Teste de software:

O que é?

Teste de software é a execução do software de maneira controlada para avaliar se ele se comporta ou não conforme o especificado. Visa a verificar e relatar o estado do software, a qualquer momento, em termos significativos a todos que precisam desta informação.

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.

Testar um software significa verificar através de uma execução controlada se o seu comportamento corre de acordo com o especificado.

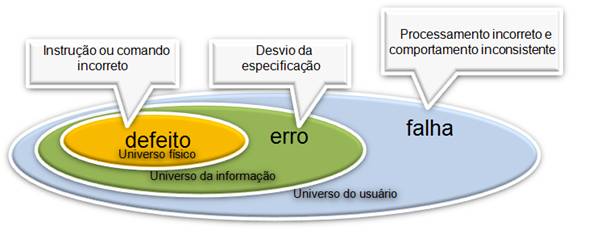
Conceitos Básicos

As definições usadas aqui seguem a terminologia padrão para Engenharia de Software do IEEE – *Institute of Electrical and Electronics Engineers* – (IEEE 610, 1990).

**Defeito** é um ato inconsistente cometido por um indivíduo ao tentar entender uma determinada informação, resolver um problema ou utilizar um método ou uma ferramenta. Por exemplo, uma instrução ou comando incorreto.

**Erro** é uma manifestação concreta de um defeito num artefato de software. Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução de um programa constitui um erro.

**Falha** é o comportamento operacional do software diferente do esperado pelo usuário. Uma falha pode ter sido causada por diversos erros e alguns erros podem nunca causar uma falha.



Níveis de Teste de Software

O planejamento dos testes deve ocorrer em diferentes níveis e em paralelo ao desenvolvimento do software (**Figura 3**). Buscando informação no Livro “Qualidade de Software – Teoria e Prática” (ROCHA *et al*., 2001), definimos que os principais níveis de teste de software são:

·        **Teste de Unidade:** também conhecido como testes unitários. Tem por objetivo explorar a menor unidade do projeto, procurando provocar falhas ocasionadas por defeitos de lógica e de implementação em cada módulo, separadamente. O universo alvo desse tipo de teste são os métodos dos objetos ou mesmo pequenos trechos de código.

·        **Teste de Integração:** visa provocar falhas associadas às interfaces entre os módulos quando esses são integrados para construir a estrutura do software que foi estabelecida na fase de projeto.

·        **Teste de Sistema:** avalia o software em busca de falhas por meio da utilização do mesmo, como se fosse um usuário final. Dessa maneira, os testes são executados nos mesmos ambientes, com as mesmas condições e com os mesmos dados de entrada que um usuário utilizaria no seu dia-a-dia de manipulação do software. Verifica se o produto satisfaz seus requisitos.

·        **Teste de Aceitação:** são realizados geralmente por um restrito grupo de usuários finais do sistema. Esses simulam operações de rotina do sistema de modo a verificar se seu comportamento está de acordo com o solicitado.

·        **Teste de Regressão:** Teste de regressão não corresponde a um nível de teste, mas é uma estratégia importante para redução de “efeitos colaterais”. 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. Pode ser aplicado em qualquer nível de teste.

O planejamento e projeto dos testes devem ocorrer de cima para baixo, ou seja:

