



# Fundamentos de Testes de Software

## Núcleo de Educação a Distância

[www.unigranrio.com.br](http://www.unigranrio.com.br)

Rua Prof. José de Souza Herdy, 1.160

25 de Agosto – Duque de Caxias - RJ



### Reitor

Arody Cordeiro Herdy

### Pró-Reitor de Administração Acadêmica

Carlos de Oliveira Varella

### Pró-Reitor de Pesquisa e Pós-Graduação

Emilio Antonio Francischetti

### Pró-Reitora de Ensino de Graduação

Hulda Cordeiro Herdy Ramim

### Pró-Reitora de Pós-Graduação *Lato Sensu* e Extensão

Nara Pires



**Produção:** Fábrica de Soluções Unigranrio

**Desenvolvimento do material:** André Luiz Braga

Copyright © 2018, Unigranrio

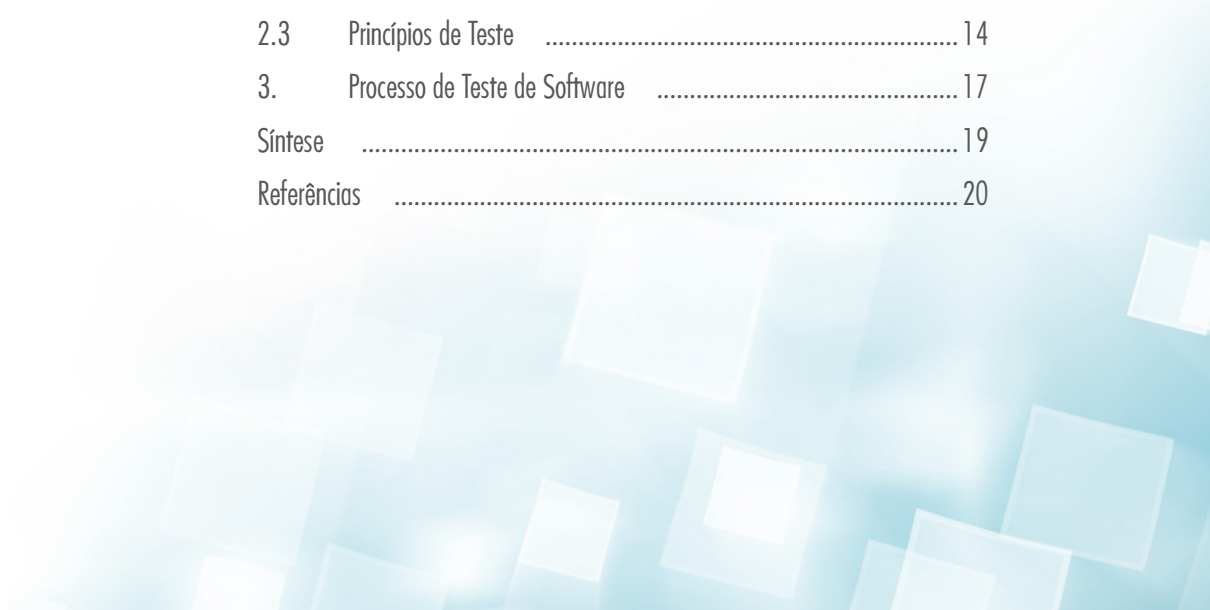
Nenhuma parte deste material poderá ser reproduzida, transmitida e gravada, por qualquer meio eletrônico, mecânico, por fotocópia e outros, sem a prévia autorização, por escrito, da Unigranrio.



# Sumário

## Fundamentos de Testes de Software

Objetivos .....	04
Introdução .....	05
1. Qualidade do Produto de Software .....	06
1.1 Visão Geral .....	06
1.2 Qualidade x Testes .....	08
1.3 Qualidade de Software .....	09
2. Verificação e Validação de Software .....	11
2.1 Verificação e Validação .....	11
2.2 Defeitos e Falhas .....	13
2.3 Princípios de Teste .....	14
3. Processo de Teste de Software .....	17
Síntese .....	19
Referências .....	20





## Objetivos

Ao final desta unidade de aprendizagem, você será capaz de:

- Entender a necessidade de se adotar um processo de testes para garantir a qualidade do desenvolvimento de software;
- Conhecer as normas e padrões de qualidade de software;
- Entender o conceito do que se deseja ao testar um software;
- Saber o significado dos termos mais importantes no teste de software;
- Conhecer os princípios do teste de software;
- Saber as tarefas a serem executadas num processo de teste de software.





