

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/266356473>

# Introdução a Teste de Software

## Article

CITATIONS

2

READS

1,344

2 authors, including:



[Arilo Claudio Dias Neto](#)

Federal University of Amazonas

57 PUBLICATIONS 423 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Developers' Governance in Mobile Software Ecosystems from Developer Experience [View project](#)



MSECO-CERT: Process-Based Approach to Support Apps Certification in Mobile Software Ecosystem  
[View project](#)

All content following this page was uploaded by [Arilo Claudio Dias Neto](#) on 14 May 2015.

The user has requested enhancement of the downloaded file. All in-text references [underlined in blue](#) are added to the original document and are linked to publications on ResearchGate, letting you access and read them immediately.



# Introdução a Teste de Software

**T**este 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. 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, assim como exemplos práticos da aplicação de cada tipo de técnica ou critério de teste.

## Conceitos básicos associados a Teste de Software

Antes de iniciarmos uma discussão sobre teste de software precisamos esclarecer alguns conceitos relacionados a essa atividade. Inicialmente, precisamos conhecer a diferença entre Defeitos, Erros e Falhas. As definições que iremos usar aqui seguem a terminologia padrão para Engenharia de Software do IEEE – Institute of Electrical and Electronics



**Arilo Cláudio Dias Neto**

(ariloclaudio@gmail.com)

É Bacharel em Ciência da Computação formado na Universidade Federal do Amazonas, Mestre em Engenharia de Sistemas e Computação formado na COPPE/UFRJ, e atualmente está cursando doutorado na área de Engenharia de Software da COPPE/UFRJ. Possui 5 anos de experiência em análise, desenvolvimento e teste de software. É editor técnico das Revistas SQL Magazine e WebMobile, gerenciadas pelo Grupo DevMedia.

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.

A **Figura 1** expressa a diferença entre esses conceitos. Defeitos fazem parte do universo físico (a aplicação propriamente dita) e são causados por pessoas, por exemplo, através do mal uso de uma tecnologia. Defeitos podem ocasionar a manifestação de erros em um produto, ou seja, a construção de um software de forma diferente ao que foi especificado (universo de informação). Por fim, os erros geram falhas, que são comportamentos inesperados em um software que afetam diretamente o usuário final da aplicação (universo do usuário) e pode inviabilizar a utilização de um software.

Dessa forma, ressaltamos que teste de software revela simplesmente falhas em um produto. Após a execução dos testes é necessária a execução de um processo de depuração para a identificação e correção dos defeitos que originaram essa falha, ou seja, Depurar não é Testar!

A atividade de teste é composta por alguns elementos essenciais que auxiliam na formalização desta atividade. Esses elementos são os seguintes:

- **Caso de Teste.** descreve uma condição particular a ser testada e é composto por valores de entrada, restrições para a sua execução e um resultado ou comportamento esperado (CRAIG e JASKIEL, 2002).

- **Procedimento de Teste.** é uma descrição dos passos necessários para executar um caso (ou um grupo de casos) de teste (CRAIG e JASKIEL, 2002).

- **Critério de Teste:** serve para selecionar e avaliar casos de teste de forma a aumen-



**Figura 1.** Defeito x erro x falha (Uma versão similar pode ser obtida em <http://www.projectcartoon.com/cartoon/611>)

tar as possibilidades de provocar falhas ou, quando isso não ocorre, estabelecer um nível elevado de confiança na correção do produto (ROCHA et al., 2001). Os critérios de teste podem ser utilizados como:

- o **Critério de Cobertura dos Testes:** permitem a identificação de partes do programa que devem ser executadas para garantir a qualidade do software e indicar quando o mesmo foi suficientemente testado (RAPPS e WEYUKER, 1982). Ou seja, determinar o percentual de elementos necessários por um critério de teste que foram executados pelo conjunto de casos de teste.

- o **Critério de Adequação de Casos de Teste:** Quando, a partir de um conjunto de casos de teste T qualquer, ele é utilizado para verificar se T satisfaz os requisitos de teste estabelecidos pelo critério. Ou seja, este critério avalia se os casos de teste definidos são suficientes ou não para avaliação de um produto ou uma função (ROCHA et al., 2001).

- o **Critério de Geração de Casos de Teste:** quando o critério é utilizado para gerar um conjunto de casos de teste T adequado para um produto ou função, ou seja, este critério define as regras e diretrizes para geração dos casos de teste de um produto que esteja de acordo com o critério de adequação definido anteriormente (ROCHA et al., 2001).

