**FACULDADE INTERAMERICANA DE PORTO VELHO**

**PAULO JOSÉ DOS SANTOS**

Teste de Software unitário e Teste de Software Automatizado

PORTO VELHO/RO

2014

**INTRODUÇÃO**

O presente trabalho tem por objetivo apresenta alguns conceitos e característica referente a teste de software, abordando falando um pouco sobre as técnicas e níveis de teste tem como foco principal dentre os inúmeros conceitos estados de poder, apresentando o que grandes filósofos brigaram e discutiram por uma sociedade melhor, para tirar o poder soberano das mãos de um único individuo. Explanando um pouco sobre o Estado Liberal, e comparando com o nosso atual período histórico.

[Teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado.](http://www.devmedia.com.br/introducao-a-testes-de-software-parte-1-definicoes-iniciais/16075)  O seu objetivo é revelar falhas em um produto, para que as causas dessas falhas sejam identificadas e possam ser corrigidas pela equipe de desenvolvimento antes da entrega final. Por conta dessa característica das atividades de teste, dizemos que sua natureza é “destrutiva”, e não “construtiva”, pois visa ao aumento da confiança de um produto através da exposição de seus problemas, porém antes de sua entrega ao usuário final.

O conceito de teste de software pode ser compreendido através de uma visão intuitiva ou mesmo de uma maneira formal. Existem atualmente várias definições para esse conceito. De uma forma simples, testar um software significa verificar através de uma execução controlada se o seu comportamento corre de acordo com o especificado. O objetivo principal desta tarefa é revelar o número máximo de falhas dispondo do mínimo de esforço, ou seja, mostrar aos que desenvolvem se os resultados estão ou não de acordo com os padrões estabelecidos.

Ao longo deste artigo, iremos discutir os principais conceitos relacionados às atividades de teste, as principais técnicas e critérios de teste que podem ser utilizados para [verificação ou validação de um produto](http://www.devmedia.com.br/curso/definicoes-iniciais-sobre-validacao-verificacao-e-testes/309), assim como exemplos práticos da aplicação de cada tipo de técnica ou critério de teste.

Não se pode garantir que todo [software](https://pt.wikipedia.org/wiki/Software) funcione corretamente, sem a presença de erros, visto que os mesmos muitas vezes possuem um grande número de estados com fórmulas, atividades e [algoritmos](https://pt.wikipedia.org/wiki/Algoritmo) complexos. O tamanho do projeto a ser desenvolvido e a quantidade de pessoas envolvidas no processo aumentam ainda mais a complexidade. Idealmente, toda permutação possível do software deveria ser testada. Entretanto, isso se torna impossível para a ampla maioria dos casos devido à quantidade impraticável de possibilidades. A qualidade do teste acaba se relacionando à qualidade dos profissionais envolvidos em filtrar as permutações relevantes.

Falhas podem ser originadas por diversos motivos. Por exemplo, a [especificação](https://pt.wikipedia.org/wiki/Projeto_de_Software) pode estar errada ou incompleta, ou pode conter [requisitos](https://pt.wikipedia.org/wiki/Requisito) impossíveis de serem [implementados](https://pt.wikipedia.org/w/index.php?title=Implementa%C3%A7%C3%A3o&action=edit&redlink=1), devido a limitações de [hardware](https://pt.wikipedia.org/wiki/Hardware) ou software. A implementação também pode estar errada ou incompleta, como um erro de um algoritmo. Portanto, uma falha é o resultado de um ou mais defeitos em algum aspecto do sistema.

O teste de software pode ser visto como uma parcela do processo de [qualidade de software](https://pt.wikipedia.org/wiki/Qualidade_de_software). A qualidade da aplicação pode e, normalmente, varia significativamente de sistema para sistema.

