

DAMIÃO MARTINS DA SILVA

# **Benefícios da Aplicação de Processos de Testes em Projetos de Desenvolvimento de Softwares**

São Paulo

2013

DAMIÃO MARTINS DA SILVA

# **Benefícios da Aplicação de Processos de Testes em Projetos de Desenvolvimento de Softwares**

Trabalho de conclusão de curso para obtenção do título  
de Especialista em Gerenciamento de Projetos  
pela Faculdade de Tecnologia de São Paulo.

Orientador: Dra. Fernanda M. P. F. Ramos Ferreira

São Paulo

2013

## **AGRADECIMENTOS**

Agradeço a todos que acreditaram em mim para a realização desse trabalho quando nem mesmo eu acreditava.

Agradeço a FATEC-SP e a todos os Professores, na pessoa da Dra. Fernanda M.P.F. Ramos Ferreira, que com paciência e incentivo na orientação tornou possível a conclusão da monografia desse curso de especialização.

Aos amigos Marcos Marangoni e Márcia Shimada pelo companheirismo nessa árdua caminhada de conhecimento.

A todos que de alguma forma colaboraram para que eu conseguisse concluir todas as etapas acadêmicas até hoje.

Em especial, a minha esposa Camila e minha filha Laís, todo esse esforço e dedicação não seriam possíveis sem elas.

MUITO OBRIGADO!

*Em memória de meus pais,*

*Domiciano Lameu da Silva & Margarida Martins da Silva*

## RESUMO

A proposta desse trabalho é realizar um breve estudo sobre as visões de alguns autores que são referências no processo de Gestão da Qualidade e que conceberam com representatividade a revolução nessa área, analisando conceitos, criando ferramentas e técnicas de qualidade em gerenciamento de projetos de software. Com base nessas obras, esse trabalho foi elaborado com o intuito de realizar uma breve análise do Processo de Desenvolvimento de Software especificamente na importância do Processo de Testes na Gestão da Qualidade. Finalmente será mostrado um estudo de caso baseado em uma grande instituição financeira brasileira de nome fictício “Banco X”, onde será possível visualizar que até empresas com grandes estruturas tecnológicas e financeiras encontram muita dificuldade de atingir níveis de qualidade fundamentais por comprometerem fases do Processo de Testes em função da redução de custos e comprometimento de prazos de entrega.

Palavras-chave: Qualidade, Teste, Software, Processo de Testes, Gerenciamento de projeto de Software.

## **ABSTRACT**

The purpose of this work is to make a brief study on the views of some authors who are references in the Quality Management process and representativeness conceived with the revolution in this area, analyzing concepts, creating tools and techniques of quality management of software projects. Based on these works, this work was done in order to make a brief analysis of the Software Development Process specifically on the importance of Testing Process in Quality Management in companies. Finally a case study based on a big Brazilian financial institution fictitious name " Bank X" , where you can see that even big companies with technological and financial structures are very difficult to achieve basic levels of quality for the commit phase process will be shown tests due to the reduction of costs and commitment to deadlines.

Keywords: Quality, Testing, Software, Test Process, Project Management Software.

## LISTA DE ILUSTRAÇÕES

Figura 1 - Modelo Cascata .....	- 18 -
Figura 2 – Modelo Incremental .....	- 19 -
Figura 3 – Modelo Espiral .....	- 20 -
Figura 4 – Os Fatores da Qualidade.....	- 24 -
Figura 5 - Visão Geral do Gerenciamento da Qualidade do Projeto.....	- 29 -
Figura 6 - Planejar a Qualidade .....	- 30 -
Figura 7 - Realizar Garantia da Qualidade .....	- 31 -
Figura 8 - Realizar Controle da Qualidade .....	- 32 -
Figura 9 - Estrutura da Norma ISO/IEC 12119 .....	- 42 -
Figura 10 - Detalhamento dos documentos produzidos durante o teste.....	- 66 -
Figura 11 - Detalhamento da composição de plano de testes.....	- 68 -

## **LISTA DE QUADROS**

Quadro 1 - Estrutura dos Guias da Norma ISO/IEC 14598. ....	- 42 -
Quadro 2 - Requisitos de qualidade para a Descrição do Produto.....	- 44 -
Quadro 3 - Requisitos de qualidade para a Documentação do Usuário .....	- 45 -
Quadro 4 - Requisitos de Qualidade para Programas e Dados.....	- 45 -
Quadro 5 - Instruções para Testes .....	- 46 -
Quadro 6 - Diagrama de Correspondência Requisitos-Teste .....	- 69 -
Quadro 7 – Testes Aplicados e seus Responsáveis .....	- 79 -
Quadro 8 - Método de Dupla Custódia de Testes .....	- 82 -



## LISTA DE ABREVIATURAS E SIGLAS

<b>ABNT</b>	–	Associação Brasileira de Normas Técnicas
<b>ASQ</b>	–	American Society for Quality Sociedade Americana para Qualidade
<b>C/S</b>	–	Cliente/Servidor
<b>CMM</b>	–	Capability Maturity Model Modelo de Maturidade em Capacitação
<b>CMMI</b>	–	Capability Maturity Model Integration Modelo de Maturidade em Capacitação Integração
<b>ERP</b>	–	Enterprise Resource Planning Sistemas Integrados de Gestão Empresarial
<b>GUI</b>	–	Graphical User Interfaces Interface Gráfica do Usuário
<b>IEC</b>	–	<i>International Electrotechnical Commission</i> Comissão Eletrotécnica Internacional
<b>IEEE</b>	–	Instituto de Engenheiros Eletricistas e Eletrônicos
<b>ISO</b>	–	<i>International Organization for Standardization</i> Organização Internacional para Padronização
<b>ITG</b>	–	Independent Test Group Grupo de Teste Independente
<b>KPA</b>	–	Key Process Areas Áreas Chaves do Processo
<b>NBR</b>	–	Norma Brasileira
<b>OO</b>	–	Orientado a Objetos
<b>PMI</b>	–	Project Management Institute Instituto de Gerenciamento de Projetos
<b>SEI</b>	–	Software Engineering Institute Instituto de Engenharia de Software
<b>SQA</b>	–	Software Quality Assurance Garantia de Qualidade de Software
<b>SQE</b>	–	Software Quality Engineer

Engenharia de Qualidade de Software

**SW-CMM** – *Software Engineering* Capability Maturity Model

Engenharia de Software Modelo de Maturidade em Capacitação

**TI** – Tecnologia da Informação

**TIC** – Tecnologia da Informação e Comunicação

## SUMÁRIO

<b>1. INTRODUÇÃO.....</b>	<b>- 12 -</b>
1.1. JUSTIFICATIVA E ANTECEDENTES .....	- 12 -
1.2. PROBLEMAS DESSA PESQUISA.....	- 12 -
1.3. OBJETIVO .....	- 12 -
1.4. DELIMITAÇÃO DA PESQUISA.....	- 13 -
1.5. ESTRUTURAÇÃO DO TRABALHO.....	- 13 -
<b>2. REVISÃO DA LITERATURA.....</b>	<b>- 14 -</b>
2.1. SOFTWARE.....	- 14 -
2.2. PROBLEMAS OCORRIDOS EM DESENVOLVIMENTO DE SOFTWARES .....	- 15 -
2.2.1. CAUSAS .....	- 15 -
2.2.2. CUSTOS E PRAZOS .....	- 16 -
2.2.3. DESENVOLVIMENTO DE SOFTWARE .....	- 16 -
2.2.4. PROTOTIPAÇÃO .....	- 21 -
2.2.5. AS ETAPAS DE DESENVOLVIMENTO .....	- 21 -
2.3. GARANTIA DA QUALIDADE DE SOFTWARE .....	- 23 -
2.4. OS FATORES DE QUALIDADE .....	- 24 -
2.4.1. GARANTIA DA QUALIDADE DE SOFTWARE.....	- 25 -
2.4.2. PROCESSOS DE GERENCIAMENTO DA QUALIDADE DO SOFTWARE .....	- 26 -
2.4.3. PLANEJAR A QUALIDADE.....	- 30 -
2.4.4. REALIZAR A GARANTIA DA QUALIDADE .....	- 30 -
2.4.5. REALIZAR O CONTROLE DA QUALIDADE .....	- 31 -
2.4.6. CUSTO DA QUALIDADE DE SOFTWARE .....	- 32 -
2.4.7. A QUALIDADE SEGUNDO A ISO .....	- 34 -
2.4.8. PLANO SQA (PLANO DE GARANTIA DE QUALIDADE DE SOFTWARE).....	- 34 -

2.4.9.	MODELOS E NORMAS DE QUALIDADE.....	- 37 -
2.4.10.	CMM / CMMI.....	- 38 -
2.4.11.	NORMAS ISO/IEC .....	- 40 -
2.5.	TESTES DE SOFTWARE .....	- 47 -
2.6.	DOCUMENTAÇÃO DE TESTES .....	- 66 -
2.7.	PLANO DE TESTES.....	- 67 -
2.8.	ESPECIFICAÇÃO E AVALIAÇÃO DOS TESTES .....	- 68 -
2.9.	DESCRIÇÃO DOS TESTES .....	- 70 -
2.10.	RELATÓRIO DE ANÁLISE DE TESTE .....	- 70 -
2.11.	FORMULÁRIO DE RELATÓRIO DE PROBLEMAS .....	- 71 -
<b>3.</b>	<b>METODOLOGIA DE PESQUISA .....</b>	<b>- 73 -</b>
3.1.	PROTOCOLO DA PESQUISA.....	- 73 -
<b>4.</b>	<b>ESTUDO DE CASO.....</b>	<b>- 74 -</b>
4.1.	BREVE HISTÓRICO DA EMPRESA “BANCO X S.A.” .....	- 74 -
4.2.	GERENCIAMENTO DA QUALIDADE APLICADOS NO DESENVOLVIMENTO DE SOFTWARE.....	- 74 -
4.2.1.	PRÁTICAS ATUAIS DO PROCESSO DE DESENVOLVIMENTO .....	- 74 -
4.2.2.	PRÁTICAS ATUAIS DE CONTROLE DE QUALIDADE .....	- 76 -
4.2.3.	COMPARAÇÃO ENTRE LITERATURA E PRÁTICA .....	- 78 -
<b>5.</b>	<b>CONCLUSÃO.....</b>	<b>- 80 -</b>

# **1. INTRODUÇÃO**

## **1.1. Justificativa e Antecedentes**

A preocupação com a qualidade de software está se tornando cada vez mais significativa em função do grande volume de produção das empresas desenvolvedoras de sistemas informatizados e a exigência dos usuários que almejam antes de tudo softwares confiáveis e eficientes.

O teste de software é uma das atividades que buscam contribuir para a melhoria da qualidade do software tendo como objetivo revelar a presença de defeitos que inviabilizam a qualidade do produto final.

Porém, grande parte das empresas desenvolvedoras de software não aplicam metodologias adequadas para a implantação do processo de teste em função dos altos custos e da grande quantidade de tempo exigida pelas atividades de teste, muitas vezes negligenciando-as ou reduzindo-as, tendo como consequência, frequentes ocorrências de entrega de software para o usuário com a presença de defeitos não revelados.

## **1.2. Problemas dessa Pesquisa**

Essa pesquisa teve início a partir da necessidade de identificar por qual motivo existiam vários problemas de pós-implantação dos sistemas desenvolvidos. Vários programas que entravam em processo de produção apresentavam erros nos primeiros processos executados. Esses erros aconteciam mesmo o sistema tendo cumprido todas as etapas de testes especificadas no programa de testes adotado pela equipe de qualidade de software. O problema é evidenciado na dificuldade que a organização encontra para implementar as metodologias de testes de software existentes dentro de sua estrutura sem impactar os projetos na relação custo e prazo.

## **1.3. Objetivo**

Os objetivos deste trabalho são:

- Analisar as visões de alguns autores que são referências no processo de Gestão da Qualidade, principalmente na fase de testes, no processo de desenvolvimento de software;
- Apresentar um estudo de caso baseado em uma grande instituição financeira brasileira de nome fictício “Banco X”, demonstrando como até empresas com

grandes estruturas tecnológicas e financeiras encontram muita dificuldade de atingir níveis de qualidade fundamentais por comprometerem fases do Processo de Testes em função da redução de custos e comprometimento de prazos de entrega;

#### **1.4. Delimitação da Pesquisa**

Essa pesquisa esta delimitada aos projetos de tecnologia da informação voltados para o desenvolvimento de softwares do setor bancário, especificamente no Processo de Teste dos softwares. Utilizando as melhores práticas de Processos já elaborados e Normas como referência, principalmente os processos que gerenciam a qualidade.

O PMI® apresenta 42 processos para gerenciar projetos, porém apenas o processo de Gerenciamento da Qualidade é abordado nesse trabalho.

#### **1.5. Estruturação do trabalho**

No capítulo dois, estudou-se especificamente a área da Tecnologia responsável pelo desenvolvimento de softwares, sobre a garantia de sua qualidade e a importância da atividade de testes. A relação existente entre testes e garantia de qualidade é que ambos trabalham com o objetivo de minimizar os erros que possam ocorrer no processo de desenvolvimento de software. Apresentam-se também os modelos e normas de qualidade nos padrões mundiais de qualidade existentes, muito importantes para se estabelecer quais os tipos de testes existem e podem ser utilizados no desenvolvimento do software.

No capítulo três é apresentada a metodologia da pesquisa.

No capítulo quatro é apresentado um Estudo de Caso de uma instituição financeira no processo de melhoria contínua dos índices de qualidade dos softwares desenvolvidos aplicando Processos de Testes e métodos de Gestão da Qualidade.

No capítulo cinco, apresenta-se a conclusão final desse trabalho de pesquisa.

## 2. REVISÃO DA LITERATURA

### 2.1. SOFTWARE

Para Pressman (2006), o software de computador é uma tecnologia muito importante e indispensável para negócios, ciência e engenharia. Ele permite a criação de novas tecnologias, a extensão das existentes e a extinção das antigas. Nada disso poderia ser previsto, inclusive que os softwares tivessem que ser corrigidos e adaptados com o passar do tempo, e muito menos que sua manutenção exigiria mais trabalho e tempo do que a criação de novos. Com o aumento da importância dos softwares, a busca por tornar mais rápido e fácil construir e manter programas de computadores de alta qualidade será uma constante. Define software de computador como o produto que os profissionais de software constroem e, depois, mantêm ao longo do tempo. Abrange programas que executam em computadores de qualquer tamanho e arquitetura, conteúdo que é apresentado ao programa a ser executado e documentos tanto em forma impressa quanto virtual que combinam todas as formas de mídia eletrônica.

Para Sommerville (2003) é comum acreditar que os softwares são programas de computadores, porém software não é apenas o programa, mas também toda a documentação associada a ele e todos os dados de configuração para que ele opere corretamente.

