I WORKSHOP PARA DESENVOLVEDORES

TESTE DE SOFTWARE

CAMPUS COLATINA

VERSÃO BETA 1.0.0

SUMÁRIO

LISTA DE FIGURAS

1.1 QUALIDADE DO SOFTWARE

É o conjunto de atividades que tem como objetivo atender de forma correta os requisitos do cliente efetuando também a prevenção e eliminação de defeitos na aplicação.

A norma internacional ISO/IEC 9126, publicada em 1991, na versão brasileira publicada em agosto de 1996 recebeu o número NBR 13696, define Qualidade de Software como: "*A totalidade de características de um produto de software que lhe confere a capacidade de satisfazer necessidades explicitas e implícitas".*

- *ISO 9126 são as características da qualidade de produtos de software.*
- *NBR 13696 é a versão brasileira da ISO 9126.*

A produção de um software possuiu diversas necessidades explicitas, onde correspondem a objetivos e condições estabelecidos por quem fez o software. As necessidades implícitas pertencem tanto ao universo do usuário quanto dos desenvolvedores, são as necessidades subjetivas do usuário.

Como podemos ver existem diversas definições respectivas a qualidade de software, porém temos que ter em mente o seguinte;

- *Obter software de qualidade com processos de desenvolvimento frágeis e deficientes é praticamente impossível;*
- *Notar que a qualidade do produto é algo bom, mas que qualidade do processo de produção é ainda mais importante;*
- *Ter requisitos funcionais e de desempenho explicitamente declarados e padrões de desenvolvimento claramente documentados.*

Desta forma podemos concluir, que para a obtenção da qualidade de um software necessitamos de duas vertentes, a qualidade do processo de desenvolvimento e consequentemente a qualidade do produto de software.

1.1.1 TESTE E MEDIÇÃO DA QUALIDADE

Cada etapa do desenvolvimento gera um agregado de documentos que torna possível a avaliação da qualidade nas fases do ciclo de desenvolvimento do software através de testes aplicados na documentação. Esses testes são chamados de testes de verificação e visam mensurar a qualidade do processo. Para que a qualidade do processo seja garantida, deve-se assegurar a qualidade dos documentos produzidos nas etapas de desenvolvimento.

Os testes que garantem a qualidade do produto são os testes de validação, pois ao produzir uma unidade do software, valida a estrutura interna e sua conformidade com os requisitos especificados e são aplicados em diversos estágios do desenvolvimento do programa.

1.1.2 CUSTO DA QUALIDADE DE SOFTWARE

A qualidade e os preços dos produtos são sempre muito questionados e exigidos pelos clientes, a competitividade seja no processamento, projetos e preços vem demonstrando que o mercado consumidor está muito atento ao adquirir um produto.

O termo custo da qualidade é usado para quantificar o custo total de prevenção e de avaliação, além dos custos de produção do software. O custo da qualidade é o investimento feito com a intenção de um produto ou serviço obter a qualidade esperada. A qualidade não é adquirida sem custos.

1.2 O QUE É TESTE DE SOFTWARE?

A resposta para essa pergunta, é que existem diversas definições para este termo, porém aqui estão algumas das definições formais consagradas de teste de software:

- *Teste é um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente. Por essa razão, deverá ser definido para o processo de software um modelo (template) para teste - um conjunto de etapas na qual pode-se colocar técnicas específicas de projeto de caso de teste e métodos de teste.* ***(Pressman)***

- *Testar é o processo de executar um programa ou sistema com a intenção de encontrar defeitos* ***(Myers)***

- *O teste de programas pode ser usado para mostrar a presença de defeitos, mas nunca para mostrar a sua ausência* ***(Dijkstra).***

Existem diversas conceituações formais para teste de software que por sua vez pode ser entendido também de forma intuitiva, em suma teste de software é detectar falhas com o objetivo de melhorar a qualidade da aplicação.

O teste visa assegurar a satisfação do cliente perante o produto, garantir que os requisitos sejam corretamente atendidos, reduzir manutenção corretiva e retrabalho.

Deve-se levar em consideração também que um software mal testado provoca prejuízos à organização e compromete a funcionalidade do teste.

2.1 O QUE É UMA FALHA, UM DEFEITO, E UM ERRO?

Normalmente quando um sistema trava ou algo do tipo, popularmente dizemos que o sistema está “bugado”, porém na programação existem termos relacionados, que costumam provocar uma certa confusão, entre: falha, defeito e erro.