Os atributos qualitativos previstos na norma [ISO 9126](https://pt.wikipedia.org/wiki/ISO_9126) são:

[Funcionalidade](https://pt.wikipedia.org/wiki/ISO_9126#Funcionalidade), [Confiabilidade](https://pt.wikipedia.org/wiki/ISO_9126#Confiabilidade), [Usabilidade](https://pt.wikipedia.org/wiki/ISO_9126#Usabilidade), [Eficiência](https://pt.wikipedia.org/wiki/ISO_9126#Efici.C3.AAncia), [Manutenibilidade](https://pt.wikipedia.org/wiki/ISO_9126#Manutenibilidade),

[Portabilidade](https://pt.wikipedia.org/wiki/ISO_9126#Portabilidade).

De forma geral, mensurar o bom funcionamento de um software envolve compará-lo com elementos como especificações, outros softwares da mesma linha, versões anteriores do mesmo produto, inferências pessoais, expectativas do cliente, normas relevantes, leis aplicáveis, entre outros. Enquanto a especificação do software diz respeito ao processo de verificação do software, a expectativa do cliente diz respeito ao processo de validação do software. Por meio da verificação será analisado se o produto foi feito corretamente, se ele está de acordo com os requisitos especificados. Por meio da validação será analisado se foi feito o produto correto, se ele está de acordo com as necessidades e expectativas do cliente.

Um [desenvolvimento de software](https://pt.wikipedia.org/wiki/Desenvolvimento_de_software) organizado tem como premissa uma [metodologia](https://pt.wikipedia.org/wiki/Metodologia) de trabalho. Esta deve ter como base conceitos que visem a construção de um produto de software de forma eficaz. Dentro desta metodologia estão definidos os passos necessários para chegar ao produto final esperado.

Assim, quando se segue uma metodologia para o desenvolvimento de um produto de software, espera-se um produto final que melhor agrade tanto aos clientes quanto ao próprio fornecedor, ou seja, a empresa de desenvolvimento. Observando este aspecto, não faz sentido iniciar a construção de um produto de software sem ter uma metodologia de trabalho bem solidificada e que seja do conhecimento de todos os envolvidos no processo. Porém, além de uma crescente demanda por softwares de qualidade, as empresas de desenvolvimento de software sofrem cada vez mais pressão por parte dos clientes para que o produto seja entregue num curto período de tempo. Este fato pode fazer com que uma sólida metodologia de trabalho acabe por se desequilibrar.

Independentemente da metodologia de trabalho empregada no desenvolvimento de um software, para que se obtenha um produto final com um certo nível de qualidade é imprescindível a melhoria dos processos de engenharia de software.

Uma maneira viável para se assegurar a melhoria de tais [processos](https://pt.wikipedia.org/wiki/Processos) seria tomar como base modelos sugerido por entidades internacionais respeitadas no assunto. Dentro de uma gama de modelos, sejam eles para situações e ambientes específicos ou para soluções genéricas, existem alguns que são mais utilizados e tidos como eficientes, como por exemplo, os [SW-CMM](https://pt.wikipedia.org/w/index.php?title=SW-CMM&action=edit&redlink=1), [SE-CMM](https://pt.wikipedia.org/w/index.php?title=SE-CMM&action=edit&redlink=1), [ISO/IEC 15504](https://pt.wikipedia.org/wiki/ISO/IEC_15504) e o mais conhecido [CMMI](https://pt.wikipedia.org/wiki/CMMI).

Outro fator com grande influência sobre a qualidade do software a ser produzido é o que diz respeito aos testes que serão executados sobre tal produto. Todas as metodologias de desenvolvimento de software têm uma disciplina dedicada aos testes. Atualmente esta é uma tarefa indispensável, porém muitas vezes efetuada de maneira ineficiente, seja pelo subestimar dos que desenvolvem, pela falta de tempo ou mesmo pela falta de recursos humanos e financeiros.

De acordo com um estudo conduzido pelo [NIST](https://pt.wikipedia.org/wiki/NIST) em [2002](https://pt.wikipedia.org/wiki/2002), os [defeitos](https://pt.wikipedia.org/wiki/Defeito_de_software) resultam num custo anual de 59,5 [bilhões](https://pt.wikipedia.org/wiki/Bilh%C3%A3o) de [dólares](https://pt.wikipedia.org/wiki/D%C3%B3lar_americano) à economia dos [Estados Unidos](https://pt.wikipedia.org/wiki/Estados_Unidos). Mais de um terço do custo poderia ser evitado com melhorias na [infraestrutura](https://pt.wikipedia.org/wiki/Infraestrutura) do teste de software.