Para o mesmo autor, existem dois tipos de produtos de software:

- Os produtos genéricos  
São sistemas desenvolvidos por empresas de softwares e disponibilizados no mercado para qualquer comprador capaz de adquiri-lo.
- Os produtos sobre encomenda  
São os sistemas encomendados por um cliente em particular, geralmente trata-se de sistemas escritos para serem compatíveis com um determinado processo de negócio.

Para o mesmo autor, a principal diferença entre esses tipos de software é o controle da especificação dos mesmos, feito pela empresa desenvolvedora e pelos clientes, respectivamente. A importância dos softwares, desde a criação dos computadores, cresceu

perceptivelmente. Durante as primeiras décadas dos computadores, o desafio era desenvolver um hardware que reduzisse o custo de processamento e armazenagem de dados. Hoje, o principal desafio é melhorar a qualidade de soluções baseadas em computador, que são os softwares. Os softwares devem ter “inteligência” embutida para atender ao usuário e a empresa, com foco no negócio empresarial. O mesmo pode ser aplicado em qualquer situação em que exista um conjunto de passos predefinidos, um algoritmo.

## **2.2. Problemas ocorridos em desenvolvimento de softwares**

Para Pressman (2006), são vários os tipos de problemas que afetam o desenvolvimento de Software, dentre eles, os mais superficiais são:

- Não cumprimento do prazo e do custo;
- Produtividade das pessoas na área de software que não acompanha a demanda de serviços;
- Alto índice de falhas;

Para o mesmo autor, todos esses problemas geram custos excessivos, os prazos se arrastam por meses e nada tem sido feito para melhorar a produtividade dos profissionais da área. Todas estas dificuldades são resultado de outros problemas:

- Falta de dedicação durante o Levantamento de Dados;
- Falta de comunicação entre cliente e desenvolvedor;
- Recente valorização dos testes e da qualidade dos softwares;
- Despreocupação com o custo da manutenção do software;

Segundo o mesmo autor, cada um dos problemas descritos pode ser corrigido com uma abordagem de engenharia ao desenvolvimento de software, aliada com a contínua melhora de técnicas e ferramentas.

### **2.2.1. Causas**

Para Pressman (2006), os problemas dos softwares foram causados por sua complexidade e pelas falhas das pessoas responsáveis por desenvolvê-lo. Isto se deve ao curto espaço de tempo em que estamos nos dedicando a isso.

Para o mesmo autor, o desafio de desenvolver softwares é grande, porém os principais problemas ocorrem por falhas humanas corriqueiras. Os profissionais na área de softwares não recebem o treinamento necessário, o que compromete a qualidade do



mesmo. É essencial que o projeto de software seja desenvolvido com base em métodos de controle e que o gerente de software se comunique com todos os envolvidos no projeto, para prevenir que o projeto seja mal compreendido e ocorram os problemas de softwares.

### **2.2.2. Custos e Prazos**

Segundo Paula Filho (2003), grande parte dos sistemas informatizados é entregue com atraso e custam mais do que o previsto. Estourar cronogramas e orçamentos é parte da rotina da maioria dos profissionais de desenvolvimento de software. Porém esses prazos e custos são reais?

De acordo com o autor, clientes e gerentes se desesperam com os atrasos dos projetos de software, entretanto, no próximo contrato, eles provavelmente escolherão o fornecedor que prometer menor prazo e/ou menor custo. Eles buscam desenvolvedores que prometam prazos politicamente agradáveis, embora irreais.

Para o mesmo autor, estimar prazos e custos serve para um produto ser viável. Se isto não for possível, o produto pode não ser viável do ponto de vista de mercado, ou pode ser preferível adquirir outro produto, ainda que sacrificando alguns dos requisitos. Ter estimativas de prazos e custos, portanto, é uma expectativa mais que razoável de clientes e gerentes. O problema é que existem alguns desenvolvedores que não conhecem métodos técnicos de estimativa de prazos e custos do desenvolvimento de software ou que trabalham em organizações onde não se podem apresentar avaliações reais das perspectivas dos projetos. Fica fácil observar a relação direta desses fatores, quando o profissional compromete prazo e/ou custo, o resultado afetará diretamente na queda da qualidade do produto final.

### **2.2.3. Desenvolvimento de Software**

Os autores Peters e Pedrycz (2001) classificam o desenvolvimento de software como processo de pré-desenvolvimento e processo de desenvolvimento. Durante o pré-desenvolvimento existem dois processos principais: a exploração de conceitos e a alocação de sistemas. A alocação de sistemas liga a exploração de conceitos e o processo de desenvolvimento, fornecendo uma descrição das necessidades do sistema e identificando a arquitetura e os requisitos de hardware e software. Durante a fase de desenvolvimento existem três processos principais a serem seguidos:

- **Identificar os requisitos**

Decidindo o que o sistema deve fazer as prioridades, os riscos e os planos de teste;

- **Fazer o projeto do sistema**

Determinando como são a estrutura e as funções do mesmo, concretizando os requisitos de softwares;

- **Implementar**

Produzindo o código-fonte, a documentação e fazendo os testes para validação e verificação do atendimento dos requisitos;

- **Ciclos de Vidas**

De acordo com Sommerville (2003), o modelo de processo de software ou ciclo de vida, é uma representação abstrata de um processo de software. Os ciclos de vida representam um resumo do processo de desenvolvimento, proporcionando informações sobre as principais atividades do projeto. Um sistema grande não pode ser representado apenas por um modelo de processo, processos diferentes são utilizados para representar diferentes partes do sistema. Os principais modelos de software são: Cascata, Incremental e Espiral.

- **Modelos de Desenvolvimento de Software**

- **Modelo Cascata**

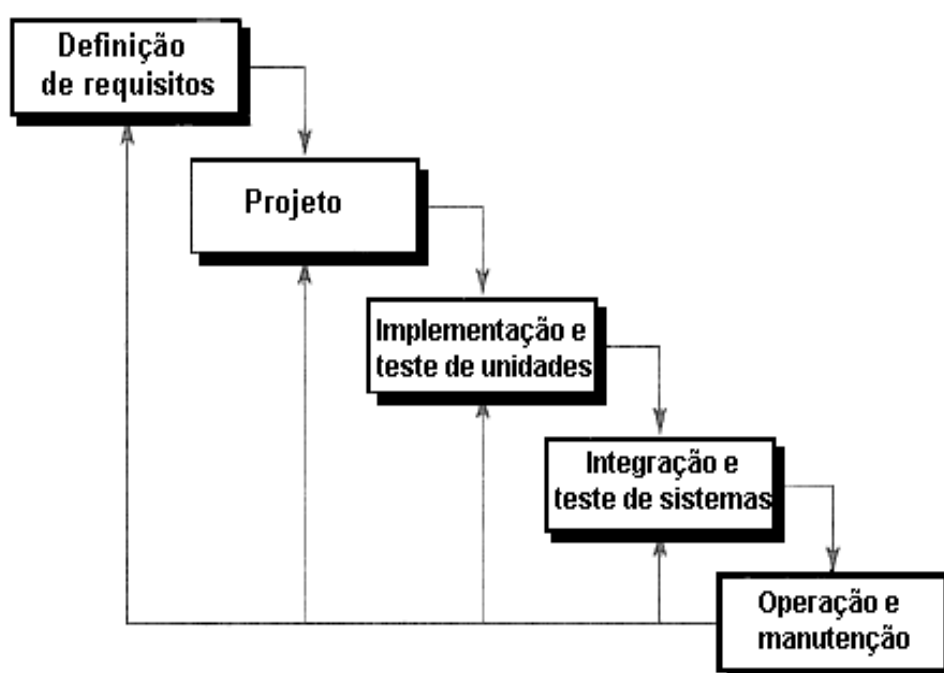
Para Sommerville (2003), o modelo cascata requer uma abordagem sistemática e sequencial que se inicia na análise e definição de requisitos e avança ao longo do projeto, implementação e teste de unidade, integração e teste de sistemas, operação e manutenção, conforme a Figura 1.

Para o mesmo autor, durante a análise e definição de requisitos, por meio de consultas aos usuários, são estabelecidas as funções, restrições e objetivos do sistema. Logo após a conclusão desta fase, é realizado o processo de projeto, onde é estabelecida uma arquitetura geral, com as abstrações fundamentais do sistema e suas relações. O próximo estágio é a implementação do sistema e os testes para verificação de cada unidade. Então, todos os programas implementados são integrados e testados como um sistema completo. Na fase final e geralmente mais longa, o sistema é colocado em uso e, à

medida que são descobertas falhas ou novos requisitos, é realizada a manutenção, onde pode haver a repetição de alguns ou de todos os estágios do processo. Normalmente, a cada fase são descobertos problemas com a fase anterior.

O mesmo autor salienta que os problemas apresentados pelo Modelo Cascata, são a sua falta de flexibilidade na divisão do projeto e a dificuldade encontrada em relação à frequente modificação dos requisitos do cliente. Por outro lado, apresenta a vantagem de ser um modelo simples de gerenciamento.

Figura 1 - Modelo Cascata

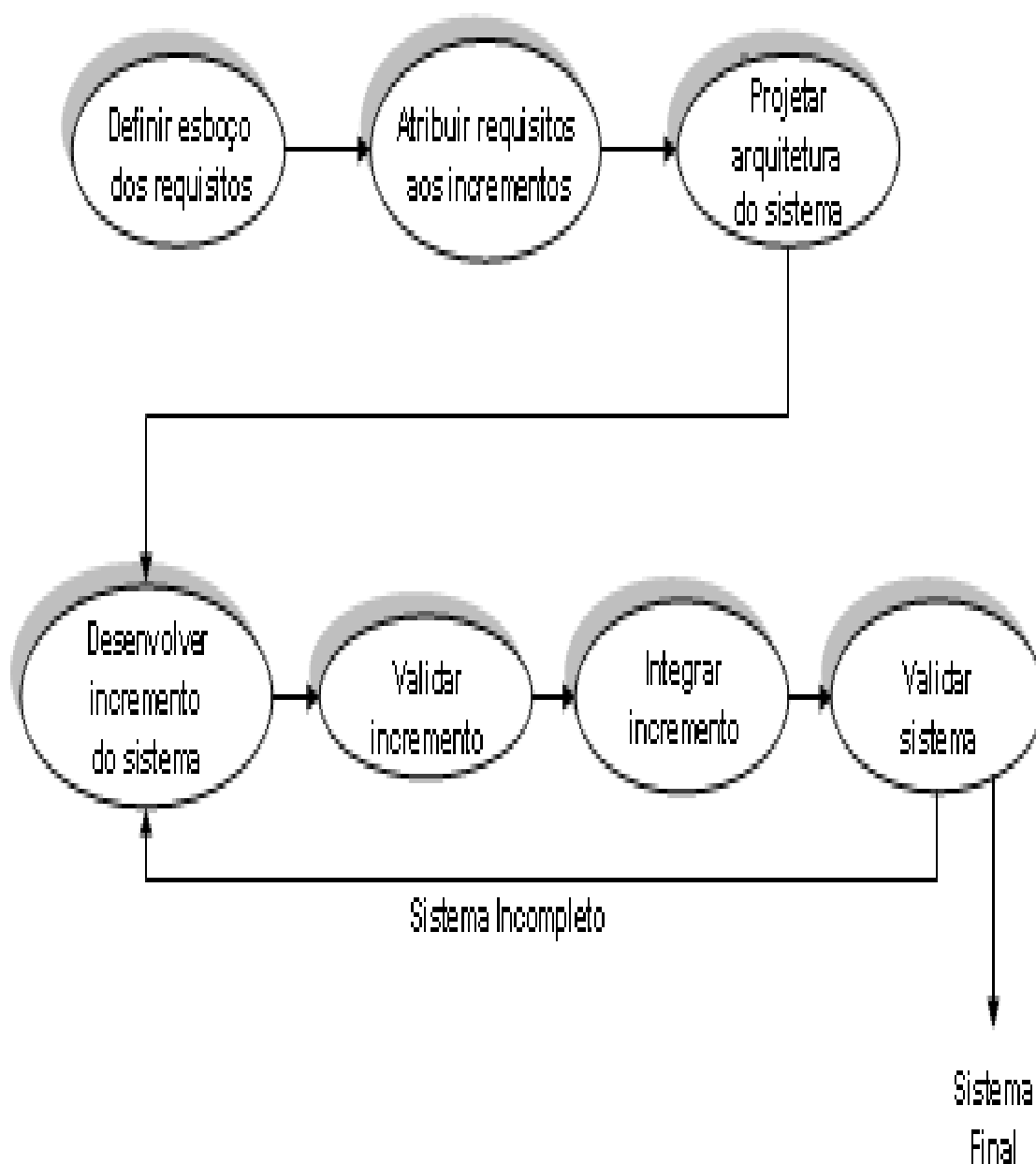


Fonte: SOMMERVILLE (2003)

#### ▪ **Modelo Incremental**

Para Sommerville (2003), o Modelo Incremental é um intermédio entre o Cascata e o Espiral. O Modelo Cascata requer que os requisitos do sistema sejam definidos antes do início do projeto, já no modelo espiral há uma abordagem evolucionária do sistema. O Ciclo de Vida Incremental permite aos clientes oportunidades de adiar a definição de alguns requisitos até que se tenha uma experiência maior com o sistema, reduzindo o retrabalho. Quanto maior o grau de importância do requisito, mais cedo ele terá que ser definido e será entregue ao cliente, conforme mostra a Figura 2.

Figura 2 – Modelo Incremental



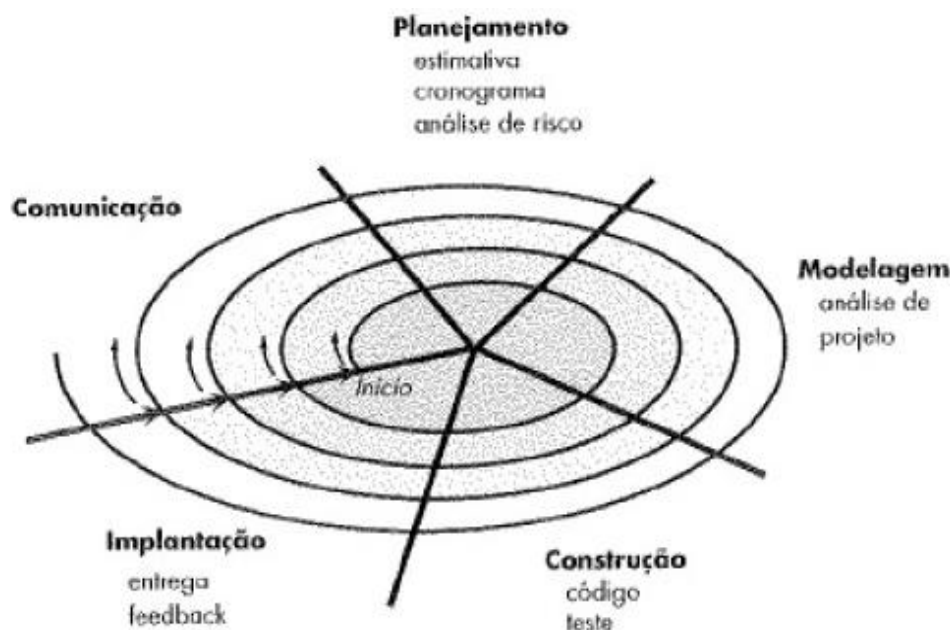
Fonte: SOMMERVILLE (2003)

Para o mesmo autor, após a definição dos primeiros requisitos, esses são detalhados. Logo após inicia-se o desenvolvimento deste incremento e não é mais permitida a mudança de requisitos para o incremento atual. Este incremento é concluído e esta funcionalidade é entregue ao cliente, que pode experimentá-lo antes de definirem os próximos requisitos. À medida que cada incremento novo é desenvolvido, este é integrado ao sistema. O problema deste tipo de modelagem é que ela parte do princípio de que os requisitos são estáveis, o que é irreal devido às mudanças na tecnologia e experiência.

