# Fundamentos Engenharia de Software

UNIDADE 05

## Testes de software

*Esta Unidade apresenta os conceitos fundamentais relacionados à atividade de testes de software, que é fundamental para garantir a qualidade do software. A Unidade tem início com uma breve introdução a respeito da importância de testes, seus conceitos e desafios. Posteriormente, os testes são apresentados sob perspectivas diferentes: níveis e tipos. Na perspectiva de níveis, será visto que teste é uma atividade a ser realizada em todos os níveis de desenvolvimento, desde o menor componente implementado (método, função etc.) até a entrega de um sistema com inúmeros componentes integrados para o usuário validar. Na perspectiva de tipos, será abordado que tanto requisitos funcionais quanto não*

*funcionais devem ser testados. Por fim, a Unidade é finalizada com a apresentação de duas técnicas mais utilizadas para o teste de funcionalidades do sistema.*

## Introdução ao teste de *software*

Antes de falarmos diretamente sobre a atividade de testes de *software*, é importante abordar a necessidade da garantia da qualidade de produtos ou serviços.

A qualidade é garantida por meio de um conjunto sistemático e planejado de atividades, necessárias para proporcionar a confiança adequada de que produtos e serviços estarão em conformidade com requisitos especificados e atenderão às necessidades do usuário.

O teste é uma atividade fundamental para garantir a qualidade de qualquer produto ou serviço. Isso se aplica a produtos e serviços de qualquer natureza, como para um novo carro, brinquedo, serviço de hotel, alimento, serviço de beleza etc. Portanto, ao desenvolver um novo produto de *software*, é fundamental executar atividades que garantam a qualidade deste produto, ou seja, que ele atenda plenamente às necessidades do usuário.

Uma das atividades essenciais para a garantia da qualidade de *software* é o **teste de *software*.**

Quando falamos em testes de *software*, há quatro questões fundamentais que precisamos ter em mente:

---

1) Por que testar?

São vários os motivos para se testar um *software*, dentre eles: garantir a qualidade *software*, reduzir o retrabalho e manter uma boa reputação da empresa.

2) O que testar?

Requisitos funcionais e não funcionais. Da menor unidade de *software* (um método, uma função etc.) ao produto final (integração entre componentes, sistemas etc.).

3) Quando testar?

Ao longo de todo o ciclo de desenvolvimento do *software* (desde o levantamento dos requisitos à entrega das funcionalidades).

4) Quem testa?

Todos os participantes de um projeto de desenvolvimento de *software* (desenvolvedores, testers, usuários etc.), em algum momento, precisam estar envolvidos à atividade de teste de *software* (quanto mais pessoas de perfis diferentes testarem, melhor).

## Conceitos fundamentais

Antes de avançarmos com o entendimento da atividade de teste, é importante definirmos alguns termos, que podem ser considerados sinônimos, porém, têm significados distintos.

O teste tem como objetivo analisar o comportamento do *software* por meio de sua execução, com a intenção específica de encontrar **defeitos**, **erros** ou **falhas**, antes da entrega ao usuário final.

+ **Defeito**

  É uma linha de código, bloco ou conjunto de dados incorretos que provocam um erro observado.

+ **Erro**

É uma diferença detectada entre o resultado de uma computação e o resultado correto ou esperado.

+ **Falha**

É um não funcionamento do *software*, possivelmente provocado por um defeito, mas com outras causas possíveis. A falha é a observação de que o *software* não funciona adequadamente.

Existem falhas que são provocadas por um defeito no *software*, mas outras que podem ser provocadas por problemas tecnológicos (segurança, conexão, leitura de dados etc.). Portanto, nem todas as falhas são provocadas por defeitos.

Assim, concluímos que o teste de *software* se ocupa principalmente em identificar falhas provocadas por defeitos para que os defeitos sejam corrigidos.

## Desafios do teste de *software*

A tarefa de testar *software*, porém, não é simples. Algumas vezes, pode ser mais fácil produzir o *software* do que elaborar bons casos de testes. Desta forma, muita sistematização e critérios são necessários para que a atividade de testes deixe de ser uma tarefa aleatória e ingênua para se tornar uma atividade de engenharia com resultados efetivos.