## Técnicas de Teste de Software

Existem muitas maneiras de se testar um software. Mesmo assim, existem as técnicas que sempre foram muito utilizadas em sistemas desenvolvidos sobre [linguagens estruturadas](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_estruturada) que ainda hoje tem grande valia para os sistemas [orientados a objeto](https://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objeto). Apesar de os paradigmas de desenvolvimento ser completamente diferentes, o objetivo principal destas técnicas continua a ser o mesmo, encontrar falhas no software. Abaixo estão descritas algumas das técnicas mais conhecidas.

### Caixa-branca

Também chamado de teste estrutural ou orientado à lógica, a técnica de caixa-branca avalia o comportamento interno do [componente de software](https://pt.wikipedia.org/wiki/Componente_de_software). Essa técnica trabalha diretamente sobre o [código fonte](https://pt.wikipedia.org/wiki/C%C3%B3digo_fonte) do componente de software para avaliar aspectos tais como: teste de condição, [teste de fluxo de dados](https://pt.wikipedia.org/wiki/Teste_de_fluxo_de_dados), teste de ciclos, teste de caminhos lógicos, códigos nunca executados.

Os aspectos avaliados nesta técnica de teste dependerão da complexidade e da tecnologia que determinarem a construção do componente de software, cabendo, portanto avaliação de mais aspectos que os citados anteriormente. O testador tem acesso ao código fonte da aplicação e pode construir códigos para efetuar a ligação de [bibliotecas](https://pt.wikipedia.org/wiki/Biblioteca_de_rotinas) e componentes. Este tipo de teste é desenvolvido analisando o código fonte e elaborando casos de teste que cubram todas as possibilidades do componente de software. Dessa maneira, todas as variações relevantes originadas por estruturas de condições são testadas.

Um exemplo bem prático desta técnica de teste é o uso da ferramenta livre [JUnit](https://pt.wikipedia.org/wiki/JUnit) para desenvolvimento de classes de teste para testar classes ou métodos desenvolvidos em [Java](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o_Java). Também se enquadram nessa técnica testes manuais ou testes efetuados com apoio de ferramentas para verificação de aderência a boas práticas de codificação reconhecidas pelo mercado de software. A aderência a padrões e boas práticas visa principalmente a diminuição da possibilidade de erros de codificação e a busca de utilização de comandos que gerem o melhor desempenho de execução possível. Apesar de muitos desenvolvedores alegarem que não há ganhos perceptíveis com essa técnica de teste aplicada sobre unidades de software, devemos lembrar que, no ambiente produtivo, cada programa pode vir a ser executado milhares ou milhões de vezes em intervalos de tempo pequenos. É na realidade de produção que a soma dos aparentes pequenos tempos de execução e consumo de memória de cada programa poderá levar o software a deixar de atender aos objetivos esperados. A técnica de teste de caixa-branca é recomendada para as fases de [teste de unidade](https://pt.wikipedia.org/wiki/Teste_de_unidade) e [teste de integração](https://pt.wikipedia.org/wiki/Teste_de_integra%C3%A7%C3%A3o), cuja responsabilidade principal fica a cargo dos desenvolvedores do software, que por sua vez conhecem bem o código fonte produzido.

### Caixa-preta

Também chamada de teste funcional, teste comportamental, orientado a dado ou orientado a entrada e saída, a técnica de caixa-preta avalia o comportamento externo do componente de software, sem se considerar o comportamento interno do mesmo. Dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado previamente conhecido. Como detalhes de implementação não são considerados, os casos de teste são todos derivados da especificação.

