**Qualidade de Software**

**1 – Histórico e Conceitos**

**1.1 Introdução**

Quando se fala em qualidade pode-se remontar ao início da organização do homem para comercializar seus produtos há milhares de anos atrás. O conceito de qualidade muda de pessoa para pessoa, de local para local, de época para época e, por isso, pode ser considerado como uma noção subjetiva do valor de um produto.

No entanto, somente ao final do século XIX é que o homem, com o surgimento da industrialização, passou a estudar e implantar conceitos de padrões de qualidade com o intuito de evitar os retrabalhos e a entrega dos produtos com problemas ao consumidor final. Nesta época era o próprio operário que com o conhecimento adquirido, com a experiência e com os ensinamentos recebidos procedia a aceitação ou provocava ajuste ou rejeitava o produto.

Já no século XX, teve início a verificação da qualidade como uma tarefa para pessoas mais qualificadas, denominadas de supervisores, que ao fazer a inspeção do trabalho executado pelos seus subordinados atuavam para evitar os retrabalhos e a rejeição de produtos. Com o passar do tempo essa forma de atuação avançou em direção à inspeção dos produtos, realizadas por inspetores de qualidade.

**1.2 O que é qualidade?**

*A qualidade é relativa. O que é qualidade para uma pessoa pode ser falta de qualidade para outra. – G. Weinberg*

Existem diversas definições. Algumas pessoas que tentaram uma definição simples chegaram a frases como:

* Qualidade é estar em conformidade com os requisitos dos clientes;
* Qualidade é antecipar e satisfazer os desejos dos clientes;
* Qualidade é escrever tudo o que se deve fazer e fazer tudo o que foi escrito.

Segundo a atual norma brasileira sobre o assunto (NBR ISO 8402), qualidade é: *“A totalidade das características de uma entidade que lhe confere a capacidade de satisfazer às necessidades explícitas e implícitas”.*

De acordo com Pressman (2001), qualidade de software é a conformidade a: (1) requisitos funcionais e não funcionais que têm sido explicitamente declarados, (2) padrões de desenvolvimento que tenham sido claramente documentados e (3) características implicitamente esperadas de todo software a ser desenvolvido.

Alexandre Bartié em seu livro enfatiza que "Qualidade de software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos" (BARTIÉ, 2002).

De forma resumida, qualidade consiste de um conjunto de requisitos e de um produto ou serviço que esteja em conformidade com estes requisitos e, por esta razão, atenda completamente às necessidades dos clientes. Não se pode pensar em qualidade como sinônimo de perfeição. Trata-se de algo factível, relativo, substancialmente dinâmico e evolutivo, de acordo com os objetivos a serem atingidos. Considerá-la como algo absoluto e definitivo seria como igualar ao inatingível e, com base neste pensamento, dificilmente conseguiríamos produzi-la (Belchior, 1997).

Atualmente, muitas instituições se preocupam em criar normas para permitir a correta avaliação de qualidade tanto de produtos de software quanto de processos de desenvolvimento de software. Apenas para ter uma visão geral, observe o quadro abaixo com as principais normais nacionais e internacionais nesta área:

|  |  |
| --- | --- |
| **Norma** | **Comentário** |
| ISO 9126 | Características da qualidade de produtos de software. |
| NBR 13596 | Versão brasileira da ISO 9126 |
| ISO 14598 | Guias para a avaliação de produtos de software, baseados na utilização prática da norma ISO 9126 |
| ISO 12119 | Características de qualidade de pacotes de software (software de prateleira, vendido com um produto embalado) |
| IEEE P1061 | Standard for Software Quality Metrics Methodology (produto de software) |
| ISO 12207 | Software Life Cycle Process. Norma para a qualidade do processo de desenvolvimento de software. |
| NBR ISO 9001 | Sistemas de qualidade - Modelo para garantia de qualidade em Projeto, Desenvolvimento, Instalação e Assistência Técnica (processo) |
| NBR ISO 9000-3 | Gestão de qualidade e garantia de qualidade. Aplicação da norma ISO  9000 para o processo de desenvolvimento de software. |
| NBR ISO 10011 | Auditoria de Sistemas de Qualidade (processo) |
| CMM | Capability Maturity Model. Modelo da SEI (Instituto de Engenharia de Software do Departamento de Defesa dos EEUU) para avaliação da qualidade do processo de desenvolvimento de software. Não é uma norma ISO, mas é muito bem aceita no mercado. |
| SPICE  ISO 15504 | Projeto da ISO/IEC para avaliação de processo de desenvolvimento de software. Ainda não é uma norma oficial ISO, mas o processo está em andamento. |

Entre as atividades que podem ser utilizadas para verificar a qualidade de software se encontram: Revisões de software, testes, etc.

A importância das revisões na garantia da qualidade é destacada pelo modelo CMMI, que exige a realização de revisões como uma prática específica do processo de verificação. Sabe-se ainda que inspeções de software, em particular, capturam em torno de 60% dos defeitos de artefatos (BOEHM e BASILI, 2001) o que deixa explícita a sua contribuição para a melhoria da qualidade. Uma outra maneira de detectar defeitos em artefatos é através da aplicação de testes. No entanto, a aplicação de testes descobre apenas sintomas de problemas e, desta forma, pode ocasionar um trabalho refinado e custoso para detectar o defeito que causou o sintoma.

BOEHM e BASILI (2001) ressaltam ainda que inspeções e testes capturam diferentes tipos de defeito e em diferentes momentos do processo de desenvolvimento de software. Portanto, é interessante aplicar tanto as inspeções quanto testes para detectar defeitos em artefatos de software. Além disto, precedendo os testes com as inspeções, defeitos podem ser removidos nas fases iniciais do processo de desenvolvimento e os desenvolvedores terão uma visão mais ampla da complexidade do sistema. Esta visão os deixa mais bem preparados para o momento em que se confrontam com esta complexidade e com os defeitos que podem possivelmente surgir durante os testes.

