Engenharia de Software

Anhanguera


AVALIE
SUA PROFISSÃO
QUANDO APARECER EM SEU PORTAL UMA AVALIAÇÃO SOBRE SEU CURSO, RESPONDA:
NOTAS
9 OU 10
SIGNIFICA QUE VOCÊ INDICA
NOTAS
7 OU 8
SIGNIFICA QUE VOCÊ NÃO INDICA
Anhanguera

Anhanguera

TIPOS DE TESTE

Como já se sabe, um produto de software necessita de inúmeros elementos para oferecer ao seu usuário um funcionamento pleno. A adequação do hardware e da infraestrutura de rede são apenas dois desses elementos, e o dimensionamento incorreto de um deles terá o potencial de arruinar a experiência do usuário. Há, noentanto, um aspecto que sempre terá papel decisivo na percepção do usuário em relação ao sistema que utiliza: as suas funções. Afinal, será por meio delas que eleinteragirá com o programa e será através delas que alcançará seus objetivos traçados lá na fase de requisitos. Funções mal projetadas ou defeituosas podem, inclusive, desestimular o uso do sistema. Com tamanha importância, a área detestes não poderia deixar de direcionar muita atenção às funções do sistema e o faz por meio da aplicação da Técnica de Teste Funcional e do Teste de Funcionalidades, assuntos que desenvolveremos na sequência.

ESTRATÉGIAS DE TESTES

Entretanto, antes de abordarmos especificamente técnicas de teste, vale a pena assituarmos no contexto da estratégia de teste, a qual equivale a uma prática cujo objetivo é estabelecer um roteiro que descreve os passos a serem executados no processo de teste ou, em outras palavras, um modelo para os testes. Pressman e Maxim (2016) descrevem genericamente alguns elementos presentes em todas asestratégias adotadas para os testes:

Revisões de software: boas revisões eliminam problemas no código antes da efetiva aplicação dos testes.

Progressão do teste: os testes devem começar em uma unidade do software e progredir em direção à integração do sistema como um todo.

Depuração: esta atividade deve estar associada ao teste embora sejamelementos distintos entre si.

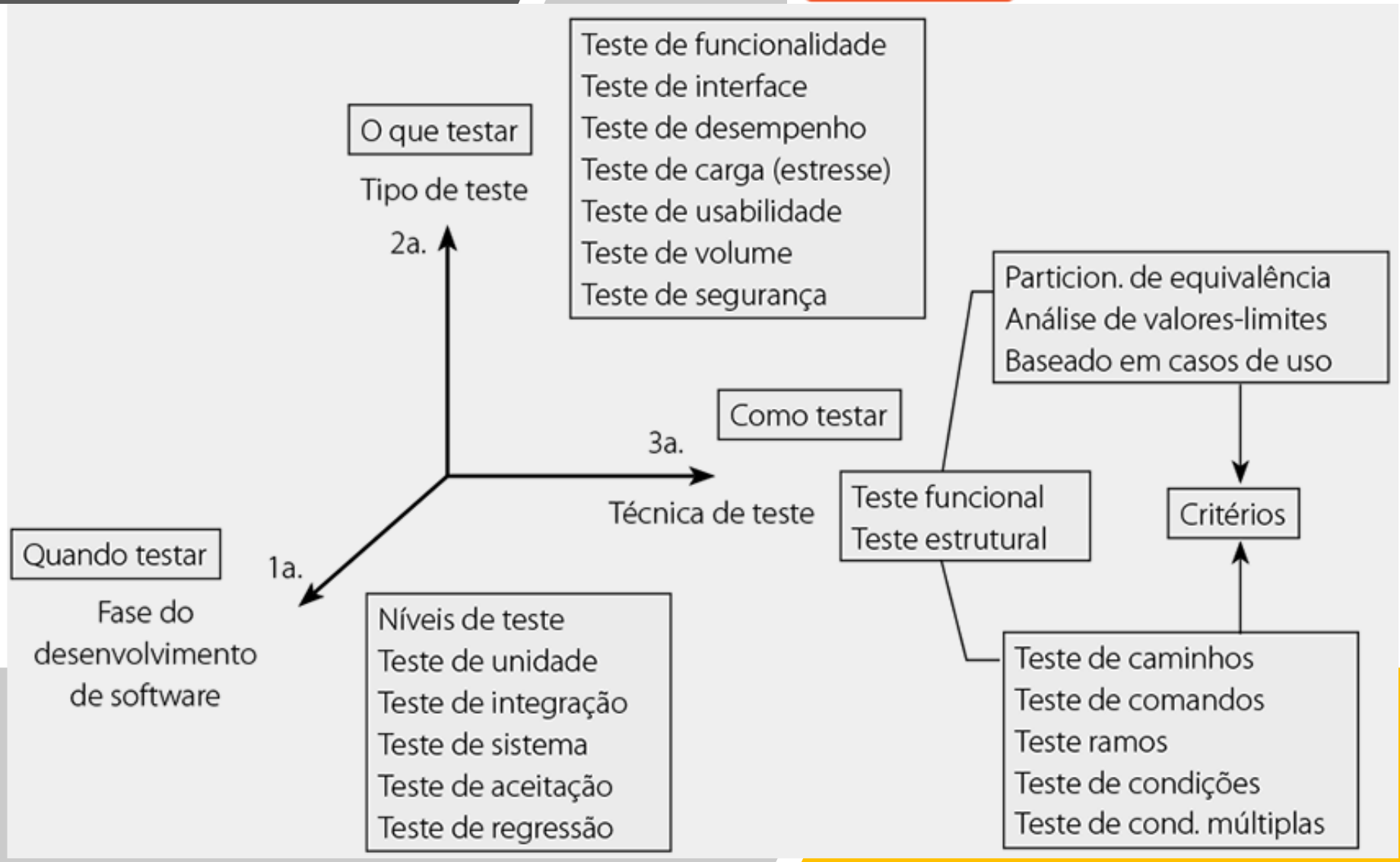
Técnicas: diferentes técnicas de teste são adequadas a diferentes sistemas esão aplicadas conforme os recursos disponíveis à equipe.