A maioria dos sistemas de médio e grande porte pode conter defeitos ocultos, pois a quantidade de testes necessária para garantir que estejam livres de defeitos pode ser impraticável. Assim, a área de testes de *software* ocupa-se em definir conjuntos finitos e possíveis de testes que mesmo não garantindo que o *software* esteja livre de defeitos, consigam localizar os mais prováveis permitindo a sua correção.

Diante de todos os processos, modelos e ferramentas que a engenharia do *software* disponibiliza para obter qualidade dos produtos desenvolvidos, ainda assim, pode ocorrer incidência de erros e falhas. As técnicas de teste têm se mostrado importantes recursos para minimizá-los.

Para exemplificar o quanto a tarefa de testar um *software* pode não ser tão simples, vamos realizar um exercício.

## REFLEXÃO

**Analise quantos casos de testes são necessários para garantir que o seguinte programa tenha sido construído com sucesso.**

**Cenário**

- O programa lê três valores inteiros.
- Estes três valores devem representar o comprimento dos lados de um triângulo.
- O programa apresenta uma mensagem, dizendo se o triângulo é escaleno, isósceles ou equilátero.

**Regras**

- O comprimento de um lado do triângulo é sempre menor do que a soma dos outros dois lados.
- Triangulo equilátero: todos os lados iguais.
- Triângulo isósceles: dois lados iguais.
- Triângulo escaleno: todos os lados diferentes.

**Resultado**

Você tem um caso de que avalia:

1. Se é um triângulo? Testes com 2, 5, 10 não são um triângulo escaleno, na verdade, não formam um triângulo.
2. Os testes do caso 1 consideram a permuta dos três valores? (ex.: 2, 5, 10; 2, 10, 5; 10; 2, 5)
3. Um triângulo equilátero?
4. Um triângulo isóscele? Tentando três permutas de dois lados iguais? (ex.: 3, 3, 4; 3, 4, 3; 4, 3, 3)
5. Um triângulo equilátero?
6. Um dos lados tem valor zero?
7. Um dos lados tem valor negativo?

8. A entrada válida para os três valores inteiros?
9. Etc.

Perceba a quantidade de casos de testes que pode ser necessária para obter 100% de cobertura da execução do código, de forma a identificar erros e, consequentemente, a correção de defeitos.

## Níveis de testes

A atividade de testes deve ser realizada em fases distintas do desenvolvimento do *software* e, portanto, pode ser aplicada em níveis distintos. A seguir, conheça alguns deles!

### Teste de unidade

Os testes de unidade são os mais básicos e consistem em verificar se um componente individual foi implementado corretamente. Estes componentes podem ser um método, uma classe ou um pacote de funções de tamanho pequeno a moderado. Geralmente, esse teste é realizado pelo próprio programador, inclusive na técnica de desenvolvimento orientado a testes (TDD), recomenda-se que o programador antes de desenvolver uma unidade de software, desenvolva os casos de testes com objetivo de automatização.

Para o teste de unidade, podem ser aplicadas as técnicas de testes funcionais (caixa-preta) e de teste estruturais (caixa-branca). No teste funcional, garante-se que as saídas estão de acordo com as especificações. Já no teste estrutural, procura-se garantir que todos os comandos e as decisões dos componentes implementados tenham sido executados com objetivo de encontrar falhas. Estas técnicas serão detalhadas nas próximas Unidades de Estudo desta disciplina.

### Teste de integração

Os testes de integração têm o objetivo de identificar falhas associadas às interfaces entre os componentes (método, classe, pacote etc.), quando esses são integrados.

Um grande desafio dos testes de integração é o fato de que parte dos componentes individuais que precisam ser integrados, muitas vezes não foram codificados ou testados. Nestes casos, as interfaces faltantes precisam ser suprimidas por implementações incompletas ou simplistas desenvolvidas de forma provisória para possibilitar o teste de integração de parte dos componentes.

Para contornar estas situações, pode-se aplicar estratégias distintas, com pontos favoráveis e desfavoráveis.

### + Integração *big bag*

Consiste em desenvolver todos os componentes individuais e depois integrar tudo. É uma técnica mais alinhada com o ciclo de vida cascata, que tem como vantagem não precisar de implementações provisórias; porém, não se consegue adotar um método mais incremental para avançar no desenvolvimento até que todas as implementações dependentes sejam realizadas.