# Introdução

Existe uma visão comum de que o teste e a garantia de qualidade de um software consistem em executar passos, inserir valores, registrar resultados e enviar os defeitos de volta aos desenvolvedores para que sejam reparados. A falta de conhecimento dos fundamentos que estão por trás do processo de testes e do significado da qualidade tem resultado em produtos de péssima eficiência e reputação. Muitos de nós já usamos softwares extremamente populares e internacionalmente conhecidos e nos deparamos com falhas que levaram a perdas de dias ou meses de trabalho.

Podemos, inicialmente, alegar que tais softwares foram mal projetados e desenvolvidos, mas, mesmo assim, isso seria apenas uma pequena parte do problema. Por melhor que seja o processo de desenvolvimento, sem um sólido processo de testes, não há como garantir o seu funcionamento.

Nesta unidade de aprendizagem, entenderemos melhor toda essa área de teste e qualidade de software e como ela se apresenta. Veremos, também, os conceitos básicos que precisam ser entendidos e que fazem parte do nosso racional ao planejarmos, projetarmos e implementarmos testes de software.



# 1. Qualidade do Produto de Software


## 1.1 Visão Geral

A área de testes e qualidade de software é um tema que gera grande dificuldade, não só de ensino como de aprendizado. Qualidade e teste de produtos é uma área bastante estudada e desenvolvida, mas em relação à sua aplicação em software, temos ainda muitas deficiências na forma como ela é apresentada.

Uma rápida pesquisa na literatura já nos traz inúmeras abordagens diferentes, além de vários tópicos postos de forma independente. Encontramos informação sobre o que é e quais são os processos de teste, e em outro tópico, vemos a definição da qualidade aplicada aos testes, muitas vezes com conceitos que se interseccionam. Metodologias, características, técnicas, categorias, normas, boas práticas e conceitos estão frequentemente postos em sequências e relações diversas, tornando o estudo difícil e complexo. Podemos olhar dois livros diferentes sobre o assunto e termos a sensação de que estão falando de coisas completamente diferentes. Um entendimento claro de toda a área de qualidade e testes poderia ser resumido pelos seguintes pontos:

- **Conceitos Básicos** – A maioria das referências que vamos encontrar no material introdutório sobre testes e qualidade de software apresenta os fundamentos e define os termos básicos para que se possa entender como executar os testes de software.
- **Objetivos** – O que desejamos obter por meio dos testes de software. Por que executá-los e o que esperar.
- **Termos e elementos** – Entendimento preciso dos termos e elementos usados nos testes de software e o seu significado. Por exemplo, no linguajar leigo, “falha” e “erro” podem ser sinônimos, mas na área de testes de software, eles têm significados distintos.



- 
- **Características** – Define os atributos que os testes irão ter, como escopo, entradas, critério de escolha de conjuntos de teste etc.
  - **Princípios** – Atitudes e posturas que devem ser tomadas ou esperadas nos testes, como o entendimento de que um teste pode apenas revelar a existência de defeitos, mas que não prova sua inexistência.

**Processo de Testes** – Esta é a parte mais prática e direta, em que iremos encontrar o que precisamos para atingir os objetivos de qualidade e teste de software.

- **Fases** – Descreve o conjunto de processos que deve ser feito sequencialmente nos testes de software e suas interdependências.
- **Processos** – Descreve, mais detalhadamente, o que devemos fazer em cada processo, como em testes de integração, em que devemos testar a conexão entre vários módulos.
- **Artefatos** – Quais os produtos e documentações que serão resultado de cada processo e em cada fase, assim como a entrada para os processos seguintes. Por exemplo, a descrição das entradas e saídas esperadas ou “casos de teste” são utilizados como entradas para as fases de teste de sistema.

**Métodos e Técnicas de Teste** – Esta área, apesar de se confundir um pouco com as fases e processos, apresenta, na verdade, como fazer as tarefas que estão sendo especificadas nos processos de teste. De certa forma, estes dois estão diretamente ligados, porém, existem várias técnicas que são igualmente aplicadas às várias fases e processos. Por exemplo, um teste de caixa preta, que basicamente significa verificar se um conjunto de entrada gera um conjunto de saída esperado, pode ser aplicado tanto em um teste de componentes individuais, como em um conjunto integrado de módulos ou no sistema como um todo.