O último item da estratégia de teste embasa nosso próximo assunto e, para entendermos as razões de uma técnica, vale uma breve digressão: os testes sãorealizados tendo em vista objetivos específicos e são projetados para verificar diferentes propriedades de um sistema. A depender da técnica escolhida, os casos de teste podem ser planejados, por exemplo, para verificar se as especificações funcionais estão implementadas corretamente. Em outros casos, eles são selecionados para averiguação da adequação do desempenho, da confiabilidade ea da usabilidade, por exemplo. Assim, abordagens diferentes de teste motivam aaplicação de técnicas também diferentes de testes (PRESSMAN; MAXIM, 2016).

TIPOS DE TESTES

Embora as técnicas de teste sejam de grande relevância, por remeterem mais diretamente a metodologias de aplicação de testes, esta é não a única dimensão que podemos identificar nesse contexto. Na verdade, podemos estabelecer outros tipos de relações entre um teste e seu objetivo e, para isso, basta fixarmos o momento, o objeto e, como já mencionado, a maneira (ou metodologia) da aplicação.

Parece complicado? Observe a Figura 3.7: ela posiciona os vários tipos de teste nas três dimensões que acabamos de mencionar: o “quando”, o “que” e o “como”.

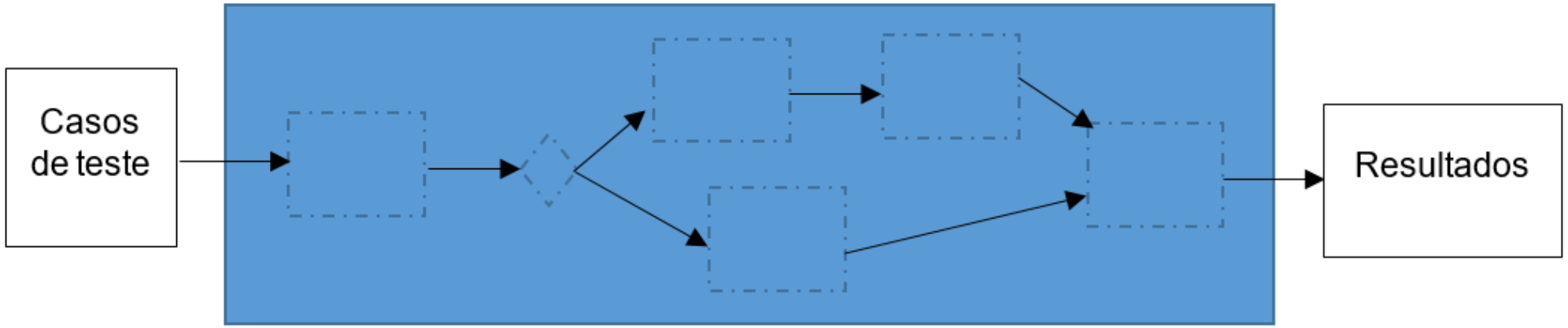

O que testar
Tipo de teste
2a.
Teste de funcionalidade
Teste de interface
Teste de desempenho
Teste de carga (estresse)
Teste de usabilidade
Teste de volume
Teste de segurança
Como testar
3a.
Técnica de teste
Particion. de equivalência
Análise de valores-limites
Baseado em casos de uso
Teste funcional
Teste estrutural
Critérios
Teste de caminhos
Teste de comandos
Teste ramos
Teste de condições
Teste de cond. múltiplas
Quando testar
Fase do desenvolvimento de software
1a.
Níveis de teste
Teste de unidade
Teste de integração
Teste de sistema
Teste de aceitação
Teste de regressão

