# TESTE DE SOFTWARE E QUALIDADE DE SOFTWARE: UMA VISÃO GERAL

Renato de Oliveira Moraes
Faculdades Tibiriçá – Lab. de Engenharia de Software - E-mail: renato.moraes@peceptron.com.br
Rua Pe Benedito Maria Cardoso, bl f-10 apto 404 03169-060 – Moóca – São Pauo, SP
Fernando José Barbin Laurindo
Depto. de Engenharia de Produção da Escola Politécnica da USP. E-mail: fjblau@usp.br
Av. Prof. Almeida Prado, 128 Tr.2 Biênio 2° and. - 05508-900 – C. Universitária - São Paulo, SP

ABSTRACT

Nowadays, there are many software applications affecting people in their day-by-day activities in professional and personal life. This wide range of applications facilitates the work and the life of many people, but , on the other hand, exposes the users to software failure risks. This paper emphasizes the importance of systematic process for testing software, based on two well known models: ISO9000 and CMM.

KEYWORDS: software process, software quality, software process.

RESUMO:

Atualmente, há inúmeras aplicações de software afetando as pessoas no cotidiano de suas atividades pessoais e profissionais. Este amplo universo de aplicações facilita o trabalho e a vida inúmeras pessoas, mas, por outro lado, expõe os usuários aos riscos de falhas de software. Este artigo ressalta a importância de processos sistemáticos para testar softwares, baseando-se em dois modelos bastante conhecidos: ISO9000 e CMM

# 1 INTRODUÇÃO

A cada dia o *software* torna-se um elemento cada vez mais corriqueiro na vida das pessoas. Ele está no forno de microondas, no elevador, nas mesas de trabalho, nas agências e serviços bancários, nos automóveis, nos aviões e aeroportos. Todos estes bens e serviços têm algum tipo de recurso de Tecnologia da Informação suportando sua operação.

O tamanho e a complexidade dos produtos de software têm crescido fazendo com que a incidência de erros e não conformidades também aumentasse. Modelos de gestão de processo propõem atividades de garantia da qualidade para a obtenção de melhores produtos nas organizações de desenvolvimento de software. Uma destas atividades é a de teste de software. Este trabalho apresenta um visão geral da atividades de teste e a sua relação com a qualidade do produto de software.

# 2 GESTÃO DA QUALIDADE E TESTE DE SOFTWARE

Existem vários modelos que podem ser utilizados nas organizações de desenvolvimento de software. Contudo, no Brasil, dois deles têm tido uma utilização significativamente maior que os demais. São eles o modelos ISO 9000 e o CMM (Capability Maturity Model).

Dentro da série o ISO 9000, a norma ISO 9001 –Sistemas da Qualidade – Modelos para garantia da qualidade em projeto, desenvolvimento, produção, instalação e serviços associados - é a indicada para organizações de desenvolvimento de software.

O CMM, mais recente e menos popular (dada sua natureza de modelo específico para processo de software) tem despertado grande interesse entre os profissionais de desenvolvimento. Prova disto é a existência de grupos (em São Paulo, Rio de Janeiro e Curitiba) de pessoas interessadas em discutir e trocar experiências relacionadas a este modelos. Estes grupos chamam-se SPIN – *Software Process Improvement Network.*

## 2.1 ISO 9001

Esta norma tem caráter bastante genérico para que ela possa ser aplicada em qualquer tipo de organização. A ISO, reconhecendo as especificidades do processo de desenvolvimento de software, criou a ISO 9000-3 – Diretrizes para a aplicação da ISO 9001 ao desenvolvimento, fornecimento e manutenção de software. Esta norma, dedicada à processo de desenvolvimento de software estabelece um sistema da qualidade com uma série de atividades do ciclo de vida. Dentre estas atividades, que

independem do modelo de ciclo de vida adotado pela organização, esta a atividade de testes de software: item 5.7 Ensaios e Validação (NBR ISO 9000-3, ANTONIONI) .

## 2.2 CMM - Capability Maturity Model

O desenvolvimento do modelo CMM começou em 1986, na Universidade de Carnegie Mellon (PAULK, 1994) , para atender a uma necessidade do Departamento de Defesa norte americano de avaliar seus fornecedores de *software*. A primeira versão do CMM foi divulgada em 1991 e revista em 1993. Até o momento da produção deste artigo, a segunda versão do modelo ainda não estava concluída.