### + Integração *bottom-up*

Consiste em integrar os componentes de mais baixo nível e ir integrando com componentes de nível imediatamente superior. Assim, um componente superior só é integrado e testado quando todos os componentes do qual ele depende foram integrados e testados. Desta forma, não há a necessidade de desenvolvimentos provisórios, porém, os testes de mais alto nível do sistema serão testados tardiamente.

### + Integração *top-down*

Consiste em integrar inicialmente os módulos de nível mais alto, deixando os mais básicos para o fim. A vantagem é poder verificar inicialmente os comportamentos mais importantes do

sistema, no entanto, muito desenvolvimentos provisórios são necessários para resolver as dependências.

+ **Integração incremental**

  Consiste em integrar componentes à medida em que vão ficando prontos, independentemente do nível. O planejamento do desenvolvimento dos componentes é que vai determinar se a integração ocorrerá com características *top-down*, *bottow-up* ou outra estratégia mais conveniente.

As estratégias descritas têm como objetivo realizar testes de integração de forma mais organizada e assertiva. Embora os componentes, depois do teste de unidade, possam funcionar corretamente de forma isolada, o teste de integração é necessário, pois quando colocados juntos, várias situações inesperadas podem acontecer. E quando as integrações ocorrem de forma desordenada ou ainda com implementações provisórias para possibilitar a integração, diversas falhas podem ocorrer mesmo após os testes de integração.

## Teste de sistema

Depois de realizar todas as integrações, teremos o sistema completo. Podemos, assim, simular o ambiente de uso final e realizar testes que apontarão falhas em aspectos gerais do sistema.

O teste de sistema visa a verificar se a versão corrente do sistema permite executar um fluxo de um processo ou de um caso de uso completo sobre o ponto de vista do usuário. Neste nível de teste, se utiliza a técnica funcional, em que não se avalia a estrutura interna da implementação (código), mas o comportamento do sistema em relação a sua especificação.

Exemplos de fluxos que são testados neste nível:

- Comprar um livro em uma livraria virtual.

- Realizar a transferência bancária por meio do internet banking.
- Registrar matrícula de um estudante em uma disciplina.
- Pesquisar imóveis à venda em um portal.

Em suma, temos a seguinte sequência dos testes:

- Primeiro, são testados os componentes individualmente pelo teste de unidade, a fim de verificar falhas específicas nas funções de cada componente.
- Em seguida, são agrupados os componentes para verificar erros na interface entre eles e medir seus níveis de desempenho e confiabilidade. Isso é feito por grupos, não considerando o sistema completo.
- Por fim, após testar todos os grupos de integração, é realizado o teste de sistema.

## Teste de aceitação

Os testes de aceitação, também conhecidos como homologação, são semelhantes ao teste de sistema; no entanto, deve ser realizado pelo usuário final ou cliente, e não pela equipe de desenvolvimento.

Neste nível de teste, o objetivo principal é validar se os requisitos do *software* foram desenvolvidos de acordo com o esperado. Neste momento, o usuário aprova a versão ou solicita modificações.

Apesar de todos os testes internos (unidade, integração e sistema), é possível que ainda existam falhas no sistema tanto de codificação quanto de entendimento dos requisitos. O desejável é que neste nível dos testes, o usuário final receba uma aplicação livre de falhas de codificação e que ele possa se concentrar na validação dos requisitos implementados.

## Teste de regressão

O teste de regressão é executado sempre em um sistema em operação como uma manutenção. Ou seja, ele é a reaplicação dos testes nas novas versões do sistema para verificar possíveis falhas após a correção de defeitos.

A correção de um defeito ou modificação de alguma parte pode gerar novos defeitos. Neste caso, devem ser executados novamente todos os testes. Se aplicar novos testes na nova versão e ela não passar, considera-se que o sistema regrediu, por isso o nome regressão.

Por se tratar por uma atividade de alto custo, pois pode ser executada diversas vezes ao longo do ciclo de vida do *software*, é importante que os testes sejam automatizados.

## Tipos de testes

Os tipos de testes estão diretamente relacionados ao tipo de requisito que está sendo testado e, portanto, classificam-se em duas categorias principais: não funcionais e funcionais.

A seguir, confira o detalhamento dos tipos de testes não funcionais (interface, performance, segurança, recuperação de falhas etc.) e, posteriormente, o teste de funcionalidade.