Apesar de não termos tido contato com os testes apontados na figura, é possível destacar que o teste funcional está relacionado a uma maneira de se realizar um teste, daí ter sido posicionado na dimensão “Como testar”. Já o teste de funcionalidade – que também será abordado na sequência – guarda associação com “o que testar”. Feita essa contextualização, passamos à abordagem inicial datécnica de teste funcional e do teste de funcionalidade.

TÉCNICA DE TESTE FUNCIONAL E TESTE DE FUNCIONALIDADE

Essa técnica baseia-se nas especificações do software para derivar os requisitos deteste. O teste é realizado nas funções do programa, daí o nome funcional. Não éseu objetivo verificar como ocorrem internamente os processamentos, mas se oalgoritmo inserido produz os resultados esperados (BARTIÉ, 2002).

Uma das vantagens dessa estratégia de teste é o fato de ela não requerer conhecimento sobre detalhes da implementação do programa. Sequer o código-fonte é necessário. Observe uma representação dessa técnica na Figura 3.8:


Casos
de teste
Resultados

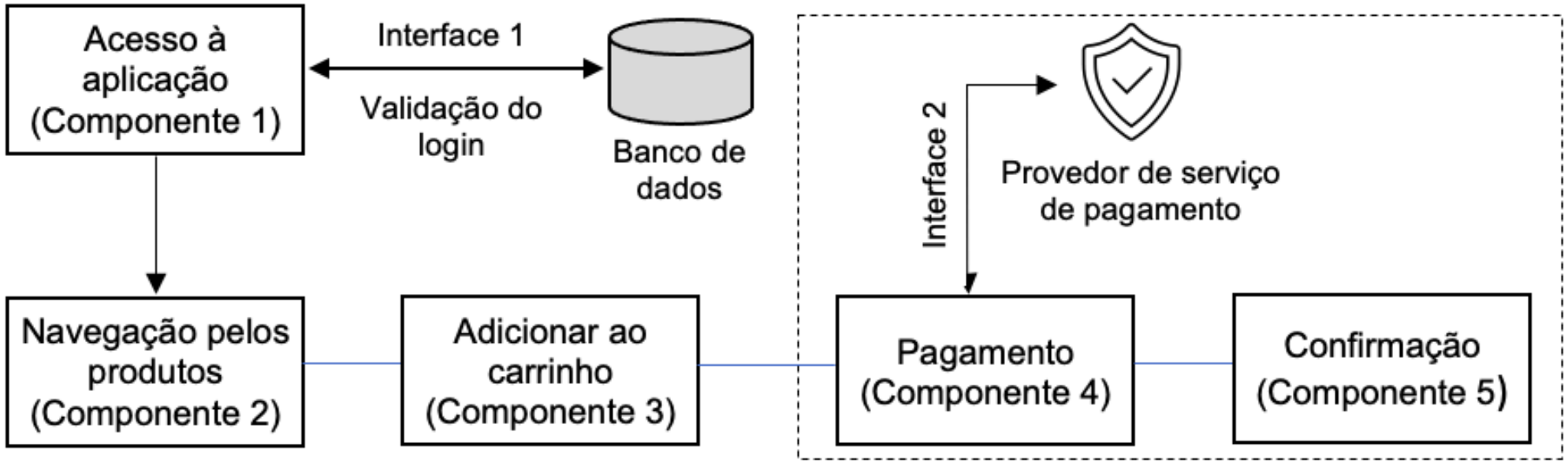
A ideia ilustrada na figura nos faz entender que o testador não conhece os detalhes internos do sistema e baseia seu julgamento apenas nos resultados obtidos a cada entrada fornecida. A esse respeito, Pressman e Maxim (2016) afirmam que um produto de software pode ser testado caso o responsável conheça a função específica para a qual aquele software foi projetado e, com esse conhecimento, estará em condições de realizar testes que demonstrem que cada uma das funções é operacional, embora o objetivo final do teste seja o deencontrar defeitos no produto. Os autores concluem que essa abordagem de testese vale de uma visão externa do produto e não se preocupa com a lógica internado software, a qual é opaca ao testador. Por esse motivo, a técnica funcional também é conhecida como teste de caixa preta.

