- MÓDULO 4 -
TESTE DE SOFTWARE

1. INTRODUÇÃO

O teste do software é a investigação do software a fim de fornecer informações sobre sua qualidade em relação ao contexto em que ele deve operar. Isso inclui o processo de utilizar o produto para encontrar seus defeitos.

O teste é um processo realizado pelo testador de software, que permeia outros processos da engenharia de software, e que envolve ações que vão do levantamento de requisitos até a execução do teste propriamente dito.

Falhas no sistema podem ser originadas por diversos motivos, como especificação errada ou incompleta, ou ainda conter requisitos impossíveis de serem implementados devido a limitações de 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. A qualidade da aplicação pode e, normalmente, varia significativamente de sistema para sistema.

2. 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 que ainda hoje têm grande valia para os sistemas orientados a objeto. Apesar de os paradigmas de desenvolvimento serem completamente diferentes, o objetivo principal destas técnicas continua a ser o mesmo, encontrar falhas no software.

Nesta fase de testes, o engenheiro cria uma série de testes que têm a intenção de demolir o software que ele construiu.

Este teste tem os seguintes objetivos:

- A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro;

- Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto;

- Um teste bem sucedido é aquele que revela um erro ainda não descoberto.

Se a atividade de teste for bem conduzida, ela descobrirá erros no software. Como benefício secundário, a atividade de teste demonstrará que as funções de software aparentemente estão trabalhando de acordo com as especificações, que os requisitos de desempenho aparentemente foram cumpridos. Além disso, os dados compilados quando a atividade de testes é levada a efeito proporcionam uma boa indicação da confiabilidade de software e alguma indicação da qualidade do software como um todo. Mas há uma coisa que a

atividade de teste não pode fazer: a atividade de teste não pode demonstrar a ausência de bugs; ela só pode mostrar se defeitos de software estão presentes.

2.1. Projeto de Casos de Teste

Qualquer produto trabalhado por engenharia pode ser testado de duas maneiras:

- Conhecendo-se a função específica que um produto projetado deve executar, testes podem ser realizados para demonstrar que cada função é totalmente operacional;

- Conhecendo-se o funcionamento interno de um produto, testes podem ser realizados para garantir que "todas as engrenagens se encaixam", ou seja, que a operação interna do produto tem um desempenho de acordo com as especificações e que os componentes internos foram adequadamente postos à prova.

2.2. Caixa Preta (*Black box ou behaviour testing*)

Encara um programa/componente como uma “caixa preta”. É baseado na especificação do sistema, e não no código fonte. O analista não tem acesso ao código fonte e desconhece a estrutura interna do sistema. É também conhecido como teste funcional, pois é baseado nos requisitos funcionais do software. O foco, nesse caso, é nos requisitos da aplicação, ou seja, nas ações que ela deve desempenhar.

Para mostrar quais problemas que esse tipo de teste rastreia, podemos citar alguns exemplos:

- Data de nascimento preenchida com data futura;

- Campos de preenchimento obrigatório que não são validados;

- Utilizar números negativos em campos tipo valor a pagar;

- Botões que não executam as ações devidas;

Enfim, todo tipo de falha funcional, ou seja, falhas que contrariam os requisitos da aplicação.

Figura 1 - Teste de Caixa Preta

2.3. Caixa Branca (*White box*)

É baseado no código fonte do sistema. O analista tem acesso ao código fonte, conhece a estrutura interna do produto sendo analisado e possibilita que sejam escolhidas partes específicas de um componente para serem avaliadas. Esse tipo de teste, também conhecido como teste estrutural, é projetado em função da estrutura do componente e permite uma averiguação mais precisa do comportamento dessa estrutura. Perceba que o acesso ao código facilita o isolamento

de uma função ou ação, o que ajuda na análise comportamental das mesmas.

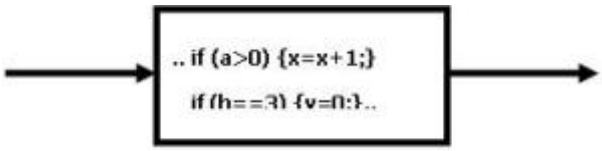

Figura 2 - Teste de Caixa Branca

Realizar um teste de caixa branca é importante, pois quando um programa é traduzido para código-fonte de linguagem de programação, é provável que alguns erros de digitação ocorram. Muitos deles serão descobertos por mecanismos de verificação de sintaxe, mas outros passarão a ser detectados até que os testes se iniciem. É provável que exista um erro tipográfico tanto num caminho lógico obscuro como num caminho da corrente principal.

