O Processo de Teste é separado em quatro etapas:

* Planejamento: E definido quem executa o teste, o período, os recursos e a técnica utilizada.
* Projeto de Caso de Teste: E definido o Caso de Teste que serão executados no processo.
* Execução do Programa com os Casos de Teste: O Teste é efetivamente realizado.
* Análise dos Resultados: Verificação dos resultados.

Tipos de teste:

* **Teste de regressão:**Toda vez que algo for mudado, deve ser testada toda a aplicação novamente.
* **Teste operacional:**Garante que a aplicação pode rodar muito tempo sem falhar.
* **Teste de configuração:**Testar se a aplicação funciona corretamente em diferentes ambientes de hardware ou de software.
* **Teste de integração:**Garante que um ou mais componentes combinados (ou unidades) funcionam. Podemos dizer que um teste de integração é composto por diversos testes de unidade. Pressman trata o Teste de Integração em diferentes abordagens:

***1 - Integração Descendentes:*** teste de integração descendente (top-down) é uma abordagem incremental para a construção da arquitetura de software. Os componentes são integrados movimentando-se **de cima para baixo através da hierarquia de controle da aplicação**, começando com o módulo de controle principal (programa principal). Módulos subordinados ao módulo de controle principal são incorporados à estrutura de uma maneira: primeiro-em-profundidade ou primeiro-em-largura (depth-firstou breadth-first).

***1.1 - Breadth-first (largura):*** incorpora todos os componentes diretamente subordinados a cada nível, **movimentando-se horizontalmente ao longo da arquitetura.**

***1.2 -Depth-first (profundidade):*** **integra** todos os **componentes** **num caminho de controle principal** da arquitetura.A escolha de um caminho de controle principal é bastante arbitrária e depende das características específicas da aplicação.

***2 - Integração Ascendentes:*** o teste de integração ascendente (bottom-up), como o nome diz, começa a construção e o teste com módulos atômicos (isto é, componentes nos níveis mais baixos na estrutura do programa). Devido aos componentes serem **integrados de baixo para cima**, a funcionalidade proporcionada por componentes subordinados a um dado nível está sempre disponível e a necessidade de pseudocontrolados (*stubs*) é eliminada.

* **Teste de fumaça**: o teste fumaça é uma abordagem de teste de integração usada em tempo de desenvolvimento que permite que o projeto seja avaliado frequentemente. É projetado como um mecanismo de marcapasso para projetos com prazo crítico, permitindo que a equipe de software avalie o projeto frequentemente. Em essência, a abordagem teste fumaça abrange as seguintes atividades:

1. códigos são integrados a uma construção (build);
2. testes são criados para identificar erros;
3. a construção é integrada com outras construções, o produto inteiro passa diariamente pelo teste fumaça e pode ser ascendente ou descendente.

* **Teste de carga:**Verifica o funcionamento da aplicação com a utilização de uma quantidade grande de usuários simultâneos.
* **Testes de performance** (em português desempenho) é o mais abrangente e genérico de todos. Ele é usado para referenciar qualquer tipo de avaliação que esteja sendo feito em uma aplicação ou ambiente e seu resultado não é simples como “sucesso” ou “falha”. Os resultados dos testes são usados para determinar um padrão de comportamento e capacidade da aplicação que está sendo testada. Esses limites e padrões são chamados de baseline e servem de referência para testes de carga, stress, spike ou outros que estejam sendo planejados/executados no ambiente.
* **Teste de Stress (esforço) -** Em um teste de stress, além de uma grande carga disparada contra a aplicação, alguns cenários de crash da aplicação são testados, com o objetivo também de determinar a capacidade de recuperação e estabilidade do sistema. Um exemplo é a retirada de um nó do cluster de servidores de aplicação ou uma eventual indisponibilidade do banco de dados.
* **Teste de Carga** - O Teste de Carga é aquele onde o volume gerado pela ferramenta de geração de carga é crescente no decorrer do tempo. O objetivo principal é encontrar o limite de capacidade da aplicação e identificar qual o limitante (codificação, hardware, tempo de resposta excessivo). Da mesma forma que os outros testes, não é objetivo do teste de carga encontrar problemas funcionais na aplicação.