**Ferramentas** – O conjunto de ferramentas disponíveis para garantia da qualidade e testes é bem vasto, mas irá se dividir em duas abordagens. A primeira abordagem é documental, ou seja, irá auxiliar os desenvolvedores e testadores a registrar os conjuntos de teste e erros encontrados. A segunda abordagem, muitas vezes incorporada à primeira nas ferramentas, diz respeito aos componentes auxiliares de desenvolvimento e automação; em outras palavras, bibliotecas de código nas linguagens utilizadas no sistema para inserção de códigos de teste, possibilitando também a automação.

**Qualidade** – A qualidade e os testes são dois tópicos bastante ligados, mas diferentes. Os estudos de qualidade focarão em todo o processo de desenvolvimento para melhora e garantia da qualidade, e todos os processos de teste serão um dos fatores do desenvolvimento do sistema que vão garantir sua qualidade, sendo, não obstante, um dos mais importantes.

**Normas** – Todos os padrões a serem seguidos, para garantir a qualidade dos processos de desenvolvimento e testes, assim como os próprios processos de teste em si, foram, ao longo do tempo, sendo solidificados em normas internacionais a serem seguidas pelo desenvolvimento de software e testes. É muito comum encontrarmos o estudo dessas normas aplicadas nas várias áreas de teste, como veremos a seguir.


## 1.2 Qualidade x Testes

As duas áreas que mais se confundem em termos de testes de software são qualidade e testes. É muito comum acharmos que o conhecimento de uma leva a outra, mas é um pouco mais complexo que isso. Originariamente, o estudo e análise da qualidade surgem bem antes da formalização das técnicas de teste de software, visto que qualidade se refere a um espectro mais amplo de produção do que somente software. As primeiras normas ISO de qualidade surgiram na década de 1990, e a formalização das normas de teste, em 2013.

A razão disso vem do fato de que a ideia de garantir que um software seja construído de acordo como foi especificado e sem erros pode ser alcançada por meio da sua qualidade dos processos de desenvolvimento. Obviamente que testes em software já vinham sendo executados desde antes mesmo das normas de qualidade, porém, não se via a necessidade de formalizá-las. A área







de qualidade de software, por outro lado, vinha crescendo por meio da análise dos vários fatores e características, tanto dos processos de desenvolvimento quanto dos softwares em si.


É importante entendermos que a aplicação correta dos testes de software é um dos fatores que irá garantir a qualidade, mas o oposto não é necessariamente verdade, pois a qualidade também depende do processo de desenvolvimento, assim como várias características do software em si. Por exemplo, um software pode ser muito bem testado, obedecer a todos os rigores do processo de desenvolvimento, mas ainda assim, sua arquitetura pode ser extremamente fechada e de difícil manutenção ou expansão. Estas são características que estarão bem mais presentes nas normas de qualidade do que nos processos de teste.

## 1.3 Qualidade de Software

Podemos definir a qualidade de alguma coisa, em geral, um produto, e no caso, um software, por meio da sua capacidade de atender às necessidades para as quais ele foi criado, além de ter sido construído de acordo como foi especificado; contudo, existem muitas nuances nesse processo, e uma delas, por exemplo, é que um software pode ter sido muito bem especificado e construído de acordo, mas não atender às necessidades do usuário, algumas vezes, por ter sido mal especificado, ou outras, porque o usuário ou cliente não soube expressar o que queria.

O estudo da qualidade de software começou, algumas décadas, focando, principalmente, nestas discrepâncias e tentando levantar as características básicas que um software deveria ter para garantir sua qualidade. O problema dessa abordagem é que, como cada software tem um propósito distinto, a maioria das abordagens focava nos métodos que os softwares eram construídos e nos detalhes arquiteturais de seu design. Por exemplo, ter uma arquitetura mais fácil de ser expandida e ser bem documentado são métricas de qualidade do software, mas nada dizem a respeito de esse software atender melhor ou pior às suas necessidades.

Ainda assim, as características e métricas de qualidade ajudaram a melhorar os softwares a serem produzidos, contudo, é importante entendermos




que “qualidade” não é uma ferramenta e nem uma tecnologia, mas sim, uma orientação ou uma série delas; em geral, corresponde a um grande número de documentações. Por esta razão, sempre houve muita desconexão entre a área de qualidade e o real desenvolvimento de software.