Dos testes de caixa branca, identificam-se os seguintes:

2.3.1. TESTE DO CAMINHO BÁSICO

O teste é uma técnica de caixa branca, onde calcula se a complexidade lógica do software e utiliza esta medida como base para descobrir os caminhos básicos do software e exercendo o teste de modo que todos os caminhos sejam efetuados (PRESSMAN, 2006).

2.3.2. NOTAÇÃO DE GRAFO DE FLUXO

Para usar a técnica de teste de caixa branca o código fonte deve estar pronto, neste código fonte terminada, extrai se o grafo de fluxo que representa a lógica do código fonte.

O grafo de fluxo é um gráfico que demonstra a lógica do código fonte através de fios e ramos. De acordo PRESSMAN(2006) na construção do grafo de fluxo existem representações simbólicas correspondentes do grafo de fluxo. Usando o grafo de fluxo consegui visualizar os controles lógicos do software. Para cada círculos (ramos) demonstra uma ou varias linhas do código fonte e para cada setas (arestas) mostra o caminho ou caminhos que o código fonte pode fazer. Quando a existência de condições composta torna se mais difícil à construção do grafo de fluxo, encontra se quando ocorrem operações booleanas (ou, e, não-e, não-ou lógicos).

Todos os grafo de fluxo são construído através uma base principal de grafos tais como: sequência, se, enquanto, até que e o caso.

A representação da figura 3 demonstra conjuntos de notações bases de grafos que auxiliam para construções de grafos de fluxo.

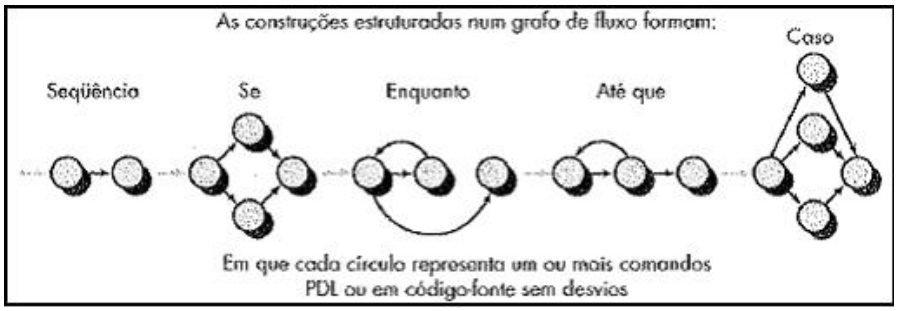

Figura 3 - Representação base para o grafo de fluxo (PRESSMAN, 2006).

Na representação da listagem 1 demonstra um código fonte que realiza um calculo potencial.

Listagem 1- Representação do código fonte de calculo potencial

```
/*1*/      import javax.swing.JOptionPane;
/*1*/      public class PotencialTeste {
/*1*/      public static void main( String args[] ){
/*1*/           String PrimeiroNumero,SegundoNumero;
/*1*/           float Base, Expoente,Resultado, Potencial;
/*1*/           PrimeiroNumero = JOptionPane.showInputDialog("Entra com a base:");
/*1*/           SegundoNumero = JOptionPane.showInputDialog("Entra com o expoente:");
/*1*/           Base = Integer.parseInt(PrimeiroNumero);
/*1*/           Expoente = Integer.parseInt(SegundoNumero);
/*1*/           if (Expoente < 0 ){
/*2*/               Potencial=0-Expoente;
/*3*/           }
/*3*/           else {
/*3*/               Potencial=Expoente;
/*4*/           }
/*4*/           Resultado=1;
/*4*/           while (Potencial !=0 ){
/*5*/               Resultado = Resultado * Base;
/*5*/               Potencial=Potencial-1;
/*5*/           }
/*6*/           if (Expoente<0 && Base !=0){
/*7*/               Resultado=1/Resultado;
/*8*/           }
/*8*/           else{
/*8*/               if(Base ==0){
/*9*/                JOptionPane.showMessageDialog(null, "A potencia é um valor finito");
                    }
/*10*/          }
/*10*/     JOptionPane.showMessageDialog(null, "A potencia é " + Resultado, "Resultado",
JOptionPane.PLAIN_MESSAGE);
/*10*/     System.exit( 0 );
```

```
/*10*/    }
/*10*/}
```

