salto2, salto3, maior;
  System.out.println("informe a altura do salto 1");
  salto1 = entrada.nextDouble();
  maior = salto1;
  System.out.println("Informe a altura do segundo salto");
  salto2 = entrada.nextDouble();
  System.out.println("Informe a altura do terceiro salto");
  salto3 = entrada.nextDouble();
  if (salto2 > maior) {
      maior = salto2;
      if (salto3 > maior) {
          maior = salto3;
      }
  } else {
      if (salto3 > maior) {
          maior = salto3;
      }
  }
  System.out.println("A maior marca é de: " + maior + " ms");
}
}
```

Agora, pense no seguinte caso de teste para verificar a cobertura das instruções: o primeiro salto teve altura de 1,89 m, o segundo salto de 1,95 m e o terceiro foi anulado, portanto fica com 0,0 m. Dessa forma, teste seu código com esses valores para verificar quais instruções serão executadas. Para facilitar a análise, crie um fluxograma com as instruções do programa.

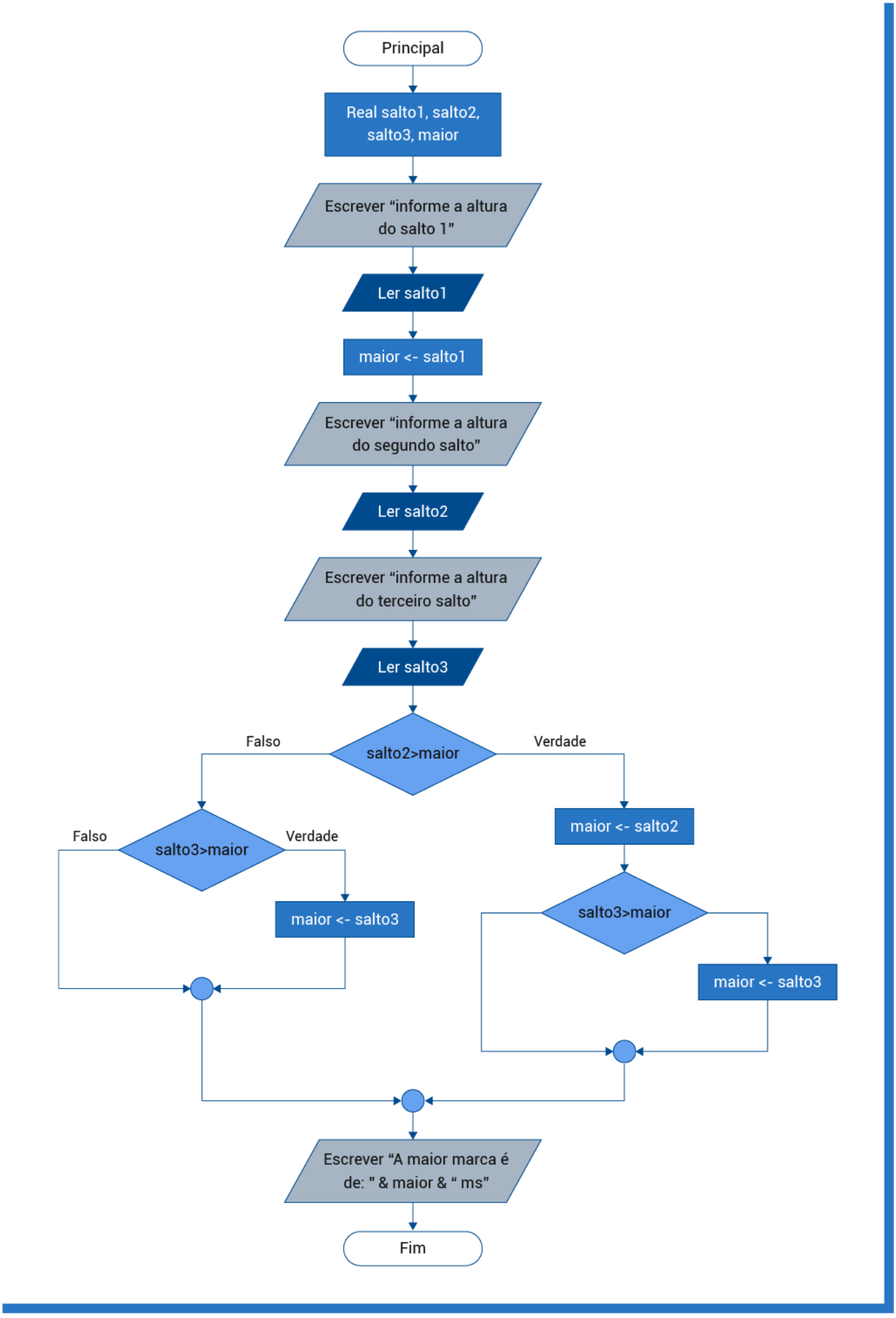
Principal
Real salto1, salto2, salto3, maior
Escrever "informe a altura do salto 1"
Ler salto1
maior <- salto1
Escrever "informe a altura do segundo salto"
Ler salto2
Escrever "informe a altura do terceiro salto"
Ler salto3
salto2>maior
Falso
Verdade
salto3>maior
Falso
Verdade
maior <- salto3
maior <- salto2
salto3>maior
maior <- salto3
Escrever "A maior marca é de: " & maior & " ms"
Fim

Figura 3 – Método de teste cobertura de comandos

Fonte: Senac EAD (2023)

Cada figura geométrica no fluxograma significa uma instrução, portanto há 15 (ficarão de fora os círculos rosas, que estão vazios, as setas e, em roxo, as elipses **Principal** e **Fim**).

Agora, analise as instruções. O programa realizará as declarações, as entradas/saídas na tela e as atribuições até chegar na primeira estrutura de decisão. Até a estrutura de decisão, oito instruções foram realizadas. A nona instrução a ser executada será a verificação de atendimento da condição. Com os dados desse caso de teste, o segundo salto é de 1,95 m, portanto maior que o valor que já está armazenado na variável maior. Logo, será executada também a instrução de atribuição de valor maior na variável maior, que é a décima instrução a ser testada.