- ***Falha*** *é o comportamento operacional do software diferente do esperado pelo usuário. Uma falha pode ser provocada por diversos erros, porém alguns erros podem não provocar uma falha.*

- ***Defeito*** *é a imperfeição de um produto, e surge por causa de um erro. Quando uma aplicação não está livre de erros, naturalmente é uma aplicação defeituosa.*

- ***Erro*** *é a manifestação concreta de um defeito em uma aplicação. Qualquer resultado imprevisto na execução de um software é estabelecido como erro.*

2.2 VERIFICAÇÃO E VALIDAÇÃO

Verificação e validação são atividades envolvidas com a garantia da qualidade de software. A função dessas atividades é prevenir os defeitos e diminuir os riscos do projeto.

A verificação diz respeito ao conjunto de atividades que garante que o software implemente uma função específica. Avalia se o software está sendo desenvolvido conforme os padrões e metodologia estabelecidos no projeto.

A validação diz respeito a um conjunto de atividades que garante que o software construído é equivalente ás exigências do cliente. Avalia a conformidade do software implementado em relação ao comportamento descrito nos requisitos.

2.3 DIRETRIZES BÁSICAS PARA UM TESTADOR

- *O programador não deve testar o seu programa.*
- *O teste deve ser planejado antes de ser executado.*
- *O teste deve atender as necessidades do cliente.*
- *Os resultados de cada teste devem ser profundamente analisados.*
- *50% do teste deve avaliar se o programa faz o que deveria e os outros 50% se o programa faz o que não deveria.*
- *Não planejar o teste assegurando que não haverá erros.*
- *Melhoria contínua do processo, redefinindo as técnicas e a confiabilidade, através da melhoria em: politica, norma, procedimentos e ferramentas de teste.*

2.4 DICAS PARA AUMENTAR A QUALIDADE DOS TESTES

- *Escolher uma metodologia de testes adequada ao processo de desenvolvimento;*
- *Estruturar uma organização orientada para testes com pessoal capacitado.*
- *Utilizar um ambiente de testes adequado;*
- *Utilizar ferramentas de automação dos testes;*
- *Utilizar regras de medição do projeto de testes;*
- *Utilizar mecanismos de aferição do nível de maturidade do processo de teste;*
- *Usar técnicas de gerencia de projetos (PMI).*

2.5 QUANDO PARAR OS TESTES?

Assegurar que o software não tenha nenhum defeito pode consumir muito tempo e dinheiro. Dessa forma é necessário estipular qual é o melhor momento para interromper os testes.

Dicas:

- *Diversas vezes o tempo de mercado indica o melhor momento de liberar o software e interromper os testes, embora esses possam continuar paralelamente ao lançamento do produto. O Windows XP foi lançado com milhares de defeitos e estes foram sendo corrigidos através do tempo.*

- *Quantidade de defeitos: é sempre bom manter informações sobre o número de defeitos encontrados relacionados com casos de uso, de forma que os casos de uso tem que estar identificados por algum marcador. Ou seja tem-se informações histórica de que um caso de uso complexo tem de 10 defeitos em média. Ele está sendo testado e já foram encontrados 11 defeitos. Caso o prazo estoure, tem-se um critério para interromper os testes.*

- *Quando o custo da correção dos defeitos forma superior ao custo da ocorrência do defeito.*

- *Garantir que todos os requisitos foram testados fazendo uma cobertura dos testes.*

Porém seja qual for a situação os guias para a interrupção dos testes são o custo e o tempo.

3.1 IDENTIFICANDO AS CONDIÇÕES DE TESTES

A identificação das condições de testes com a documentação do sistema fica bastante fácil, pois se trata basicamente de “cruds”, funções específicas da lógica do negócio, ou seja, qualquer parte do sistema que você possa ter exceções fora do fluxo normal do sistema são grandes concorrentes a teste.

3.2 CASOS DE TESTE

O caso de teste é um conjunto de condições usadas para teste de software. Ele pode ser elaborado para identificar defeitos na estrutura interna do software por meio de situações que exercitem adequadamente todas as estruturas utilizadas na codificação, ou ainda, garantir que os requisitos do software que foi construído sejam plenamente atendidos. Podemos utilizar os casos de uso para criar e rastrear um caso de teste, facilitando assim identificação de possíveis falhas.

O caso de teste deve especificar a saída esperada e os resultados esperados do processamento. Numa situação ideal, no desenvolvimento de casos de teste, se espera encontrar o subconjunto dos casos de teste possíveis com a maior probabilidade de encontrar a maioria dos erros.