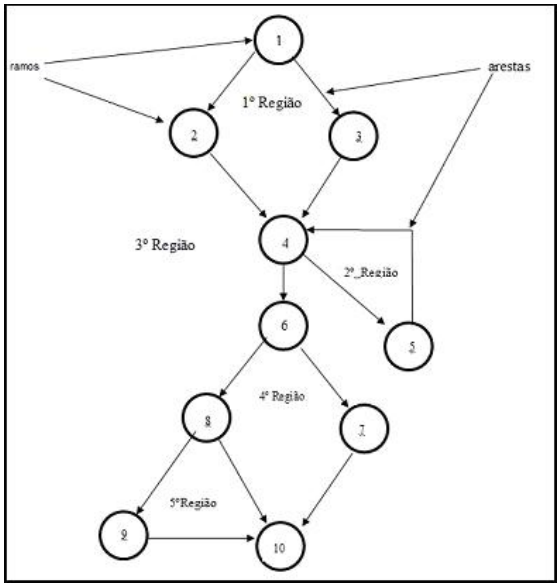

Figura 4 - Representação do Grafo de Fluxo da Listagem 1

A demonstração da figura 4 representa o grafo de fluxo do código fonte do calculo potencial representado na listagem 1, que podemos ver que os ramos estão sendo representados pelos números que estão ao lado esquerdo do código fonte e as arestas são as direções possíveis.

2.3.3. CAMINHO INDEPENDENTE DE PROGRAMA

Seria algum caminho ao longo do código fonte que execute um novo comando e no grafo de fluxo seria uma nova área que não foi exercida antes (PRESSMAN, 2006). Com esta ferramenta permite calcular um valor para os atributos do software e o calculo de complexidade ciclomática (V(G)) para o grafo de fluxo.

V(G) é o calculo de complexidade da decisão estrutural do código fonte. É o numero de caminhos independentes possíveis e o numero mínimo de caminhos que pode ser testado para garantir que o código esteja livre de defeito (MCCABE, 2010).

Conforme Pressman (2006) a V(G) podem ser calculadas de três formas:

1. Pelos números de regiões do grafo de fluxo.

2. V(G)= E – N + 2.

Onde:

V(G) = e a complexidade ciclomática.

G = representa o grafo de fluxo.

E = representa a quantidade de arestas no grafo.

N = representa a quantidade de ramos no grafo.

3. V(G)= P + 1.

Onde:

V(G) = e a complexidade ciclomática.

G = representa o grafo de fluxo.

P = representa a quantidade de ramos predicativos.

Através dos grafos de fluxo representados na figura 4 serão retirados os valores dos V(G) dessas formas:

1. Cinco regiões.
2. V(G) = 13 arestas – 10 ramos + 2 = 5.
3. V(G) = 4 ramos predicativos + 1 = 5.

Por estas formulas calculada chegamos que o grafo de fluxo usado tem cinco caminhos diferentes para testar o código fonte por completo. Tais como:

1. Caminho 1: 1-2-4-5-4-6-7-10.
2. Caminho 2: 1-2-4-5-4-6-8-9-10.
3. Caminho 3: 1-3-4-6-8-10.
4. Caminho 4: 1-3-4-5-4-6-8-10.
5. Caminho 5: 1-3-4-5-4-6-8-9-10.

2.3.4. MATRIZES DE GRAFOS

A maneira para transforma o grafo de fluxo e executar os caminhos possíveis no grafo de fluxo de forma automática, é usada uma estrutura de dados. Uma matriz quadrada é usada e o tamanho é igual à quantidade de ramos encontrados no grafo de fluxo, onde cada linhas e colunas da matriz são correspondentes às quantidades de ramos (PRESSMAN, 2006).