Definidos os elementos básicos associados aos testes de software, iremos a seguir discutir a origem dos defeitos em um software.

## Defeitos no desenvolvimento de software

No processo de desenvolvimento de software, todos os defeitos são humanos e, apesar do uso dos melhores métodos de desenvolvimento, ferramentas ou

profissionais, permanecem presentes nos produtos, o que torna a atividade de teste fundamental durante o desenvolvimento de um software. Já vimos que esta atividade corresponde ao último recurso para avaliação do produto antes da sua entrega ao usuário final.

O tamanho do projeto a ser desenvolvido e a quantidade de pessoas envolvidas no processo são dois possíveis fatores que aumentam a complexidade dessa tarefa, e consequentemente aumentam a probabilidade de defeitos. Assim, a ocorrência de falhas é inevitável. Mas o que significa dizer que um programa falhou? Basicamente significa que o funcionamento do programa não está de acordo com o esperado pelo usuário. Por exemplo, quando um usuário da linha de produção efetua consultas no sistema das quais só a gerência deveria ter acesso. Esse tipo de falha pode ser originado por diversos motivos:

- A especificação pode estar errada ou incompleta;
- A especificação pode conter requisitos impossíveis de serem implementados devido a limitações de hardware ou software;
- A base de dados pode estar organizada de forma que não seja permitido distinguir os tipos de usuário;
- Pode ser que haja um defeito no algoritmo de controle dos usuários.

Os defeitos normalmente são introduzidos na transformação de informações entre as diferentes fases do ciclo de desenvolvimento de um software. Vamos seguir um exemplo simples de ciclo de vida de desenvolvimento de software: os requisitos expressos pelo cliente são relatados textualmente em um documento de especificação de requisitos. Esse documento é então transformado

em casos de uso, que por sua vez foi o artefato de entrada para o projeto do software e definição de sua arquitetura utilizando diagramas de classes da UML. Em seguida, esses modelos de projetos foram usados para a construção do software em uma linguagem que não segue o paradigma orientado a objetos. Observe que durante esse período uma série de transformações foi realizada até chegarmos ao produto final. Nesse meio tempo, defeitos podem ter sido inseridos. A **Figura 2** expressa exatamente a metáfora discutida nesse parágrafo.

Essa série de transformações resultou na necessidade de realizar testes em diferentes níveis, visando avaliar o software em diferentes perspectivas de acordo com o produto gerado em cada fase do ciclo de vida de desenvolvimento de um software. Esse será o foco da seção seguinte.

## 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:

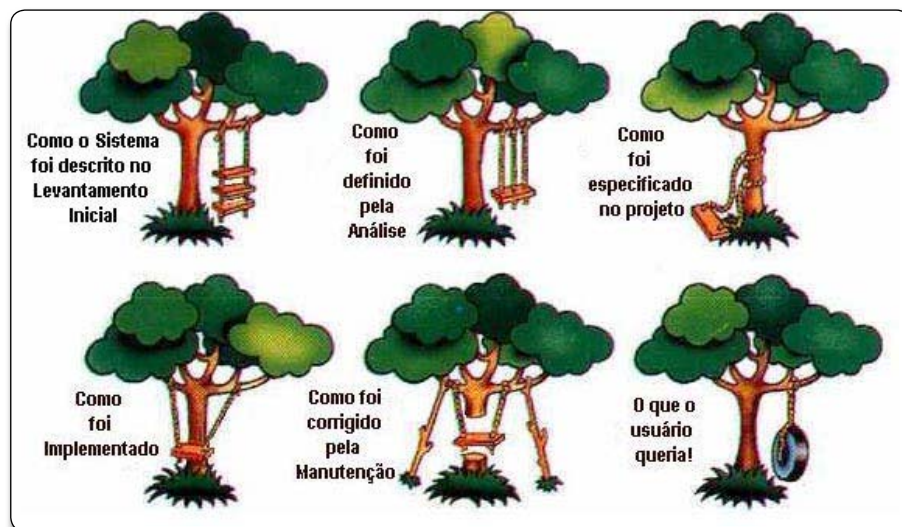
- **Teste de Unidade:** também conhecido como testes unitários. Tem por objetivo explorar a menor unidade do projeto, procurando provocar falhas ocasionadas por defeitos de lógica e de implementação em cada módulo, separadamente. O universo alvo desse tipo de teste são os métodos dos objetos ou mesmo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