O caso de teste é composto pelos seguintes fatores:

- ***Identificador:*** *identifica de forma única o caso de teste perante os outros casos envolvidos no projeto de teste.*
- ***Itens de teste:*** *são considerados itens de teste requisitos, projetos de teste, guia do usuário, guia operacional, guia de instalação entre outros.*
- ***Especificações de entrada:*** *especificar a origem dos dados de entrada.*
- ***Especificações de saída:*** *o que é esperado após a execução do caso de teste.*
- ***Necessidades de ambiente:*** *definir todas as necessidades de ambiente para a execução do caso de teste.*
- ***Requisitos ou procedimentos especiais****: qualquer outro requisito ou procedimento necessário par que o caso de teste possa ser executado.*
- ***Dependências entre casos de teste:*** *definir e especificar as relações de hierarquia entre os casos de teste.*

Escopo do caso de teste:

CASO DE TESTE	
CÓDIGO	Identificador do caso de teste
FINALIDADE	Definição dos objetos do casos de teste.
ENTRADAS	Valores das entradas para o caso de teste, ou identificação dos arquivos a serem utilizados como entrada.
RESULTADOS ESPERADOS	Resultados que são aceitos como corretos para o caso de teste
DEPENDÊNCIAS	Identificadores dos casos de teste que devem ser executados antes dele.
PROCEDIMENTO DE TESTE	
PREPARAÇÃO	Descrever as tarefas a serem realizadas antes do início da execução dos casos de teste, como por exemplo, preparação de ambiente.
INICIALIZAÇÃO	Descrever as tarefas necessárias para iniciar a execução dos casos de teste. EX: 1. Ligar o serviço do Apache 2. Ligar o serviço do MySQL
EXECUÇÃO	Descrever as tarefas a serem realizadas durante a execução dos casos de teste, como por exemplo, a intervenção de um operador. Descrever também como os testes são acompanhados e, se for o caso, como medidas serão observadas e registradas
REORGANIZAÇÃO	Descrever as tabelas a serem realizadas após o término normal da execução dos casos de teste.
INTERRUPÇÃO	Descrever as tarefas a serem executadas com caso de interrupção não planejada dos testes.
RECURSOS ESPECÍFICOS	Hardware, Software, pessoal ou outro recurso necessário à execução desse procedimento de teste especificadamente
AVALIAÇÃO DO CASO DE TESTE	
TESTADOR RESPONSÁVEL	
PERÍODO DE TESTE	
RESULTADOS OBTIDOS	
OBSERVAÇÃO DO TESTADOR	

Figura 1

3.3 CATEGORIAS DAS TÉCNICAS DE TESTE

Os testes são definidos basicamente em duas diferentes técnicas, técnicas estruturais e técnicas funcionais, e essas técnicas são constituídas pelos níveis de teste.

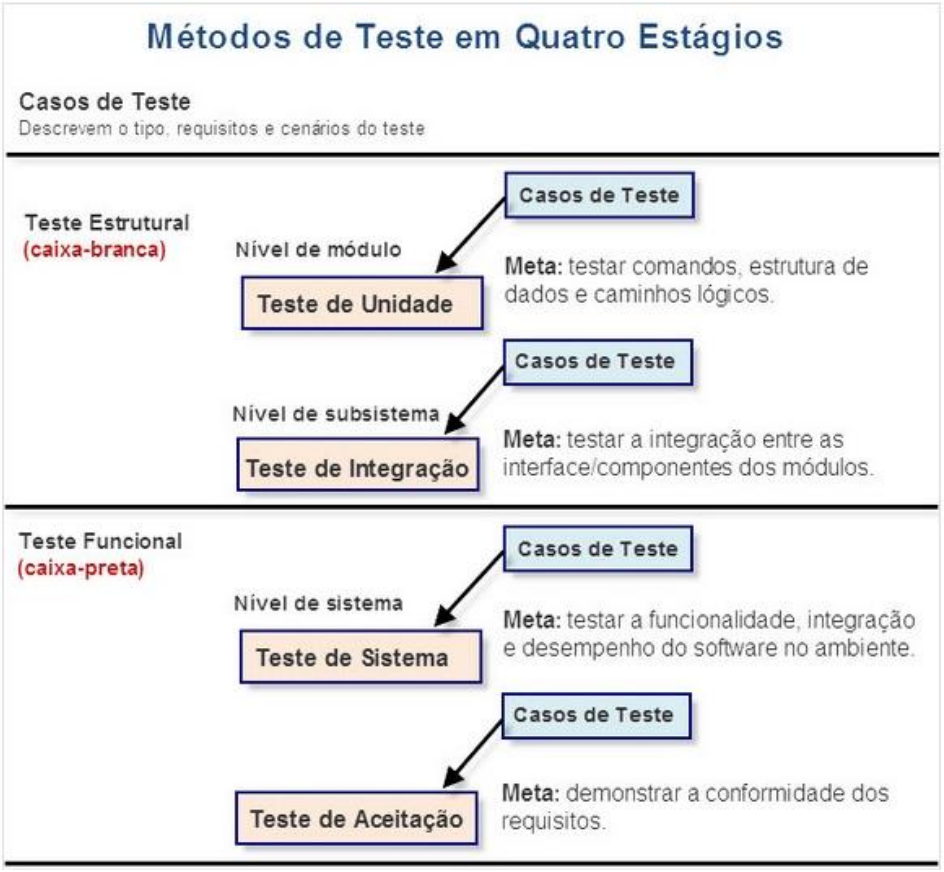

Figura 2

3.4 TÉCNICA FUNCIONAL OU CAIXA-PRETA

O teste de funcional também conhecido como teste de caixa-preta, seu nome caixa-preta tem o significado de dizer que não há a necessidade do conhecimento da estrutura interna da tecnologia ou do código utilizado.