O planejamento do teste funcional envolve dois passos principais: **identificaçãodas funções** que o software deve realizar (por meio da especificação dosrequisitos) e a **criação de casos de teste** capazes de checar se essas funções estão sendo executadas corretamente. Apesar da simplicidade da técnica e apesar de sua aplicação ser possível em todos os programas cujas funções são conhecidas, não podemos deixar de considerar uma dificuldade inerente: não se pode garantir que partes essenciais ou críticas do software serão executadas, mesmo com um bom conjunto de casos de teste.

Um teste funcional não deve ser aplicado simultaneamente em todas as funções do sistema e nem em apenas uma única ocasião. Em vez disso, ele deve examinar elementos específicos em ocasiões previamente conhecidas, característica que levou esse tipo de teste a ser dividido em subtipos. Alguns exemplos ajudarão a esclarecer essa circunstância: quando aplicado para verificar funções de um módulo, função ou classe do sistema, ele recebe a denominação de teste de unidade. Já o teste de integração é executado quando se deseja avaliar como asunidades funcionam em conjunto ou integradas e o teste de regressão é um subtipo do teste funcional, que visa garantir que uma alteração feita em uma função não tenha introduzido outros problemas no código (PRESSMAN; MAXIM,2016).

A menção ao quarto subtipo de teste funcional nos dará a oportunidade para o desenvolvimento de um exemplo prático. Quando testamos um componente do sistema de modo independente para verificar sua saída esperada, chamamos tal procedimento de teste de componentes. Geralmente, ele é executado paraverificar a funcionalidade e/ou usabilidade de componentes, mas não se restringe apenas a eles. Um componente pode ser qualquer coisa que receba entradas eforneça alguma saída, incluindo uma página web, telas e até mesmo um sistema dentro de um sistema maior.

Como aplicação prática desse teste, imaginemos um sistema de vendas pelainternet. Em uma análise mais minuciosa, seria possível identificar muitos componentes nessa aplicação, mas escolheremos apenas alguns, os quais estão representados na Figura 3.9.



O componente de login (identificado como componente 1) efetiva a validação de acesso do usuário ao sistema, usando, para isso, uma interface específica, que realiza a conexão com o banco de dados. Esse componente dá acesso à navegação pelos produtos, à inclusão de itens no carrinho, ao pagamento da compra e à sua confirmação. Para efetivação do pagamento, há uma interface que conecta o sistema em teste a um provedor de serviço de pagamentos. Considerando essecenário, vejamos o que deve ser testado em específico no componente 1 (login):

A interface de usuário nos itens de usabilidade e acessibilidade.

O carregamento da página, como forma de garantir bom desempenho da aplicação.

A suscetibilidade a ataques ao banco de dados por meio de elementos da interface de usuário.

A resposta da funcionalidade de login por meio do uso de credenciais falsas eválidas.

Se considerarmos o componente 3 (adição do produto ao carrinho), o testador deverá verificar o que ocorre, por exemplo, quando o usuário coloca um item, o qual supostamente deseja comprar, e fecha aplicação em seguida. Um cuidado especial também deverá ser tomado com a comunicação entre o componente 4 e o provedor de serviço de pagamento.

Pelas suas características, o teste de componente nos prepara para o teste quevem a seguir. Se a técnica de teste funcional se preocupa com as funções do sistema, qual seria, então, sua diferença para o teste de funcionalidade? Os testesde funcionalidades priorizam interações com o usuário e a navegação no sistemae são executados para verificar se um aplicativo de software funciona corretamente, de acordo com as especificações do projeto, no que se refere àentrada de dados, validação de dados, funções de menu e tudo o que está relacionado à interação do usuário com as interfaces. Além das verificações mencionadas, um teste de funcionalidades deve se preocupar também em averiguar se funções de “copia e cola” funcionam corretamente e se os ajustes de padrões regionais estão de acordo com a localidade em que o software está sendo executado.

No escopo da qualidade de software, não há apenas uma acepção relacionada à funcionalidade. Existem as funcionalidades do sistema, que são objetos de teste, e a funcionalidade entendida como um atributo fundamental da qualidade. Neste segundo caso, trata-se do grau com o que o software satisfaz as necessidades declaradas pelo cliente, conforme indicado pelos subatributos de adequabilidade, exatidão, interoperabilidade, conformidade e segurança (PRESSMAN; MAXIM,2016). Em outras palavras, a funcionalidade oferecida pelo sistema não pode ser confundida com o grau com que um programa atende a requisitos de adequação ao seu propósito, à sua exatidão e à facilidade com que opera com outros sistemas.

