UNIVERSIDADE FEDERAL DE PERNAMBUCO
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

# TESTES BASEADOS EM MODELO

## QUALIDADE, PROCESSOS E GESTÃO SOFTWARE

**Alunos:** Diana Rúbia Rodrigues Ricardo (drrr@cin.ufpe.br)
Paulo César (paulocol@gmail.com)

Recife, 27 de novembro de 2008.

UNIVERSIDADE FEDERAL DE PERNAMBUCO
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

# TESTES BASEADOS EM MODELO

## QUALIDADE, PROCESSOS E GESTÃO SOFTWARE

*Trabalho disciplina Qualidade, Processos e Gestão de Software*
*do Programa de Pós-Graduação em*
*Ciência da Computação do Departamento de Sistemas*
*de Computação da Universidade Federal de Pernambuco.*

**Professor:** Alexandre Vasconcelos (amlv@cin.ufpe.br)

Recife, 27 de novembro de 2008.

## Resumo

.....



**Palavras-chave:**

Engenharia de Software, Testes, Testes Baseados em Modelo.

# Sumário

# Índice de Figuras



# Índice de Tabelas

# 1. Introdução

A importância do software tem crescido cada vez mais nas organizações e na sociedade. A cada dia ele desempenha atividades mais importantes, que trazem benefícios e agregam valor ao meio no qual atua. Geralmente, essas atividades podem ser realizadas pelo homem. Porém, algumas delas só podem ser executadas por computadores. A tendência é que o mundo se torne cada vez mais dependente dos softwares.

Determinadas atividades desempenhadas por computadores têm impacto direto na saúde financeira de uma instituição. Outras delas estão extremamente relacionadas à vida.

Torna-se cada vez mais necessário produtos com alto nível de qualidade e que satisfaçam o cliente. Por outro lado, os softwares que estão disponíveis no mercado, em sua grande maioria, apresentam uma grande quantidade de defeitos. Esses problemas afetam a funcionalidade, o desempenho, a segurança, a confiabilidade e a usabilidade do sistema, tendo impacto direto no ambiente no qual ele atua e podendo trazer conseqüências graves.

Informações de mercado dizem que mais de 90% dos sistemas são liberados com graves defeitos [1].

A insuficiência de testes é um dos principais motivos de falha nos Projetos de Desenvolvimento de Software [2].

Uma das formas de diminuir os custos com a Arquitetura e Execução de Testes é automatizar essas atividades. E uma das formas de automatizar essas atividades de Testes é utilizar Testes baseados em Modelos.

## *1.1 Organização do documento*

O restante desse documento está organizado em quatro outras seções da seguinte maneira:

Seção 2 – Mostra uma Visão geral sobre Teste de Software.

Seção 3 – Apresenta o Conceito de Teste Baseado em Modelo.

Seção 4 – Dedica-se a explicar algumas técnicas de Teste Baseado em Modelo.

Seção 5 – Traz uma conclusão sobre o trabalho desenvolvido e a perspectiva para trabalhos futuros.

# 2. Visão Geral sobre Testes de Software

## *2.1 Evolução da Atividade de Teste*

A atividade de testar o software até alguns anos estava em segundo plano. Era uma atividade que não recebia muita atenção e nem investimentos. Não se investia nem em tempo e nem em recurso para a realização dessa tarefa. Há poucos anos não se encontrava muito sobre o assunto na literatura.

Em 1950, Alan Turing escreveu o primeiro artigo científico que abordava questões sobre teste de software. Ele definia um teste operacional para demonstrar o comportamento da inteligência de um programa semelhante ao de um ser humano.

Antes de 1957, teste era a simples tarefa de navegar pelo código e corrigir erros já conhecidos. Foi em 1957 que o conceito Teste de Software se tornou um processo de detecção de erros de software. Mas, essa atividade só ocorria no final do processo de desenvolvimento.

Quando a Engenharia de Software, no início da década de 1970, passou a ser utilizada como modelo para organizações e universidades o processo de desenvolvimento de software passou a ter abordagens mais profundas. Em 1972 ocorreu a primeira conferência sobre testes na Universidade da Carolina do Norte.

Mas, um dos primeiros trabalhos mais completos e profundos sobre um processo de teste só foi produzido em 1979 por Glenford Myers. Foi nesse trabalho que a atividade de teste foi definida como um "processo de trabalho com a intenção de encontrar erros". Até esse momento, o objetivo do teste era somente provar o bom funcionamento de uma aplicação. Todos os esforços da atividade estavam voltados para a comprovação desse fato e conseqüentemente poucos defeitos eram encontrados. Myers propôs que se o objetivo do teste for encontrar erros, uma quantidade maior de problemas será encontrada. Uma vez que vários cenários serão buscados para testar o comportamento do aplicativo em várias circunstâncias.