Quanto mais entradas são fornecidas, mais rico será o teste. Numa situação ideal todas as entradas possíveis seriam testadas, mas na ampla maioria dos casos isso é impossível.Outro problema é que a especificação pode estar ambígua em relação ao sistema produzido, e como resultado as entradas especificadas podem não ser as mesmas aceitas para o teste.Uma abordagem mais realista para o teste de caixa-preta é escolher um subconjunto de entradas que maximize a riqueza do teste. Pode-se agrupar subconjuntos de entradas possíveis que são processadas similarmente, de forma que testar somente um elemento desse subconjunto serve para averiguar a qualidade de todo o subconjunto. Por exemplo, em um sistema que aceita um inteiro como entrada, testar todos os casos possíveis pode gerar pelo menos dezenas de milhares de casos de testes distintos. Entretanto, a partir da especificação do sistema, pode-se encontrar um subconjunto de inteiros que maximizem a qualidade do teste. Depende do propósito do sistema, mas casos possíveis incluem inteiros pares, inteiros ímpares, zero, inteiros positivos, inteiros negativos, o maior inteiro, o menor inteiro.

### Caixa-cinza

A técnica de teste de caixa-cinza é uma mescla do uso das técnicas de caixa-preta e de caixa-branca. Isso envolve ter acesso a estruturas de dados e algoritmos do componente a fim de desenvolver os casos de teste, que são executados como na técnica da caixa-preta. Manipular entradas de dados e formatar a saída não é considerado caixa-cinza, pois a entrada e a saída estão claramente fora da caixa-preta. A caixa-cinza pode incluir também o uso de engenharia reversa para determinar, por exemplo, os limites superiores e inferiores das classes, além de mensagens de erro.

### Regressão

Essa é uma técnica de teste aplicável a uma nova versão de software ou à necessidade de se executar um novo ciclo de teste durante o processo de desenvolvimento. Consiste em se aplicar, a cada nova versão do software ou a cada ciclo, todos os testes que já foram aplicados nas versões ou ciclos de teste anteriores do sistema. Inclui-se nesse contexto a observação de fases e técnicas de teste de acordo com o impacto de alterações provocado pela nova versão ou ciclo de teste. Para efeito de aumento de produtividade e de viabilidade dos testes, é recomendada a utilização de ferramentas de automação de teste, de forma que, sobre a nova versão ou ciclo de teste, todos os testes anteriores possam ser executados novamente com maior agilidade.

### Técnicas não funcionais

São técnicas utilizadas para verificar a operação correta do sistema em relação a casos inválidos ou inesperados de entrada. Outras técnicas de teste existem para testar aspectos não funcionais do software, como por exemplo, a adequação a restrições de negócio, adequação a normas, ou restrições tecnológicas. Em contraste às técnicas funcionais mencionadas acima, que verificam a produção pelo sistema de respostas adequadas de suas operações, de acordo com uma especificação, as técnicas não funcionais verificam atributos de um componente ou sistema que não se relacionam com a funcionalidade (por exemplo, confiabilidade, eficiência, usabilidade, manutenibilidade e portabilidade).

Uma delas é o uso conjunto de teste de desempenho e teste de carga, que verifica se o software consegue processar grandes quantidades de dados, e nas especificações de tempo de processamento exigidas, o que determina a escalabilidade do software. O teste de usabilidade é necessário para verificar se a interface de usuário é fácil de aprender e utilizar. Entre verificações cabíveis estão à relação da interface com conhecimento do usuário, a compreensibilidade das mensagens de erro e a integridade visual entre diferentes componentes. Já o teste de confiabilidade é usado para verificar se o software é seguro em assegurar o sigilo dos dados armazenados e processados. O teste de recuperação é usado para verificar a robustez do software em retornar a um estado estável de execução após estar em um estado de falha.

## Níveis de teste de software

Uma prática comum é testar o software após uma funcionalidade ser desenvolvida, e antes dela ser implantada no cliente, por um grupo de profissionais diferente da implementação. Essa prática pode resultar na fase de teste ser usada para compensar atrasos do projeto, comprometendo o tempo devotado ao teste. Outra prática é começar o teste no mesmo momento que o projeto, num processo contínuo até o fim do projeto.