A melhor forma de reforçar tal desconexão e garantir a qualidade dos produtos, de modo geral, e no caso, o desenvolvimento de software, foi a criação de normas internacionais e órgãos de reforço que davam “Selos” de qualidade ou “Certificações” para as empresas e produtos que seguissem essas normas. Segundo Gallotti (2016), elas obedeceram, razoavelmente, a seguinte progressão cronológica:

1. (ISO/IEC, 1991) - Norma ISO/IEC 9126 (BRASIL NBR13596 em 1996): Primeira norma de qualidade que surgiu com o foco em software; seu objetivo e escopo consistiam em validar a qualidade dos produtos de software, focando, basicamente, na **qualidade dos processos de desenvolvimento**.
2. (ISO/IEC, 1994) - Norma NBR 12119: Focava na determinação da qualidade dos **softwares** “de prateleira”, ou seja, os softwares desenvolvidos sem um usuário em específico, mas, para propósitos comuns, como um editor de texto ou um programa de cálculo de imposto de renda. Nesta norma, já se incluíam aspectos da área de teste com as instruções e orientações de teste de produto.
3. (ISO/IEC, 1998) - Norma ISO/IEC 9241: Esta norma foca em entrar em conceitos gerais importantes de **usabilidade dos softwares**, novamente, como algo que não garante que um software faça necessariamente o que o usuário precisa, mas que indica vários elementos e métricas de usabilidade universais, como a forma que uma interface com o usuário deve ser apresentada para melhor adequação ergonômica.
4. (ISO/IEC, 1999) - Norma ISO/IEC 14598: Evolui as especificações anteriores para garantir a **qualidade dos processos de desenvolvimento de software**, mais especificamente, diferente das normas de 1991, que ainda estavam mais focadas em produtos genéricos e qualidade de processos.



- 
5. (ISO/IEC, 2008) - Norma ISO/IEC 12207: Define o ciclo de vida do desenvolvimento do software, ou seja, esta norma veio para aperfeiçoar as normas de qualidade de processo anteriores, indicando como cada processo iria interagir com os outros. Neste momento, a parte de testes de software se torna obviamente algo de destaque, uma vez que testes levam naturalmente um “loop” no ciclo de vida, ao serem encontrados defeitos, levando-os de volta ao processo de desenvolvimento anterior.
  6. (ISO/IEC, 2013, 2013) - Norma ABNT NBR ISO/IEC/IEEE 29119-1: Neste momento, os testes de software se encontram com a área de qualidade reconhecidos como um componente indispensável. Além disso, eles especificam todo o processo de **testes de desenvolvimento de software** e mostram como tal teste irá se encaixar no ciclo de vida do desenvolvimento.
  7. (ISO/IEC, 2013) - Norma ISO/IEC 15504 – Revisão da 12207: Uma das normas mais recentes, que passam a focar em como **os processos de desenvolvimento podem ser melhorados** ao longo do ciclo de vida do desenvolvimento e do nível de maturidade das organizações.

## 2. Verificação e Validação de Software

### 2.1 Verificação e Validação

O teste de software tem duas perspectivas principais. A primeira se relaciona muito mais com o processo de desenvolvimento, que consiste em “verificar” se o software está sendo construído de acordo com as especificações; esta perspectiva está ligada ao aspecto de qualidade de processos. A segunda perspectiva é “validar” se o software atende às necessidades reais do usuário. Boehm & Yourdon (1979) possuem expressões para definir a diferença entre as duas perspectivas:

- **Validação:** “Estamos construindo o produto certo?”
  - O software atende às necessidades dos usuários?
- **Verificação:** “Estamos construindo o produto da maneira certa?”
  - O software está sendo construído de acordo com as especificações?

Uma pergunta que surge naturalmente ao observarmos estas duas perspectivas é: Como um software pode não atender às necessidades do usuário, se as suas especificações são feitas a partir de requisitos fornecidos pelo usuário? Existem mais razões para tal discrepância do que se supõe. A primeira razão, mais trivial, está no erro dos analistas ao documentar os requisitos do usuário ou, ainda mesmo, entender o que o usuário está tentando transmitir.

Uma outra razão está na própria engenharia de requisitos. Nem sempre o usuário tem conhecimento suficiente de tecnologia para especificar o que será melhor para seu negócio. Alguns analistas de negócios possuem a função de entender o problema básico do usuário e sugerir a tecnologia apropriada, contudo, sendo esta uma atividade humana, ela também está sujeita a erros.