### ▪ Modelo Espiral

Para Sommerville (2003), o modelo Espiral ao invés de representar o processo como uma sequencia de atividades, utiliza uma espiral onde cada loop representa uma fase do sistema e cada fase é dividida em quatro setores, conforme mostra a Figura 3.

Figura 3 – Modelo Espiral



Fonte: SOMMERVILLE (2003)

Para o mesmo autor, cada setor do modelo representado na Figura 3 representa uma importante atividade. O primeiro é o planejamento, onde são determinados os objetivos e as restrições, o segundo é um elemento presente primeiramente neste tipo de modelo, a análise de riscos, o seguinte é a engenharia, o desenvolvimento do produto e o último correspondem à avaliação dos resultados feita pelo cliente. O processo evolucionário inicia-se no centro, em sentido horário, realizando as atividades indicadas. Cada passagem no setor de planejamento resulta em ajustes no projeto e à medida que a evolução cresce, aumentam os riscos. O gerente de projetos é quem define o número de iterações necessárias para completar o software. Usando o modelo espiral, o software evolui a cada loop do modelo, iniciando como um protótipo e terminando como um completo sistema submetido à engenharia. O autor considera o modelo espiral realista, sendo mais fácil para os desenvolvedores entenderem os riscos, dividindo-os em cada nível evolucionário.

#### **2.2.4. Prototipação**

De acordo com Pressman (2006), o objetivo principal da prototipação é o desenvolvimento rápido de software, onde o cliente pode previamente ver um protótipo do sistema produzido com desempenho limitado antes de se comprometerem com o sistema final. Desse modo, o cliente pode pedir alterações antes do início do projeto. Os eventos para a prototipação são semelhantes a qualquer abordagem de desenvolvimento de software, iniciando-se com a coleta dos requisitos, a elaboração de um projeto rápido, levando a construção de um protótipo que será avaliado pelo cliente para ajudar nos requisitos do software que será desenvolvido. Um problema citado pelo autor em relação aos protótipos: os clientes quererem, pela pressa de colocar o sistema em operação, apenas fazer alterações no próprio protótipo e não iniciar o desenvolvimento do sistema. O autor afirma ainda que a solução para isso seja simples, apenas definir as “regras do jogo”.

Para o mesmo autor, ainda que por se concentrarem nos requisitos principais, os protótipos podem ignorar requisitos de qualidade, confiabilidade, manutenibilidade e segurança, não garantindo que o sistema satisfaça o cliente.

#### **2.2.5. As Etapas de Desenvolvimento**

De acordo com Sommerville (2003), um processo de desenvolvimento de software é um conjunto de atividades ou etapas a serem cumpridas para obter um produto de software, é um dos principais mecanismos para se desenvolver software de qualidade.

Para o mesmo autor, algumas etapas de desenvolvimento de software são definidas como:

- **Análise Econômica**

Trata-se de uma análise do projeto para estabelecer se o mesmo gerará lucro e se a receita gerada será o suficiente para cobrir os custos.

- **Análise de requisitos de software**

A primeira etapa a ser realizada para a criação de um desejado produto é a extração dos requisitos. Esta tarefa requer habilidade, pois ela é a base para o projeto, é necessário reconhecer a incompletude, ambiguidade ou contradição dos requisitos para produzir um software de qualidade.

- **Especificação**

A especificação é a tarefa de descrever precisamente o software que será escrito, estabelecendo quais funções são requeridas pelo sistema e as restrições sobre a operação e o desenvolvimento do mesmo.

- **Arquitetura de Software**

A arquitetura de um sistema de software corresponde a uma representação abstrata daquele sistema. A etapa da arquitetura também direciona as interfaces entre os sistemas de software e outros produtos de software, como também com o hardware básico ou com o sistema operacional.

- **Implementação (ou codificação)**

Trata-se da transformação de um projeto para um código, ou seja, a elaboração e preparação dos módulos necessários à sua execução.

- **Teste**

Teste de partes 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 e utilizar o produto para encontrar seus defeitos.

- **Documentação**

Uma importante tarefa é a documentação do projeto interno do software para propósitos de futuras manutenções e aprimoramentos.

- **Suporte e Treinamento de Software**

Trata-se do suporte e treinamento ao usuário para a utilização do sistema. Usuários irão ter muitas questões e problemas de software os quais conduzirão para a próxima fase.

- **Manutenção**

A manutenção e melhoria de software lidam com a descoberta de novos problemas e requisitos. Ela pode tomar mais tempo que o gasto no desenvolvimento inicial do mesmo. A maioria das manutenções é para

ampliar os sistemas para novas funcionalidades, as quais, de diversas formas, podem ser consideradas um novo trabalho.

### **2.3. Garantia da Qualidade de Software**

Segundo Pressman (2006), é necessário conhecer o que significa qualidade de software para se implementar uma estratégia de testes de software, pois ambos estão ligados: ao se definir a estratégia de testes, se define também os objetivos de qualidade dos produtos e os serviços oferecidos. A garantia da qualidade de software (*Software Quality Assurance*, SQA) é uma atividade de proteção que é aplicada ao longo do processo de software. Não é suficiente dizer que a qualidade de software é importante, segundo o mesmo autor, deve-se:

- Definir explicitamente o que quer dizer “qualidade de software”;
- Criar um conjunto de atividades que ajudarão a garantir que todo o produto de trabalho de engenharia de software exibe alta qualidade;
- Realizar atividades de garantia de qualidade em todo o projeto de software;
- Usar métricas para desenvolver estratégias para aperfeiçoar seu processo de software e, como consequência, a qualidade do produto final.

Para o mesmo autor, todos os envolvidos no processo de engenharia de software são responsáveis pela qualidade, portanto devem ser implantados procedimentos de melhoria de qualidade e acompanhados pelos responsáveis nas organizações. Se uma equipe de software enfatiza a qualidade em todas as atividades de engenharia de software, aumenta o planejamento do processo, análise, definições e testes, reduz a quantidade de trabalho que tem que refazer. Isso resulta em menores custos e, mais importante, menor prazo para colocação no mercado.

O autor ainda define a qualidade de software como a satisfação de requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo software desenvolvido profissionalmente.

Entretanto para Paula Filho (2003), a qualidade de um produto corresponde ao grau de conformidade com seus requisitos. O que decide a qualidade é o confronto entre a



promessa e a realização de cada produto. Todos os desenvolvedores de produtos ou serviços sabem que qualidade é algo importante para o cliente. Os desenvolvedores de software também concordam que software de alta qualidade é uma meta importante. A qualidade de software é medida pela boa execução de todas as etapas do processo de desenvolvimento. Muitas vezes o processo de desenvolvimento utilizado não é exatamente o descrito, por falta de qualificação das pessoas ou porque devido aos prazos algumas etapas diretamente relacionadas com a qualidade do mesmo são puladas. Por exemplo, quando não se prioriza a etapa de definição dos requisitos, a falta de conformidade com os mesmos é diretamente relacionada com falta de qualidade.

## 2.4. Os Fatores de Qualidade

Segundo Paula Filho (2003), os fatores de qualidade do software dividem-se em:

- **Fatores que podem ser medidos diretamente**  
Defeitos funcionais
- **Fatores que podem ser medidos indiretamente**  
Manutibilidade e Usabilidade.

Segundo o mesmo autor, em cada caso deve ocorrer medição, é preciso comparar o software com algum dado e chegar a uma indicação de qualidade. Ele apresenta um modelo como métricas para qualidade de software, este modelo é conhecido como Fatores da Qualidade, estes fatores avaliam o software em três aspectos: suas características operacionais, sua habilidade de passar por modificações e sua adaptabilidade a novos ambientes, como mostra a Figura 4.

Figura 4 – Os Fatores da Qualidade



Fonte: PAULA FILHO (2003)

De acordo com Paula Filho (2003), as descrições dos fatores são:

- **Correção**  
À medida que o software cumpre seus objetivos.
- **Confiabilidade**  
À medida que o software execute suas funções com precisão.
- **Usabilidade**  
O esforço exigido para apreender e operar o sistema.
- **Integridade**  
À medida que o acesso ao software por pessoa seja seguro.
- **Eficiência**  
A quantidade de recursos de computação exigida para que um software execute suas funções.
- **Manutenibilidade**  
O trabalho exigido para reparar os erros.
- **Flexibilidade**  
O trabalho exigido para modificar um software.
- **Testabilidade**  
O trabalho exigido para testar um software a fim de garantir que ele execute sua função pretendida.
- **Portabilidade**  
O trabalho exigido para transferir o programa de um ambiente para outro.
- **Reusabilidade**  
À medida que o software ou parte dele pode ser reusado em outras aplicações.
- **Interoperabilidade**  
O trabalho exigido para acoplar um sistema ao outro.

Para o mesmo autor, cada fator apresentado deve ser utilizado como parâmetro para medir a qualidade dos softwares, o peso dado a cada um varia conforme o produto e as preocupações locais.

#### **2.4.1. Garantia da Qualidade de Software**

Para Paula Filho (2003), a qualidade é consequência dos processos, das pessoas e da tecnologia. A relação entre a qualidade do produto e cada um desses fatores é

complexa. Em todas as etapas do desenvolvimento do software as pessoas introduzem defeitos. Esses defeitos são decorrentes de limitações humanas como:

- Erros lógicos;
- Erros de interpretação;
- Desconhecimento de técnicas;
- Falta de motivação.

Segundo o mesmo autor, durante o desenvolvimento de software existem etapas realizadas para garantir a qualidade, tais como revisões, testes e auditorias. Nestas etapas, grande parte dos defeitos são removidos. Estas etapas são indispensáveis para garantir a qualidade do produto. Os defeitos que não são removidos precocemente são detectados depois e a sua correção torna-se cada vez mais cara. O pior caso acontece quando o defeito chega ao produto final. Neste caso, ele só será removido através de uma operação de manutenção. Esta é a forma mais cara de remoção de defeitos. O tempo de desenvolvimento é geralmente reduzido com o aumento da qualidade do processo, pois um processo de qualidade é mais eficiente na detecção e eliminação precoce dos defeitos. Em geral, o tempo gasto com a correção precoce é mais do que compensado pela eliminação do tempo que seria gasto com a correção tardia. Vários métodos de garantia da qualidade levam em conta uma limitação humana, somos mais eficazes para achar os defeitos dos outros do que nossos próprios defeitos.

De acordo com o autor, a busca constante pela qualidade não se faz apenas no começo do projeto ou no seu final realizando testes, mas sim em um processo que visa abranger toda a engenharia de software bem como a colaboração de todos os membros do time do projeto. Para tanto, todas as etapas do ciclo de vida de engenharia de software devem ser contempladas com atividades que visam garantir a qualidade tanto do processo quanto do produto.

#### **2.4.2. Processos de Gerenciamento da Qualidade do Software**

De acordo com o PMI® (2008), os processos de gerenciamento da qualidade do projeto correspondem a todas as atividades que determinam as responsabilidades, os objetivos e as políticas de qualidade, para que o projeto atenda às necessidades que motivaram sua realização com qualidade. Os processos de gerenciamento da qualidade são:

✓ **Planejamento da qualidade**

O processo de identificar os requisitos e/ou padrões de qualidade do projeto e do produto, bem como documentar de que modo o projeto demonstrará a conformidade.

✓ **Realizar a garantia da qualidade**

O processo de auditoria dos requisitos de qualidade e dos resultados das medições do controle de qualidade para garantir que sejam usados os padrões de qualidade e as definições operacionais apropriadas.

✓ **Realizar o controle da qualidade**

O processo de monitoramento e registro dos resultados da execução das atividades de qualidade para avaliar o desempenho e recomendar as mudanças necessárias.

De acordo com o PMI® (2008), esses processos interagem entre si e com os processos das outras áreas de conhecimento. Cada processo pode envolver o esforço de uma ou mais pessoas ou grupos de acordo com os requisitos do projeto. Cada processo ocorre pelo menos uma vez em todo o projeto e em uma ou mais fases do mesmo, se o projeto for dividido em fases. Embora os processos sejam apresentados como elementos distintos com interfaces bem definidas, na prática eles podem se sobrepor e interagir de formas que não foram detalhadas nesse trabalho. O Gerenciamento da Qualidade dos Projetos engloba o gerenciamento do projeto e do produto do projeto, e se aplica a todos os projetos, independentemente da natureza do produto. As medidas e técnicas de qualidade do produto são específicas do tipo de produto resultante do projeto. Enquanto o gerenciamento da qualidade de produtos de software utiliza abordagens e medidas diferentes de uma construção de uma usina nuclear, as abordagens do gerenciamento da qualidade do projeto se aplicam aos dois tipos exemplificados. Nos dois casos, deixar de cumprir os requisitos de qualidade do produto ou do projeto pode ter consequências negativas graves para uma ou todas as partes interessadas do projeto.

De acordo com o PMI® (2008), no Gerenciamento Moderno da Qualidade as seguintes disciplinas tem sua importância reconhecida:

✓ Satisfação do Cliente

Entender, avaliar, definir e gerenciar as expectativas para os requisitos do cliente sejam atendidos. Para isso, é necessária uma combinação de conformidade com os requisitos(para garantir que o projeto produza o que ele foi criado para conduzir) e adequação ao uso (o produto ou serviço devem satisfazer às necessidades reais).

✓ Prevenção ao invés da inspeção

Um dos princípios fundamentais do moderno gerenciamento da qualidade determina que a qualidade deve ser planejada, projetada e incorporada em vez de inspecionada. O custo de prevenir os erros geralmente é muito menor do que o custo de corrigi-los quando são encontrados pela inspeção.