TÉCNICA DE TESTE ESTRUTURAL

Se voltarmos um pouco no texto e observarmos novamente a Figura 3.7,encontraremos o teste estrutural posicionado na mesma dimensão do teste funcional. Essa dimensão, que chamamos de “Como testar”, agrupam técnicas (ou maneiras) de se realizar testes. Os testes estruturais (também chamados de caixa branca) são assim conhecidos por serem baseados na arquitetura interna do programa. Contando com o código-fonte e com a estrutura do banco de dados, o testador poderá submeter o programa a uma ferramenta automatizada de teste. Vale aqui a menção de que um teste funcional também pode (e deve) ser realizado de forma automática.

Há várias ferramentas capazes de realizar testes estruturais, mas não usaremos nenhuma em especial para o nosso propósito de detalhar esse teste. Em vez disso,nós nos apoiaremos em uma possível forma de representação do código do programa para construir um método de teste. Vamos considerar que, ao analisar o código do programa, nossa ferramenta construa uma representação dele, conhecida como grafo. De acordo com Delamaro (2004), em um grafo, os nós equivalem a blocos indivisíveis de código, o que, em outras palavras, significa que não existe desvio de fluxo do programa para o meio do bloco e, uma vez que oprimeiro comando dele é executado, os demais comandos são executados sequencialmente. Outro elemento que integra um grafo são as arestas (ou arcos), eeles representam o fluxo existente entre os nós. Se considerarmos o código da aplicação que calcula o fatorial de um valor, exibido no Código 3.3, teremos que o trecho entre a linha 1 e a linha 8 obedece aos requisitos para se transformar em um nó do grafo.

```c
#include <stdio.h>
main()
{
    int i = 0;
    valor = 0;
    fatorial = 1;
    printf("Programa que calcula o fatorial de um valor informado pelo usuario\n");

    do {
        printf("\nInforme um valor entre 1 e 10: ");
        scanf("%d",&valor);
        } while ((valor<1) || (valor>10));
    for (i=1; i<=valor; i++)
        fatorial=fatorial*i;
    printf("\nO Fatorial de %d = %d", valor, fatorial);
    printf("\n");
}
```

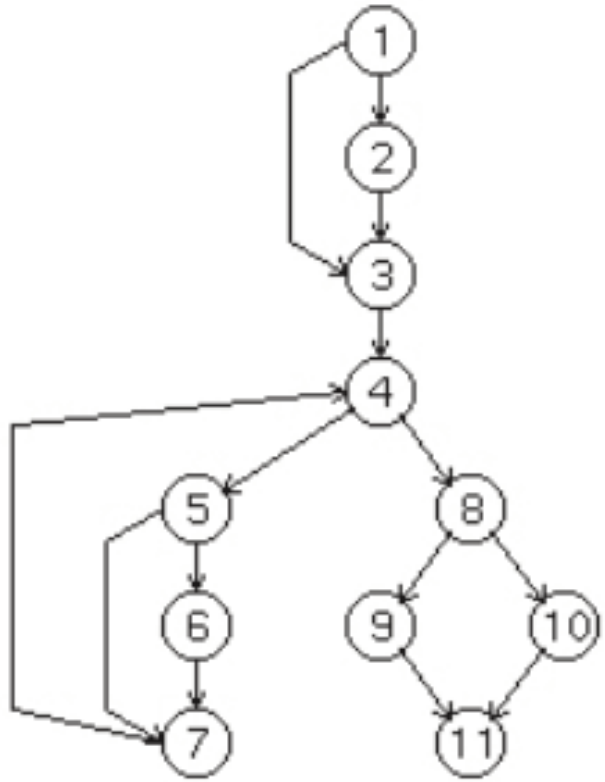
O programa que usaremos para ilustrar um teste estrutural será aquele cujocódigo pode ser visto no Código 3.4. Sua função é a de verificar a validade de umnome de identificador fornecido com base nas seguintes regras:

- Tamanho t do identificador entre 1 e 6 (1 <= t <= 6): condição válida.
- Tamanho t do identificador maior que 6 (t>6): condição inválida.
- Primeiro caractere c é uma letra: condição válida.
- Primeiro caractere c não é uma letra: condição inválida.
- O identificador possui apenas caracteres válidos: condição válida.
- O identificador possui um ou mais caracteres inválidos: condição inválida.