Ainda existem outras situações em que a tecnologia pode ser tão complexa que o resultado esperado para o usuário pode divergir das suas necessidades. Vamos supor, por exemplo, que um usuário queira um sistema para automatizar o seu processo de registro de vendas de sua loja. Ele quer um sistema automatizado para garantir que os seus funcionários não percam tempo anotando pedidos de vendas, estoque, preenchendo notas fiscais etc. Contudo, após a construção do software, ele percebeu que, apesar da automação ter facilitado alguns aspectos, o seu funcionário está gastando mais tempo para apertar todos os botões na tela e usar o teclado para preencher as informações do que antes, quando fazia-se uma anotação rápida no caderno. Como o volume de vendas é muito alto no horário de pico, os funcionários acabam registrando tudo em papéis de rascunho que, muitas vezes, são perdidos ou mal transcritos. Podemos observar que todos os requisitos foram capturados apropriadamente, mas não foi levada em conta a transição tecnológica das pessoas.

As diferenças encontradas ao final dos processos de verificação e validação deverão ser apropriadamente analisadas e levadas ao cliente junto com a equipe de análise de requisitos e arquitetura.

## 2.2 Defeitos e Falhas

Na área de desenvolvimento de sistemas, no que se refere a testes, existem vários termos que têm significados bem específicos, apesar de serem sinônimos na linguagem comum (ISO/IEC/IEEE, 2017). Entre testes, temos uma categoria destinada a especificar cada tipo de “problema” (genericamente falando) que ocorre durante os testes de software (BARBOSA et al 2004; DELAMARO, MALDONADO & JINO, 2016; IEEE COMPUTER SOCIETY, 1990; MARTINS, 2016). São eles:

- **Engano** (do inglês, *mistake*) – É uma ação, por definição humana, que irá produzir um resultado incorreto. Por exemplo, se um sistema manda salvar dados de um formulário antes mesmo de termos preenchido.
- **Defeito** (do inglês, *fault*) – É o passo ou processo incorreto, ou seja, que não devia ter ocorrido no fluxo de operações do sistema. Note que a indicação de um defeito não necessariamente indica qual o *Erro* ocorrido; este deve ser posteriormente identificado. Os *Defeitos* podem ser ocasionados por uma entrada incorreta ou não esperada de um módulo, assim como por uma implementação incorreta da especificação do módulo ou sistema.
- **Erro** (do inglês, *error*) – É o objeto resultante da ocorrência de um defeito. Ele representa a diferença entre o que foi obtido pelo teste e o que foi esperado, sejam valores ou ações. Note que qualquer valor ou ação que não seja a esperada em qualquer passo dos testes será considerado um erro. Este constitui justamente o objeto da análise do sistema para sua correção.
- **Falha** (do inglês, *failure*) – É o nome dado a situação em que o valor ou ação esperada é diferente daquela indicada na especificação de desenvolvimento do sistema. Note que a diferença entre o defeito e a falha é que o primeiro é uma causa, e o segundo, a sua consequência.

O ciclo desses quatro itens se conecta, consecutivamente, e também nos diferentes níveis de teste. De um modo geral, um **Defeito** produz um **Erro**, que leva o sistema, ou seja, o módulo a ser testado a ter uma **Falha**, sendo que, os defeitos poderão ocorrer tanto por problemas no sistema como por **Enganos**. Note que os **Defeitos** podem ocorrer com ou sem **Engano**.

De uma forma geral, os erros são classificados em:

- **Erros Computacionais** – Provocam uma computação incorreta, mas o caminho executado (sequências de comandos) é igual ao caminho esperado.
- **Erros de Domínio** – O caminho efetivamente executado é diferente do caminho esperado.

Em um primeiro momento, muitas vezes, apenas as **Falhas** são identificadas, ou seja, resultados finais diferentes dos especificados, mas, os **Erros** são descobertos assim como os **Defeitos** que os ocasionaram. Delamaro; Maldonado & Jino (2016) consideram os termos **Engano**, **Defeito** e **Erro** como causas, e a **falha** como a consequência.




### Importante

Devemos entender a sutileza do conceito “Engano”. Por princípio, ele significa que uma certa ação que gerou uma falta, erro ou falha não deveria ter sido executada em primeiro lugar. Definimos esta ação como “humana” porque, por definição, processos automatizados só fazem o que lhe é especificado e estas especificações são sempre fornecidas por seres humanos. Contudo, nos dias de hoje, com a inteligência artificial e muitos outros recursos menos determinísticos, não se pode dizer que um engano somente possa ser cometido por origens humanas. Do ponto de vista de modularização, por exemplo, dados de teste de entrada fornecidos a um módulo podem vir de outro módulo por causa de engano ou defeito, mas do ponto de vista do módulo, dados de entrada que nunca deveriam ter ocorrido são enganos, seja de onde tenham vindo.