Algumas respostas que podem ser encontradas com um teste de carga são:

* Throughput
* Limite de capacidade de hardware
* Requisitos de balanceamento de carga

Um entregável interessante de um projeto com o teste de carga é a quantidade de transações simultâneas, para sistemas baseados em serviços (web services, por exemplo), e qual a quantidade de usuários simultâneos, para sistemas de interação com o usuário (como um site).

* **Teste de Desempenho(Segundo Sommerville)**: "Os testes de desempenho devem ser projetados para assegurar que o sistema pode operar na carga necessária. **Isso envolve, geralmente, o planejamento de uma séria de testes em que a carga é constantemente aumentada até que o desempenho se torne inaceitável. No teste de desempenho, isso significa estressar o sistema (por isso o nome teste de estresse) por meio de demandas fora dos limites do projeto do software**".
* **Teste de Release** é o processo de testar um release particular de um sistema que se destina para uso fora da equipe de desenvolvimento. Geralmente, o release de sistema é para uso dos clientes e usuários.
* **Teste de Cenário** é uma abordagem de teste de release em que você imagina cenários típicos de uso e os usa para desenvolver casos de teste para o sistema.  Um cenário é uma estória que descreve uma maneira de usar o sistema. Cenários devem ser realistas, e usuários reais do sistema devem ser capazes de se relacionar com eles.

Ainda no tópico de teste de desempenho ele cita:

"O teste de estresse é particularmente relevante para sistemas distribuídos baseados em uma rede de processadores. Esses sistemas frequentemente exibem degradações graves ao serem submetidos a pesadas cargas de demanda."

Os testes de desempenho podem ser de dois tipos: carga e stress. O sommerville ao falar em testes de desempenho aborda mais explicitamente o teste de stress. Essa diferenciação fica clara no pressman. Um teste de carga, você varia as condições da carga, mas ainda dentro das condições do projeto. Já o teste de stress leva o sistema para fora das condições de projeto.

**O que contém um caso de teste:** um conjunto de entradas, uma saída esperada e o escopo do teste. Além disso, ela apresenta a impossibilidade de conseguirmos gerar casos de testes para todas as entradas possíveis de um programa, sendo necessária a adoção de estratégias como partições de equivalência para definir um conjunto de casos de teste suficiente para testar um programa.

Técnicas de Teste se dividem entre **Funcional e Estrutural**, sendo que o Teste Funcional, ou Teste de Caixa Preta (Black Box), é aquele que tem como alvo verificar se a implementação está de acordo com o que foi especificado. Já o Teste Estrutural, também chamado de Teste de Caixa Branca (White Box), busca garantir que o software desenvolvido esteja bem estruturado internamente, portanto, funcionando corretamente.

"**Objetos Mock**, **objetos simulado** ou simplesmente **Mock** (do inglês **Mock object**) em desenvolvimento de software são objetos que simulam o comportamento de objetos reais de forma controlada. São normalmente criados para testar o comportamento de outros objetos. Em outras palavras, os objetos mock são objetos “falsos” que simulam o comportamento de uma classe ou objeto “real” para que possamos focar o teste na unidade a ser testada. "

**DEF** -> **D**efeitos (universo físico) causam **E**rros (universo informação) que causam **F**alhas (universo usuário)

**Defeitos**são observados sob uma perspectiva interna, i.e., código está incorreto, lógica está inconsistente, funções estão ausentes, há problemas de hardware, etc.

**Falhas** são observadas sob uma perspectiva externa, i.e., sob o ponto de vista da percepção do usuário – travamento do sistema, terminação anormal, tela azul, etc.

**Defeitos**causam **Erros**que podem causar **Falhas**. Quando há uma diferença entre o resultado observado e o resultado esperado, temos um **erro**; quando há uma diferença entre o comportamento observado e o comportamento esperado, temos uma **falha**!

**Desenvolvimento Guiado Por Testes - TDD (Test-Driven Development)**

O **desenvolvimento dirigido a testes (TDD, do inglês Test-Driven Development) é** uma abordagem para o desen­volvimento de programas em que se intercalam testes e desenvolvimento de código. Essencialmente, você desenvolve um código de forma incremental, em conjunto com um teste para esse incremento. Você não caminha para o próximo incremento até que o código desenvolvido passe no teste. O desenvolvimento dirigido a testes foi apresentado como parte dos métodos ágeis, como o**Extreme Programming.**

Seu objetivo é criar um “código limpo que funcione”. Trabalha com a estratégia**Red - Green - Refactor**:  
  
**Codifique o teste**; “criar um teste para tarefa."  
Faça-o compilar e executar. O teste não deve passar (**Red**).  
Implemente o requisito e faça o teste passar (**Green**).  
Refatore o código (**Refactor**).