✓ Melhoria Contínua

O ciclo PDCA é a base para a melhoria da qualidade conforme definida por Shewhart e modificada por Deming. Além disso, as iniciativas de melhoria da qualidade empreendidas pela organização executora, tais como GQT e Seis Sigma devem aprimorar a qualidade do gerenciamento do projeto e também a qualidade do produto do projeto.

✓ Responsabilidade da Gerência

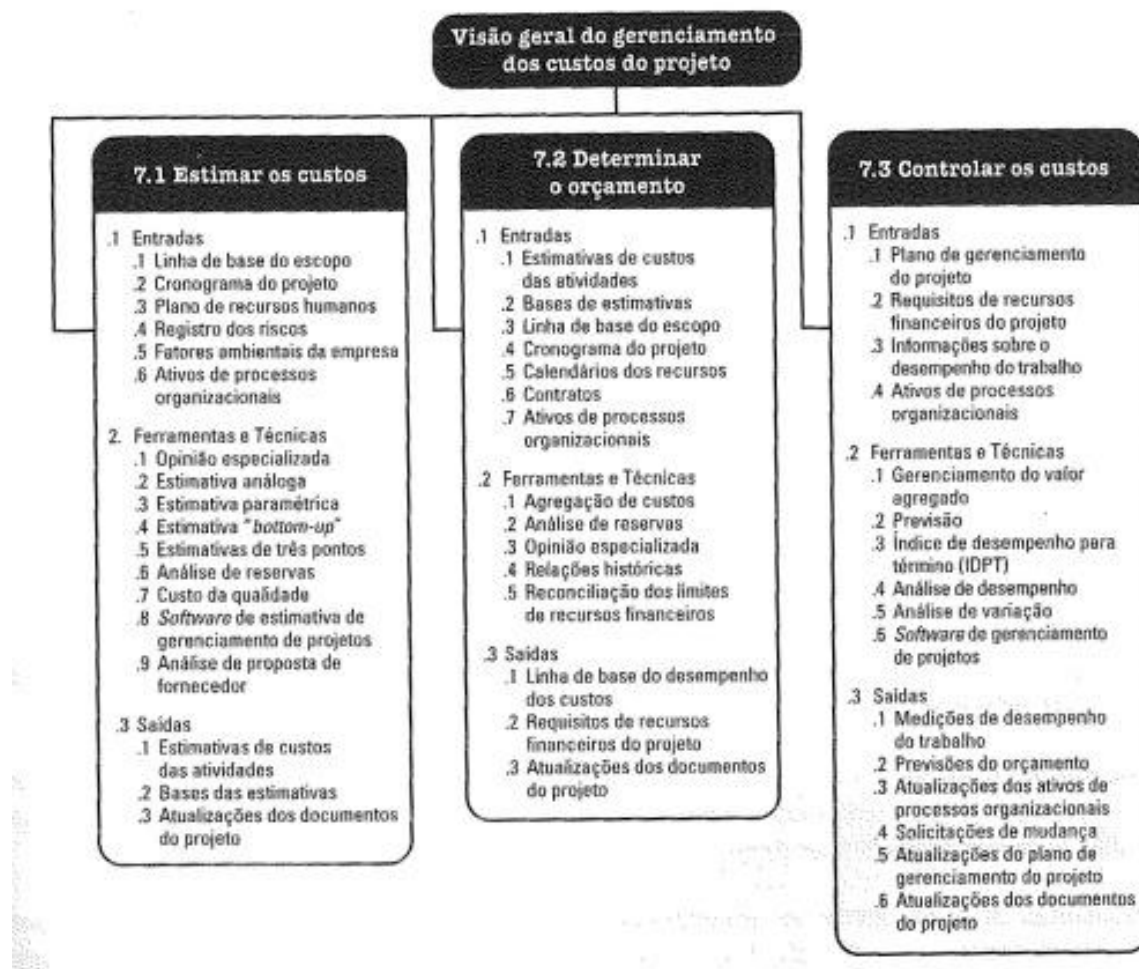
O sucesso exige a participação de todos os membros da equipe do projeto, mas continua sendo a responsabilidade da gerência fornecer os recursos necessários ao êxito.

De acordo com o PMI® (2008), o planejamento da qualidade envolve a identificação dos padrões de qualidade relevantes para o projeto e a determinação de como satisfazê-los. O planejamento deve avaliar fatores como:

- ✓ Fatores ambientais;
- ✓ Políticas e procedimentos da empresa do cliente.

A Figura 5 demonstra a visão geral do gerenciamento da qualidade.

Figura 5 - Visão Geral do Gerenciamento da Qualidade do Projeto



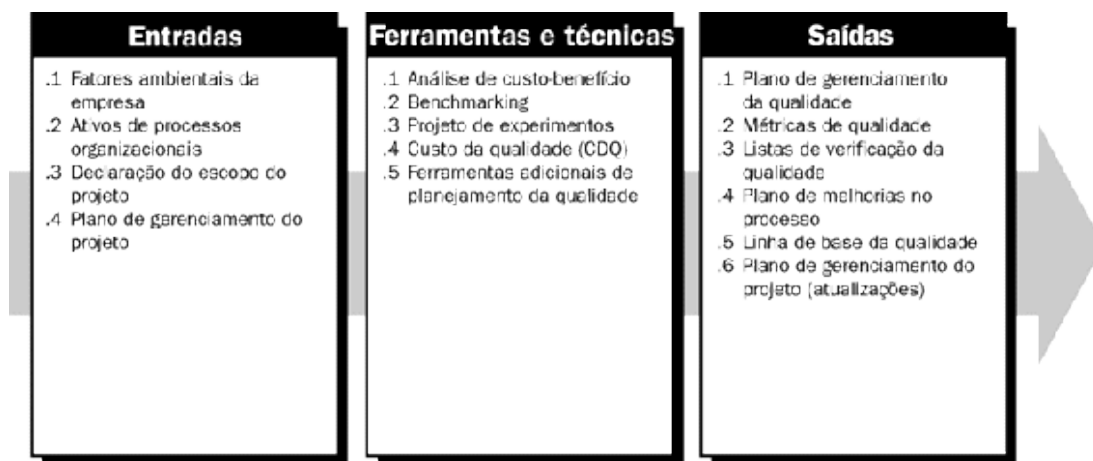
Fonte - PMI® (2008)

De acordo com o PMI® (2008), o planejamento da qualidade deve ser realizado a partir do escopo do projeto de software, com cronogramas, objetivos e limites, como custo. O mesmo deve considerar custo-benefício e variáveis que podem influenciar o projeto. Para realizar a garantia da qualidade, as atividades devem ser planejadas e sistemáticas para garantir que o projeto emprega todos os processos necessários para atender aos requisitos. O controle da qualidade deve ser realizado através do monitoramento de resultados do projeto a fim de determinar se eles estão de acordo com os padrões relevantes de qualidade e identificação de maneiras de eliminar as causas de um desempenho insatisfatório.

### 2.4.3. Planejar a Qualidade

De acordo com o PMI® (2008), planejar a Qualidade é o processo de identificação dos requisitos e/ou padrões de qualidade do projeto e do produto, além da documentação de como o projeto demonstrará a conformidade. O Planejamento da Qualidade deve ser realizado em paralelo com ou outros processos de planejamento do projeto. Por exemplo, modificações propostas no produto para atender aos padrões de qualidade identificados podem exigir custos ou ajustes nos cronogramas e uma análise de riscos detalhada dos seus impactos nos planos. As técnicas de planejamento da qualidade descritas nesse trabalho são usadas com maior frequência nos projetos. Existem muitas outras que podem ser úteis em determinados projetos ou em algumas áreas de aplicação, como podemos ver na Figura 6.

Figura 6 - Planejar a Qualidade



Fonte: PMI® (2008)

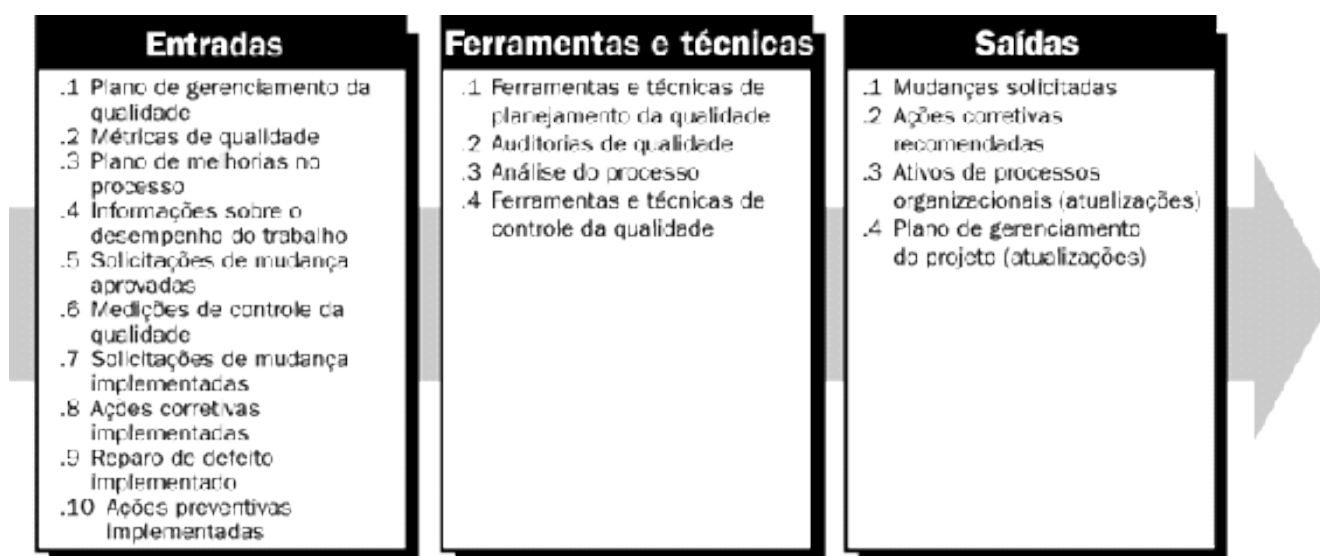
### 2.4.4. Realizar a garantia da Qualidade

De acordo com o PMI® (2008), realizar a garantia da qualidade é o processo de auditoria dos requisitos de qualidade e dos resultados das medições do controle da qualidade para garantir que sejam usados os padrões de qualidade e definições operacionais apropriados. Realizar a garantia da qualidade é um processo de execução que usa dados criados durante o processo Realizar o controle da qualidade. O departamento de garantia da qualidade, ou organização similar, em geral supervisiona as atividades de garantia da qualidade. O suporte da garantia da qualidade, independentemente do título da unidade, pode ser fornecido à equipe do projeto, à

gerência da organização executora, ao cliente ou ao patrocinador, bem como a outras partes interessadas que não estejam envolvidas ativamente no trabalho do projeto.

De acordo com o PMI®, o processo Realizar a garantia da qualidade também inclui a melhoria contínua do processo, que é um meio iterativo de melhorar a qualidade de todos os processos. A melhoria contínua de processos reduz o desperdício e elimina as atividades que não agregam valor, permitindo que os processos sejam operados com níveis mais altos de eficiência e eficácia.

Figura 7 - Realizar Garantia da Qualidade



Fonte: PMI® (2008)

#### 2.4.5. Realizar o controle da Qualidade

De acordo com o PMI® (2008), realizar o controle da qualidade é o processo de monitoramento e registro dos resultados da execução das atividades de qualidade para avaliar o desempenho e recomendar as mudanças necessárias. O controle da qualidade é realizado durante todo o projeto. Os padrões de qualidade incluem os processos do projeto e as metas do produto. Os resultados do projeto incluem em entregas e os resultados do gerenciamento do projeto, tais como desempenho de custos e prazos. O controle da qualidade em geral é realizado por um departamento de controle da qualidade ou uma unidade da organização com nome semelhante. As atividades de controle da qualidade identificam as causas da baixa qualidade do processo ou produto e recomendam e/ou executam as ações para eliminá-las. A equipe de gerenciamento do projeto deve ter um conhecimento prático de controle estatístico da qualidade, principalmente de amostragem e

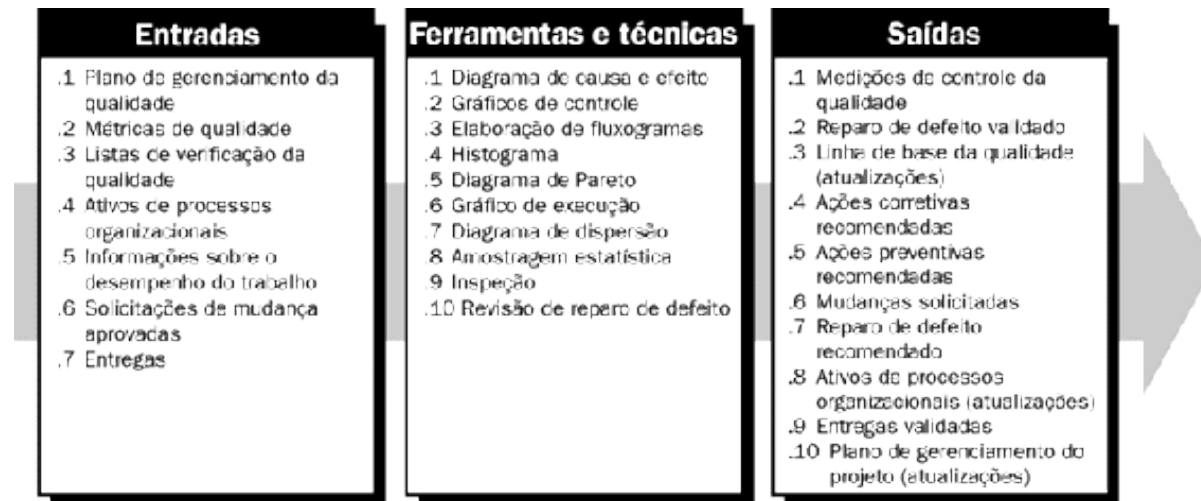


probabilidade, para ajudar a avaliar as saídas do controle da qualidade. Entre outros assuntos, é recomendar que a equipe conheça as diferenças entre os seguintes pares de termos:

- ✓ Prevenção - Manter os erros fora do processo  
Inspeção - Manter os erros fora do alcance do cliente
- ✓ Amostragem de atributos – O resultado está em conformidade ou não está em conformidade  
Amostragem de variáveis – O resultado é classificado em uma escala contínua que mede o grau da conformidade
- ✓ Tolerâncias – Intervalo especificado de resultados aceitáveis  
Limite de controle – Limites que podem indicar se o processo está fora de controle

Realizar o controle da qualidade pode ser visualizado na Figura 8.

Figura 8 - Realizar Controle da Qualidade



Fonte: PMI® (2008)

#### 2.4.6. Custo da Qualidade de Software

Segundo Pressman (2006), para garantir a qualidade de um produto, investimentos financeiros, treinamentos e softwares são necessários. O custo para garantir a qualidade de um software varia conforme a etapa em que são descobertos os defeitos. Para tanto, o principal objetivo das técnicas para descoberta precoce dos defeitos de software é que o

defeito possa ser corrigido antes do passo seguinte do processo de engenharia de software. Ao corrigir precocemente os defeitos, os custos dos passos subsequentes nas fases de desenvolvimento e manutenção reduzem substancialmente.