## 2.3 Princípios de Teste

Se seguirmos todos os procedimentos dos processos de teste, estaremos provavelmente seguros de termos verificado e validado nosso



software adequadamente; contudo, muitas vezes, no processo de criação dos casos de teste ou na sua execução, somos tentados a vários vícios ou caminhos que ilusoriamente parecem óbvios. Existem alguns princípios já bastante conhecidos que foram empiricamente levantados por meio das décadas de experiência nessa área (ISTQB, 2018). São eles:

1. **Teste mostra a presença de defeitos, mas não a ausência** – Os testes não podem demonstrar se o software é livre de defeitos ou se ele se comportará conforme especificado em qualquer situação. É sempre possível que um teste que você tenha esquecido seja aquele que poderia descobrir mais problemas no sistema (SOMMERVILLE, 2011). Dahl, Dijkstra e Hoare (1972, *apud* BOEHM & YOURDON, 1979) são um dos primeiros colaboradores para o desenvolvimento da engenharia de software e citam que os testes podem mostrar apenas a presença de erros, e não sua ausência.
2. **Testes exaustivos são impossíveis** – Esta impossibilidade se dá, não somente pela enorme quantidade, tempo e custo, mas também, pela falta do conhecimento. Um software pode adquirir uma quantidade de possibilidades e dimensões que pode até não ser previsível, mesmo que tivéssemos a intenção de testar todos os casos. Este princípio, não se fundamenta apenas na área de software, mas na dedução científica e teórica básica. Einstein já mencionava que a verdade sobre uma teoria nunca poderia ser definitiva, pois seria impossível saber se algum experimento futuro não poderia prová-la errada (EINSTEIN, 1919).
3. **Testes antecipados poupam tempo e custo** – Os defeitos de um software se propagam exponencialmente, à medida que os dados e o fluxo do sistema vão passando a cada novo módulo implementado. É possível, por exemplo, que dúzias de erros em um módulo final possam ser resolvidos por meio da correção de poucos defeitos nos módulos de entrada do sistema. A execução de testes o mais cedo possível no desenvolvimento irá poupar muito tempo na implementação de novos módulos incorretos ou testando vários defeitos que, na verdade, nunca ocorreriam se os módulos iniciais já estivessem reparados.

4. **Defeitos se aglomeram** – Normalmente, um pequeno grupo de módulos que irá conter a maior quantidade de defeitos. Geralmente, módulos mais complexos obviamente terão muito mais caminhos de fluxo com possível chance de erros.
5. **Paradoxo do Pesticida** – Analogamente a um pesticida, um teste ou conjunto de valores, após ser executado muitas vezes, torna-se ineficiente, sendo necessário criar novos testes ou conjunto de valores. De certa forma, este princípio se estende a partir dos princípios 1 e 2. Já que não há como abranger com 100% de certeza todos os possíveis defeitos, sempre haverá novos testes que poderão mostrar novos erros, o que implica que os mesmos testes e valores, em algum momento, já tenham demonstrado todos os defeitos do domínio que eles abrangem. Existe uma exceção nesse princípio, que são os testes de regressão, que consistem em reexecutar testes passados ou mesmo em plataformas de compatibilidade anteriores, após mudanças e correções no software.
6. **Testes são dependentes do contexto** – Obviamente, as diferentes áreas de negócios terão diferentes níveis de exigência, segurança, precisão etc. e, portanto, o contexto da aplicação influirá no projeto e especificação dos testes. Por exemplo, a exigência de um software médico é muito superior à de um software de registro de ponto de funcionários.
7. **Ausência de erros é ilusório** – A gerência de projetos e, geralmente, o próprio cliente esperam que, após um grande número de ciclos de teste e reparo, o sistema funcione corretamente e livre de defeitos. Há dois erros de raciocínio nessa expectativa: a primeira já é demonstrada pelos princípios 1 e 2; além disso, achar defeitos é uma ação muito mais focada na “VERIFICAÇÃO”, e não na “VALIDAÇÃO”. Ou seja, um sistema pode não apresentar nenhum defeito num primeiro momento, e ainda não atender às expectativas do usuário.