Essa nova visão revolucionou a forma de abordar o problema, no entanto, os testes continuavam sendo executados no final do processo de desenvolvimento.

Os primeiros conceitos de qualidade de software surgiram no início dos anos 1980. Nesses novos conceitos, desenvolvedores e testadores trabalham juntos desde o início do processo de desenvolvimento.

A partir deste período, teste passou a ser visto como um processo paralelo ao processo de desenvolvimento. Passou a possuir atividades específicas de planejamento, análise de requisitos dos testes, design, implementação, execução e manutenção de testes [3].

As primeiras ferramentas para realizar as atividades teste só começaram a ser fabricadas em 1990. Elas possibilitam a execução de teste que não podiam ser feitos manualmente. Como testes de carga, performance, entre outros.

A visão de testes tem evoluído para uma atitude mais pró-ativa, não se limitando apenas à detecção de falhas, mas atuando fortemente na prevenção [3].

## 2.2 *Teste de Software*

Atualmente, o mercado está cada vez mais exigente competitivo e as empresas buscam estratégias para sobreviver. Melhorar a qualidade do produto final e reduzir os custos do desenvolvimento deste produto são importantes objetivos das organizações. O processo de teste de software visa aumentar a qualidade do produto, reduzir os custos do desenvolvimento e de manutenção, reduzir os defeitos dos produtos, diminuir o esforço e o custo de retrabalho, aumentar a confiabilidade do produto e, consequentemente aumentar a satisfação do cliente. Os testes são as últimas oportunidades de se encontrar defeitos antes que o produto entre em produção.

Vários autores apresentam uma definição para testes de software, algumas delas são:

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 encontrar o máximo número de erros dispondo do mínimo de

esforço, ou seja, mostrar aos que desenvolvem se os resultados estão ou não de acordo com os padrões estabelecidos [4].

Teste é um processo sistemático e planejado que tem por finalidade única a identificação de erros [5].

Operação de um sistema ou aplicação, em um ambiente controlado, de situações normais e anormais, e a avaliação dos resultados para verificar se o sistema se comporta conforme previsto [6].

Segundo a *IEEE Standard Glossary of Software Engineering Terminlogy*, teste de software é um processo de exercitar um software ou componente sobre condições específicas, observando ou gravando os resultados, e realizando uma avaliação sobre algum aspecto.

Os testes podem ser usados para mostrar a existência de erros, mas nunca a ausência deles. Por mais que você teste sempre existirão erros, zero-defeito é algo inatingível [7]. É impossível ter um produto livre de erros devido ao número enorme de situações existentes, cenários possíveis. Depois de um determinado tempo testando a probabilidade de encontrar erros diminui bastante. O momento certo de parar de testar é o momento em que o custo para testar e corrigir os erros é mais caro que o custo da falha ocorrer em produção [8]. A Figura 1 mostra o momento de interromper os testes.

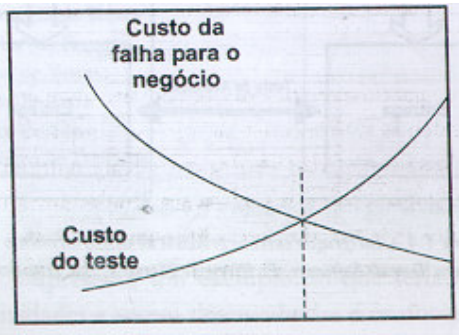


**Figura 1: Gráfico mostrando o momento de interromper os testes**

A criticidade do sistema é quem define a quantidade de testes. Quanto mais crítico for o sistema mais testes terão que ser executados para garantir a qualidade dele.

Todo tipo de erro custa dinheiro. Além da alocação de recursos e tempo para produzir algo com defeito, existem os custos que o erro provoca como:

identificação e correção do erro, implantação da correção. Segundo Myers, quanto mais tarde um erro é descoberto mais caro ele é. Ele aplica a "regra de 10" aos custos de correção do erro. Ou seja, quando um erro não é encontrado em uma fase do processo de desenvolvimento, os custos de correção são multiplicados por 10 para cada fase que ele se propaga. Isso significa dizer que erros encontrados em produção são extremamente caros.

## 2.3 *Verificação e Validação*

As atividades de teste começam no início do processo de desenvolvimento com os testes de verificação. Teste de software inclui atividades de verificação e validação.

A verificação diz respeito ao que foi construído, se a aplicação foi construída corretamente. Já a validação diz respeito ao entendimento do que era pra ser construído, se foi construído o sistema correto. A figura 2 exemplifica bem o que acontece durante a construção de um sistema.