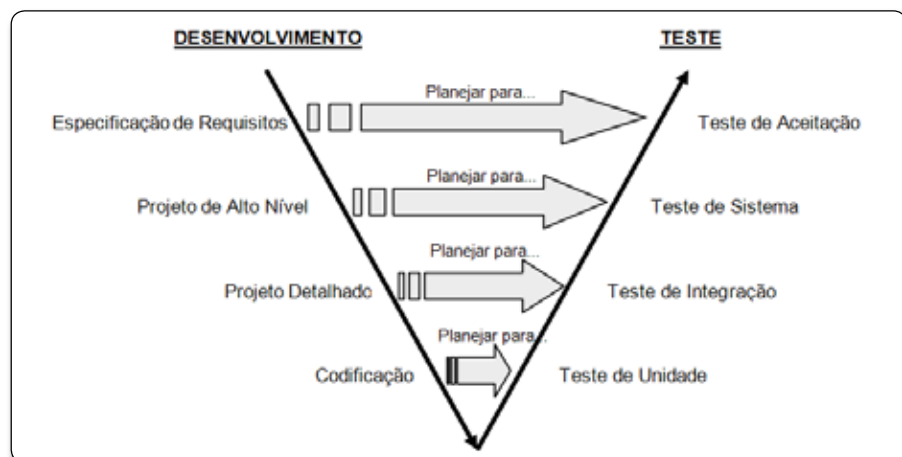
- **Teste de Sistema:** avalia o software em busca de falhas por meio da utilização do mesmo, como se fosse um usuário final. Dessa maneira, os testes são executados nos mesmos ambientes, com as mesmas condições e com os mesmos dados de entrada que um usuário utilizaria no seu dia-a-dia de manipulação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



**Figura 2.** Diferentes Interpretações ao longo do ciclo de desenvolvimento de um software



**Figura 3.** Modelo V descrevendo o paralelismo entre as atividades de desenvolvimento e teste de software (CRAIG e JASKIEL, 2002)

Dessa forma, seguindo a Figura 3, 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.

Já a execução ocorre no sentido inverso.

Conhecidos os diferentes níveis de teste, a partir da próxima seção descreveremos as principais técnicas de teste de software existentes que podemos aplicar nos diferentes níveis abordados.

## Técnicas de teste de software

Atualmente 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 diferentes, o objetivo principal destas técnicas continua a ser o mesmo:



encontrar falhas no software.

As técnicas de teste são classificadas de acordo com a origem das informações utilizadas para estabelecer os requisitos de teste. Elas contemplam diferentes perspectivas do software e impõe-se a necessidade de se estabelecer uma estratégia de teste que contemple as vantagens e os aspectos complementares dessas técnicas. As técnicas existentes são: técnica funcional e estrutural.

A seguir conheceremos um pouco mais sobre cada técnica, mas iremos enfatizar alguns critérios específicos para a técnica funcional.

### Técnica Estrutural (ou teste caixa-branca)

Técnica de teste que avalia o comportamento interno do componente de software (Figura 4). Essa técnica trabalha diretamente sobre o código fonte do componente de software para avaliar aspectos tais como: teste de condição, teste de fluxo de dados, teste de ciclos e teste de caminhos lógicos (PRESSMAN, 2005).

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 outros aspectos além dos citados anteriormente. O testador tem acesso ao código fonte da aplicação e pode construir códigos para efetuar a ligação de bibliotecas e componentes.

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 origi-

nadas por estruturas de condições são testadas. A Listagem 1 apresenta um código fonte, extraído de (BARBOSA et al., 2000) que descreve um programa de exemplo que deve validar um identificador digitado como parâmetro, e a Figura 5 apresenta o grafo de programa extraído a partir desse código, também extraído de (BARBOSA et al., 2000). A partir do grafo deve ser escolhido algum critério baseado em algum algoritmo de busca em grafo (exemplo: visitar todos os nós, arcos ou caminhos) para geração dos casos de teste estruturais para o programa (PFLEEGER, 2004).

Um exemplo bem prático desta técnica de teste é o uso da ferramenta livre JUnit para desenvolvimento de casos de teste para avaliar classes ou métodos desenvolvidos na linguagem Java. 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. Dessa forma, podemos auxiliar na redução dos problemas existentes nas pequenas funções ou unidades que compõem um software.