Como não há conhecimento sobre a operação interna do programa, o analista se concentra nas funções que o software deve desempenhar, é recomendado a utilização desta técnica em conjunto com a técnica de caixa-branca, pois também é de suma importância efetuar a análise da estrutura interna da aplicação.

O teste de caixa-preta pode ser relacionado como se fosse a visão do usuário, ou seja, quer utilizar o programa sem se atentar aos detalhes de sua construção. Este técnica é considerada fácil para ser realizada, pois necessita apenas do conhecimento dos requisitos, características e comportamentos esperados do sistema, para que a partir dessas especificações seja determinada as saídas esperadas para certos conjuntos de entrada de dados.

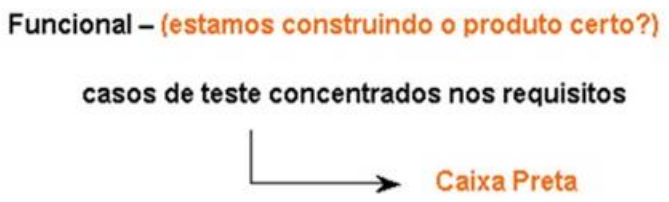

Figura 3

O teste de caixa-preta é eficiente especificadamente para encontrar problemas dos tipos:

- *Funções incorretas ou omitidas;*
- *Erros de interface;*
- *Erros de comportamento ou desempenho;*
- *Erros de início e término.*

3.5 TÉCNICA ESTRUTURAL OU CAIXA-BRANCA

Os testes estruturais também denominados de teste de caixa-branca, é basicamente o inverso do teste de caixa-preta, ou seja, este visa sua análise na arquitetura interna do sistema, ele tem por objetivo encontrar defeitos no código ou na estrutura interna do sistema, além disso também visa garantir que após os testes, todas as linhas de código e condições estejam corretas.

Este teste é feito com relação à estrutura do componente permitindo uma verificação mais precisa do produto de software. Então proporciona ao avaliador focar sua atenção para os pontos mais perigosos ou importantes do código de uma forma mais precisa.

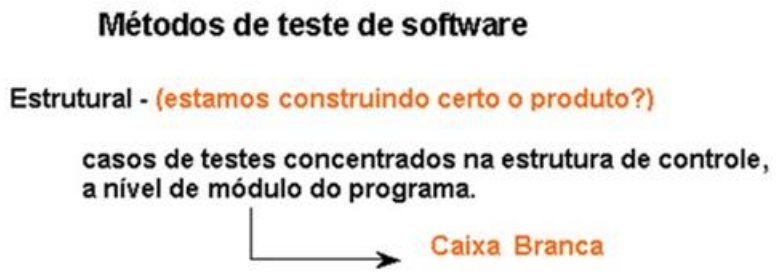

Figura 4

O teste de caixa-branca prega a filosofia de garantir que:

- *Todos os caminhos independentes de um módulo tenham sido executados pelo menos uma vez;*
- *Exercitem todas as decisões lógicas para valores falsos ou verdadeiros;*
- *Executem todos os laços em suas fronteiras e dentro de seus limites;*
- *Exercitem as estruturas de dados internas para garantir a sua validade.*

4.1 MODELOS DE DESENVOLVIMENTO DE SOFTWARE

Este tópico serve somente como uma simples abordagem dos modelos de desenvolvimento de software e também para dar ênfase neste fator, pois todo o processo de teste está ligado diretamente com o tipo do modelo de ciclo de vida que o produto de software está utilizando no seu processo de construção.

4.2 NÍVEIS DE TESTE OU ESTÁGIO DE TESTE

Os níveis de teste tem o intuito de efetuar a indicação sobre o foco do teste e os tipos de problemas a serem encontrados, dependendo do nível em que o teste será realizado. O termo níveis de teste está relacionado com o modelo V de desenvolvimento, que representa quando as atividades de teste acontecem em decorrência do ciclo de vida do software. No entanto, o conceito de estágios de teste também pode ser aplicado ao desenvolvimento iterativo. Quando o teste é realizado no momento da codificação e desenvolvimento do sistema.

Estes níveis de teste possuem uma hierarquia, que quando comparado com o processo de desenvolvimento de software, nota-se facilmente esta hierarquia. Os mesmos são divididos basicamente em: *teste unitário, teste de integração, teste de sistema, teste de aceitação,* logo em seguida iremos fazer uma conceituação de cada nível de teste.