Anhanguera

```
/* 01 */ {
/* 01 */ char achar;
/* 01 */ int length, valid_id;
/* 01 */ lenght = 0;
/* 01 */ printf ("Indentificador");
/* 01 */ achar = fgetc(stdin);
/* 01 */ valid_id = valid_s(achar);
/* 01 */ if (valid_id)
/* 02 */ lenght = 1;
/* 03 */ achar = fgetc (stdin);
/* 04 */ while (achar != '\n')
/* 05 */ {
/* 05 */ if (!(valid_f(achar)))
/* 06 */ valid_id = 0;
/* 07 */ lenght++;
/* 07 */ achar = fgetc (stdin);
/* 07 */ }
/* 08 */ if (valid_id && (length >= 1) && (lenght < 6))
/* 09 */ printf ("Valido\n");
/* 10 */ else
/* 10 */ printf ("Invalido\n");
/* 11 */ }
```

Quando analisado pela ferramenta de teste, o Código 3.4 será transformado no grafo da Figura 3.10. Note que cada trecho identificado com um número no código é representado em um nó no grafo gerado.



Ao testador cabe selecionar casos de teste capazes de percorrer os nós, os arcos e os caminhos representados no grafo do programa. Apenas para fins de exemplificação, um caminho que pode ser percorrido pelo fluxo do programa é o formado pela sequência de nós 2, 3, 4, 5, 6 e 7. Casos de testes diferentes tenderão a exercitar sequências diferentes do grafo e, se considerarmos a existência de um conjunto infinito desses casos, teremos que as escolhas incorretas podem arruinaro teste. Ainda, se consideramos que os casos de teste são potencialmente infinitos, alguns critérios de cobertura do código devem ser previamente definidos, o que nos leva a uma reflexão.

TESTES DE APLICAÇÕES WEB E MÓVEIS

Houve um tempo em que códigos executáveis desempenhavam suas funções apenas em computadores de grande porte ou nos poucos computadores pessoais que existiam. Como os dispositivos móveis eram apenas um sonho, o planejamento e a execução dos testes eram atividades adequadas ao perfil dos programas da época e aos equipamentos que os executavam. Em nossos dias, noentanto, há dispositivos com uma ampla variedade de aplicações capazes de executar programas para as mais variadas finalidades e essa realidade exige algumas adequações nos procedimentos de testes aplicados nas plataformas mais comuns.

A fim de delimitar nosso estudo, trataremos aqui dos testes voltados a aplicações feitas para dispositivos móveis e dos testes executados em aplicações para a internet, nessa mesma ordem. Pressman e Maxim (2016) destacam o que consideram abordagens de teste especializadas para aplicativos móveis:

Teste de experiência do usuário: ninguém melhor do que um futuro usuário da aplicação móvel para expressar suas expectativas de usabilidade e acessibilidade, por exemplo. Se considerarmos uma aplicação móvel de vendas,os vendedores deverão ser incluídos no processo de desenvolvimento desde o seu início, para que o nível de experiência desejado por eles seja atingido.

Teste de compatibilidade do dispositivo: este item sugere que os testadoresdevem verificar se o aplicativo móvel funciona corretamente com o hardware escolhido para executar o aplicativo.

Teste de desempenho: neste quesito, os testadores devem verificar se indicadores de desempenho exclusivos dos dispositivos móveis correspondem à necessidade do cliente. Esses indicadores incluem tempo de download e autonomia de carga, por exemplo.

Teste de conectividade: aqui são testados a capacidade da aplicação em realizar conexões em redes de diferentes modalidades (ad hoc ou com infraestrutura, por exemplo) e o comportamento da aplicação em caso dequeda da conexão.

Essas considerações nos permitem criar um panorama dos testes de aplicaçõesmóveis, começando com sua definição: são atividades organizadas de teste,executadas com o objetivo de descobrir erros em aspectos fundamentais de seufuncionamento em um dispositivo móvel. Esses testes são realizados pelos profissionais técnicos, além dos usuários, do cliente e do gerente do projeto. Umteste típico começa por verificar as funções visíveis da aplicação para, então, voltar sua atenção para aspectos de infraestrutura e tecnologia. Normalmente, as etapas desse teste incluem testes de conteúdo, de interface, de navegação, de componente, de configuração, de desempenho e de segurança.

Como qualquer teste, temos aqui também a necessidade de apurar se o processo foi corretamente desempenhado. Como não é possível aplicar absolutamente todas as possibilidades existentes de entradas, nem apurar todas as situações quepodem ocorrer durante o uso da aplicação, novamente os critérios de paradaserão o parâmetro para o fim dos testes.