## Saiba Mais

Segundo Pinheiro (2018), o exame de certificação *Certified Tester Foundation Level* (CTFL), do *International Software Testing Qualifications Board* sempre avalia os conhecimentos do candidato sobre cada termo e princípio de teste. Para conhecer mais sobre o tema, consulte o site do BSTQ.



| Leia mais



### 3. Processo de Teste de Software

De acordo com a norma ISO/IEC (2013), o processo de teste de software não é universal e vai depender do contexto e do software que deverão ser testados. Existe um conjunto básico comum de tarefas que costumam estar presentes no desenvolvimento de softwares de médio e grande portes, contudo, algumas dessas tarefas podem ser agregadas, opcionais ou até mesmo mais detalhadas:

1. **Planejamento** – Atividades que envolvem definir os objetivos do teste e quais abordagens serão usadas para atingi-los.
2. **Controle e Monitoramento** – Atividades de avaliação do progresso dos testes, checando as diferenças entre os resultados correntes e os objetivos.
3. **Análise** – Esta é a fase em que determinaremos o que deve ser testado, quais critérios serão usados, as prioridades de cada item a ser testado e quais as condições de cada teste.
4. **Projeto** – A fase anterior define o que tem que ser testado. A fase de projeto vai especificar “como” estes itens serão testados; esta tarefa compreende a criação e documentação dos casos de teste e inclui as seguintes atividades:
  - a. Criar e priorizar os casos de teste;
  - b. Identificar os dados de teste para suportar as condições dos casos de teste;

- c. Especificar o ambiente de teste e toda infraestrutura necessária;
  - d. Traçar a relação bidirecional entre os casos de teste, as condições e os procedimentos.
5. **Implementação** – Se fossemos comparar as tarefas de teste com as tarefas de desenvolvimento, a implementação seria efetivamente o ato de criar o código do software. É na área de testes que criamos tudo que é necessário para a posterior execução dos testes (criar o conjunto de dados, scripts de automação de testes etc).
  6. **Execução** – Fase real de execução dos testes. Os elementos dessa tarefa poderão variar bastante, mas isso vai depender das categorias de teste especificadas.
  7. **Conclusão** – Consolida todos os resultados da execução dos testes, assim como as condições gerais, verificando se todos os objetivos definidos foram atingidos. Esses resultados devem ser apresentados por meio de um relatório geral para o cliente.

Estas tarefas apresentadas são genéricas para as várias fases e categorias de teste, pois primeiramente, é necessário interagir com a equipe de desenvolvimento e obter as fases e especificações do sistema, para então, determinar todo esse processo de teste para cada fase.



### Saiba Mais

O mundo está cada vez mais conectado, integrado e fácil, porém, mesmo com todo o avanço tecnológico, as equipes de desenvolvimento de software continuam encarando as questões ligadas à qualidade. A IBM tem soluções e ferramentas para orientar o relato dos testes de software para garantir a qualidade. Acesse o link e saiba mais sobre o assunto.



| Leia mais





## Síntese

Nesta unidade de aprendizagem, aprendemos, de forma geral, três fundamentos principais: o que é a qualidade de um software e como ela está relacionada aos testes de software; os conceitos básicos do teste de software e qual o processo que deve ser realizado para teste de um software.

Verificamos, também, que a qualidade não é garantida apenas pelos testes de software; o próprio processo de desenvolvimento contribui para ela. Vimos as normas internacionais que devem ser seguidas para a garantia da qualidade dos processos, produtos e também dos processos de teste de software.

Aprendemos o significado específico dos termos fundamentais da área de teste de software, como a diferença entre defeito e erro, verificar e validar, e principalmente, entendemos que existe uma grande diferença entre o produto que é desenvolvido e o produto que é esperado.

Por fim, entendemos as tarefas necessárias para a completa execução de um teste de software. Iremos aprofundar essas tarefas e ver como elas se encaixam em suas etapas nas unidades seguintes.

## Referências

BARBOSA, E. F et al. (2004). Introdução ao Teste de Software. **Notas didáticas do ICMC**. 2004. Disponível em: <[http://pbjug-grupo-de-usuarios-java-da-paraiba.1393240.n2.nabble.com/attachment/2545944/0/Introducao Ao Teste De Software - Maldonado,Jose.pdf](http://pbjug-grupo-de-usuarios-java-da-paraiba.1393240.n2.nabble.com/attachment/2545944/0/Introducao-Ao-Teste-De-Software-Maldonado,Jose.pdf)>. Acesso em: 20 ago. 2018.