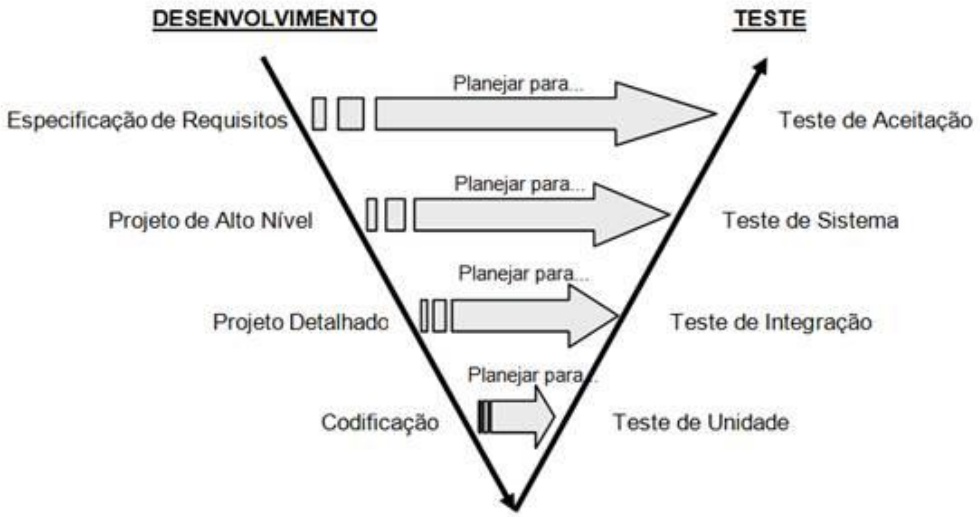

Figura 5

Esta figura 5 tem o intuito de basicamente dar ênfase que os processos do ciclo de desenvolvimento e teste têm início simultâneo, ou seja, o ciclo de vida do desenvolvimento e o ciclo de vida de testes são interligados, porém, o ciclo de teste necessita que os produtos das atividades de desenvolvimento sejam concluídos, para poder começar o seu ciclo.

Desde que o software começou a ser construído hierarquicamente (unidades criam módulos, módulos geram subsistemas e subsistemas criam sistemas), testar cada entidade separadamente pode ser entendido como uma vantagem para os testes. Sucessivas fases de testes visam avaliar largamente as entidades do produto, cada estágio objetiva trabalhar cada entidade com os requisitos do usuário (se houver), especificados e projetados.

Produtos diferentes terão diferentes estágios, pois depende da estrutura do produto e das práticas de desenvolvimento de software utilizadas para o produto. Nesse sentido, se uma organização utiliza o modelo em cascata, então provavelmente os estágios de testes serão divididos em unitários e testes de sistemas, devido ao tipo de estrutura

Os testes são organizados em fases, o que simplifica para o entendimento por parte da equipe que está conduzindo, bem como ampla a cobertura dos testes e a eficiência da detecção de defeitos. Algumas categorias, por terem menor criticidade, podem ser agrupadas, bem como deve haver uma preocupação maior com as categorias, mais complexas e importantes.

4.2.1 TESTE DE UNIDADE

Os testes de unidade são realizados especificadamente em uma unidade independente do produto de software, este teste é considerado o primeiro nível de teste, ele é quem envolve assegurar que cada funcionalidade especificada no fase de análise do desenvolvimento do software tenha sido implementada de forma correta nesta unidade ou componente, ou seja, seu foco é validar a menor unidade de projeto de software (módulo ou função) para garantir que a lógica do programa esteja completa e correta.

E para que o teste de unidade possa garantir essa validação o mais próximo do seu objetivo, deve-se validar as seguintes partes:

- *A **interface com o módulo** para validar se as informações que entram e saem estão compatíveis com o resultado esperado.*

- *A **estrutura de dados local** para validar se realmente os dados que estão armazenados temporariamente estão com sua integridade preservada durante todos os passos da execução do código.*

- *As **condições de limite** para validar se os limites que foram declarados para restringir ou demarcar o processamento estão garantindo a execução de forma coerente da unidade.*

- *Os **caminhos básicos** neste caso deve ser efetuado a execução pelo menos uma vez de todas as instruções contidas na unidade, com o objetivo de verificar se os resultados condizem com o esperado.*

- *Os **caminhos de tratamento de erro** além do fluxo normal (caminhos básicos), os de tratamento de erro todos também devem ser executados, com o intuito de verificar se o tratamento está fazendo o que lhe foi estipulado.*

Neste ponto é importante deixar claro que testes de unidade são diferentes que depuração de código fonte. A depuração trabalha com problemas específicos da linguagem e do compilador utilizado, já os testes de unidade trabalham com a função do sistema, a qual o desenvolvedor entende estar pronto.