A demonstração da figura 5 representa a utilização da matriz de grafos para um grafo de fluxo qualquer. As linhas e colunas são as mesmas quantidades de ramos e as arestas que são representadas as ligações entre os ramos por letras.

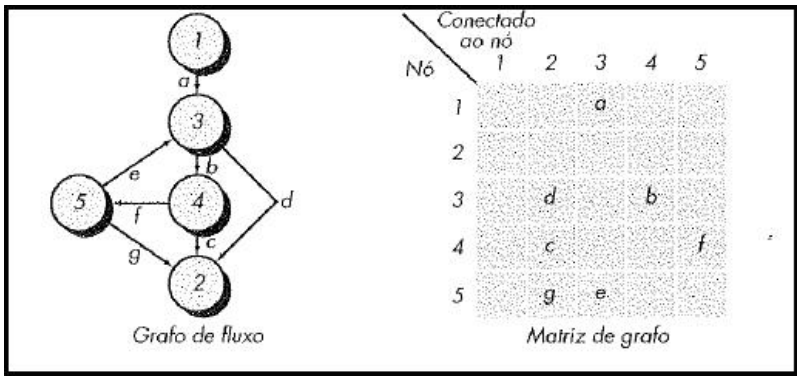

Figura 5 - Representação da Matriz de Grafos (Pressman, 2006)

2.3.5. TESTE DE ESTRUTURA DE CONTROLE

Com o uso do teste de estrutura de controle serve como um complemento para o teste do caminho básico e assim garantir uma alta qualidade para técnica de caixa branca (PRESSMAN, 2006).

2.3.6. TESTE DE CONDIÇÃO

Este tipo de teste são feitos nas condições booleana simples ou composto para analisar os desvios possíveis existentes, onde o teste examina os lados positivos ou falsos da condição booleana.

Uma condição encontra-se errada das seguintes formas:

1. Erros de operador booleano,
2. Erros de variáveis booleana,
3. Erros de parêntese booleano,
4. Erros de operador relacional e
5. Erros de expressão aritmética (PRESSMAN, 2006).

2.3.7. TESTE DE CICLO

O teste de ciclo é uma técnica de caixa branca que concentra se na validação da construção de ciclo (PRESSMAN, 2006).

Os ciclos têm uma utilização frequente na estrutura de controle. Com isso muitos erros são encontrados nas construções deles.

São classificados quatro tipos de ciclos como: ciclos simples, concatenados, aninhados e desestruturados (PRESSMAN, 2006).

- **Ciclos simples**: O teste é aplicado em no ciclo com o tamanho n de modo que:

a) Pule o ciclo completamente;

b) Uma passagem pelo ciclo;

c) Duas passagens pelo ciclo;

d) k passagens pelo ciclo em que k< n.

e) n -1 passagens pelo ciclo;

f) n +1 passagens pelo ciclo (BURNSTEIN, 2003).

- **Ciclos concatenados**: Utiliza no ciclo concatenado a abordagem de ciclos simples para o caso dos ciclos forem independente um do outro, caso contrario é recomendado o uso da abordagem de ciclos aninhados (PRESSMAN, 2006).

- **Ciclos aninhados**: Para esta abordagem testa se os ciclos mais internos que são aplicados à abordagem de ciclos simples e os outros ciclos externos permanecem com o valor

mínimo (PRESSMAN, 2006).

- **Ciclos desestruturados**: Sempre que ocorrer este caso deve se refazer o ciclo para pensar no uso da construção da programação estruturada (PRESSMAN, 2006).

Na demonstração da figura 6 representam os quatros tipos dos ciclos.