### Teste Funcional (ou teste 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 (Figura 6). Dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um re-

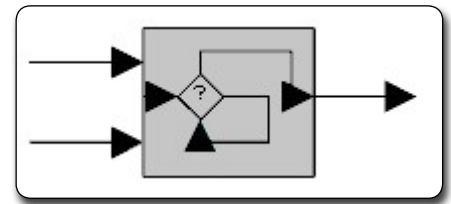


Figura 4. Técnica de Teste Estrutural.

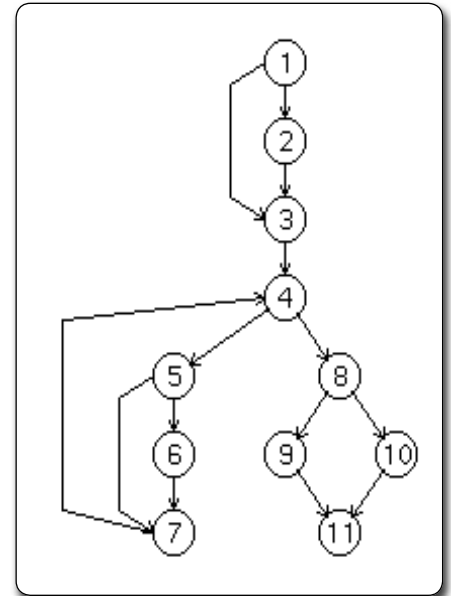


Figura 5. Grafo de Programa (Identifier.c) (BARBOSA et al., 2000).

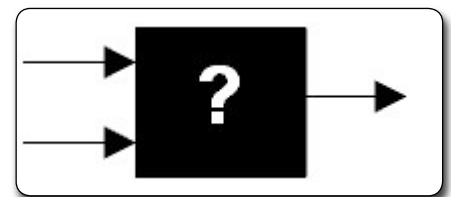


Figura 6. Técnica de Teste Funcional.



Listagem 1. Código fonte do programa identifier.c (BARBOSA et al., 2000)

```
/* 01 */ {
/* 01 */ char achar;
/* 01 */ int length, valid_id;
/* 01 */ length = 0;
/* 01 */ printf ("Digite um possível identificador\n");
/* 01 */ printf ("seguido por <ENTER>: ");
/* 01 */ achar = fgetc (stdin);
/* 01 */ valid_id = valid_starter (achar);
/* 01 */ if (valid_id)
/* 02 */     length = 1;
/* 03 */     achar = fgetc (stdin);
/* 04 */     while (achar != '\n')
/* 05 */     {
/* 05 */         if (!valid_follower (achar))
/* 06 */             valid_id = 0;
/* 07 */             length++;
/* 07 */             achar = fgetc (stdin);
/* 07 */     }
/* 08 */     if (valid_id && (length >= 1) && (length < 6) )
/* 09 */         printf ("Valido\n");
/* 10 */     else
/* 10 */         printf ("Invalido\n");
/* 11 */ }
```

sultado 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).

Um conjunto de critérios de teste pode ser aplicado aos testes funcionais. A seguir conheceremos alguns deles.

Particionamento em classes de equivalência

Esse critério divide o domínio de entrada de um programa em classes de equivalência, a partir das quais os casos de teste são derivados. Ele tem por objetivo minimizar o número de casos de teste, selecionando apenas um caso de teste de cada classe, pois em princípio todos os elementos de uma classe devem se comportar de maneira equivalente. Eventualmente, pode-se também considerar o domínio de saída para a definição das classes de equivalência (ROCHA et al., 2001).

Uma classe de equivalência representa um conjunto de estados válidos e inválidos para uma condição de entrada.

Tipicamente uma condição de entrada pode ser um valor numérico específico, uma faixa de valores, um conjunto de valores relacionados, ou uma condição lógica. As seguintes diretrizes podem ser aplicadas:

- Se uma condição de entrada especifica uma faixa de valores ou requer um valor específico, uma classe de equivalência válida (dentro do limite) e duas inválidas (acima e abaixo dos limites) são definidas.
- Se uma condição de entrada especifica um membro de um conjunto ou é lógica, uma classe de equivalência válida e uma inválida são definidas.

Devemos seguir tais passos para geração dos testes usando este critério:

1. Identificar classes de equivalência (é um processo heurístico)
  - o condição de entrada
  - o válidas e inválidas
2. Definir os casos de teste
  - o enumeram-se as classes de equivalência
  - o casos de teste para as classes válidas
  - o casos de teste para as classes inválidas

Em (BARBOSA et al., 2000) é apresentado a aplicação do critério de particionamento por equivalência para o programa

identifier.c. Iremos apresentá-lo como exemplo do uso deste critério de teste. Relembrando, o programa deve determinar se um identificador é válido ou não.