1.      Inicialmente é planejado o teste de aceitação a partir do documento de requisitos;

2.      Após isso é planejado o teste de sistema a partir do projeto de alto nível do software;

3.      Em seguida ocorre o planejamento dos testes de integração a partir o projeto detalhado;

4.      E por fim, o planejamento dos testes a partir da codificação.

Tipos e Técnicas de Teste de Software

|  |  |
| --- | --- |
| Teste de Unidade | Também conhecido como testes unitários. Tem por objetivo explorar a menor unidade do projeto, procurando provocar falhas ocasionadas por defeitos de lógica e de implementação em cada módulo, separadamente. O universo alvo desse tipo de teste são os métodos dos objetos ou mesmo pequenos trechos de código. |
| Teste de Integração | visa provocar falhas associadas às interfaces entre os módulos quando esses são integrados para construir a estrutura do software que foi estabelecida na fase de projeto. |
| Teste Operacional | Garante que a aplicação pode rodar muito tempo sem falhar. |
| Teste Positivo-negativo | Garante que a aplicação vai funcionar no “caminho feliz” de sua execução e vai funcionar no seu fluxo de exceção; ou seja, garante que a aplicação funcione para os fluxos de execução previstos e tratados. |
| Teste de regressão | Toda vez que algo for mudado, deve ser testada toda a aplicação novamente. |
| Teste de Interface | Verifica se a navegabilidade e os objetivos da tela funcionam como especificados e se atendem da melhor forma ao usuário. |
| Teste de Performance | Verifica se o tempo de resposta é o desejado para o momento de utilização da aplicação. |
| Teste de carga | Verifica o funcionamento da aplicação com a utilização de uma quantidade grande de usuários simultâneos. |
| Teste de aceitação do usuário | Testa se a solução será bem vista pelo usuário. Ex: caso exista um botão pequeno demais para executar uma função, isso deve ser criticado em fase de testes. (aqui, cabem quesitos fora da interface, também). |
| Teste de Volume | Testar a quantidade de dados envolvidos (pode ser pouca, normal, grande, ou além de grande). |
| Testes de stress | Testar a aplicação sem situações inesperadas. Testar caminhos, às vezes, antes não previstos no desenvolvimento/documentação. |
| Testes de Configuração | Testar se a aplicação funciona corretamente em diferentes ambientes de hardware ou de software. |
| Testes de Instalação | Testar se a instalação da aplicação foi OK. |
| Testes de Segurança | Testar a segurança da aplicação das mais diversas formas. Utilizar os diversos papéis, perfis, permissões, para navegar no sistema. |

Teste Estrutural / Caixa Branca:

Técnica de teste que avalia o comportamento interno do componente de software. Essa técnica trabalha diretamente sobre o código fonte do componente de software.

Este tipo de teste é desenvolvido analisando-se o código fonte e elaborando-se casos de teste que cubram todas as possibilidades do componente de software. Dessa maneira, todas as variações originadas por estruturas de condições são testadas.

A técnica de teste de Estrutural é recomendada para os níveis de Teste da Unidade e Teste da Integração, cuja responsabilidade principal fica a cargo dos desenvolvedores do software, que são profissionais que conhecem bem o código-fonte desenvolvido e dessa forma conseguem planejar os casos de teste com maior facilidade.

Teste Funcional / Caixa Preta:

Técnica de teste em que o componente de software a ser testado é abordado como se fosse uma caixa-preta, ou seja, não se considera 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. Haverá sucesso no teste se o resultado obtido for igual ao resultado esperado. O componente de software a ser testado pode ser um método, uma função interna, um programa, um componente, um conjunto de programas e/ou componentes ou mesmo uma funcionalidade. A técnica de teste funcional é aplicável a todos os níveis de teste (PRESSMAN, 2005).

Teste de Análise do Valor Limite

Por razões não completamente identificadas, um grande número de erros tende a ocorrer nos limites do domínio de entrada invés de no “centro”. Esse critério de teste explora os limites dos valores de cada classe de equivalência para preparar os casos de teste (Pressman, 2005).

Se uma condição de entrada especifica uma faixa de valores limitada em *a* e *b*, casos de teste devem ser projetados com valores *a* e *b* e imediatamente acima e abaixo de *a* e *b*. Exemplo: Intervalo = {1..10}; Casos de Teste à {1, 10, 0,11}.

Fontes:  
 [www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035](http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035)

Pressman, 2005

Ferramentas de Teste “Livres”

Aqui apresentamos algumas ferramentas gratuitas para realização de testes de software, junto de suas principais características.

JUnit

JUnit é um simples framework para criar teste repetíveis para serem usados em programas escritos em Java. É uma instancia da arquitetura XUnit.

Focado em realização de testes de unidade (caixa-branca), o JUnit permite que funções e métodos de objetos sejam testados de forma automatizada.

Algumas vantagens de se utilizar JUnit:

-Permite a criação rápida de código de teste enquanto possibilita um aumento na qualidade do sistema sendo desenvolvido e testado;

-Não é necessário escrever o próprio framework;

-Amplamente utilizado pelos desenvolvedores da comunidade código-aberto, possuindo um grande número de exemplos;

-Uma vez escritos, os testes são executados rapidamente sem que, para isso, seja interrompido o processo de desenvolvimento;

-JUnit checa os resultados dos testes e fornece uma resposta imediata;

-Pode-se criar uma hierarquia de testes que permitirá testar apenas uma parte do sistema ou todo ele;

-Escrever testes com JUnit permite que o programador perca menos tempo depurando seu código;

-JUnit é livre.

O JUnit também possui versões para C#, C++, Fortran e Phyton.

Nota: *www. opensourcetesting.org*

Site com uma extensa lista de ferramentas gratuitas para teste de software.B

Fontes: wikipedia.org, junit.org, opensourcetesting.org