Este modelo, como o da ISO, prevê que organizações com processos maduros tendem a produzir produtos de melhor qualidade. Estabelece cinco níveis de maturidade hierarquizados que, ao serem percorridos, levam as organizações a um nível de excelência em seus processo de desenvolvimento de software. Estes níveis são mostrados na tabela 1 e na figura 1.

| **Nível de Maturidade** | **Característica** |
|---|---|
| 1. Inicial | Qualquer organização, por pior que seja, está neste nível. Não existe nenhuma condição para uma organização ocupar este nível. |
| 2. Repetível | As organizações, neste nível, confrontam seus produtos contra requisitos preestabelecidos. |
| 3. Definido | Os processos são planejados e executados segundo procedimentos conhecidos e entendidos pelas pessoas envolvidas. |
| 4. Gerenciado | A administração possui indicadores do processo que permitem um acompanhamento quantitativo do seu desenvolvimento. |
| 5. Otimizado | A organização possui processos extremamente confiáveis e é capaz de prever falhas e alterar o processo e/ou a tecnologia envolvida para obter melhores resultados. |

Tabela 1 Níveis de Maturidade do CMM

Fonte: PAULK, 1994

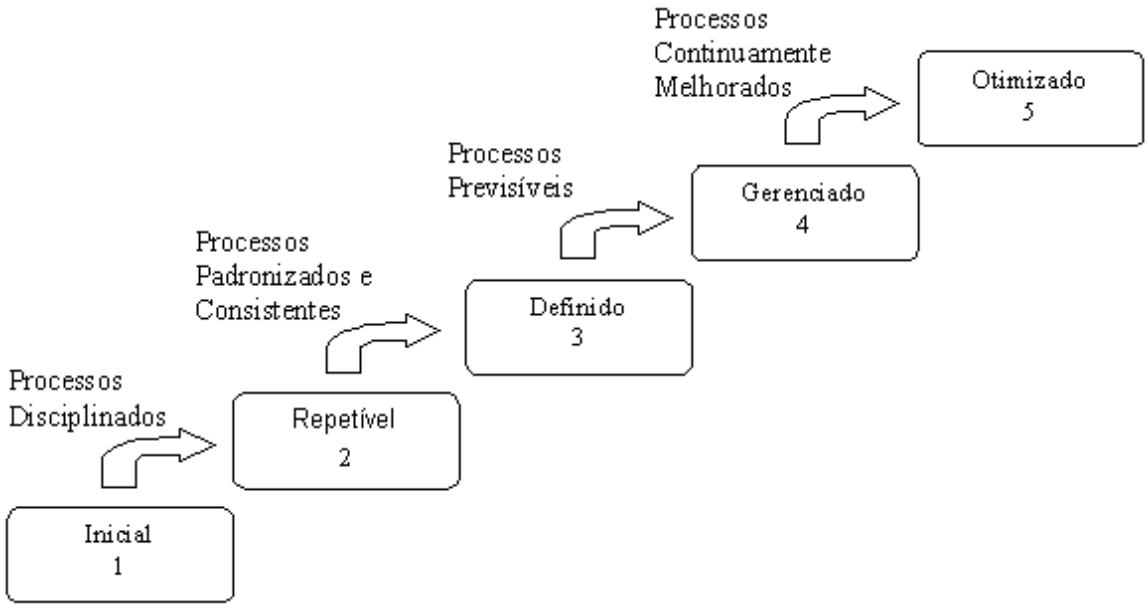


Figura 1 CMM – níveis do maturidade

Fonte: adaptado de PAULK, 1994

Em cada um dos níveis de maturidade, excetuando o primeiro, existe um conjunto de áreas chave de processo. Para atingir um determinado nível, todas as áreas-chave deste nível devem estar implementadas.