Outro elemento que demanda providências específicas durante o processo deteste são as aplicações web. Da mesma forma que em qualquer outro teste, aqui também estamos diante da missão de executar um sistema para encontrar e corrigir defeitos. No entanto, as aplicações web podem operar em conjunto com diferentes sistemas operacionais, navegadores, protocolos de comunicação e plataformas de hardware, o que pode representar dificuldades adicionais na buscapor defeitos. Com essas especificidades, os seguintes elementos de qualidade devem ser testados (PRESSMAN; MAXIM, 2016):

Conteúdo da aplicação: deve ser avaliado em dois níveis:
-Sintático: neste nível devem ser avaliadas a ortografia, a pontuação e agramática do conteúdo textual da aplicação web.
-Semântico: aqui devem ser avaliadas a exatidão, a consistência e a ausênciade ambiguidade da informação.
Funcionalidades da aplicação: são testadas para fins de descoberta deproblemas que colocam a ferramenta em situação de não conformidade comos requisitos do cliente.
Estrutura da aplicação: a facilidade com que a estrutura da aplicação podecrescer e, ainda assim, passar por manutenções é testada neste elemento.
Usabilidade: testes são feitos para que vários perfis de usuários possam seadaptar às características da interface da aplicação.
Navegabilidade: atenção especial é dedicada a links inativos e incorretos nesteelemento.

Adicionalmente, desempenho, interoperabilidade e segurança também recebematenção especial em um teste de aplicação web.

TESTES DE APLICAÇÕES ORIENTADAS A OBJETOS

Seguindo nosso caminho pelos testes feitos em aplicações ou paradigmas específicos, chegamos até as aplicações Orientadas a Objetos (OO). Por causa dascaracterísticas próprias dos programas O O, os testes aqui aplicados devemconsiderar a existência de subsistemas em camadas que encapsulam outras classes. De acordo com Pressman e Maxim (2016), é preciso testar um sistema O O em vários níveis diferentes, de modo que erros eventualmente ocorridos durante as interações entre as classes sejam descobertos. Ainda de acordo com os autores,três providências básicas são necessárias para se testar adequadamente umsistema OO:

A definição do teste deve ser ampliada para incluir as técnicas da descoberta de erros aplicadas à análise orientada a objetos e aos modelos do projeto.

A estratégia para teste de unidade e de integração deve ser alterada significativamente.

O conjunto de casos de teste devem levar em conta as características especiais do paradigma de Orientação a Objetos.

Uma análise mais minuciosa da primeira providência pode nos levar a umaaparente inconsistência: como é possível que aspectos da análise orientada aobjetos e modelos do projeto sejam testados? De fato, no sentido convencional, não podem. Ocorre que a criação de uma aplicação OO começa com a criação demodelos de análise e de projeto, os quais começam com representações quaseinformais dos requisitos e se tornam modelos detalhados de classes e de suasrelações conforme o desenvolvimento avança. Por isso, em cada estágio da sua evolução, esses modelos devem ser revisados para que erros sejam descobertos logo em fases iniciais do desenvolvimento (PRESSMAN; MAXIM, 2016).

Entre os vários benefícios trazidos pela adoção da Orientação a Objetos para odesenvolvimento de software, está o fato de que esse paradigma reduz a necessidade de testes, segundo Schach (2009). O autor complementa que a reutilização de código via herança é um dos fatores que torna essa característica ainda mais forte: em tese, uma vez que a classe mãe tenha sido testada, as classes provenientes dela não precisam ser novamente testadas. Quando os desenvolvedores inserem novos métodos na subclasse de uma classe já testada, é natural que eles precisem ser testados. No entanto, métodos herdados não precisam de teste adicional.

Por mais lógicas e confiáveis que nos pareçam, essas afirmações precisam seranalisadas com um certo cuidado. Para começar, a classe é um tipo de da doabstrato e o objeto é a instância de uma classe. Esse fato nos induz ao raciocínio deque uma classe não tem uma realização concreta e que, portanto, não é possível aplicar testes baseados em execução (ou seja, aqueles que temos abordado nesta unidade) diretamente nela. Outra característica da Orientação a Objetos que tem efeitos em um teste é quantidade normalmente elevada de métodos em um objeto que, ao invés de retornarem um valor para o chamador, simplesmente alteram o estado do objeto ao modificarem seus atributos. Segundo Schach (2009), a dificuldade, nesse caso, é a de testar se a alteração de estado foi realizada corretamente.

Tomemos a seguinte situação como exemplo: uma aplicação bancária possui ummétodo chamado deposito, cujo efeito é o de aumentar o valor da variável de estado chamada saldo DaConta. No entanto, como consequência do ocultamento de informações, a única maneira viável de se testar se o método deposito tem uma execução correta é pela invocação de um outro método, chamado determinar Saldo, tanto antes como depois da chamada do método deposito, a fim de verificar como a mudança do saldo se dá.