Em contrapartida, algumas práticas emergentes como a programação extrema e o desenvolvimento ágil focam o modelo de desenvolvimento orientado ao teste. Nesse processo, os testes de unidade são escritos primeiro (TDD), por engenheiros de software. Antes da implementação da unidade em questão, o teste falha. Então o código é escrito, passando incrementalmente em porções maiores dos casos de teste. Os testes são mantidos junto com o resto do código fonte do software, e geralmente também integra o processo de construção do software.

Existem vários níveis de teste de software, porém o presente trabalho somente abordara teste unitário e teste automatizado.

### Teste de unidade

Também conhecida como teste unitário ou teste de módulo, é a fase em que se testam as menores unidades de software desenvolvidas (pequenas partes ou unidades do sistema). O universo alvo desse tipo de teste são as sub-rotinas, métodos, classes ou mesmo pequenos trechos de código. Assim, o objetivo é o de encontrar falhas de funcionamento dentro de uma pequena parte do sistema funcionando independentemente do todo.

**Automação de teste**

Automação de teste é o uso de software para controlar a execução do teste de software, a comparação dos resultados esperados com os resultados reais, a configuração das pré-condições de teste e outras funções de controle e relatório de teste. De forma geral, a automação de teste pode iniciar a partir de um processo manual de teste já estabelecido e formalizado.

Apesar do teste manual de software permitir encontrar vários erros em uma aplicação, é um trabalho maçante e que demanda um grande esforço em tempo. Também, pode não ser efetivo na procura de classes específicas de defeitos. A automação é o processo de escrita de um programa de computador para realizar o teste. Uma vez automatizado, um grande número de casos de teste podem ser validados rapidamente. As vantagens da automação tornam-se mais evidentes para os casos de softwares que possuem longa vida no mercado, devido ao fato de que até mesmo pequenas correções no código da aplicação podem causar a quebra de funcionalidades que antes funcionavam.

A automação envolve testes de caixa-preta, em que o desenvolvedor não possui conhecimento sobre a estrutura interna do sistema, ou caixa-branca, em que há pleno conhecimento da estrutura interna. Para os dois casos, a cobertura de teste é determinada respectivamente pela experiência do desenvolvedor ou pela métrica de cobertura de código.

A automação de teste pode ser cara, e geralmente é usada em conjunto com técnicas manuais. Entretanto, pode cortar custos a longo prazo, especialmente na fase de teste de regressão. Uma forma de gerar casos de teste automaticamente é o teste baseado em modelo, em que um modelo do sistema é usado para a geração de casos de teste.

## Abordagens

Há duas abordagens usadas na automação.

No teste de interface gráfica do utilizador, uma plataforma gera os eventos de entrada na interface de utilizador do sistema e observa as mudanças na saída. Várias ferramentas fornecem funcionalidades para gravar e reproduzir as ações do utilizador. Sendo aplicável a qualquer aplicação que use uma interface gráfica, esta técnica possui a vantagem de exigir menos codificação, mas implica dificuldades de manutenção do teste, como quando a interface gráfica é alterada.

Já no teste baseado em código, a interface pública das classes, módulos ou bibliotecas são testadas com uma variedade de argumentos de entrada, observando-se a saída. Uma tendência atual no desenvolvimento de software é o uso de plataformas xUnit, que permitem realizar testes de unidade para determinar se as seções do código estão processando de forma esperada em diversas circunstâncias. O teste baseado em código é uma funcionalidade chave no desenvolvimento ágil de software.

**Bibliografia**

# Artigo Engenharia de Software - Introdução a Teste de Software

Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>. Acesso em: 10 abril de 2016.

# Wikipédia Teste de software

Disponível em: < https://pt.wikipedia.org/wiki/Teste\_de\_software>. Acesso em: 10 abril de 2016.

# Wikipédia Automação de teste

Disponível em: < https://pt.wikipedia.org/wiki/Automa%C3%A7%C3%A3o\_de\_teste>. Acesso em: 10 abril de 2016.