### Teste de interface com o usuário

O teste de interface tem como objetivo verificar se as interfaces, por meio das quais as funcionalidades são executadas, são eficazes e eficientes. Mesmo que a funcionalidade seja aprovada no teste do sistema, ou seja, as saídas são as esperadas, aspectos de usabilidade não atendidos podem provocar modificações na aplicação.

Exemplos do que pode ser verificado em testes de interface:

- Acessibilidade (deficientes visuais, auditivos etc.).
- Ergonomia (uso do mouse que exige muitos movimentos de braço).
- Mensagens do sistema (apropriação do texto e localização das mensagens).
- Sequência de execução das funcionalidades (otimização da sequência das funcionalidades que facilite a execução de caminhos principais e alternativos).

### Teste de performance (carga, estresse e resistência)

Este tipo de teste tem o objetivo de executar as funcionalidades e mensurar o seu tempo, avaliando se está dentro dos padrões definidos. Os testes de performance também têm o objetivo de avaliar a estabilidade de um sistema. Eles podem ser classificados em três tipos:

### + Teste de carga

Tem como objetivo realizar cargas de dados e simular transações que normalmente seriam esperadas pela aplicação em situações típicas. Com o resultado destas simulações, pode-se antecipar problemas, como identificar gargalos, e preventivamente buscar soluções para eles.

### + Teste de estresse

É um caso extremo do teste de carga, em que se procura colocar a aplicação em situações de limite máximo e verificar como ele se comporta. Exemplo deste tipo de situação: datas específicas em que há um grande volume de acesso (*black friday*, Natal, promoções relâmpagos etc.).

### + Teste de resistência

Verifica como o sistema se comporta após um período maior de exposição a situações atípicas. Normalmente, no teste de resistência, é verificado o uso de memória após um período prolongado de uma carga expressiva de trabalho.

## Teste de segurança

São diversos os aspectos de segurança que precisam ser verificados em um sistema. A seguir, confira seis tipos de teste de segurança que devem ser testados:

- Integridade: garantir que a informação recebida é correta e completa.
- Autenticação: garantir que o usuário é quem diz que é.
- Autorização: garantir que apenas pessoas autorizadas tenham acesso à informação.
- Confidencialidade: garantir que pessoas não autorizadas não tenham acesso à informação.
- Disponibilidade: garantir que pessoas autorizadas consigam obter a informação.
- Não repúdio: garantir que emissor ou receptor não possam alegar que não enviaram ou receberam uma mensagem.

Alguns tipos de testes podem se complementam mutuamente (ex.: confidencialidade e disponibilidade).

## Teste de recuperação de falhas

O teste de recuperação de falhas consiste em verificar a capacidade de o sistema se tornar operacional novamente após a ocorrência de uma falha. Situações em que se aplica:

- Queda de energia no cliente ou servidor.
- Rompimento de fibra ótica.
- Discos corrompidos.

## Teste de funcionalidade

Conforme o nome sugere, é o tipo de teste que verifica e valida as funções do sistema, ou seja, os processos que podem ser efetivamente realizados por meio do fluxo entrada, processamento e saída.

Exemplos de funções:

- Incluir um cliente.

- Validar um CPF.
- Pesquisar um livro em uma livraria.
- Transferir valores entre contas financeiras.

## Técnicas de testes de funcionalidade

Até este momento, conhecemos os testes sobre duas perspectivas – níveis e tipos.

- Os níveis demonstram que a atividade de teste precisa ser realizada desde a implementação das menores unidade de *software* (métodos, funções etc.) até a operacionalização completa do sistema incluindo a aceitação do cliente.
- Os tipos demonstram que os testes devem ser realizados tanto para requisitos não funcionais quanto para requisitos funcionais.

O teste de funcionalidade tem uma particularidade especial em relação aos requisitos não funcionais: está mais focado no código implementado. Para os testes de funcionalidade, há duas técnicas mais conhecidas: teste funcional e teste estrutural.

### Teste funcional (caixa preta)

O teste funcional, também conhecido como caixa-preta (Figura 1), é executado sobre as entradas e as saídas do programa, sem que se tenha necessariamente conhecimento do seu código-fonte.