A próxima instrução a ter sua cobertura verificada é a de análise do terceiro salto (não perca a conta ,11º), se ele é maior que o atual valor em maior. Como o terceiro salto foi anulado, a condição não é atendida e, assim, o programa segue para o 12º comando, que encerra a execução exibindo uma mensagem com a maior altura saltada. Considerando o cálculo de cobertura, foram executadas 12 instruções em um total de 15, portanto a cobertura desse teste é de 80%.

Teste de cobertura de decisão

Esse método de teste verifica as estruturas de decisões presentes no código, assim como as instruções que serão executadas conforme o resultado das condições. Os casos de testes devem ser criados de forma a validar situações condicionais. Por exemplo, se for uma estrutura **if**, deve-se criar um caso de teste para quando a condição é atendida e um para quando a condição não é atendida (verdadeiro ou falso, respectivamente). Em uma estrutura **switch case**, todos os casos devem ter um caso de teste que cobrirá o resultado, incluindo a resposta padrão.

A cobertura é medida por meio do seguinte cálculo:

(Número de resultados de decisão executados / Total de decisões existentes) *100

Em algumas literaturas, utilizam-se termos como “ramificação” e “desvio”.

Agora, pense no seguinte exemplo: uma empresa está considerando subsidiar um auxílio para seus funcionários realizarem um curso de especialização em qualquer área, independentemente de ter ou não relação com seu cargo na empresa, no entanto o funcionário deve aceitar que o percentual restante seja descontado diretamente em folha de pagamento. O colaborador poderá, ainda, definir se, ao escolher realizar a especialização, receberá um outro subsidio de transporte, novamente descontado em folha.

Quantos e quais seriam os casos de uso necessários para cobrir todas as estruturas de decisão presentes no seguinte trecho de código?