De acordo com o autor, o custo para garantir a qualidade de software varia se o defeito já existe, se é realizada a detecção precoce ou se o projeto não se preocupa com a detecção de defeitos precocemente. As descobertas de falhas em relação à variação de custos são:

- **Custo da Detecção de Defeitos**

O foco está exatamente no produto, as atividades realizadas são orientadas ao produto desenvolvido incluindo:

- Revisões de Requisitos;
- Revisões de Modelagem;
- Revisões de Planos de Teste;
- Inspeções de Código;
- Testes de Software.

- **Custo da Prevenção de Defeitos**

O foco está exatamente no processo, as atividades realizadas são orientadas ao processo incluindo:

- Definição de Metodologias;
- Treinamentos;
- Ferramentas de apoio ao processo de desenvolvimento;
- Definição de Políticas;
- Procedimentos;
- Padrões;
- Especificações e convenções;
- Planejamento do SQA;
- Relatórios de Qualidade para melhoria de processo.

- **Custo pela opção de não detectar precocemente os defeitos**

Está relacionado às perdas que o projeto terá, não optando pela detecção e prevenção de defeitos:

- Re-revisões;

- Re-testes;
- Correções de código-fonte e documentação muito constantes;
- Reestruturação;
- Redistribuição das versões do software;
- Atrasos no cronograma;
- Falhas na produção.

#### **2.4.7. A Qualidade segundo a ISO**

A ISO apresenta as características da qualidade de software através da norma NBR ISO/IEC 9126-1(2001). A antiga norma brasileira para as características da qualidade de software era a NBR 13.596. No entanto, a mesma se integrou a ISO, formando a NBR ISO/IEC 9126-1(2001). A norma ISO 9126 foi desenvolvida para identificar os atributos que significam qualidade para software de computador. Alguns desses atributos são:

- **Funcionalidade**

Quando o software satisfaz as necessidades declaradas, atende os requisitos.

- **Confiabilidade**

Quando no período de tempo em que o software está disponível, este apresenta tolerância a falhas e recuperabilidade.

- **Usabilidade**

Quando o software é simples para o usuário utilizar.

- **Eficiência**

Quando o software faz uso otimizado dos recursos do sistema.

- **Manutenibilidade**

Facilidade com a qual podem ser feitos reparos no software.

- **Portabilidade**

Facilidade com a qual o software pode ser transposto de um ambiente para o outro.

#### **2.4.8. Plano SQA (Plano de Garantia de Qualidade de Software)**

Para Pressman (2006), um Plano de Garantia de Qualidade de Software é criado para definir a estratégia de SQA de uma equipe de software. Durante a análise, projeto e geração de código, o principal produto do trabalho de SQA é o relatório resumido da

revisão técnica formal. Durante o teste, planos e procedimentos de testes são produzidos. Outros produtos de trabalho associados com o aperfeiçoamento do processo podem também ser gerados. O ideal seria encontrar erros antes que eles se transformem em defeitos, isto é, trabalhar para aperfeiçoar a eficiência na remoção de defeitos, reduzindo assim a quantidade de trabalho a ser feito que a equipe de software tem que realizar.

Para o mesmo autor, o processo de SQA abrange:

- Uma abordagem de gestão de qualidade;
- Tecnologia efetiva de engenharia de software (métodos e ferramentas);
- Revisões técnicas formais, que são aplicadas ao longo do processo de software;
- Uma estratégia de testes multicamada;
- Controle da documentação de software e das modificações nela feita;
- Um procedimento para garantir a satisfação de padrões de desenvolvimento de software (quando aplicável);
- Mecanismos de medida e relatório;

Para o mesmo autor, o plano de SQA fornece um roteiro para a instituição de garantia de qualidade de software. Desenvolvido pelo grupo de SQA, o plano serve como gabarito para as atividades SQA que são instituídas para cada projeto de software. A garantia de qualidade de software é composta de uma variedade de tarefas associadas a duas partes diferentes – os engenheiros de software que fazem o trabalho técnico, e um grupo de SQA, que tem responsabilidade pelo planejamento, supervisão, registro, análise e relato da garantia de qualidade. O grupo SQA atua como o representante do cliente verificando o software do ponto de vista do cliente. A missão do grupo de SQA é ajudar a equipe de software a conseguir um produto final de alta qualidade.

Segundo Pressman (2006), as atividades do grupo de SQA são:

- Preparar um plano SQA para um projeto: o plano é desenvolvido durante o planejamento do projeto e é revisado por todas as partes interessadas. As atividades de garantia de qualidade realizadas pela equipe de engenharia de software e pelo grupo de SQA são regidas pelo plano.
- Participar no desenvolvimento da descrição do processo de software do projeto.
- Rever as atividades de engenharia de software para verificar a satisfação do processo de software definido.

- Auditar os produtos do trabalho de software encomendado para verificar a satisfação do que foi definido como parte do processo de software.
- Garantir que os desvios do trabalho de software e dos produtos do trabalho são documentados e manipulados de acordo com um procedimento documentado.
- Registrar qualquer eventual não satisfação e relatar a gerência superior.

Para o mesmo autor, defeitos e falhas do sistema são problemas de qualidade que são encontrados depois que o software é entregue aos usuários finais. O objetivo principal das revisões técnicas formais é achar erros durante o processo, de modo que eles não se transformem em defeitos depois da entrega do software. O benefício óbvio das revisões técnicas formais é a descoberta antecipada de erros, de modo que eles não se propaguem para o passo seguinte do processo de software. Para realizar as revisões, o engenheiro de software precisa investir tempo e esforço, e a organização de desenvolvimento precisa gastar dinheiro. Porém, os resultados são mais lucrativos. As revisões formais fornecem um custo-benefício demonstrável. As revisões de software são um filtro para o processo de engenharia de software, são aplicadas em vários pontos durante o desenvolvimento de software e servem para descobrir erros e defeitos que podem depois ser removidos. Uma revisão é um modo de usar a diversidade de um grupo de pessoas para:

- Indicar os aperfeiçoamentos necessários no produto de uma pessoa ou equipe;
- Confirmar as partes do produto nas quais o aperfeiçoamento não é desejado ou não é necessário;
- Conseguir trabalho técnico de qualidade mais uniforme, ou pelo menos mais previsível do que aquele que pode ser conseguido sem revisões, a fim de torná-lo mais gerenciável;
- Uma revisão técnica formal é uma atividade de garantia de qualidade de software realizada por engenheiros de software (e outros). Os objetivos são:
  - Descobrir erros na função, na lógica ou na implementação para qualquer representação do software;
  - Verificar que o software sob revisão satisfaz seus requisitos;
  - Garantir que o software tenha sido representado de acordo com padrões pré-definidos;
  - Conseguir software que seja desenvolvido de modo uniforme;
  - Tornar os projetos mais administráveis;

Para o mesmo autor, deve existir uma reunião para a revisão composta de 3 a 5 pessoas, com pautas definidas e a duração da reunião menor que duas horas, focalizando uma parte específica de todo o software. A reunião da revisão tem a participação do líder de revisão, de todos os revisores e do produtor. Um dos revisores assume o papel de redator da reunião. No fim da revisão, todos os participantes devem decidir sobre o produto. Com a decisão tomada, todos os participantes assinam uma lista na qual indicam sua participação na revisão e sua concordância com os resultados da equipe de revisão. Durante a revisão, um revisor registra os tópicos que foram levantados. Esses são resumidos no fim da reunião de revisão e uma lista de tópicos é produzida. O relatório deve responder as questões:

- O que foi revisado?
- Quem fez a revisão?
- Quais foram as descobertas e conclusões?

Para o mesmo autor, as diretrizes para a condução de revisões técnicas formais devem ser estabelecidas previamente e distribuídas a todos os revisores, receber a concordância de todos e depois devem ser seguidas. A qualidade de software é serviço de todos, ela pode ser conseguida pela análise, projeto, codificação e teste competente, bem como pela aplicação de revisões técnicas formais, e uma estratégia de testes multicamadas, de melhor controle dos produtos do trabalho de software e das modificações feitas neles, e da aplicação de padrões amplamente aceitos de engenharia de software.

#### **2.4.9. Modelos e Normas de Qualidade**

Segundo Pressman (2006), um sistema de garantia de qualidade pode ser definido como:

- A estrutura organizacional;
- As responsabilidades;
- Os procedimentos;
- Os processos;
- Os recursos para implementar a gestão da qualidade.

Para o mesmo autor, Sistemas de garantia de qualidade são criados para ajudar as organizações a garantir que seus produtos e serviços satisfaçam as expectativas do cliente, estando de acordo com as especificações. Para o desenvolvimento de software

com qualidade, dentro de prazos e custos controlados e compatíveis com o mercado, é fundamental a melhoria de processos de engenharia de software. Para tanto, abordagens e experiências para a melhoria de processo de software baseadas em modelos têm sido utilizadas com sucesso pelas organizações de software.

Os modelos mais utilizados têm sido:

- CMM
- CMMI

E as normas:

- ISO/IEC 9126(2001)
- ISO/IEC 14598(1997)
- ISO/IEC 12119(1994)
- SPICE ISO/IEC 15504(1998)

Segundo o autor, estes modelos e normas identificam processos fundamentais para a engenharia de software. Todos eles identificam, direta ou indiretamente, teste de software com um destes processos. Teste é fundamental para avaliação do software desenvolvido. Entretanto, testar software não é uma atividade trivial e exige conhecimentos, habilidades e infraestrutura específicos. A seguir serão descritas as principais características dos modelos mais utilizados.

#### **2.4.10. CMM / CMMI**

- **CMM**

Segundo os autores Weber, Rocha e Nascimento (2001), o modelo CMM (*Capability Maturity Model*) foi desenvolvido pelo SEI (*Software Engineering Institute*), com o objetivo de estabelecer um padrão de qualidade para o software desenvolvido para as Forças Armadas dos Estados Unidos. Ele foi concebido para o desenvolvimento de grandes projetos militares, portanto para aplicação em projetos menores e em outras áreas é necessário um trabalho cuidadoso de interpretação e adequação à realidade da organização. O modelo CMM baseou-se nos conceitos de qualidade total estabelecidos por Philip Crosby, onde ele mostrou que a implantação de sistemas de qualidade em empresas segue um amadurecimento gradativo que são: incerteza, espertar, esclarecimento, sabedoria e certeza. Estabeleceu também cinco níveis de maturidade:

- ✓ Nível 1 – inicial;
- ✓ Nível 2 – repetível;
- ✓ Nível 3 – definido;
- ✓ Nível 4 – gerenciado;
- ✓ Nível 5 – otimizado.

Para os mesmos autores, cada nível de maturidade possui um conjunto de Áreas-Chave de Processo (KPAs – *Key Process Areas*). O CMM descreve 18 KPAs: seis no nível 2, sete no nível 3, duas no nível 4 e três no nível 5. O nível 1 não possui KPAs, pois este nível não possui organização formalmente estabelecida. No nível 2 do CMM, tem a KPA (*Key Process Area*) referente à Garantia de Qualidade de Software, que tem como objetivo a aderência dos produtos e atividades de software aos padrões, procedimentos e requisitos aplicáveis. Um dos instrumentos para a implementação da Garantia de Qualidade de Software é o teste de software, disciplina exigida pela ASQ - *American Society for Quality*, para a obtenção da certificação SQE - *Software Quality Engineer*.

Para os mesmos autores, A KPA que diz respeito à Garantia de Qualidade de Software inclui revisões formais e informais dos produtos e auditorias dos processos. Para cumprir esta KPA, a empresa deve instituir um grupo de garantia de qualidade (ou ao menos um comitê), procedimentos que garantam que as atividades necessárias de garantia de qualidade são planejadas, executadas e revisadas. Estes procedimentos devem incluir medidas objetivas de verificação de qualidade, e também padrões de qualidade que possam ser usados como medida de comparação com os produtos e processos efetivamente utilizados. A cada projeto, portanto, são desenvolvidos planos de garantia de qualidade dentro do planejamento geral do projeto. Estes planos incluem pontos e procedimentos de verificação, tais como revisões de documentos, inspeções de códigos e, naturalmente, testes. O cumprimento destes planos permite a identificação de desvios de qualidade e a adoção de ações corretivas o mais cedo possível. O teste sistemático é uma atividade fundamental para a ascensão ao nível 3 do modelo CMM, além de ser significativo para as atividades de depuração, manutenção e estimativa de confiabilidade de software, devendo ser planejado para sua execução ao longo de todo o processo de desenvolvimento.



Para os mesmos autores, apesar do CMM descrever a importância dos testes na busca da qualidade do produto e processo, para a passagem do nível de maturidade dois para os três, o mesmo não diz como devem ser feitos os testes.

- **CMMI**

Segundo os autores Weber, Rocha e Nascimento (2001), o modelo CMMI (*Capability Maturity Model Integrated*), é uma evolução do SW-CMM. No CMMI está definida uma área de gerência de projeto composta por seis áreas de processo: planejamento de projeto, acompanhamento e controle de projeto, gerenciamento de acordos com fornecedores, gerenciamento integrado de projetos, gerenciamento de riscos e gerenciamento quantitativo de projeto. O CMMI define duas representações de modelos de maturidade: em estágios e contínua. A representação em estágios é similar ao modelo de maturidade definido pelo CMM. A representação contínua, como a Norma ISO/IEC 15504(1998), introduz o conceito de nível de capacidade, que difere do conceito de nível de maturidade usado pela representação em estágios por focar de forma individual nas áreas chave de processo, ao invés de lidar com o processo organizacional como um todo. A representação contínua do CMMI é composta por seis níveis de capacidade:

- ✓ Incompleto;
- ✓ Executável;
- ✓ Gerenciável;
- ✓ Definido;
- ✓ Quantitativamente Gerenciável;
- ✓ Otimizado.

Para os mesmos autores, cada um desses níveis de capacidade, a partir do nível 1, têm objetivos genéricos associados. Para cada objetivo genérico, são definidas práticas genéricas que devem ser atendidas para que a área do processo cumpra o objetivo genérico e possa ser classificada no nível de capacidade que esse objetivo genérico está relacionado.

#### **2.4.11. Normas ISO/IEC**

- ✓ ISO/IEC 9126(2001)

A norma ISO/IEC 9126(2001) define um conjunto de seis características básicas de qualidade que um produto de software deve ter. Para cada característica, há

subcaracterísticas associadas. As subcaracterísticas não serão mencionadas por não fazerem parte do escopo deste trabalho. As características desta norma são:

➤ **Funcionalidade**

Evidencia a capacidade do produto de software fornecer funções que satisfazem as necessidades explícitas e implícitas para a finalidade a que se destina o produto;

➤ **Confiabilidade**

Capacidade do produto de software manter o nível de desempenho especificado, quando usado sob as condições especificadas;

➤ **Usabilidade**

Evidencia a facilidade de uso do produto, capacidade do produto de software ser entendido, ser aprendido e ser atraente ao usuário quando usado sob as condições especificadas.