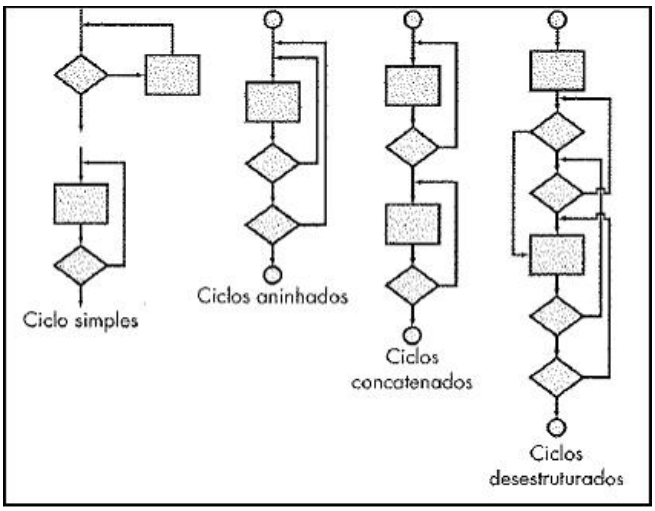

Figura 6 - Classificação de Ciclos Existentes (PRESSMAN, 2006)

2.3.8. TESTE DE DECISÃO

Conseguir a cobertura inteira dos comandos sem executar todos os desvios existentes do código fonte. E conseguir com um único caso de teste a adequação de desvio para que o desvio do código fonte seja exercido.

2.3.9. TESTE DE FLUXO DE DADOS

Conforme PRESSMAN (2006) o teste de fluxo de dados descobre os caminhos para que teste o código fonte, selecionar as definições (DEF) e uso das variáveis do código fonte. Para demonstrar o fluxo de dados de cada comando do código fonte é representada por números.

Esta abordagem de teste de software demonstra que na DEF das variáveis ao longo do grafo de fluxo localiza se caminhos simples. Assim torna o teste de caixa branca mais poderoso.

Com este teste de software verifica se o comportamento das variáveis ate localizar algum defeito que tenha passado despercebido e a propagação dele no código fonte.

As principais categorias do fluxo de dados são:

- Bloco básico ou ramos,
- Todos os usos,
- Todos usos computacionais (c-uso),
- Todos usos predicativos (p-uso) e
- Caminho livre de def (Todos-du-Caminhos).

O bloco básico ou ramo são trechos de códigos fonte que são executados todos de uma vez. Todos o c-uso, todos o p-uso e todos usos são categorias mais mencionadas do fluxo de

dados.

Todos os usos solicitam todas DEF de variáveis, seus p-uso e seus c-uso que sejam realizados no caso de teste pelo menos uma vez pelo caminho livre de def. Toda DEF solicita que cada def seja exercida pelo menos uma vez, não implicam os c-uso e os p-uso.

Todos os usos c-uso seriam quando uma variável utilizada na def troca o seu valor ou como uma produção de valor. Todos os p-uso são utilizados para definir o verdadeiro ou falso de um valor predicativo e um predicativo associado para um limite.

Os Todos-du-Caminhos solicitam que todas as DEF de variáveis e subsequências como c-uso e p-uso das variáveis sejam executadas por todos os caminhos livres de DEF e de laços.

Na representação da figura 7 demonstra como usar o teste de fluxo de dados em um grafo de fluxo.