Outro aspecto a ser analisado é a necessidade de aplicação de um novo teste em métodos herdados. Observe a hierarquia de classes ilustrada no trecho de código contido no Código 3.5. Na primeira classe (ClasseArvoreComRaiz), são definidos dois métodos: o que exibe o conteúdo de um nó e a rotina de impressão, utilizada pelo método exibe ConteudoNo.

```
1   class ClasseArvoreComRaiz {
2           void exibeConteudoNo (No a);
3           void rotinaImpressao (No b);
4           // o método exibeConteudoNo usa o método rotinaImpressao
5   }
6
7   class ClasseArvoreBinaria extends ClasseArvoreComRaiz {
8           void exibeConteudoNo (No a);
9           // o método exibeConteudoNo definido aqui usa
10          // o método rotinaImpressao herdado de ClasseArvoreComRaiz
11  }
12
13  class ClasseArvoreBinariaBalanceada extends ClasseArvoreBinaria {
14          void rotinaImpressao (No b);
15          // o método exibeConteudoNo (herdado de ClasseArvoreBinaria)
16          // usa essa versão local de rotinaImpressao dentro da classe
17          // ClasseArvoreBinariaBalanceada
18  }
```

Observe que a subclasse ClasseArvoreBinaria herda o método rotinaImpressaoda classe mãe chamada ClasseArvoreComRaiz. Além disso, o método exibe ConteudoNo é novamente definido nessa subclasse, o que anula o mesmo método definido na classe ClasseArvoreBinaria. Já a subclasse ClasseArvoreBinariaBalanceada herda o método exibeConteudoNo da superclasse ClasseArvoreBinaria. No entanto, nela é definido um novo método, o rotina Impressao, que anula aquele definido em ClasseArvoreComRaiz. Quando o método exibe ConteudoNo usa o método rotina Impressao no contexto de ClasseArvoreBinariaBalanceada , as regras do Java especificam que a versãolocal de rotina Impressao deve ser usada e, em consequência disso, o código real do método rotina Impressao, executado quando exibe ConteudoNo é chamado no contexto da instanciação da classe ClasseArvoreBinaria, é diferente daqueleexecutado quando esse mesmo método é executado na instanciação de ClasseArvoreBinariaBalanceada. Esse fenômeno ocorre normalmente apesar de o método exibe ConteudoNo ser herdado sem modificação, o que a carreta a necessidade do novo teste (SCHACH, 2009).

Também considerando características próprias de sistemas Orientados a Objetos, Masiero et al. (2015) dividem o procedimento de testes em três fases:

Teste de unidade: os autores consideram que as menores unidades a seremtestadas em um programa OO são os métodos. Por isso, indicam que o teste de unidade consiste no teste de cada método de forma isolada, o que também échamado de teste intra-método.

Teste de módulo: trata-se do teste de um conjunto de unidades que interagementre si por meio de chamadas. Essa fase pode ainda ser assim dividida:

Inter-método: aplicação de teste nos métodos públicos de uma classe, em conjunto com os outros métodos da mesma classe.

Intra-classe: consiste em testar as interações entre os métodos públicos de uma classe quando chamados em diferentes sequências.

Inter-classe: consiste em testar as interações entre classes diferentes.

Teste de sistema: consiste em testar a integração entre todos os módulos, o que equivale a um sistema completo. Para esta fase geralmente é usado o teste funcional.

Ao executar o teste da caixa preta, o testador não levará em consideração ocomportamento interno do sistema. Essa técnica leva esse nome por considerar o processamento do sistema de forma desconhecida e, por isso, apenas as entrada se as saídas do sistema serão avaliadas.

Um teste de caixa branca não poderá ser executado quando o testador não tiver a posse do:

a. Registro formal do sistema.

b. Grafo do sistema.

c. Código-fonte do sistema.

d. Conjunto de funcionalidades do sistema.

e. Arestas e caminhos do sistema.

Uma questão clássica surge todas as vezes que se discute teste de software:“Quando podemos dizer que terminamos os testes – como podemos saber que já testamos o suficiente?” Há algumas respostas pragmáticas e algumas tentativasempíricas para essa questão.

Considerando o conceito de teste e as características de seu procedimento, assinale a alternativa que contém a sentença indicativa do momento em que o testador deve interromper um procedimento de teste.

a. Quando ele tiver certeza de que o programa está completamente livre de defeitos.

b. Quando o grafo do programa for totalmente coberto pelos casos de teste.

c. Quando todos os casos de teste existentes tiverem sido usados no procedimento.

d. Quando os critérios estabelecidos para o término tiverem sido atingidos.

e. Quando os critérios de total ausência de defeitos tiverem sido todos cumpridos.