```
import java.util.*;
import java.lang.Math;

public class JavaApplication {
private static Scanner input = new Scanner(System.in);

public static void main(String[] args) {
  String curso;
  String transporte;

  String curso;
  String transporte;

  System.out.println("Informe sim,se desejar realizar o curso de especialização ou não para re
cusar!");
  curso = input.nextLine();
  if (curso.equals("sim")) {
      System.out.println("Informe sim,se desejar ganhar o subsidio para transporte ou não para
recusar!");
      transporte = input.nextLine();
      if (transporte.equals("sim")) {
          System.out.println("Você optou não receber o subsidio do transporte e fará a especia
lização");
      } else {
          System.out.println("Você optou por não receber o subsidio do transporte, mas fará a
especialização");
      }
  } else {
      System.out.println("Você optou por não realizar a especialização");
  }
  System.out.println("Obrigado por participar");
}
}
```

Use novamente um fluxograma para auxiliar na análise da execução.

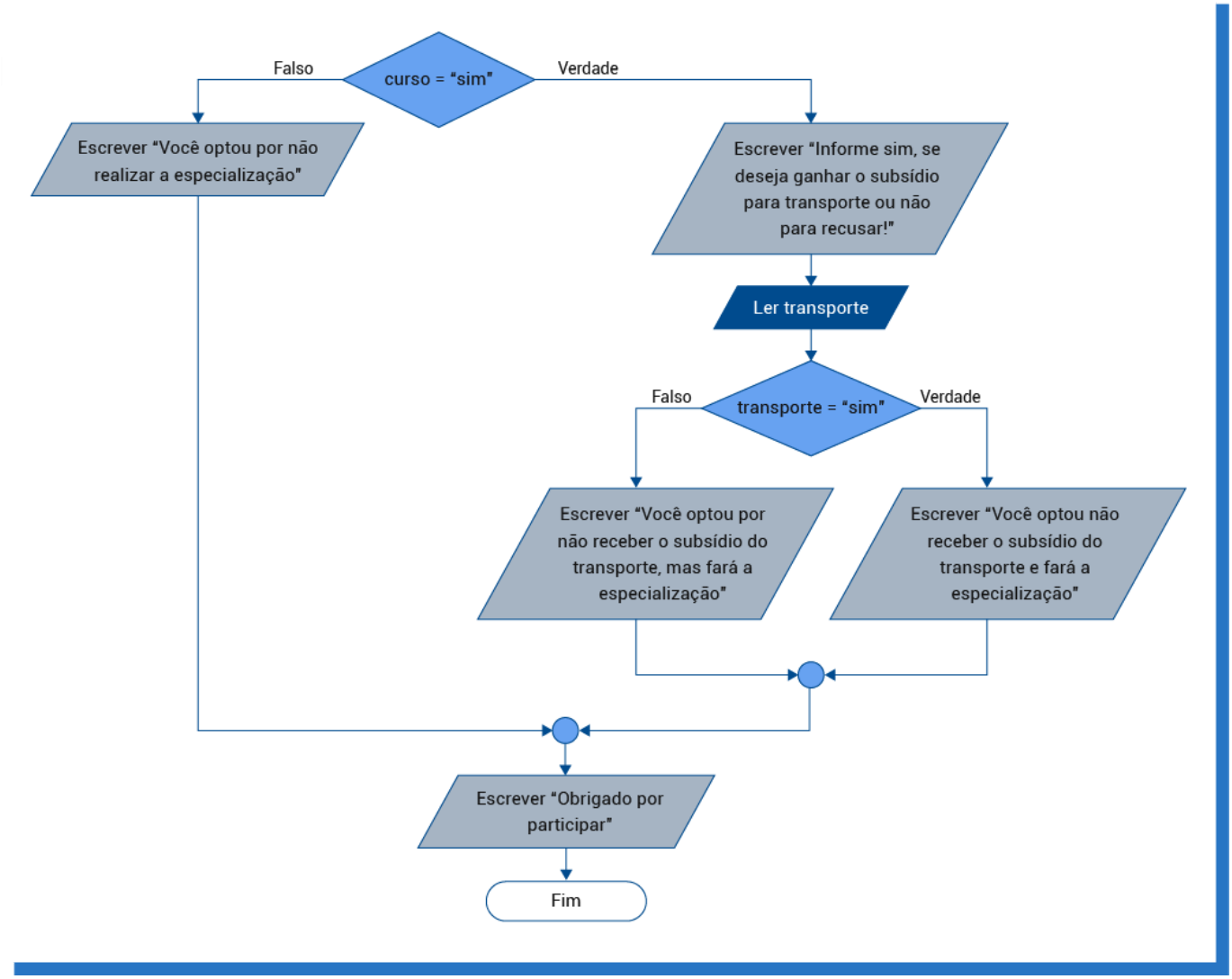

Figura 4 – Método de teste cobertura de decisão

Fonte: Senac EAD (2023)

O primeiro caso de teste seria definir a resposta para a participação no curso como “não”. Dessa forma, encerra a execução do programa, pois irá pelo caminho de decisão falso.

O segundo caso de teste seria definir a primeira resposta como “sim” e a segunda como “sim” novamente. Assim, passaria por todas as condições verdadeiras.

O terceiro caso de teste seria definir a primeira resposta como “sim” e a segunda como “não”. Dessa forma, percorreria pelo caminho da primeira decisão sendo verdadeiro e o caminho da segunda decisão sendo falso. Logo, seriam necessários três casos de teste para atingir a cobertura de 100% dos caminhos disponíveis conforme as estruturas de condição existentes.

Dessa forma, pode-se afirmar que cobrir 100% de um teste de cobertura de decisão cobrirá 100% de um teste de comando.

Desafio 1: Desafio de teste de tabela de decisão

Você é um desenvolvedor de *software* para uma empresa de seguros automotivos. Sua equipe está desenvolvendo um sistema para calcular o valor do seguro de carros, levando em consideração vários fatores, como idade do motorista, modelo do carro e ano do carro.