Além disso, os testes devem seguir o modelo F.I.R.S.T.  
  
**F (Fast)**- Rápidos: devem ser rápidos, pois testam apenas uma unidade;  
**I (Isolated)** - Testes unitários são isolados, testando individualmente as unidades e não sua integração;  
**R (Repeateble)**- Repetição nos testes, com resultados de comportamento constante;  
**S (Self-verifying)** - A auto verificação deve verificar se passou ou se deu como falha o teste;  
**T (Timely)**- O teste deve ser oportuno, sendo um teste por unidade.

Testes limpos(clean code) também seguem as regras do acrônimo **FIRST (Fast, Indepedent, Repeatable, Self-validation, Timely).**

**Método de caixa preta (abordagem funcional)** - 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. Principais técnicas:

* Testes baseados em grafos: Graph-based testing methods. Toda aplicação é construída por “objetos”. Essa técnica identifica todos estes objetos e gera gráficos para representá-los. Os objetos e relacionamentos são testados para descobrir erros e comportamentos inesperados.
* Partição de equivalência: é um método que divide o domínio de entrada em categorias de dados. Cada categoria revela uma classe de erros, permitindo que casos de testes na mesma categoria sejam eliminados sem que se prejudique a cobertura dos testes.
* Analise de valor limite: Em geral, erros nas fronteiras do domínio da entrada são mais frequentes do que nas regiões centrais. A análise de valor limite é uma técnica p/ seleção de casos de teste que exercitam os limites. O emprego dessa técnica deve ser complementar ao emprego da partição de equivalência. Assim, em vez de se selecionar um elemento aleatório de cada classe de equivalência, selecionam-se os casos de teste nas extremidades de cada classe.
* Matriz ortogonal

**Técnica, Nível e Tipos de Teste**

* Técnica: caixa branca, preta e cinza (COMO)
* Nível: Unidade, Integração, Sistema e Aceitação (ONDE)
* Tipo: Segurança, Funcional, Volume, Regressão, Usabilidade etc. (O QUE)

Estas dimensões não são excludentes, ou seja, qualquer teste se encaixa em algum dos itens das 3 dimensões. Um teste de sistema também é um teste de caixa preta e engloba diversos tipos de teste (usabilidade, carga, segurança etc.)

Características entre Teste estático e dinâmico

Principais características do teste estático

A análise estática de softwares, também conhecida como **whitebox**, trabalha diretamente com o código de uma ferramenta. Nesse caso, os componentes de uma ferramenta são verificados sem que o produto seja executado. Seja por meio de uma ferramenta automatizada ou dos testes manuais, o principal objetivo dessa técnica é identificar erros de programação, tais como:

* Práticas ruins;
* Erros de sintaxe;
* Falhas de segurança.

A análise estática auxilia gestores de TI a identificar todas as linhas de código que foram mal escritas durante a criação de um software. Todos os caminhos de execução, processamento e exibição de valores são examinados. Como consequência, erros mais comuns são descobertos mais rapidamente.

Principais características do teste dinâmico

O teste dinâmico pode ser empregado de forma complementar a análise estática. Esse tipo de abordagem vê o software como uma “caixa preta” (daí o nome popular “**blackbox**”) e trabalha, principalmente, com as informações que são inseridas nas rotinas de entrada e saída de dados. Além disso, são verificados itens como:

* O tempo de resposta;
* A performance da aplicação;
* A capacidade do software se adaptar a diferentes ambientes;
* O comportamento funcional.

Muitas empresas adotam a análise dinâmica por ela permitir que problemas mais sutis sejam identificados. Não importa o grau de complexidade, as chances de um bug passar por uma análise estática e uma análise dinâmica sem ser rastreado é consideravelmente baixa. Dessa forma, o teste dinâmico consegue dar mais segurança e confiabilidade ao produto final.