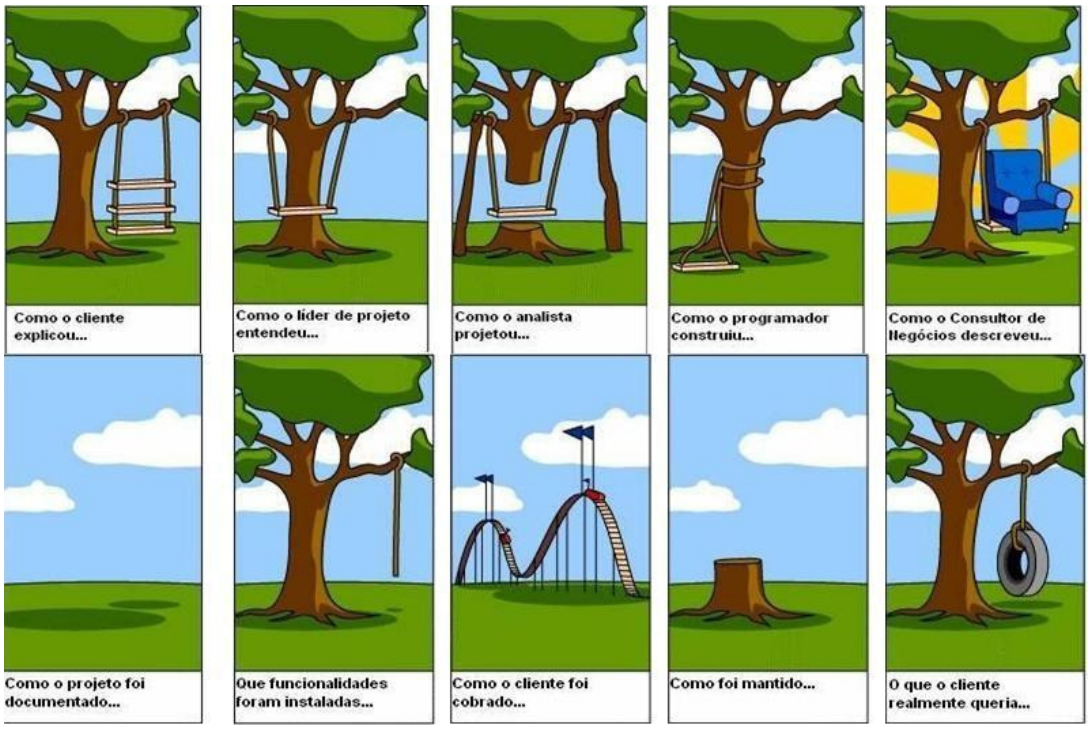


**Figura 2: Validação x Verificação**

### Testes de Verificação

Esse tipo de teste também é conhecido como teste estático, pois geralmente são realizados sobre os documentos que são produzidos durante todo o ciclo de desenvolvimento. Essas atividades são iniciadas antes da codificação do sistema, elas começam na fase de requerimento e continuam através da codificação do produto. O teste consiste na realização de inspeções, revisões e/ou auditorias sobre os produtos gerados em cada etapa do processo, evitando que dúvidas e assuntos mal resolvidos passem para a próxima fase do processo de desenvolvimento. Algumas atividades de verificação são:

- Revisões de requisitos;
- Revisões de modelos;
- Revisões e inspeções técnicas em geral.

A verificação tem por objetivo provar que o produto vai ao encontro dos requerimentos especificados. Ela garante a qualidade do software na produção e na manutenção.

### Testes de Validação

Os testes de validação também são conhecidos como testes dinâmicos ou testes de software, uma vez que são aplicados diretamente no software. Eles podem ser aplicados em componentes isolados, módulos existentes ou em todo o sistema. Esse tipo de teste avalia se o sistema atende aos requisitos e especificações analisadas nas fases iniciais do projeto, se ele vai ao encontro dos requerimentos do usuário. Algumas atividades de validação são:

- Teste unitário
- Teste de integração
- Teste de sistema
- Teste de aceitação
- Homologação
- Teste de Regressão.

Testes de verificação e validação são complementares, eles não são atividades redundantes. Eles possuem objetivos e naturezas distintas, contribuem para a detecção de erros e aumentam a qualidade final do produto.

## *2.4 Automação de Teste*

A automação dos testes é a automação dos casos de testes, a geração automática da massa de dados de testes e execução automatizada dos testes.