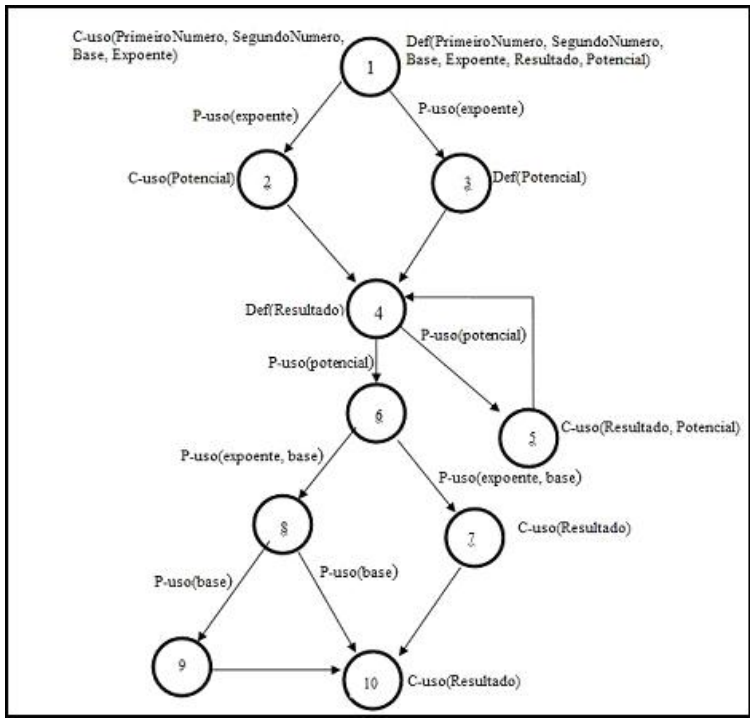

Figura 7 - Representação do Teste de fluxo de dados

Há que se destacar, contudo, que existe um elemento comum aos dois tipos de teste. Tanto no teste de caixa branca quanto no teste de caixa preta, o analista não sabe qual será o comportamento da aplicação ou do alvo de teste em uma determinada situação. A imprevisibilidade de resultados é comum aos dois casos.

2.4. Ferramentas de Teste Automatizadas

Para reduzir o tempo de teste, sem reduzir a eficácia, é claro, pesquisadores e profissionais desenvolveram uma primeira geração de ferramentas de teste automatizadas:

a) Analisadores Estáticos: Suporta a "comprovação" de afirmações estáticas - afirmações fracas sobre a estrutura do programa.

b) Auditores de Código: Filtros especiais que são usados para verificar a qualidade de

software, a fim de garantir que ele atenda aos padrões mínimos de codificação.

c) Processadores de Asserção: Sistemas pré-processadores e pós-processadores são empregados para dizer se as informações fornecidas pelo programador, denominadas asserções, sobre o comportamento de um programa são de fato cumpridas durante as execuções reais do programa.

d) Geradores de arquivos de teste: Processadores que geram e preenchem com valores previamente determinados, arquivos de entrada típicos para programas que estão em teste.

e) Geradores de dados de teste: São sistemas de análise automatizados que auxiliam o usuário a selecionar dados de teste que fazem um programa comportar-se de uma forma particular.

f) Verificadores de Teste: São ferramentas que medem a cobertura interna dos testes, frequentemente expressa em termos que estão relacionados à estrutura de controle do objeto de teste, e relatam o valor da cobertura ao especialista em garantia da qualidade.

2.5. Estratégias de Teste de Software

A estratégia de teste de software proporciona um mapa rodoviário para o desenvolvedor de software, para a organização de garantia de qualidade e para o cliente, no sentido que descreve os passos a serem dados como parte da atividade de teste.

Uma estratégia de teste de software deve ser flexível o bastante para promover a criatividade e a customização necessárias para testar adequadamente todos os grandes sistemas baseados em software. Ao mesmo tempo, deve ser rígida o bastante para promover um razoável planejamento e rastreamento administrativo à medida que o projeto progride.

Existe um grande número de estratégias de teste de software. Todas oferecem ao desenvolvedor um esqueleto para testar as seguintes características genéricas:

A atividade de teste inicia-se no nível de módulos e prossegue "para fora", na direção da integração de todo o sistema baseado em computador.

Diferentes técnicas de teste são apropriadas a diferentes pontos do tempo.

A atividade de teste é realizada pela equipe de desenvolvimento do software e (para grandes projetos) por um grupo de teste independente.

As atividades de teste e de depuração são atividades diferentes, mas a depuração deve ser acomodada em qualquer estratégia de teste.

2.5.1. VERIFICAÇÃO E VALIDAÇÃO

Verificação refere-se ao conjunto de atividades que garante que o software implemente corretamente uma função específica.

Validação refere-se a um conjunto diferente de atividades que garantem que o software que foi construído é rastreável às exigências do cliente.

Verificação: " Estamos construindo certo o produto ? "
Validação: " Estamos construindo o produto certos? "

A atividade de teste não deve ser vista como uma rede de segurança.

"Não se pode testar a qualidade. Se ela não estiver lá antes de você começar a testar, não estará lá quando você tiver terminado de testar".

A qualidade é incorporada ao software em todo o processo de engenharia de software. A aplicação adequada de métodos de e ferramentas, revisões técnicas formais efetivas e um gerenciamento e medição sólidos, todos levam à qualidade que é confirmada durante o teste.