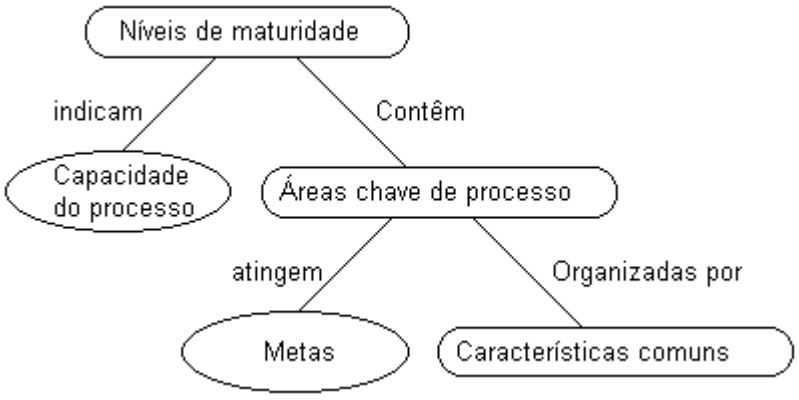


Figura 2 CMM – estrutura

Fonte: adaptado de PAULK, 1994

| Nível | Áreas Chave de Processos |
|---|---|
| 2 - Repetível | Gerenciamento de requisitos |
| | Planejamento do Projeto de Software |
| | Acompanhamento e Supervisão do Projeto de Software |
| | Gerenciamento da Subcontratação de Software |
| | Garantia da Qualidade de Software |
| | Gerenciamento da Configuração de Software |
| 3 - Definido | Foco no Processo da Organização |
| | Definição do Processo da Organização |
| | Programa de Treinamento |
| | Gerenciamento Integrado de Software |
| | Engenharia do Produto de Software |
| | Coordenação Inter-Grupos |
| | Revisões aos pares<br>*Peer reviews* |
| 4 - Gerenciado | Gerenciamento Quantitativo do Processo |
| | Gerenciamento da Qualidade de Software |
| 5 - Otimizado | Prevenção de Defeitos |
| | Gerenciamento da Mudança de Tecnologia |
| | Gerenciamento da Mudança do Processo |

Tabela 2 Áreas chave de Processos

Fonte: adaptado de PAULK, 1994

As áreas chave de processo estão organizadas por "Características comuns" (common features). Estas características comuns são atributos que indicam se uma determinada área chave foi implementada e institucionalizada dentro da organização. As cinco características comuns são:

| | |
|---|---|
| Compromissos em executar: | São as ações que a organização deve realizar para garantir que o processo seja estabelecido e seja duradouro. Incluem práticas políticas e de treinamento. |
| Habilidades em executar: | São as condições necessárias em um projeto ou organização para implementar competentemente o processo de software. Inclui práticas relacionadas a recursos, treinamento, orientação, estrutura organizacional, e ferramentas. |
| Atividades executadas: | São as funções e procedimentos necessários para implementar a área chave de processo. Inclui práticas em planejamento, procedimentos, trabalho realizado, acompanhamento, e ação corretiva. |
| Medidas e análises: | São os procedimentos necessários para medir o processo e avaliar as medições. Inclui práticas nos processos de medição e análise. |
| Verificação da implementação: | São os passos necessários para garantir que as atividades sejam executadas de acordo com o processo estabelecido. Inclui práticas de revisão administrativa (*management review*) e auditoria. |

É na área chave de processo do nível 2 Garantia da Qualidade de Software, que está a atividade de teste de software. O modelo, como o da ISO 9000, não diz como devem ser gerados os casos de teste. Ele apenas explicita a necessidade de procedimentos formais na condução dos testes de software em desenvolvimento.

## 3 A ATIVIDADE DE TESTE

A atividade de teste traz uma necessidade de mudança de postura para o desenvolvedor. Se durante as etapas anteriores do processo de desenvolvimento a preocupação é a de criar/construir um produto que atende às especificações implícitas e explícitas do produto, na atividade de teste o desenvolvedor

tentará provar que o produto que ele produziu não atende às especificações. O uso de procedimentos formais nesta etapa ajuda o desenvolvedor a realizar esta “inversão de papéis”.

Se utilizarmos o atendimento às especificações como definição de qualidade (como faz a ISO 9000-3), a conformidade do produto de software em relação às especificações já esta determinada antes de iniciar os testes. A atividade de teste deve apenas procurar evidenciar o grau de atendimento a estas especificações.

Não é possível, durante os testes, executar o programa de todas as formas possíveis de modo a verificar todas as possibilidades de erro. Isto porque a complexidade algoritmica e o tamanho do software torna o número de caminhos que o programa pode seguir extremamente grande. Tentar cobrir todos estes caminhos torna o software mais caro e o tempo de desenvolvimento maior.

Desta maneira, os casos de teste selecionados devem ter uma alta probabilidade de encontrar um erro no programa. É importante lembrar que um caso de testes não provoca um erro no software, apenas o evidencia. O que provoca erro no software é especificação de requisitos incorreta, mudança de requisitos, projeto inadequado, módulos mal especificados, erros de codificação, entre outras fontes de mão conformidade.

Se o teste pode evidenciar a existência de erros, infelizmente ele não pode mostrar que um software esta livre de erros. Um caso de teste, ou um conjunto de casos de teste, que quando executado não evidencia um erro não prova que não existe erro: apenas mostra que nenhum erro foi encontrado. Assim, um caso de teste bem sucedido é aquele que encontra um erro.