BOEHM, B.; & YOURDON, E. **Classics in software engineering**. Yourdon Press. 1979. Disponível em: <<https://dl.acm.org/citation.cfm?id=1241536>>. Acesso em: 20 ago. 2018.


BSTQB. **Certified by international software testing qualifications board**. Disponível em: <<http://www.bstqb.org.br/>>. Acesso em: 16 ago. 2018.

DAHL, O. J; DIJKSTRA, E. W; & Hoare, C. A. R. **Structured programming**. Academic Press. 1972. Disponível em: <<https://dl.acm.org/citation.cfm?id=1243380>>. Acesso em: 16 ago. 2018.

DELAMARO, M. E; Maldonado, J. C; & Jino, M. **Introdução ao teste de software**. 2 ed. Rio de Janeiro: Elsevier, 2016.

EINSTEIN, A. Indução e dedução na física. **Scientiae Studia**. 1919. Disponível em: <<https://doi.org/10.1590/S1678-31662005000400008>>. Acesso em: 16 ago. 2018.

GALLOTTI, Giocondo Marino Antonio (Org.) (2016). *Qualidade de Software*. São Paulo: Pearson Education do Brasil, 2016. Disponível em: <<http://unigranrio.bv3.digitalpages.com.br/users/publications/9788543020358/pages/-12>>. Acesso em: 16 ago. 2018.



IEEE COMPUTER SOCIETY. **IEEE Standard Glossary of Software Engineering Terminology**. IEEE 610.12. 1990. Disponível em: <<https://standards.ieee.org/findstds/standard/610.12-1990.html>>. Acesso em: 16 ago. 2018.

ISO/IEC. **ISO/IEC 9126:1991** - Software engineering - Product quality. 1991. Disponível em: <<https://www.iso.org/standard/16722.html>>. Acesso em: 16 ago. 2018.

\_\_\_\_\_. **ISO/IEC 9241-11:1998** - Guidance on usability. 1998. Disponível em: <<https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en>>. Acesso em: 16 ago. 2018.

\_\_\_\_\_. **ISO/IEC 12119** - Information technology - Software packages - Quality requirements and testing. Rio de Janeiro, 1994. Disponível em: <<https://www.iso.org/standard/1308.html>>. Acesso em: 16 ago. 2018.

\_\_\_\_\_. **ISO/IEC 15504** - Information technology -- Process assessment. 2013. Disponível em: <<https://www.iso.org/standard/61492.html>>. Acesso em: 16 ago. 2018.

\_\_\_\_\_. **ISO/IEC 29119-1** - Software and systems engineering — Software testing — Part 1: Concepts and definitions. Geneva: IEEE. 2013. Disponível em: <<https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29119:-1:ed-1:v1:en>>.

\_\_\_\_\_. **ISO/IEC 29119-2** - Software and systems engineering — Software testing — Part 2: Test processes. GENEVA: IEEE. 2013. Disponível em: <<https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29119:-2:ed-1:v1:en>>. Acesso em: 16 ago. 2018.

ISO/IEC/IEEE. **ISO/IEC/IEEE 24765:2017** - Systems and software engineering — Vocabulary. New York: IEEE. 2017. Disponível em: <<https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:24765:ed-2:v1:en>>. Acesso em: 16 ago. 2018.

ISTQB. **Certified Foundation Level Syllabus**. 2018. Disponível em: <<https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html>>. Acesso em: 16 ago. 2018.



MARTINS, M. D. C. **Testes de Software**. Rio de Janeiro: Sses, 2016.

PINHEIRO, Flávio R. Principais tópicos do exame CTFL. **Mindmesister**. 2018. Disponível em: <<https://www.mindmeister.com/pt/212814955/principais-t-picos-do-exame-ctfl>>. Acesso em: 16 ago. 2018.

SOMMERVILLE, I. **Engenharia de Software**. 9 ed. São Paulo: Pearson, 2011. Disponível em: < [http://tmv.edu.in/pdf/Diploma Syllabus/Computer/TY\\_fifth\\_sem/Fifth Semester Curriculum.pdf](http://tmv.edu.in/pdf/Diploma Syllabus/Computer/TY_fifth_sem/Fifth Semester Curriculum.pdf)>. Acesso em: 16 ago. 2018.