Figura 1: Teste funcional (caixa preta)

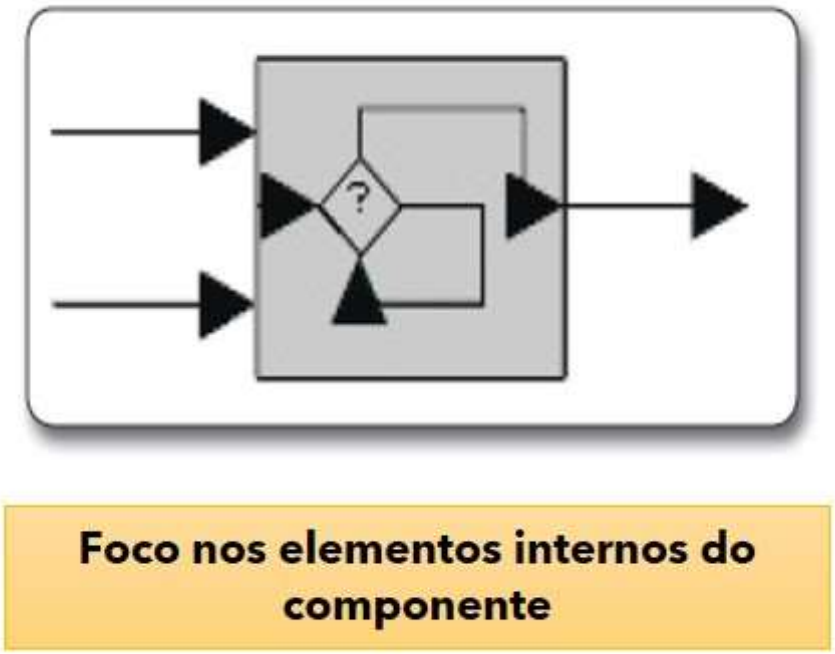

A figura representa o conceito de teste funcional, em que o foco são as entradas e saídas, sem interesse em como internamente o código está implementado. Fonte: O autor (2021).

O componente de *software* a ser testado é abordado como se fosse uma caixa-preta, ou seja, não se considera o comportamento interno dele. Os dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado previamente conhecido.

## Teste estrutural (caixa-branca)

O teste estrutural, também conhecido como caixa-branca (Figura 2), é executado com conhecimento do código implementado. Ele avalia o comportamento interno do componente de *software*, atuando diretamente no código-fonte do componente de *software*.

Figura 2: Teste estrutural (caixa-branca)




A figura representa o conceito de teste estrutural, em que o foco são os elementos internos (estrutura de seleção, repetição etc.) do componente. Fonte: O autor (2021).

Este teste pode ajudar a responder à pergunta-chave:

---

**Quais casos de testes adicionais são necessários para revelar falhas que não são aparentes, usando somente testes de caixa-preta?**

A diferença principal entre testes caixa-branca e caixa-preta está na fonte utilizada para estabelecer os casos de teste.

Nas próximas Unidades de Estudo, será apresentado detalhadamente o funcionamento das técnicas de teste funcional e estrutural.

A videoaula dessa unidade apresenta um exemplo de especificação de um programa de computador e os casos de testes necessários para identificar possíveis defeitos nele. É possível observar a importância de utilizar técnicas sistemáticas para a elaboração de casos de testes. A apresentação também ressalta que é possível combinar técnicas funcionais (caixa-branca) e técnicas estruturais (caixa-preta) para obter casos de teste que garantam a qualidade do *software*.


## Conclusão

Nesta Unidade, foi introduzido o tema “testes de *software*”. Por meio das seções, discutimos a importância, os conceitos e os desafios da atividade de testes, os quais devem ser executados em todas as fases do desenvolvimento de *software* e ser aplicados para testar tanto requisitos funcionais quanto não funcionais. Para cada tipo de teste, aspectos distintos devem ser avaliados. Concluímos a Unidade com a apresentação dos dois tipos de técnicas focados no teste de funcionalidade: teste funcional (caixa-preta) e teste estrutural (caixa-branca). As Unidades seguintes apresentarão detalhadamente a aplicação de ambas as técnicas.

## Referências bibliográficas

BRAGA, P. H. C. **Testes de software**. São Paulo: Pearson Education do Brasil, 2016. Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/150962/epub/0. Acesso em: 26 de abril de 2021.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788580555349/. Acesso em: 26 de abril de 2021.

WAZLAWICK, R. S. **Engenharia de software**: conceitos e práticas. 2. ed. Rio de Janeiro: Elsevier, 2019.