Na prática, quando o desenvolvedor testa o programa que acabou de construir ele deseja, no seu íntimo, que tudo esteja correto e livre de problemas. Nada mais natural. Contudo, esta postura não é a mais conveniente para a atividade de testes. É necessária a participação de pessoas isentas, que não tenham participado do desenvolvimento, na condução dos testes. Estas pessoas devem pertencer a uma área de garantia da qualidade. Em organizações menores onde não exista esta área, pode-se utilizar alguém da equipe de desenvolvimento que não tenha participado do software que irá ser testado para executar esta função de coordenação dos testes.

### 3.3 Tipos de teste de software

Este item descreve os tipo de teste, seus objetivos e as técnicas de geração de casos de testes

### 3.3.1 Testes de verificação

Os testes de verificação são atividades conduzidas pela organização de desenvolvimento com o objetivo de verificar se os resultados obtidos até o momento atendem às especificações realizadas em etapas anteriores, e garantir que as ações, atividades e decisões entre as etapas de desenvolvimento mantenham sua consistência. Assim, os testes de verificação tem uma ótica da equipe de desenvolvimento.

### 3.3.2 Testes de validação

Durante o teste de validação, a organização desenvolvedora procura avaliar quanto o software obtido atende às especificações de requisitos estabelecidas em conjunto com o cliente. Aqui a atividade de testes tem a ótica do cliente.

### 3.3.3 Testes de aceitação

Os testes de aceitação são conduzidos pelo cliente com objetivo de avaliar o produto que esta em vias lhe ser enviado. A organização de desenvolvimento participa destes testes.

A forma e os critérios a serem utilizados no testes de aceitação são acordados entre as partes logo nas fases iniciais do desenvolvimento.

## 3.4 Geração de casos de teste

O projeto de casos de testes pode ser feita, basicamente, utilizando duas estratégias (PRESSMAN, 1995) chamadas:

- caixa branca; e
- caixa preta.

### 3.4.4 Teste de caixa branca

Nesta estratégia os casos de testes são gerados a partir da análise da estrutura do programa a ser testado. É a través da estrutura de controle e da estrutura de dados que os casos de teste são gerados. Existem várias técnicas dentro desta estratégia de testes:

**Teste de caminho básico**

Utilizando-se esta técnica gera-se um conjunto (mínimo) de casos de testes que faz com que cada instrução do programa seja executada pelo menos uma vez. Com este tipo de teste procura-se identificar instruções incorretas e impróprias no programa. É claro que outros tipos de erros (como erros de estruturas de dados ) podem ser encontrados durante a execução destes casos de teste.

Constrói-se, para o programa ser testado, o grafo de fluxo cujo valor da complexidade ciclomática corresponde à quantidade de casos de testes que dever ser gerada para que cada instrução do programa seja executada pelo menos uma vez.

**Teste das estruturas de controle – Teste de condição**

Aqui procura-se encontrar erros nas condições lógicas do programa:

- Erro no operador booleano ( OR, AND e NOT);
- Erro de variável booleana;
- Erro de parênteses booleanos;
- Erro de operador relacional;
- Erro de expressão aritmética.

Para cada expressão relacional do tipo: $E_1$ <operador relacional> $E_2$ , são realizados 3 testes em que a expressão $E_1$ é maior, igual e menor do que $E_2$.

Para expressões compostas, as expressões simples são testadas individualmente e de forma cruzada. Assim uma expressão booleana com *n* variáveis requer $2^n$ casos de teste.

**Teste das estruturas de controle – Teste de laços**

Os casos de teste gerados através desta técnica se concentram na análise das estruturas de repetição do programa. Existem quatro tipos de laços que podem ser encontrados num programa, o que leva a maneiras ligeiramente diferentes de testá-los:

Laço simples:

- Pule o laço inteiramente
- Somente uma passagem pelo laço
- Duas passagens pelo laço
- Realize *M* passagens pelo laço.
- Realize *N-1*, *N*, *N+1* passagens pelo laço. Onde *N>M* .

Laços aninhados:

- Comece pelo laço localizado na parte mais interna da estrutura. Fixe os demais laços em valores mínimos
- Realize os testes de laços simples para este laço mais interno.

| | |
|---|---|
| | • Vá para o próximo laço mais externo mantendo os laços internos em valore padrões. |
| Laços concatenados: | • Se os contadores dos laços forem independentes, teste os laços de forma independente com os casos de laços simples.<br>• Caso contrário, utilize os casos de laços aninhados. |
| Laços não estruturados: | Reescreva seu programa, de forma estruturada, para substituir estes laços por outros dos tipos acima. Conceitualmente isto é sempre possível. |

### 3.4.5 Teste de caixa preta

A geração de casos é feita com base nos requisitos funcionais do programa a ser testado.