“Um identificador válido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo 1 caractere e no máximo 6 caracteres de comprimento. Exemplo: “abc12” (válido), “cont\*1” (inválido), “1soma” (inválido) e “a123456” (inválido).”

O primeiro passo é a identificação das classes de equivalência. Isso está descrito na Tabela 1.

A partir disso, conseguimos especificar quais serão os casos de teste necessários. Para ser válido, um identificador deve atender às condições (1), (3) e (5), logo é necessário um caso de teste válido que cubra todas essas condições. Além disso, será necessário um caso de teste para cada classe inválida: (2), (4) e (6). Assim, o conjunto mínimo é composto por quatro casos de teste, sendo uma das opções: T0 = {{a1,Válido), (2B3, Inválido), (Z-12, Inválido), (A1b2C3d, Inválido)}.

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 → {1, 10, 0,11}.

Como exemplo, extraído de (BARBOSA et al., 2000), iremos considerar a seguinte situação:

“... o cálculo do desconto por dependente é feito da seguinte forma: a entrada é a idade do dependente que deve estar restrita ao intervalo [0..24]. Para dependentes até 12 anos (inclusive) o desconto é de 15%. Entre 13 e 18 (inclusive) o desconto é de 12%. Dos 19 aos 21 (inclusive) o desconto é de 5% e dos 22 aos 24 de 3%...”

Aplicando o teste de valor limite

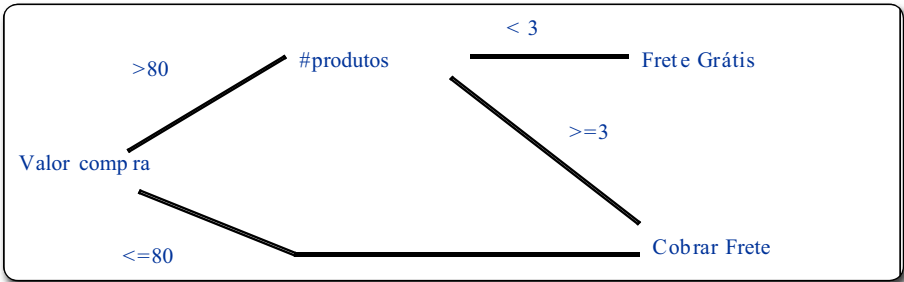


Figura 7. Árvore de Decisão – Grafo de Causa-Efeito.

Condições de Entrada	Classes	Classes
Tamanho t do identificador	(1) 1 ≤ t ≤ 6	(2) t > 6
Primeiro caractere c é uma letra	(3) Sim	(4) Não
Só contém caracteres válidos	(5) Sim	(6) Não

Tabela 1. Classes de Equivalência do programa identifier.c.

Causa	Valor da compra	> 60	> 60	<= 60
	#Produtos	< 3	>= 3	--
Efeito	Cobrar frete		V	V
	Frete grátis	V		

Tabela 2. Tabela de decisão para o programa de compra pela Internet.

convencional serão obtidos casos de teste semelhantes a este: {-1,0,12,13,18,19,21,22,24,25}.

### Grafo de causa-efeito

Esse critério de teste verifica o efeito combinado de dados de entrada. As causas (condições de entrada) e os efeitos (ações) são identificados e combinados em um grafo a partir do qual é montada uma tabela de decisão, e a partir desta, são derivados os casos de teste e as saídas (ROCHA et al., 2001).

Esse critério é baseado em quatro passos, que exemplificaremos utilizando o exemplo, também extraído de (BARBOSA et al., 2000):

“Em um programa de compras pela Internet, a cobrança ou não do frete é definida seguindo tal regra: Se o valor da compra for maior que R\$ 60,00 e foram comprados menos que 3 produtos, o frete é gratuito. Caso contrário, o frete deverá ser cobrado.”

1. Para cada módulo, **Causas** (condições de entrada) e **efeitos** (ações realizadas às diferentes condições de entrada) são relacionados, atribuindo-se um identificador para cada um.