Essa automação inclui a automação do processo de planejamento dos testes, permite aumentar a profundidade e abrangência dos casos de testes envolvidos, viabiliza a execução de alguns tipos de testes, como é o caso dos testes de carga e estresse. Porém, os projetos de automação de teste devem ser implementados com muito cuidado. A seleção da ferramenta de automação de teste bem como o que deverá ser automatizado ou não devem ser analisados de forma clara porque dependendo das técnicas de teste utilizadas, o retorno de investimento pode ser negativo ou positivo.

Algumas expectativas em relação à automação são criadas, mas nem todas são verdadeiras, exemplo:

- Todos os testes podem ser automatizados;
- Uma única ferramenta de automação atende todos os tipos de teste;
- Automação é uma alternativa para testar um sistema pouco documentado;
- Automação é uma alternativa para testar um sistema mal planejado;
- Ao automatizar a redução de esforço e tempo para realização das atividades de teste é imediata;
- Ferramentas de automação de testes são fáceis de usar.

A automação só deve ser realizada quando existe um processo maduro na organização e na equipe de testes, conscientização por parte da organização e os testes levam muito tempo para serem executados manualmente. Os testes automáticos são indicados para testes de produto, testes com múltiplos usuários, testes com múltiplas configurações de hardware e software.

# 3. Testes Baseados Em Modelos

Antes de falar em Testes Baseados em Modelos, precisamos saber o que é um modelo. Modelo é uma representação abstrata de algo, concentrando nos aspectos principais e ignorando o resto. Um outro conceito que pode ser utilizado é Um grafo de estados que descreve o comportamento de um *Software*. Modelos podem ser utilizados para Testes, pois: Uma aplicação pode ser representada como um modelo, uma máquina finita de estados é uma ferramenta de modelagem muito útil para o Teste Funcional. Logo, Modelo pode ser usado para geração de Caso de Teste. Tarefas repetitivas e sensíveis a erros podem ser automatizadas.

Testes Baseados em Modelos é: técnica de teste em que os casos de teste são derivados do modelo do software a testar [9].

Um outro conceito é: Testes Baseados em Modelos (TBM) consiste em uma técnica para geração automática de um conjunto de casos de testes, com entradas e saídas esperadas, utilizando modelos extraídos a partir dos requisitos do software [10].

No Teste Baseado em Modelo a especificação do *Software* deve estar definida em um modelo apropriado a forma utilizada para automatizar os testes. A especificação pode ser feita utilizando-se: Métodos formais, Máquinas de Estado Finito, UML.

Os testes baseados em modelos são baseados nos usos dos modelos dessas máquinas e são usados para testar:

- A estrutura do *Software*;
- O comportamento do *Software*;
- Especificação do comportamento do Software;

Essa técnica de teste permite verificar a conformidade da implementação relativamente à especificação do sistema introduzindo mais sistematização e automação no processo de teste. O processo de construção do modelo a testar ajuda a clarificar os requisitos o que por si só conduz a código de melhor qualidade. Uma vez construído, o modelo do sistema serve como documentação

e como "oráculo" de teste (fornece os resultados esperados para determinados dados de entrada). As técnicas de geração de casos de teste a partir de modelos de software são diversas e dependem das características desses modelos. Quando os modelos são executáveis consegue-se atingir níveis maiores de automatização de todo o processo de MBT (*Model Based Test*).

# 4. Técnicas de Testes Baseados Em Modelo

A utilização de modelos para representação do comportamento de um sistema permite que sejam gerados casos de teste automaticamente partindo destes modelos. Para isto, várias técnicas já foram propostas para geração de testes baseados em modelos, como o Método TT [11], Método UIO [11], Método W [11], Método DS [12], Algoritmo do Carteiro Chinês [13], *Fault Transition Pair Coverage* [14], Método BZ-TT [15] e *User Defined Test Objectives*[16].

Um caso de teste é uma seqüência de transições em um modelo que permite verificar que esta seqüência está coerente com o que está implementado no software e o que está especificado no modelo.

## *4.1 Método TT*

Este método é bastante simples, mas ele garante que todas as transições do modelo são testadas pelo menos uma vez. O algoritmo para criar os testes depende das propriedades que o modelo possui.

Pode existir apenas uma seqüência que percorra todas as transições, ou pode existir mais de uma, caso ela não seja completamente conectada.

## *4.2 Método UIO*

Este método utiliza seqüências UIO de forma a gerar várias seqüências que levem o modelo ao estado inicial, aplique uma entrada (transição), para em seguida aplicar a seqüência UIO que levará o modelo ao estado final desejado.

Uma seqüência UIO (seqüência única de entrada e saída) é utilizada para verificar se um modelo está em um determinado estado. Isto faz com que cada estado de um modelo tenha uma seqüência UIO diferente.