Normalmente os testes de unidade podem ser complicados, mas devem ser sempre feitos, porque uma das tarefas de um desenvolvedor é entregar seu código testado, além disso, é muito mais fácil encontrar problemas nesta parte do desenvolvimento do que em testes de sistemas.

Normalmente um teste de unidade é feito logo quando um módulo acaba de ser programado, para efetuar estes testes, pode ser feito da seguinte maneira:

- *Criar um "main" onde será chamado o módulo com os dados de teste e que imprima os resultados.*

- *Outra opção, seria efetuar a criação de funções simples, onde o módulo as chamas, e que elas executem o mínimo de processamento possível ou que as mesmas retornem os dados esperados.*

É bom lembrar que estas funções extras geram despesas extras, pois também são programas que têm que ser desenvolvidos, por isso, atualmente ferramentas para auxiliar nesses processo, estaremos abordando estas ferramentas em outro tópico mais à frente na apostila.

4.2.2 TESTE DE INTEGRAÇÃO

Este teste é considerado o segundo nível de teste, ou seja, estes testes são efetuados após os testes de unidade. Os testes de integração não tem seu foco voltado para o quê os componentes fazem (papel do teste de unidade), este tipo de teste foca em dar ênfase na verificação dos componentes, analisando se os mesmos estão se comunicando de forma coerente de acordo com o que foi especificado no projeto.

Então pode-se dizer explicitamente que o teste de integração tem o intuito de verificar se as unidades que foram testadas individualmente, se quando são integradas funcionam como o esperado.

A integração destas unidades deve ser feita de forma incremental, ou seja, não é recomendado efetuar a integração de todas as unidades ao mesmo tempo, pois, se torna mais fácil encontrar os erros quando o teste avalia partes menores do que quando se testa o sistema como um todo. Portanto a utilização do ciclo de teste iterativo é bem adequada para esta situação.

Quando utiliza-se essa nível de teste incremental existe a necessidade de ser construído módulos para o apoio, esses módulos são divididos em: Drivers e Stubs.

- ***Driver** é o módulo que será responsável de chamar os módulos que irão ser testados, no seu corpo existe apenas a inicialização de variáveis globais, inicialização de parâmetros e chamada de rotinas necessárias.*

- ***Stub** este módulo por sua vez é chamado pelos módulos que estão sendo testados, no seu corpo basicamente é composto por atribuição de valores que serão retornados.*

O teste incremental ainda tem outros fatores de importância, que é a estratégia que será utilizada, as principais estratégias utilizadas são: topo-down e a bottom-top,

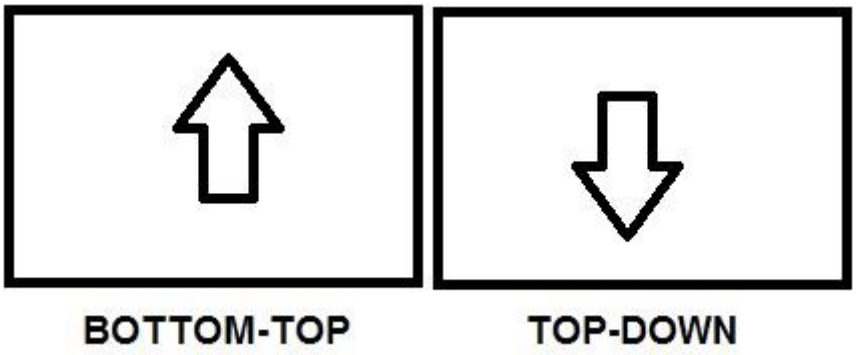

Figura 6

TOP-DOWN: neste tipo de estratégia o teste inicia-se no módulo de maior hierarquia e simulando os módulos inferiores que fazer interface, para efetuar o teste de um módulo superior, é necessário usar stubs. Onde os módulos de stubs não deveram somente conter em seu corpo mensagens relatando se o fluxo de controle passou por eles, além os sutbs devem conter em seu corpo retornos de valores ou até realização de funções específicas. Caso haja um erro na construção do stub, consequentemente existirá grandes chances de ocorrer uma falha no sistema e a detecção do mesmo levará a custos e tempo perdidos.

Vantagens:

- *Normalmente é usada na fase inicial do projeto;*
- *Mais rápido para ser informado ao cliente;*
- *Útil para medir o interesse do cliente no projeto;*
- *Utilizada em projeto quando não temos informações detalhadas.*

Desvantagens:

- *Baseado em experiências anteriores;*
- *Deve ser realizado por um especialista;*

BOTTOM-TOP: *a*o contrário da outra estratégia, esta começa sua abordagem o início do teste a partir do módulo de nível hierárquico mais baixo, usando controladores para simular a interface com os módulos de nível hierárquico mais alto. No teste de bottom-up os componentes de nível inferior são interligados e depois testados antes mesmo que os componentes de nível superior estejam prontos. É apropriado para o desenvolvimento orientado a objetos.