**Particionamento de equivalência**

(a) Se um valor de entrada especificar uma faixa continua de valores válidos, deverão ser gerados três casos de testes para esta valor: um com o valor dentro da faixa, outro com o valor abaixo do intervalo, e outro acima do intervalo.

(b) Se uma valor de entrada for booleano, dois casos de testes deverão ser gerados.

(c) Se um valor de entrada for um elemento de um conjunto, serão gerados dois casos de testes: um válido e outro inválido.

(d) Se um valor de entrada exigir um valor específico, deverão ser gerados dois casos de testes: um válido e outro inválido.

**Valor limite**

Esta técnica, parecida com a anterior, procura identificar os problemas, que costumam ocorrer, nos limites dos intervalos de valores válidos.

(a) Se um valor de entrada especificar uma faixa continua de valores válidos limitada por ***a*** e ***b***, deverão ser gerados seis casos de teste:

- com o valor logo abaixo de ***a***
- com o valor igual ***a***

- com o valor logo acima de ***a***
- com o valor logo abaixo de ***b***
- com o valor igual a ***b***
- com o valor logo acima de ***b***

(b) Se um valor de entrada for um elemento de um conjunto, serão gerados de teste

- com o valor mínimo do conjunto
- com um valor logo abaixo do valor mínimo do conjunto
- com o valor máximo do conjunto
- com um valor logo acima do valor máximo do conjunto

(c) Gere casos para os valores de saída também

**Grafo de causa-efeito**

Esta técnica procura estabelecer uma relação lógica entre as condições de entrada e a saídas possíveis do software. Ela esta dividida em quatro etapas:

(a) Identificação das causas (condições de entrada) e efeitos (ações).
(b) Desenvolvimento de um grafo que relaciona as causas aos efeitos.
(c) Conversão do grafo em uma tabela de decisão
(d) Conversão das regras da tabela de decisão e casos de teste

**Testes de comparação**

Utilizada em aplicações críticas, esta técnica consiste em criar duas versões redundantes do software, desenvolvidas independentemente, que serão testadas sob as mesmas condições. Independentemente do fato de apenas uma versão ser implementada, os resultado deverão ser idênticos nas duas versões.

## 4 CONCLUSÃO

A atividade de teste, por sí só, não é capaz de tornar o software livre defeitos. Contudo, quando formalmente feita pode reduzir o número de não conformidades e ainda fornecer dados importantes para uma estimativa da confiabilidade do software. Sua implementação exige, além da capacitação técnica da equipe de desenvolvimento para torná-la capaz de gerar casos de testes consistentes e de alta probabilidade de evidenciar erros, uma estrutura organizacional que permita incorporar a atividade de testes às atividades do ciclo de desenvolvimento e a participação de pessoas isentas na execução dos testes.

**BIBLIOGRAFIA**

ANTONIONI, J. A. & ROSA, N. B. Qualidade em software: manual de aplicação da ISO 9000, São Paulo, Makron Books, 1995.

CHATZOGLOU, Prodromos D. & MACAULAY, Linda A. A review of existing models for project planning nad estimation for project planning and estimation and the need for a new approach IN: International Journal of Project Management, Vol. 14 No. 3 pp. 173-183 1996.

NBR ISO 9000-3 Normas de gestão da qualidade e garantida da qualidade – Parte 3: Diretrizes para a aplicação da NBR 19001 ao desenvolvimento, fornecimento e manutenção de software. ABNT, 1990

NBR ISO 9001 Sistemas da qualidade – Modelo para a garantia da qualidade em projeto, desenvolvimento, produção, instalação e serviços associados ABNT, 1994

MUNNS, A. K. & BJEIRMI, B. F. The role of project management in achieving project success. IN: International Journal of Projetc Management vol 14 no. 2 pp 81-87, 1997.

PAULK, Marc C. et all "The Capability Maturity Model: Guidelines for Improving the Software Process" Addison-Wesley, 1994.

PRESSMAN, R. S. A manger's guide to the software engineering, McGraw-Hill, 1993.

PRESSMAN, R. S. Engenharia de Software, Makron Books, São Paulo 1995.

VILLAS-BOAS, A. e outros Gestão de configuração na atividade de teste - Workshop de Qualidade de Software 1997, Fortaleza.

WEBER, K. C. (org) Qualidade e Produtividade em Software 2. ed., São Paulo, Makron Books, 1997.