➤ **Eficiência**

Evidencia a compatibilidade entre os recursos e os tempos envolvidos, e o nível de desempenho requerido para o produto;

➤ **Manutenibilidade**

Evidência a facilidade para correções, atualizações e alterações do produto;

➤ **Portabilidade**

Evidencia a possibilidade de se utilizar o produto em diversas plataformas, com pequeno esforço para adaptação.

✓ **ISO/IEC 14598(1997)**

A norma ISO/IEC 14598(1997) provê um conjunto de guias que orientam o planejamento e a execução de um processo de avaliação da qualidade do produto de software. No Quadro 1 – Estrutura dos Guias da Norma ISO/IEC 14598(1997) é descrito o conjunto de guias que apoia este processo de avaliação.

Quadro 1 - Estrutura dos Guias da Norma ISO/IEC 14598.

Norma	Título Resumido	Enfoque
14598-1	Parte 1: Visão Geral	Visão geral da estrutura dessa série de Normas e dos processos de avaliação.
14598-2	Parte 2: Planejamento e Gerenciamento	Atividades de planejamento e gerenciamento do processo de avaliação.
14598-3	Parte 3: Processo para a Equipe de Desenvolvimento	Atividades de avaliação durante o processo de desenvolvimento de software.
14598-4	Parte 4: Processo para o Comprador	Atividades de avaliação no processo de seleção para aquisição de software
14598-5	Parte 5: Processo para o Avaliador	Ciclo de vida de avaliação, com definição das atividades, incluindo relações entre avaliador e requisitante.
14598-6	Parte 6: Módulos de Avaliação	São pacotes estruturados de métodos e ferramentas, para apoio de suas partes relacionadas.

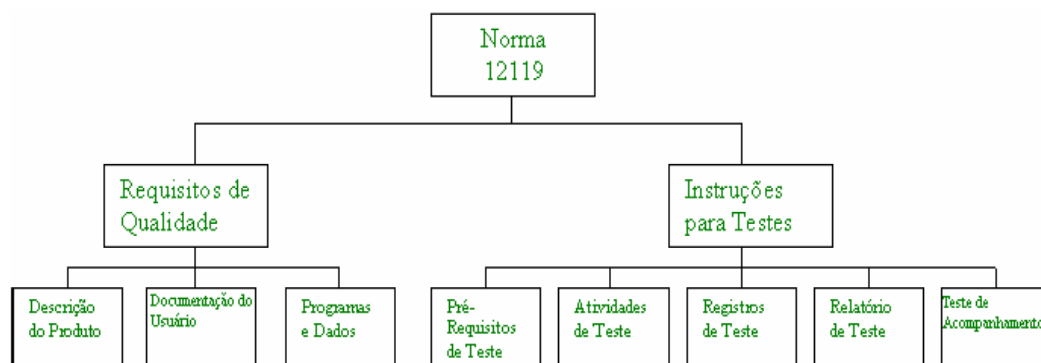
Fonte: ISO/IEC 14598(1997).

### ✓ ISO/IEC 12119(1994)

A norma ISO/IEC 12119(1994) que trata sobre testes, trata da avaliação de pacotes de software. Além de estabelecer os requisitos de qualidade para esse tipo de software, ela também destaca a necessidade de instruções para teste.

Esta Norma é aplicável à avaliação de pacotes de software na forma em que são oferecidos e liberados para uso no mercado. A figura 9 mostra a estrutura básica desta norma.

Figura 9 - Estrutura da Norma ISO/IEC 12119



Fonte: ISO/IEC 12119(1994)

## ▪ **Requisitos de Qualidade**

Os requisitos de qualidade da norma ISO/IEC 12119(1994) são compostos pela descrição do produto, documentação do usuário e programas e dados. A seguir, serão descritas as principais características dos requisitos:

### **A) Descrição do Produto**

É um documento que expõe as principais propriedades de um pacote de software, com os objetivos:

- ✓ Auxiliar o usuário ou os potenciais compradores deste produto, na avaliação da adequação do produto às necessidades dos usuários;
- ✓ Servir como base para testes.

Este documento deve estar disponível ao usuário, independentemente da aquisição do produto, através de um catálogo, de um disquete de apresentação ou qualquer outro meio disponível que alcance esse objetivo. A descrição deve ser clara, compreensível e harmônica com os outros documentos associados. A norma propõe aspectos práticos e diretos, indicando “o quê” deve conter esta descrição.

No Quadro 2 – Requisitos de Qualidade para a Descrição do Produto, estão resumidos uma parte destas indicações, as quais podem ser Mandatárias (termo “deve”) ou Recomendáveis (termo “pode”). O uso de um requisito como recomendável está diretamente relacionado com o tipo de produto, ou seja, para alguns tipos de produtos esses requisitos podem ser mandatários.

### **B) Documentação do Usuário**

É um conjunto completo de documentos disponíveis, na forma impressa ou não, que é fornecido para utilização de um produto, sendo também uma parte do produto. Ela deve incluir todos os dados necessários para instalação, para o uso da aplicação e para a manutenção do produto de software. Os principais requisitos da documentação do usuário estão descritos no Quadro 3 – Requisitos de Qualidade para a Documentação do Usuário.

### **C) Programas e Dados**

As características de Funcionalidade, Confiabilidade e Usabilidade são destacadas e devem ser verificadas através do uso do produto. Não há requisitos específicos para os aspectos de Eficiência, Manutenibilidade e Portabilidade. Qualquer requisito declarado na documentação do pacote referente às características citadas deve estar em conformidade.

Os principais requisitos para Programas e Dados estão descritos no Quadro 4 – Requisitos de Qualidade para Programas e Dados.

#### **D) Instruções para Testes**

Essa norma recomenda como um produto deve ser testado em relação aos requisitos de qualidade. O Quadro 5 – Instruções para Testes, mostra estas instruções de testes.

Quadro 2 - Requisitos de qualidade para a Descrição do Produto

Item	Requisitos
Requisitos Gerais sobre o Conteúdo da Descrição de Produto	O conteúdo da descrição deve ser inteligível, completo e possuir boa organização e apresentação, auxiliando os compradores em potencial na avaliação da adequação do produto às necessidades dos compradores, ante de adquiri-lo.
Identificações e Indicações	Deve apresentar o nome do produto, sua versão, os requisitos de hardware e software, as principais atividades realizadas e os componentes entregues com o pacote.
Declaração sobre Funcionalidade	Deve apresentar uma função geral das funções disponíveis, os valores limites se existirem e os dispositivos de segurança de acesso ao produto, quando necessário.
Declaração Sobre Confiabilidade	Deve apresentar as informações sobre os procedimentos para salvar e recuperar dados.
Declarações sobre Usabilidade	Deve apresentar o tipo de interface com o usuário, se é necessário algum conhecimento técnico específico para seu uso e se o produto pode ser adaptado às necessidades do usuário.
Declarações sobre Eficiência	Pode incluir informações a respeito do tempo de resposta.
Declarações sobre Manutenibilidade	Pode conter declarações sobre a manutenibilidade do produto.
Declaração sobre Portabilidade	Pode conter declarações sobre a portabilidade do produto.

Fonte: ISO/IEC 12119(1994)

Quadro 3 - Requisitos de qualidade para a Documentação do Usuário

Item	Requisitos
Completitude	Deve conter todas as informações necessárias para o uso do produto, tais como estabelecer todas as funções do pacote, procedimentos de instalação e os valores limites.
Correção	A informação apresentada deve estar correta e sem ambiguidade.
Consistência	Deve haver plena coerência entre a documentação e a descrição do produto. Cada termo deve ter um único significado.
Inteligibilidade	A documentação deve ser compreensível pela classe de usuários que desenvolve atividades com o produto, utilizando termos apropriados, exibições gráficas e explicações detalhadas.
Apresentação e Organização	Deve ser apresentada através de uma forma que facilite uma visão geral, através de índices e tabelas de conteúdo. Se o documento não está na forma impressa, deve haver indicação de como efetuar a impressão.

Fonte: ISO/IEC 12119(1994)

Quadro 4 - Requisitos de Qualidade para Programas e Dados

Item	Requisitos
Funcionalidade	Deve ser verificado o procedimento para a instalação do produto; a presença de todas as funções mencionadas; a execução correta destas funções; a ausência de contradições entre a descrição do produto e a documentação do usuário.
Confiabilidade	O usuário deve manter o controle do produto, sem corromper ou perder dados, mesmo que a capacidade declarada seja explorada até os limites ou fora deles, se uma entrada incorreta é efetuada, ou ainda se instruções explícitas na documentação são violadas.
Usabilidade	A comunicação entre o programa e o usuário deve ser de fácil entendimento, através das entradas de dados, mensagem e apresentação dos resultados, utilizando um vocabulário apropriado, representações gráficas e funções de auxílio (help), entre outras; o programa também deve proporcionar uma apresentação e organização que facilite uma visão geral das informações, além de procedimentos operacionais que auxiliem, por exemplo, a reversão de uma função executada e o uso de recursos de hipertexto de funções de auxílio, entre outras.

Fonte: ISO/IEC 12119(1994)

Quadro 5 - Instruções para Testes

Fases	Componentes	Recomendações
Pré-Requisitos de Testes	Presença de itens necessários ao teste.	Devem estar presentes, para a execução do teste, todos os componentes a serem entregues e os documentos de requisitos identificados na Descrição do Produto.
	Presença de Componente do Sistema	Deve estar disponível todo o ambiente de hardware e software identificados na descrição do produto.
	Treinamento	Se o treinamento for mencionado na descrição do produto, o responsável pelo teste deve ter acesso ao material e ao programa de treinamento.
Atividades de Teste	Descrição do Produto	Todo requisito nessa descrição deve ser testado. São instruções detalhadas sobre os procedimentos de teste.
	Documentação do Usuário	Todo requisito nessa descrição deve ser testado.
	Programas e Dados	Todo requisito especificado para o programa e dados deve ser testado.
Registros de Teste		Os registros devem conter informações suficientes para permitir a repetição do teste, através de um Plano de Testes com os casos de teste, os resultados associados e a identificação das pessoas envolvidas.
Relatório de Teste		Deve conter um resumo com os objetos e resultados dos testes efetuados com a seguinte estrutura: Identificação do Produto; Sistemas Computacionais utilizados; Documentos usados; resultados dos testes da Atividade de Teste; uma lista das não conformidades e a data de encerramento dos testes.
Teste de Acompanhamento		Quando um produto é testado novamente (considerando o teste anterior), todas as partes modificadas e as partes inalteradas, mas influenciáveis pelas modificações, devem ser testadas como se fosse um produto novo.

Fonte: ISO/IEC 12119(1994)

## 2.5. Testes de Software

Segundo Pfleeger (2003), Teste é um processo de execução de um programa com a finalidade de encontrar um erro. Um bom caso de teste é aquele que tem alta probabilidade de encontrar um erro ainda não descoberto. Se o teste for conduzido de maneira bem sucedida, ele descobrirá erros de software. Existem muitos tipos de testes. Os testes têm como foco a detecção de defeitos e existem muitos meios de tornarem mais eficientes e efetivos os esforços relacionados aos testes, pois teste de software é um elemento crítico da garantia de qualidade de software e representa a revisão final da especificação, projeto e geração de código. As técnicas de teste de software fornecem diretrizes sistemáticas para projetar testes que exercitam a lógica interna dos componentes de software, exercitam os domínios de entrada e saída do programa para descobrir erros na função, comportamento e desempenho do programa. Durante os primeiros estágios de teste um engenheiro de software realiza todos os testes. No entanto, à medida que os processos de testes progridem, especialistas podem ser envolvidos. Os testes são importantes para encontrar o maior número possível de erros. Testes devem ser conduzidos sistematicamente e casos de teste devem ser projetados usando técnicas disciplinadas. Um plano de testes deve contemplar as técnicas de testes existentes mais adequadas à realidade da empresa que desenvolve o software. Um erro pode ser o resultado de a especificação poder estar errada ou faltar um requisito; a especificação pode conter um requisito impossível de se implementar, considerando o *hardware* e o software estabelecidos; o projeto do sistema pode conter um defeito (banco de dados e linguagem de programação); o projeto do programa pode conter um defeito ou o código do programa pode estar errado. O software é testado de duas perspectivas diferentes: a lógica interna do programa é exercitada usando técnicas de projeto de casos de testes (de caixa-branca). Requisitos de software são exercitados usando técnicas de projeto de casos de teste caixa-preta. Em ambos os casos, o objetivo é encontrar o maior número de erros com a menor quantidade de esforço e tempo. Um conjunto de casos de teste planejado para exercitar tanto a lógica interna, quanto os requisitos externos é projetado e documentado, os resultados esperados são definidos e os resultados reais são registrados.

Segundo Pressman (2006), para garantir que os testes foram executados corretamente, deve-se modificar o ponto de vista tentando quebrar arduamente o software.



Projetar casos de teste de um modo disciplinado e revisar os casos de teste que se cria. Os princípios de testes de software são:

- Todos os testes devem ser relacionados aos requisitos do cliente;
- Os testes devem ser planejados muito antes do início do teste;
- O princípio de Pareto se aplica ao teste de software, isto é, 80% de todos os erros descobertos durante o teste vão provavelmente ser relacionados a 20% de todos os componentes do programa. O problema é isolar os componentes suspeitos e testá-los;
- O teste deve começar “no varejo”, isto é, nos componentes individuais, para depois progredir até o teste “no atacado”, em todo o sistema;
- Teste completo não é possível, mas pode-se cobrir adequadamente a lógica do programa e garantir que todas as condições no projeto ao nível de componente tenham sido exercitadas.

Para o mesmo autor, a testabilidade de software é a facilidade com que um programa de computador pode ser testado. Um software é testável quando possui as características:

- **Operabilidade**  
Quanto melhor funciona, mais eficientemente pode ser testado.
- **Observabilidade**  
O que se vê é o que se testa.
- **Controlabilidade**  
Quanto mais se pode controlar o software, mais o teste pode ser automatizado e otimizado.
- **Decomponibilidade**  
Controlando o escopo do teste, pode-se isolar problemas mais rapidamente e realizar re-testagem mais racionalmente.
- **Simplicidade**  
Quanto menos há a testar, mais rápido se pode testar.
- **Estabilidade**  
Quanto menos modificações, menos interrupções no teste.
- **Compreensibilidade**  
Quanto mais informações se têm, mais racionalmente será testado.