Vantagens:

- *Maior precisão;*
- *Desenvolve o comprometimento da equipe;*
- *Baseado na análise detalhada do projeto;*
- *Fornece a base para monitoramento e controle do projeto;*

Desvantagens:

- *Maior esforço de tempo, custo e recursos para desenvolver a estimativa;*
- *Requer que o projeto esteja bem definido e entendido;*
- *O programa funcional passa a existir somente quando o último módulo é integrado*

4.2.3 TESTE DE SISTEMA

O teste de sistema efetua uma série diferentes de testes simulando um ambiente de produção real, com um único intuito, que é validar se o software está com seu funcionamento dentro do esperado, ou seja, se todo o sistema está correto.

O tamanho do teste de sistema depende diretamente da quantidade de testes unitários executados ao longo do ciclo de teste, isso também é considerado para testes modulares e de subsistema e testes de integração, ambos feitos antes dos testes de sistema. Isso significa que os testes anteriores ao teste de sistema são relevantes e geram consequências positivas ou negativas para estes tipos de testes, ou seja, se os testes anteriores foram bem feitos, gerará consequências positivas, caso contrário trará consequências negativas.

4.2.4 TESTE DE ACEITAÇÃO

Basicamente é o teste efetuado pelo usuário final, porém com o auxílio de alguma forma dos desenvolvedores, este é o último nível de teste, então ele verifica se o produto de software desenvolvido atende a todos os requisitos.

O objetivo deste nível é evitar desagradáveis surpresas futuras, ou seja, é feito a comparação dos requisitos iniciais em relação com as necessidades do usuário final, lembrando que neste nível pequenas correções podem ser feitas para ajuste do sistema.

O teste de aceitação pode ser feito de duas maneiras: *teste alfa e teste beta.*

- ***Teste alfa** é realizado em um ambiente controlado onde o cliente usa o sistema no próprio local onde o sistema foi desenvolvido, utilizando o sistema como se estivesse em seu ambiente de trabalho, a grande é justamente que a maioria dos clientes não conseguem se comportar de forma natural, como se estivesse no seu ambiente de trabalho.*

- ***Teste beta** é o tipo mais utilizado, onde o cliente utiliza o sistema já no seu próprio ambiente de trabalho, neste caso os desenvolvedores não estão acompanhando o cliente, então, o cliente tem que registrar todos os problemas encontrados e depois repassar os mesmos para os desenvolvedores.*

Este plano teste pode ter o formato simples de um checklist, ou então ser formado por uma lista de procedimentos a serem realizados. Em ambos os casos o plano deve abranger todos os requisitos funcionais e de desempenho do sistema. Além disso a documentação do usuário também deve ser utilizada neste momento.

4.3 TIPOS DE TESTE – ALVO DO TESTE

Neste tópico será feita somente uma breve introdução ao conceitos dos tipos de testes mais utilizados, em outro capítulo da apostila abordaremos mais a fundo estes tipos e sua respectiva técnica.

TESTES ESTRUTURAIS (CAIXA-BRANCA):

- ***TESTE DE ESTRESSE****: o foco deste teste é analisar o comportamento do software sob condições críticas, tais como restrições significativas de memória, área de disco e de CPU.*

- ***TESTE DE EXECUÇÃO:*** *conhecido também como o teste de performance, tem o intuito de verificar se o sistema está se comportando como o esperado no seu ambiente, e também validar se todas as premissas relacionadas ao desempenho estão sendo atendidas.*

- ***TESTE DE RECUPERAÇÃO:*** *utilizado para analisar a integridade do sistema, ou seja, ele serve para garantir a continuidade das operações, caso ocorra uma falha, este teste irá verificar a eficácia dos componentes pertencentes ao processo e também verificar o processo de recuperação do sistema.*

- ***TESTE DE OPERAÇÃO:*** *este teste visa verificar se os procedimentos da produção e os operadores estão aptos a executar de forma adequando a aplicação, porém, essa verificação é feita antes do sistema funcionar em seu ambiente real.*

- ***TESTE DE CONFORMIDADE:*** *é o tipo de teste feito para garantir a conformidade, ou seja, para validar se realmente o produto de software foi desenvolvido de acordo com o que foi definido.*

- ***TESTE DE SEGURANÇA:*** *como seu próprio nome diz, este teste efetua o teste para garantir que a proteção dos dados e a confiabilidade das informações sejam protegidas.*

TESTES FUNCIONAIS (CAIXA-PRETA):