Seu desafio é criar uma tabela de decisão de teste que cubra os cenários mais importantes para o cálculo do valor do seguro. Para isso, considere as seguintes premissas:

- O modelo e o ano do carro também afetam o valor do seguro. Carros 0 km e de modelos acima de R$ 60 mil terão um valor de 10% da tabela Fipe (tabela da Fundação Instituto de Pesquisas Econômicas).
- O valor do seguro é influenciado pela idade do motorista. Se o motorista tiver menos de 25 anos, o valor do seguro terá um acréscimo de 40%.
- Se o motorista tiver mais de 60 anos, o valor do seguro será automaticamente reduzido em 30%.

Desafio 2: Desafio de teste de cobertura de decisão

Você foi contratado por uma empresa de desenvolvimento de *software* que está desenvolvendo um *e-commerce*. Sua equipe está desenvolvendo uma classe em Java para calcular o valor total de uma compra realizada pelo cliente.

Seu desafio é analisar o código e escrever casos de teste para cobrir todas as instruções da classe e verificar quantos são necessários para cobrir todas as situações. Você deve considerar as seguintes premissas:

- O valor da compra é calculado com base na quantidade de itens comprados, no preço unitário de cada item e no desconto que pode ser aplicado.
- O desconto é determinado com base no valor total da compra. Se o valor em reais for maior ou igual a mil, o desconto é de 10%; se o valor em reais for menor que mil, não há desconto.

O código produzido é o seguinte:

```
public class CalculadoraValorCompra {

    public static double calcular(int quantidade, double precoUnitario, double taxaDesconto) {
        double total = quantidade * precoUnitario;
        if (total >= 1000) {
            total = total - (total * taxaDesconto / 100);
        }
        return total;
```

```
    }
}
```

Encerramento

Neste material, você aprendeu a diferença de testes funcionais e testes não funcionais e verificou conceitos de técnicas de testes de caixa-preta e caixa-branca. Além disso, você foi apresentado a métodos de técnicas de testes e praticou por meio de exemplos.