Com relação aos testes, Pressman (2006), sugere os seguintes atributos para um bom resultado:

- Um bom teste tem uma alta probabilidade de encontrar um erro;
- Não é redundante;
- Deve ser boa cepa: em um grupo de teste que têm um objetivo semelhante, as limitações de tempo e recursos podem ser abrandadas no sentido da execução de apenas um subconjunto desses testes;
- Um bom teste não deve ser muito simples, nem muito complexo.

### ➤ **Técnicas de Testes de Software**

De acordo com Pressman (2006), o projeto de testes para software e outros produtos que passam por engenharia pode ser tão desafiante quanto o projeto inicial do produto propriamente dito. Qualquer produto que passa por engenharia pode ser testado de duas maneiras:

- ✓ Sabendo a função especificada que o produto foi projetado para realizar, podem ser realizados testes que demonstram que cada função está plenamente operacional, enquanto ao mesmo tempo procura erros em cada função, esta abordagem é chamada de caixa-preta;
- ✓ Sabendo como é o trabalho interno de um produto, podem ser realizados testes para garantir que todas as engrenagens combinam, isto é, que as operações internas são realizadas de acordo com as especificações e que todos os componentes internos foram exercitados, esta abordagem é chamada de caixa-branca.

Para o mesmo autor, os testes de caixa-preta referem-se a testes que são conduzidos na interface do software. Os testes de caixa-branca são baseados em exame dos procedimentos. Caminhos lógicos internos ao software são testados, definindo casos de testes que exercitam conjuntos específicos de condições e/ou ciclos. Os atributos, tanto dos testes de caixa-branca quanto da caixa-preta, podem ser combinados para obter uma abordagem que valida a interface do software e garantir seletivamente que o funcionamento interno do software está correto. Através dos métodos de caixa-branca, o engenheiro de software pode formar casos de testes que garantam que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez; exercitam

todas as decisões lógicas em seus lados verdadeiros e falsos; executam todos os ciclos nos seus limites e dentro de seus intervalos operacionais e exercitam as estruturas de dados internas para garantir sua validade.

➤ **Teste da Caixa-Branca**

Para Pressman (2006), os testes de estrutura de controle são testes que são derivados do conhecimento da estrutura e da implementação do software. São chamados de testes de caixa-branca e são aplicados a unidades de programas pequenas, como sub-rotinas, ou operações associadas com um objeto ou parte do programa. A técnica de teste de caminho básico é uma das várias técnicas de teste de estrutura de controle. Apesar do teste de caminho básico ser simples e altamente efetivo, não são suficientes por si só. Os testes de estrutura de controle são:

- **Teste de Condição**

É um método de projeto de caso de teste que exercita as condições lógicas contidas em um módulo de programa. Uma condição simples é uma variável booleana ou uma expressão relacional, possivelmente precedida por um operador NÃO.

- **Teste de Fluxo de Dados**

O método de teste de fluxo de dados seleciona caminhos de teste de um programa de acordo com a localização das definições e do uso das variáveis no programa. Como os comandos de um programa estão relacionados uns com os outros, de acordo com as definições e usos das variáveis, a abordagem de teste de fluxo de dados é efetiva para a detecção de erros. No entanto, os problemas de medida da cobertura de teste e de seleção dos caminhos de teste para o teste de fluxo de dados são mais difíceis do que os problemas correspondentes para o teste de condição.

- **Teste de Ciclo**

Ciclo existe na grande maioria dos algoritmos implementados em software. E, no entanto, frequentemente se dá pouca atenção para conduzir testes de software. Teste de ciclo é uma técnica de teste caixa-branca que focaliza exclusivamente a validade de construções de ciclo. Os testes de caminho básico são uma estratégia de teste de estrutura para exercitar cada caminho de execução independente, por meio de um componente ou programa. Se cada caminho independente for executado, então todas as declarações no componente devem ter sido executadas

pelo menos uma vez. O método de caminho básico permite ao projetista de casos de teste originar uma medida da complexidade lógica de um projeto procedimental e usar essa medida como guia para definir um conjunto básico de caminhos de execução.

Para o mesmo autor, o teste de caminho básico é uma técnica de caixa-branca, que faz uso de grafos de programa (ou matrizes de grafos) para originar um conjunto de testes linearmente independentes que vão garantir a cobertura. Os testes de condição e de fluxo de dados exercitam adicionalmente a lógica do programa e os de ciclo complementam outras técnicas caixa-branca, fornecendo um procedimento para exercitar ciclos com vários graus de complexidade. Os seguintes passos podem ser aplicados para originar um conjunto-base de testes:

- ✓ Usar o projeto ou código como base, desenhar o grafo de fluxo correspondente;
- ✓ Determinar a complexidade ciclomática do grafo de fluxo resultante;
- ✓ Determinar um conjunto-base de caminhos linearmente independentes;
- ✓ Preparar casos de teste que irão forçar a execução de cada caminho do conjunto-base.

Para o mesmo autor, um exemplo de complexidade ciclomática pode-se descrever um cadastro simples de Unidade Federativa. Deverá ser desenhado um grafo de fluxo para este programa. Determinam-se quais os caminhos que o programa poderá executar, e após prepara-se todos os casos de testes para forçar este caminho. Para desenvolver uma ferramenta de software que assista ao teste de caminho básico, uma estrutura de dados, chamada de matriz de grafo, pode ser bastante útil.

### ➤ **Teste Caixa-Preta**

Para Pressman (2006), os testes funcionais ou testes de caixa preta focalizam os requisitos funcionais do software, tentando encontrar erros das seguintes categorias: funções incorretas ou omitidas; erros de interface; erros de estrutura de dados ou de acesso à base de dados externa; erros de comportamento ou desempenho e erros de iniciação e término. O teste de caixa-preta tende a ser realizado durante os últimos estágios do teste, procurando satisfazer as questões:

- ✓ Como a validade funcional é testada;
- ✓ Como o comportamento e o desempenho do sistema são testados;
- ✓ Que classes de entrada vão construir bons casos de teste;
- ✓ O sistema é particularmente sensível a certos valores de entrada;
- ✓ Como são isolados os limites de uma classe de dados;
- ✓ Que taxas e volumes de dados o sistema pode tolerar;
- ✓ Que efeito as combinações específicas de dados vão ter na operação do sistema.

- **Método de Testes Baseados em Grafos**

Segundo Pressman (2006), o teste de software começa criando um grafo dos objetos importantes e as relações entre eles e depois estabelecendo uma série de testes que vão cobrir o grafo de modo que cada objeto e relação sejam exercitados e que os erros sejam descobertos. Para executar esses passos, o engenheiro de software começa criando um grafo – uma coleção de nós que representam objetos; ligações que representam as relações entre objetos; pesos de nó que descrevem as propriedades de um nó (por exemplo, um valor de dados específico ou comportamento de um estado); e pesos de ligações que descrevem algumas características de uma ligação. Por fim, estabelece uma série de testes que vão cobrir o grafo de modo que cada objeto e relação sejam exercitados e que os erros sejam descobertos.

- **Análise do Valor-Limite**

Segundo Pressman (2006), Análise do Valor-Limite é uma técnica de projeto de casos de teste que completa o particionamento de equivalência. Em vez de selecionar qualquer elemento de uma classe de equivalência, a análise do valor-limite leva à seleção de casos de testes nas bordas da classe. Em vez de focalizar somente as condições de entrada, a análise do valor-limite deriva casos de teste também para o domínio de saída. Por exemplo, testar os valores limites de:

- ✓ Código do representante – 1 a 999.
- ✓ Código da região – 1 a 999.

Nesse contexto, o termo objeto abrange os objetos de dados, bem como os objetos de programa, tais como módulos ou coleções de comandos de linguagem de programação.

- **Particionamento de Equivalência**

Segundo Pressman (2006), os dados de entrada para um programa se dividem em uma série de diferentes classes que têm características comuns. Os programas normalmente se comportam de maneira comparável para todos os membros de uma classe. Devido a esse comportamento equivalente, essas classes são, algumas vezes, chamadas de partições de equivalência ou domínios. Uma abordagem sistemática dos testes para detecção de defeitos baseia-se em identificar todas as partições de equivalências que tenham de ser manuseadas por um programa. Os casos de teste são projetados de modo que as entradas e as saídas fiquem dentro destas partições. Um caso de teste ideal descobre sozinho uma classe de erros que poderia, de outra forma, exigir que muitos casos fossem executados antes que um erro geral fosse observado. Aplicando as diretrizes para a derivação das classes de equivalência, casos de teste para cada item de dados do domínio de entrada podem ser desenvolvidos e executados. Casos de testes são selecionados de modo que o maior número de atributos de uma classe de equivalência é exercitado ao mesmo tempo. Por exemplo, entrada para cadastro de Pedidos de Venda, um representante pode ter acesso a empresa usando um computador pessoal, fornecer uma senha de oito dígitos e, a seguir, informar uma série de dados para acessar funções para o cadastramento de pedidos. Os dados são parametrizados assim:

- ✓ Código do Representante: número de três dígitos;
- ✓ Região: número de quatro dígitos;
- ✓ Senha: alfanumérico de oito dígitos.
- ✓ Comandos: Cadastro de Pedidos/Alteração/Consulta de Metas, etc.

Para o mesmo autor, as condições de entrada associadas a cada elemento de entrada ao sistema podem ser:

- **Código do Representante**
  - ❖ Condição de entrada – intervalo – valor entre 1 e 999.
  - ❖ Condição de Entrada – valor com três dígitos.
- **Região**
  - ❖ Condição de entrada – intervalo – valor entre 1 e 999.
  - ❖ Condição de entrada – valor – com três dígitos.

- **Senha**

- ❖ Condição de entrada – booleana – uma senha pode estar ou não presente.
- ❖ Condição de entrada – valor – cadeia de oito caracteres

- **Comando**

- ❖ Condição de entrada – conjunto contendo os comandos de cadastro de pedidos, alterações de dados do representante ou consultar metas.

- **Teste de Comparação**

Segundo Sommerville (2003), quando um software redundante é desenvolvido, equipes de engenharia de software separadas desenvolvem versões independentes de uma aplicação usando a mesma especificação. Em tais situações, cada versão pode ser testada com os mesmos dados de teste para garantir que todas fornecem saídas idênticas. Depois, todas as versões são executadas em paralelo com comparação em tempo real dos resultados para garantir consistência. Quando o programa é antigo, antes de fazer as alterações solicitadas pelo cliente, utilizando o teste de comparação, pode-se fazer uma cópia de uma versão antiga, para guardar os valores, dados processados antes da alteração e após a implementação, verificando se não foi alterada a lógica principal do programa, e se não foi afetado o que já estava funcionando.

- **Técnica de Teste Orientado a Objetos**

Para Pressman (2006), O objetivo do teste é encontrar o maior número possível de erros, com um mínimo de esforço aplicado, durante um intervalo de tempo realístico. Apesar de esse objetivo permanecer o mesmo para softwares orientados a objetos, a natureza dos programas orientados a objeto muda tanto a estratégia, como a tática de testes. A arquitetura de software orientado a objetos resulta em uma série de subsistemas em camadas que encapsulam as classes colaboradoras. É necessário um sistema OO (Orientado a Objetos) em uma variedade de níveis diferentes, num esforço para descobrir erros que podem ocorrer à medida que as classes colaboram umas com as outras e os subsistemas se comunicam entre as camadas arquiteturais. O teste orientado a objetos é realizado pelos engenheiros de software e especialistas em teste e é estrategicamente semelhante ao teste de sistemas convencionais, mas é taticamente diferente. Uma vez gerado o código, o teste OO começa em pequena escala com o teste de classe. Uma série de testes é projetada para exercitar as operações das classes e examinar se existem erros

quando uma classe colabora com outras classes. À medida que as classes são integradas para formar um subsistema, são aplicados testes baseados no caminho de execução, baseados no uso em agrupamento, junto com abordagens baseadas em erros, para exercitar plenamente as classes colaboradoras. Finalmente são usados casos de uso para descobrir erros no nível de validação do software. A construção de software orientado a objetos começa com a criação de modelos de análise e projeto. Devido à natureza evolutiva do paradigma de engenharia de software orientado a objeto, esses modelos começam como representações relativamente informais dos requisitos do sistema e evoluem para modelos detalhados de classes, conexões e relações de classes, projetos e alocação do sistema, e projeto de objetos. A notação e a sintaxe usada para representar os modelos de análise e projeto vão estar ligadas ao método específico escolhido para o projeto. Assim sendo, a correção sintática é julgada quanto ao uso adequado da simbologia; cada modelo é revisado para garantir que as convenções adequadas de modelagem foram mantidas.

Para o mesmo autor, em aplicações convencionais, o teste de unidade está voltado para a menor unidade compilável de programa – o subprograma. Após, elas são integradas em uma estrutura de programa enquanto uma série de testes de regressão é executada para descobrir erros devido à interligação entre os módulos e a efeitos colaterais causados pela adição de novas unidades. Finalmente o sistema todo é testado para garantir que os erros nos requisitos são descobertos. No software orientado a objeto, o conceito de unidade se modifica devido ao encapsulamento que guia a definição de classes e objetos. Cada classe e cada instância de uma classe (objeto) empacotam os atributos (dados) e as operações que manipulam esses dados. Ao invés de testar um módulo individual, a menor unidade testável é a classe ou objeto encapsulado. Não se pode mais testar uma única operação isoladamente, e sim como parte de uma classe. Nos sistemas orientados a funções, existe uma distinção entre as unidades básicas de programas (funções) e os conjuntos dessas unidades de programas (módulos). Em sistemas orientados a objetos não há tal distinção. Frequentemente, não existem umas hierarquias de objetos bem definidas, como é comum nos sistemas orientados a funções. As estratégias de integração dos sistemas orientados a funções são muitas vezes inadequadas aos orientados a objetos. Como na validação convencional, a validação do software OO focaliza ações usuário-notado e a saída do sistema usuário-reconhecido. Para ajudar na derivação dos testes de validação, o testador deve apoiar-se nos casos de uso. Os métodos de projeto de



casos de teste para software OO estão ainda em evolução. Este teste está voltado para o projeto de sequencias apropriadas de operações, para exercitar os estados de uma classe.

Para o mesmo autor, a classe OO é o alvo do projeto de casos de testes. Como os atributos e operações são encapsulados, o teste de operações fora da classe é geralmente improdutivo. A estrutura concisa de muitas operações de classes faz com que alguns argumentem que o esforço aplicado ao teste caixa-branca poderia ser mais bem redirecionado para testes no nível de classes. Os métodos de teste caixa-preta são tão adequados para sistemas OO quanto são para sistemas desenvolvidos usando os métodos convencionais de engenharia de software.