## *4.3 Método DS*

O método DS, utiliza uma seqüência DS para verificar que para cada estado diferente encontra-se uma saída diferente. Para se utilizar o método DS, faz-se necessário que o modelo a ser utilizado possua seqüências de sincronização e de distinção.

Uma seqüência de sincronização é uma seqüência de transições que quando aplicada a qualquer estado do modelo, gera uma mesma saída. Uma seqüência de distinção é uma seqüência de transições que quando aplicada a todos os estados do modelo, geram uma saída diferente para cada estado.

## *4.4 Método W*

Esta técnica verifica se todos os estados e transições estão corretamente implementados. Primeiro se utiliza uma seqüência T, formada a partir da árvore de teste, e objetiva verificar se todas as transições do modelo foram atingidas.

Em seguida se utiliza a seqüência P, que é derivada de um conjunto de caracterização W. Esta seqüência P é utilizada para verificar que o estado atingido é o estado correto.

## *4.5 Algoritmo do Carteiro Chinês*

O algoritmo do Carteiro Chinês é bastante conhecido e utilizado. Esta técnica garante que todas as transições ou estados sejam testados através do menos caminho possível. No início, esta técnica foi pensada para diagramas com transições não direcionadas, porém, a maioria dos modelos baseados em testes é direcionada, então a técnica foi adaptada e também funciona para modelos direcionados.

## *4.6 Fault Transition Pair Coverage*

Um par de transições de falta é um par que combina uma transição valida de entrada para um determinado estado, com uma transição de saída para uma falta, para este mesmo estado.

A técnica de *Fault Transition Pair Coverage* gera um caso de teste para cada par de transições de falta. Para isto, é gerada uma seqüência de passos mínimos que liguem o estado inicial ao estado do par.

## *4.7 Método BZ-TT*

Esta técnica consiste em testar operações em um modelo com entradas com valores limites em estados limites. Um estado limite é um estado que contém ao menos uma variável que tem um valor no extremo.

## *4.8 User Defined Test Objectives*

Utilizando a técnica de cobertura de projeção, a técnica *User Defined Test Objectives* permite que o usuário defina objetivos para seus testes, e represente na forma de restrições nos estados e transições do seu modelo.

# 5. Conclusão

Nesta seção são apresentadas as considerações finais e os trabalhos futuros.

## *5.1 Considerações Finais*

Seção será concluída no término do projeto.

## *5.2 Trabalhos Futuros*

Seção será concluída no término do projeto.

## 6. Referências


[1] RIOS, Emerson; MOREIRA, Trayahú. Teste De Software.

[2] JONES, T.Capers. Patterns of Software System Failure and Success.

[3] VIANA, Virginia. Um Método para Seleção de Testes de Regressão para Automação, 2006.

[4] Wikipedia, http://pt.wikipedia.org/wiki/Teste_de_software. Acessado em julho de 2007.

[5] BARTIE, Alexandre. Garantia da Qualidade de Software, 2002.

[6] Software QA/Test Resource Center, http://www.softwareqatest.com. Acessado em agosto de 2007.

[7] MYERS, Glenford. The Art of Software Testing.

[8] BASTOS, Anderson; RIOS, Emerson; CRISTALLI, Ricardo; MOREIRA, Trayahú. Base de Conhecimento em Testes de Software, 2006.

[9] Site: http://paginas.fe.up.pt/~softeng/wiki/doku.php?id=public_events:advanced_training:mbt. Acessado em novembro, 2008.

[10] Binder, R. Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley, 2000.

[11] DELAMARO, Márcio Eduardo; MALDONADO, José Carlos; JINO, Mario. Introdução ao Teste de Software, Editora Campus, UFRJ, 2007.

[12] GÖNEC, G. A method for the design of fault-detection experiments. IEEE Transactions on Computers, 551-558, jun, 1970.

[13]TAKAHASHI, Juichi; KAKUDA, Yoshiaki. Extended-model based testing by directed Chinese Postman Algorithm. Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering, 2002.

[14] BELLI, Fevzi; HOLLMANN, Axel. Test Generation and Minimization with "Basic" Statecharts. Proceedings of the 23rd Annual ACM Symposium on Applied Computing, 2008.

[15] LEGEARD, Bruno; PEUREUX, Fabien; UTTING, Mark. A comparison of the btt and tff test-generation methods. In Proc. Of ZB´02, 309-329, 2002.

[16] PARADKAR, Amit. Case Studies on fault detection effectiveness of model based test generation techniques. Proceedings of the Advances in Model-Based Software Testing, 2005.