- Causa: valor da compra > 60 e #produtos < 3

- Efeito: frete grátis

2. Em seguida, um grafo de causa-efeito (árvore de decisão) é desenhado (Figura 7).

3. Neste ponto, transforma-se o grafo numa tabela de decisão (Tabela 2).

4. As regras da tabela de decisão são, então, convertidas em casos de teste.

Para a elaboração dos casos de teste, devemos seguir todas as regras extraídas da tabela. Esse critério deve ser combinado com os dois outros já apresentados neste artigo para a criação de casos de teste válidos (extraídos das regras) e inválidos (valores foras da faixa definida). Os casos de teste definidos a seguir refletem somente as regras extraídas da tabela de decisão. Fica como exercício pensar nos casos de teste inválidos para este problema.

- Casos de Teste (valor, qtd produtos,

resultado esperado) = {(61,2,“frete grátis”); (61,3,“cobrar frete”); (60, qualquer quantidade,“cobrar frete”)}

### Outras técnicas

Outras técnicas de teste podem e devem ser utilizadas de acordo com necessidades de negócio ou restrições tecnológicas. Destacam-se as seguintes técnicas: teste de desempenho, teste de usabilidade, teste de carga, teste de stress, teste de confiabilidade e teste de recuperação. Alguns autores chegam a definir uma técnica de teste caixa cinza, que seria um mesclado do uso das técnicas de caixa preta e caixa branca, mas, como toda execução de trabalho relacionado à atividade de teste utilizará simultaneamente mais de uma técnica de teste, é recomendável que se fixem os conceitos primários de cada técnica.

### Conclusões

O teste de software é uma das atividades mais custosas do processo de desenvolvimento de software, pois pode envolver uma quantidade significativa dos recursos de um projeto. O rigor e o custo associado a esta atividade dependem principalmente da criticalidade da aplicação a ser desenvolvida. Diferentes categorias de aplicações requerem uma preocupação diferenciada com as atividades de teste.

Um ponto bastante importante para a viabilização da aplicação de teste de software é a utilização de uma infraestrutura adequada. Realizar testes não consiste simplesmente na geração e execução de casos de teste, mas envolvem também questões de planejamento,

gerenciamento e análise de resultados. Apoio ferramental para qualquer atividade do processo de teste é importante como mecanismo para redução de esforço associado à tarefa em questão, seja ela planejamento, projeto ou execução dos testes. Após ter sua estratégia de teste definida, tente buscar por ferramentas que se encaixem na sua estratégia. Isso pode reduzir significativamente o esforço de tal tarefa.

Além disso, é importante ressaltar que diferentes tipos de aplicações possuem diferentes técnicas de teste a serem aplicadas, ou seja, testar uma aplicação web envolve passos diferenciados em comparação aos testes de um sistema embarcado. Cada tipo de aplicação possui características específicas que devem ser consideradas no momento da realização dos testes. O conjunto de técnicas apresentado neste artigo cobre diversas características comuns a muitas categorias de software, mas não é completo.

Para finalizar, podemos destacar outros pontos importantes relacionados às atividades de teste que podemos abordar em próximos artigos, tais como processo de teste de software, planejamento e controle dos testes e teste de software para categorias específicas de software, como aplicações web. Até a próxima!

### Agradecimentos

Agradecemos aos professores José Carlos Maldonado e Ellen Barbosa por terem gentilmente autorizado a publicação deste material, cujos exemplos utilizados estão fundamentados em seus trabalhos. ●

#### Referências

- BARBOSA, E.; MALDONADO, J.C.; VINCENZI, A.M.R.; DELAMARO, M.E; SOUZA, S.R.S. e JINO, M.. “Introdução ao Teste de Software. XIV Simpósio Brasileiro de Engenharia de Software”, 2000.
- CRAIG, R.D., JASKIEL, S. P., “Systematic Software Testing”, Artech House Publishers, Boston, 2002.
- IEEE Standard 610-1990: IEEE Standard Glossary of Software Engineering Terminology, IEEE Press.
- PFFLEGER, S. L., “Engenharia de Software: Teoria e Prática”, Prentice Hall- Cap. 08, 2004.
- PRESSMAN, R. S., “Software Engineering: A Practitioner’s Approach”, McGraw-Hill, 6th ed, Nova York, NY, 2005.
- RAPPS, S., WEYUKER, E.J., “Data Flow analysis techniques for test data selection”, In: International Conference on Software Engineering, p. 272-278, Tokio, Sep. 1982.
- ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. et al., “Qualidade de software – Teoria e prática”, Prentice Hall, São Paulo, 2001.