**2 Processos e a Qualidade**

A organização de um processo de desenvolvimento de software é um fator criterioso para o sucesso do modelo adotado e tem como base para os principais modelos de qualidade atuais que citamos como o CMMI e o MPS.BR.

**2.1 CMMI**

Chrissis diz, de uma forma sucinta, que o CMMI (Capability Maturity Model Integration – Modelo Integrado de Capacitação e Maturidade) criado pela SEI (Software Engineering Institute), consiste nas melhores práticas direcionadas ao desenvolvimento e à manutenção de produtos e dos serviços, abrangendo todo o ciclo de vida do produto, desde sua concepção até a sua entrega e manutenção (CHRISSIS; KONRAD; SHRUM, 2003). É um dos modelos de maturidade mais aceitos no mundo no que diz respeito a medir a maturidade dos processos de uma organização. Quanto mais maduro forem seus processos, maior será a qualidade obtida no produto final, pois os níveis de maturidade ajudam a aprimorar seus processos. Isso ocorre, pois, prevendo o comportamento dos processos, torna-se a organização mais madura e competitiva no mercado.

**2.2 MPS.BR**

O modelo Melhoria de Processo de Software Brasileiro (MPS.BR) assim como o CMMI, é baseado em níveis de maturidades, voltado para pequenas e médias empresas e adaptado para perfil cultural das empresas brasileiras tendo como objetivo definir e aprimorar um modelo de melhoria e avaliação de processo de software de modo compatível com os padrões internacionais de qualidade de software. O MPS.BR é coordenado pela Associação para Promoção da Excelência do Software Brasileiro (SOFTEX) e conta com o apoio do Ministério da Ciência e Tecnologia (MCT), Financiadora de Estudos e Projetos (FINEP) e do Banco Interamericano de Desenvolvimento (BID) (SOFTEX, 2009). A escala de níveis pode ser expressa da seguinte forma:

* G – Parcialmente Gerenciado;
* F – Gerenciado;
* E – Parcialmente Definido;
* D – Largamente Definido;
* C – Definido;
* B – Gerenciado Quantitativamente;
* A – Em Otimização.

**3 Teste de Software**

**3.1 Conceitos básicos**

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.

* 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 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 aumentar as possibilidades de provocar falhas ou, quando isso não ocorre, estabelecer um nível elevado de confiança na correção do produto (ROCHA et al., 2001).
* 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.
* 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).
* 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).

Roger S. Pressman descreve testes de unidade da seguinte maneira:

*O teste de unidade focaliza o esforço de verificação na menor unidade de projeto do software – componente ou módulo de software. Usando a descrição de projeto no nível de componente como guia, caminhos de controle importantes são testados para descobrir erros dentro dos limites do módulo e as estruturas de dados dentro dos limites de componentes. Esse tipo de teste pode ser conduzido em paralelo para diversos componentes (PRESSMAN, 2002, p. 295)*

**3.2 Técnicas de testes 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.

**3.3 Técnica Estrutural (ou teste caixa-branca)**

Técnica de teste que avalia o comportamento interno do componente de software. 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 originadas por estruturas de condições são testadas. 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 Técnica 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.

**3.4 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. Dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado 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).

**4 Melhoria Contínua**

A melhoria contínua é o grande objetivo dos programas de qualidade e produtividade. Melhoria é a transição para um melhor estado ou condição, normalmente, gerando vantagens. Para Harrington (1993), a melhoria contínua é a busca da perfeição. Para tal, a melhoria contínua vai além da definição de qualidade que, segundo ele, “é sempre fazer corretamente o trabalho”, assumindo que, perfeição, “é sempre fazer corretamente o trabalho certo”, com o objetivo de satisfazer os clientes internos e externos. Os clientes são assim definidos pelo autor: Clientes externos são aqueles de fora da empresa, que recebem o produto ou serviço final. Clientes internos são aqueles localizados dentro da cadeia de atividades da organização, que não recebem diretamente a saída do processo, mas são afetados se o processo gerar saídas erradas ou atrasadas. Um fator diretamente relacionado para a melhoria contínua de um processo de qualidade é a gestão do conhecimento, partindo do pressuposto que o conhecimento em um projeto relacionado com pessoas, à gestão destas pessoas é uma forma de gerir o conhecimento. Nada tão importante como o conhecimento de quem faz parte do processo. Gerenciar corretamente um projeto, não é somente seguir o processo com base no que foi feito, é também incentivar e ajudar os demais a fazerem o mesmo, pois um projeto envolve a todos. Segundo Martin15 (1998), é comum pensar somente nas grandes melhorias, porém pequenas mudanças podem ser grandes nos quesitos qualidade e produtividade. A melhoria não é um fim em si próprio, portanto precisa ser contínua. O autor coloca, também, que a melhoria contínua de processos é baseada no método japonês chamado KAIZEN, em que “todos melhoram tudo, o tempo todo”. Ou seja, cada participante da organização identifica problemas, fazem análises e propõem soluções.

Referencias

professores.dcc.ufla.br/~terra/publications\_files/.../2009\_fumec\_souza\_e\_monteiro.pdf

Pressman, R. S. (2001). Software engineering: a practitioner's approach. McGraw-Hill, 5th edition.

WEINBERG, G. Software com qualidade. V. 1, 2 e 3. Makron Books, 1997.

www.martinsfontespaulista.com.br/anexos/produtos/capitulos/241804.pdf

sidneyvieira.kinghost.net/abas/disciplinas/download/ESI/es\_alta.pdf