- ***TESTE DE REQUISITOS:*** *verificar se o sistema executa corretamente as funcionalidades e se é capaz de sustentar essa correção após a sua utilização.*

- ***TESTE DE REGRESSÃO:*** *o intuito principal deste teste é basicamente verificar se alterações feitas no sistema, não ocasionaram problemas em outras áreas que antes funcionavam sem problemas.*

- ***TESTE DE TRATAMENTO DE ERROS:*** *ele determina qual é a capacidade do sistema em relação ao tratamento de erros de transações incorretas.*

- ***TESTE DE SUPORTE MANUAL:*** *este teste está voltado para verificar a qualidade do suporte que a equipe do desenvolvimento do produto de software irá proporcionar ao seu cliente final naquele sistema.*

- ***TESTE DE INTERCONEXÃO:*** *consiste em passar dados previstos entre diversos softwares envolvidos e validar se a transferência foi adequadamente realizada.*

- ***TESTE DE CONTROLE:*** *o seu intuito é garantir que os mecanismos que supervisionam o funcionamento dos sistemas, funcionem corretamente.*

- ***TESTE PARALELO:*** *é utilizado para comparar os resultados da versão antiga do sistema com relação ao novo sistema, assim verificando se os dados estão consistentes.*

Como o próprio nome diz essas ferramentas tem o intuito de auxiliar no processo de teste, atualmente no mercado existem diversas ferramentas de automatização de teste. Mas também é interessante frisar que é bastante improvável que você irá encontrar no mercado uma única ferramenta que automatize todas as atividades de teste, pois a grande parte das ferramentas dão ênfase em determinado grupo de atividades ou uma atividade em específico.

5.1 TIPOS DE FERRAMENTAS DE TESTE

As ferramentas também possuem uma divisão de tipos, de acordo com a função que exerce, abaixo iremos citar os tipos mais comuns e mais utilizados, abordando o seu conceito.

- ***Ferramentas de aquisição de dados:*** *esse tipo de ferramenta adquire os dados que serão usados nas atividades de teste, é possível adquirir esses dados por meio de conversão, extração, transformação, e até mesmo por captura dos dados existentes, ou pela geração a partir de casos de uso ou especificação suplementares.*

- ***Ferramentas de medição estática:*** *essas ferramentas tem o intuito principal de pegar as informações contidas nos modelos de design, código fonte ou em outras fontes fixas. Ao final a análise irá produzir informações respectivas ao fluxo lógico, fluxo de dados, complexidade, capacidade de manutenção ou linhas de código.*

- ***Ferramentas de medição dinâmicas:*** *este tipo de ferramenta executa uma análise em tempo de execução, ou seja, analisa as informações durante a execução do código. Essa análise obtêm medições, e essas por sua vez incluem informações como: operação do código em tempo de execução, desempenho, detecção de erros na memória entre outras.*

- ***Simuladores ou Geradores:*** *estes tem o conceito bem simples, que é executar as atividades que não estão ou não podem estar disponíveis para fins de teste, por questões como: tempo, custo ou até segurança.*

- ***Ferramentas de gerenciamento de teste:*** *como seu próprio nome exemplifica, essas ferramentas tem o intuito de auxiliar no planejamento, implementação, execução, avaliação e gerenciamento do teste.*

5.2 FERRAMENTAS EXISTENTES NO MERCADO

Como foi relatado no tópico acima as ferramentas são divididas em alguns tipo, então aqui iremos abordar o nome de algumas ferramentas existentes no mercado de acordo com o seu tipo.

Abaixo o nome de algumas ferramentas de teste pagas, porém, também existem diversas ferramentas de teste open source, onde as melhores são encontradas no site oficial de ferramentas open source para teste, basta acessar este link:

- http://www.opensourcetesting.org/functional.php

FERRAMENTAS

Medição estática:

- IBM Rational Perfomance Testing
- Rational Functional Tester
- HP QuickTest Professional
- HP LoadRunner
- TestComplete

Medição dinâmica:

- Rational Robot
- Rational Functional Tester
- HP QuickTest Professional
- HP LoadRunner
- TestComplete
- JMeter
- Bugzilla
- BugNET
- TRAC

Simuladores:

- IBM Rational Perfomance Testing
- Rational Robot
- Rational Functional Tester
- HP QuickTest Professional
- HP LoadRunner
- JMeter

Gerenciamento de Teste:

- TestLink
- Mantis
- TestMaster

6 REFERÊNCIAS BIBLIOGRÁFICAS

Pressman, R. S. (2011). *Engenharia de Software Uma Abordagem Profissional.* São Paulo: AMGH Editora.

Sommerville, I. (2003). *Engenharia de Software.* São Paulo: Addison Wesley.

TreinaWeb Tecnologia (2014). *Curso Teste de Software*. Fonte: Treina Web Cursos: http://www.treinaweb.com.br/