2.5.2. TESTE DE UNIDADE

O teste de unidades concentra-se no esforço de verificação da menor unidade de projeto de software – o módulo. Usando a descrição do projeto detalhado como guia, caminhos de controle importantes são testados para descobrirem erros dentro das fronteiras do módulo. A complexidade relativa dos testes e os erros detectados como resultado deles são limitados pelo campo de ação restrito estabelecido para o teste de unidade. O teste de unidade baseia-se sempre na caixa branca, e esse passo pode ser realizado em paralelo para múltiplos módulos.

2.5.3. TESTE DE INTEGRAÇÃO

"Se todos os módulos funcionam individualmente, por que se tem dúvida de que eles funcionarão quando colocados juntos?". O problema é exatamente "colocá-los juntos" – tendo uma interface.

O teste de Integração é uma técnica sistemática para a construção da estrutura do programa, realizando-se, ao mesmo tempo, testes para descobrir erros associados a interfaces. O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto.

2.5.4. TESTE DE VALIDAÇÃO

A validação pode ser definida de muitas maneiras, mas uma definição simples é que a validação é bem-sucedida quando o software funciona de uma maneira razoavelmente esperada pelo usuário. São definidas expectativas razoáveis na Especificação de Requisitos de Software. Esta especificação contém uma seção denominada Critérios de Validação. As informações contidas nessa seção formam a base para uma abordagem ao teste de validação.

A validação é realizada por meio de uma série de testes de caixa preta que demonstram a conformidade com os requisitos.

2.5.5. TESTE DE SISTEMA

O teste de sistema é uma série de diferentes testes, cujo propósito fundamental é por completamente à prova o sistema baseado em computador. Não obstante, cada teste tenha uma finalidade diferente, todo o trabalho deve verificar se todos os elementos do sistema foram adequadamente integrados e realizam as funções atribuídas.

2.5.6. TESTE DE RECUPERAÇÃO

É um teste de sistema que força o software a falhar de diversas maneiras e verifica se a recuperação é adequadamente executada.

Se a recuperação for automática (realizada pelo próprio sistema), a reinicialização, mecanismos de *checkpointing*, recuperação de dados e reinício são avaliados (cada um) quanto à concretude. Se a recuperação exigir intervenção humana, o tempo médio até o reparo é avaliado para determinar se ele se encontra fora dos limites aceitáveis.

2.5.7. TESTE DE SEGURANÇA

Este teste tenta verificar se todos os mecanismos de proteção embutidos num sistema o protegerão, de fato, de acessos indevidos.

Desde que tenha tempo e recursos suficientes, um bom teste de segurança penetrará por fim no sistema. O papel do projetista do sistema é fazer com que o acesso custe mais do que o valor da informação que será obtida.

2.5.8. TESTE DE ESTRESSE

Este teste executa o sistema de uma forma que exige recursos em quantidade, frequência ou volume anormais. Essencialmente, o analista tenta destruir o programa.

2.5.9. DEPURAÇÃO

A depuração ocorre em consequência de teste bem sucedidos. Quando um caso de teste revela um erro, a depuração é o processo que resulta na remoção do erro.

O processo de depuração sempre terá um dentre dois resultados:

- A causa será descoberta, corrigida e removida; ou
- A causa não será descoberta.

Assim, a pessoa que executa a depuração pode suspeitar de uma causa, projetar um caso de teste para ajudá-lo a validar sua suspeita e trabalhar na direção da correção do erro de forma iterativa.

Portanto, assim que um BUG é encontrado, ele deve ser corrigido. Porém, esta correção

pode introduzir outros erros e, portanto, fazer mais mal do que bem. Dos muitos recursos disponíveis durante a depuração, o mais valioso é o conselho de outros membros da equipe de desenvolvimento de software.

8. BIBIOGRAFIA BÁSICA

PRESSMAN, R. S. **Engenharia de Software**. São Paulo. Makron Books, 2006.

SOMMERVILLE, Ian. **Engenharia de Software** - Ed. Prentice Hall, 2007.

Técnicas de Teste de Software. Disponível em http://www2.dem.inpe.br/ijar/TesteSoftware1.html

CARDILLI, Danilo. **Uma visão da técnica de teste de caixa branca**. Disponível em http://www.devmedia.com.br/uma-visao-da-tecnica-de-teste-de-caixa-branca/15610