- **Projeto de Teste Baseado em Cenário**

Segundo Pressman (2006), o teste baseado em cenário concentra-se no que o usuário faz e não no que o produto faz. Isso significa detectar as tarefas através dos casos de uso que o usuário tem que realizar depois de aplicá-las, bem como suas variantes aos testes. Cenários descobrem erros de interação. O teste baseado em cenários tende a exercitar múltiplos subsistemas em um único teste (os usuários não se limitam ao uso de um subsistema de cada vez)

- **Teste da Estrutura Superficial e da Estrutura Profunda**

Segundo Pressman (2006), a estrutura superficial se refere à estrutura de um programa OO externamente observável. É a estrutura que é óbvia a um usuário final. Os melhores testes são originados quando o projetista olha para o sistema de um modo novo ou não convencional. A estrutura profunda refere-se aos detalhes técnicos internos de um programa OO. A estrutura que é entendida pelo exame do projeto e/ou código. O teste da estrutura profunda é projetado para exercitar dependências, comportamentos e mecanismos de comunicação que tiverem sido estabelecidos como parte do projeto do sistema e de objetos do software OO. Os modelos de análise e projeto são usados como base do teste de estrutura profunda. Por exemplo, o diagrama objeto-relacionamento ou o diagrama de colaboração de subsistemas mostra as colaborações entre os objetos e subsistemas que podem não ser externamente visíveis.

- **Estratégias de Teste de Software**

Segundo Pressman (2006), uma estratégia de teste de software integra métodos de projeto de casos de teste em uma série bem planejada de passos, que resultam na

construção de software bem sucedido. A estratégia de software deve ser flexível para promover uma abordagem de teste sob medida e também rígida para promover planejamento razoável e acompanhamento gerencial, à medida que o projeto progride. Ela é desenvolvida pelo gerente de projeto, pelo engenheiro de software e pelos especialistas de teste. O projeto efetivo de casos de teste é importante, tanto quanto a estratégia que se usa para executá-los. O teste frequentemente responde por mais esforço de projeto que qualquer outra atividade de engenharia de software e começa no varejo e progride para o atacado, isto é, de um programa para o sistema como um todo. Uma especificação de teste documenta a abordagem da equipe de software para o teste, definindo um plano que descreve uma estratégia global e um procedimento que define passos específicos de teste e os testes que serão conduzidos. Um plano e procedimento de testes efetivos vão levar à construção coordenada do software e à descoberta de erros em cada estágio do processo de construção. Um conjunto de passos no qual se pode colocar técnicas e projetos de casos de testes e métodos de teste específicos deve ser definido para o processo de testes. As características de um conjunto de testes são:

- ✓ O teste começa em nível de componente e prossegue para fora;
- ✓ Diferentes técnicas de testes são adequadas em diferentes momentos;
- ✓ O teste é conduzido pelo desenvolvedor do software e um grupo de testes;
- ✓ O teste e a depuração são atividades diferentes, mas a depuração deve ser acomodada em qualquer estratégia de testes.

Para o mesmo autor, uma estratégia de teste de software deve acomodar testes de baixo nível, bem como testes de alto nível, que validam as principais funções do sistema. Um processo geral de testes começa com o teste de unidades individuais de programas, como funções ou objetos. Esse processo é integrado em subsistemas e sistemas e as interações dessas unidades são testadas, e depois de completar o sistema, o cliente pode realizar testes de aceitação para verificar se o sistema é executado conforme o especificado. Verificação refere-se ao conjunto de atividades que garante que o software implementa corretamente uma função específica. Validação se refere a um conjunto de atividades diferentes que garante que o software construído corresponde aos requisitos do cliente. Verificação e validação abrangem um amplo conjunto de atividades SQA que inclui revisões técnicas formais, auditoria da qualidade e configuração, monitoramento de desempenho, simulação, estudo de viabilidade, revisão de documentação, revisão da base

de dados, análise de algoritmos, teste de desenvolvimento, teste de qualificação e teste de instalação. A qualidade é incorporada ao software durante o processo de engenharia de software. A aplicação adequada de métodos e ferramentas, as revisões técnicas formais efetivas, todas levam à qualidade que é confirmada durante o teste. Do ponto de vista psicológico, a análise e o projeto de software são tarefas construtivas. Quando o teste começa, há uma tentativa sutil, mas definida de descobrir problemas nesta construção. Do ponto de vista do construtor, o teste pode ser considerado destrutivo. O desenvolvedor de software é sempre responsável por testar a alteração feita. Em muitos casos, o desenvolvedor também conduz os testes de integração. O papel do grupo independente de testes (*Independent Test Group, ITG*) é remover os problemas inerentes associados em deixar o construtor testar a coisa que ele construiu. O desenvolvedor e o ITG trabalham juntos para garantir que testes rigorosos vão ser conduzidos. Durante a condução do teste, o desenvolvedor deve estar disponível para corrigir erros que são descobertos.

Segundo Pressman (2006), uma estratégia de testes de software pode ser vista no contexto espiral. O teste de unidade começa no centro da espiral e se concentra em cada unidade (componente) do software como implementada no código-fonte. O teste progride movendo-se para fora ao longo da espiral para o teste de integração, em que o foco fica no projeto e construção da arquitetura do software. Após, encontra-se o teste de validação, em que os requisitos estabelecidos como parte da análise dos requisitos do software são validados em contraste com o software que acabou de ser construído. Finalmente chega-se ao teste de sistema, testando o sistema como um todo. Os demais autores consultados também reforçam a estratégia abordada por Pressman. O teste no contexto da engenharia de software é formado por quatro passos. Inicialmente, o teste focaliza cada componente individualmente, garantindo que ele funcione adequadamente como uma unidade, fazendo uso das técnicas de teste caixa branca. Em seguida, os componentes devem ser montados ou integrados para formar o pacote de software completo. O teste de integração utiliza mais as técnicas de caixa-preta.

Para o mesmo autor, os testes de validação fornecem garantia final de que o software satisfaz todos os requisitos funcionais, comportamentais e de desempenho. As técnicas de teste caixa preta são as únicas usadas na validação. O último passo de teste é o teste de sistema, onde é verificado se todos os elementos combinam adequadamente e se a função/desempenho global do sistema é conseguida. Quando o teste de desempenho

está completo, os desenvolvedores ficam seguros de que o sistema funciona de acordo com a descrição do projeto. O próximo passo é verificar com o cliente se o sistema está de acordo com as expectativas traçadas. No desenvolvimento de software nunca se acaba de testar o encargo, simplesmente passa do desenvolvedor para o cliente.

Pressman (2006) define como aspectos estratégicos de testes os seguintes passos:

- ✓ Especificar os requisitos do produto de um modo quantificável muito antes de o teste começar;
- ✓ Enunciar explicitamente os objetivos do teste;
- ✓ Entender os usuários de software e desenvolver um perfil para cada categoria de usuário;
- ✓ Desenvolver um plano de teste que enfatiza teste de ciclo rápido;
- ✓ Construir software robusto que é projetado para testar a si próprio: diagnosticar certas classes de erros;
- ✓ Usar revisões técnicas formais efetivas como filtro antes do teste;
- ✓ Conduzir revisões técnicas formais para avaliar a estratégia de testes e os casos de testes;
- ✓ Desenvolver uma abordagem de aperfeiçoamento contínuo para o processo de teste.

#### ○ **Teste de Unidade**

Para Pressman (2006), o teste de unidade focaliza o esforço de verificação na menor unidade de projeto de software – o componente ou módulo de software. A interface do módulo é testada para garantir que a informação flui adequadamente para dentro e para fora da unidade de programa que está sendo testado. Testes de fluxo de dados através da interface de um módulo são necessários antes que qualquer teste seja iniciado. Se os dados não entram e saem corretamente, todos os outros testes são discutíveis. O teste seletivo de caminhos de execução é uma tarefa essencial nos testes de unidade. Entre os erros mais comuns no cálculo estão:

- ✓ Precedência aritmética mal entendida ou incorreta;
- ✓ Operações em modo misto;
- ✓ Inicialização incorreta;
- ✓ Falta de acurácia e precisão;

- ✓ Representação incorreta de uma expressão simbólica.

Casos de testes devem descobrir erros como:

- ✓ Comparação de tipos de dados diferentes;
- ✓ Operações ou precedência lógica incorreta;
- ✓ Expectativa de igualdade improvável;
- ✓ Comparação incorreta de variáveis;
- ✓ Terminação de ciclo inadequada ou inexistente;
- ✓ Falha na saída quando iteração divergente é encontrada e variáveis de ciclo inadequadamente modificadas.

Para o mesmo autor, entre os erros potenciais que devem ser testados quando a manipulação de erros é avaliada estão:

- ✓ A descrição do erro é ininteligível.
- ✓ O erro mencionado não corresponde ao erro encontrado.
- ✓ A condição de erro provoca a intervenção do sistema antes da manipulação do erro.
- ✓ O processamento da condição de exceção está incorreto.
- ✓ A descrição de erro não fornece informações suficientes para ajudar na localização da causa do erro.

Para o mesmo autor, teste de limite é a última tarefa do passo de teste de unidade. O software normalmente falha nos seus limites. O teste de unidade é considerado como um apêndice ao passo de codificação. Uma revisão da informação de projeto fornece diretrizes para o estabelecimento de casos de teste que tem probabilidades de descobrir erros de cada uma das categorias anteriores.

#### ○ **Teste de Integração**

Segundo Pressman (2006), o teste de integração é uma técnica sistemática para construir a estrutura do programa, enquanto, ao mesmo tempo, conduz testes para descobrir erros associados às interfaces. O objetivo é tomar componentes testados em nível de unidade e construir a estrutura de programa determinada pelo projeto. Estes testes devem ser desenvolvidos a partir da especificação do sistema e começar logo após que alguns dos componentes do sistema estejam disponíveis.

#### ✓ **Integração Descendente**

Para Pressman (2006), o teste de integração descendente (*top-down*) é uma abordagem incremental para a construção da estrutura de um programa. Os módulos são integrados movendo-se descendentemente pela hierarquia de controle, começando com o módulo de controle principal (programa principal). Os módulos subordinados ao módulo de controle principal são incorporados à estrutura de maneira primeiro-em-profundidade ou primeiro-em-largura.

### ✓ **Integração Ascendente**

Para Pressman (2006), o teste de integração ascendente (*bottom-up*): os componentes de nível inferior são integrados e testados antes que os componentes de nível superior tenham sido desenvolvidos. Uma estratégia de integração ascendente pode ser implementada com os passos:

- Componentes de baixo nível são combinados em agregados, que realizam subfunção específica do software.
- Um pseudocontrolador (programa de controle para o teste) é escrito para coordenar a entrada e saída do caso de teste.
- O agregado é testado.
- Pseudocontroladores são removidos e agregados são combinados movendo-se para cima na estrutura do programa.

Nesse tipo de teste, primeiro testam-se as partes, para após testar o todo.

### ○ **Teste de Regressão**

Para Pressman (2006), em uma estratégia de teste de integração, o teste de regressão é a re-execução de algum subconjunto de testes que já foram conduzidos para garantir que as modificações não propagaram efeitos colaterais indesejáveis.

O teste de regressão é a atividade que ajuda a garantir que modificações não introduzem comportamento indesejável ou erros adicionais. Esta técnica se faz importante, pois ao alterar parte de um programa que já havia sido testado, e após encontrar um erro, ele foi alterado. Esta correção pode repercutir nas partes já testadas, portanto, é importante o teste de regressão. Como exemplo, pode-se citar o cálculo do custo médio das notas fiscais de entrada: a cada implementação, devem ser feitos os testes novamente, pois as rotinas para o cálculo são dependentes.

### ○ **Teste Fumaça**

Para Pressman (2006), o teste fumaça é comumente usado quando produtos de software, em embalagem lacrada, estão sendo desenvolvidos. Abrange as atividades:

- ✓ Componentes de software que foram traduzidos para código são integrados em uma construção.
- ✓ Uma série de testes é projetada para expor erros que impeçam a construção de desempenhar adequadamente a sua função.

Para o mesmo autor, a construção é integrada com outras construções e o produto inteiro passa pelo teste fumaça diariamente. A abordagem de integração pode ser descendente ou ascendente. A principal desvantagem da abordagem descendente é a necessidade de pseudocontrolados e as dificuldades de teste consequentes que podem ser a eles associadas. Os problemas associados com pseudocontrolados podem ser compensados pela vantagem de testar logo as principais funções de controle. A principal desvantagem da integração ascendente é que o programa, como uma entidade, não existe até que o último módulo seja adicionado. À medida que o teste de integração é conduzido, o testador deve identificar os módulos críticos. Um módulo crítico normalmente tem as características: aborda vários requisitos de software; tem um alto nível de controle; é complexo ou propenso a erro ou tem requisitos de desempenho bem definidos. Um plano global para integração do software e uma descrição dos testes específicos são documentados numa Especificação de Teste. O plano de teste descreve a estratégia global de integração.

Para o mesmo autor, os critérios e testes correspondentes são aplicados a todas as fases de teste:

- ✓ Integridade da Interface;
- ✓ Validade funcional;
- ✓ Conteúdo informacional;
- ✓ Desempenho.

Uma estratégia de integração, contida no plano de teste e detalhes de teste, descritos num procedimento de teste são ingredientes essenciais e devem aparecer.

- **Teste de Validação**

Segundo Pressman (2006), a validação é conseguida por intermédio de uma série de testes caixa-preta que demonstram conformidade com os requisitos. Um plano de testes descreve as classes de teste a serem conduzidas e um procedimento de teste define os casos de teste específicos, que serão usados para demonstrar a conformidade com os requisitos. Tanto o plano quanto o procedimento são projetados para garantir que todos os requisitos funcionais sejam satisfeitos, todas as características comportamentais sejam alcançados, a documentação esteja correta e tenha passado por um trabalho de engenharia humana, e outros requisitos sejam satisfeitos. (PRESSMAN, 2006, pg. 3)

Um importante elemento do processo de validação é a revisão da configuração.

- **Teste Alfa e Beta**

Segundo Pressman (2006), quando um software sob encomenda é construído para um cliente, uma série de testes de aceitação é conduzida para permitir ao cliente validar todos os requisitos. Se o software é desenvolvido como um produto a ser usado por vários clientes, não é prático realizar testes formais de aceitação com cada um. A maioria dos construtores de produtos de software usa um processo chamado de teste alfa e beta para descobrir erros que apenas o usuário final parece ser capaz de descobrir. O teste alfa é conduzido na instalação do desenvolvedor com o cliente. O software é usado num ambiente natural com o desenvolvedor olhando sobre o ombro do usuário e registrando erros e problemas de uso. Testes alfa são conduzidos num ambiente controlado. O teste beta é conduzido em uma ou mais instalações do cliente pelo usuário final do software. Diferente do teste alfa, o desenvolvedor geralmente não está presente. Como resultado dos problemas relatados durante os testes beta, os engenheiros de software fazem modificações e depois se preparam para liberar o produto de software para toda a base de cliente.

- **Teste de Sistema**

Segundo Pressman (2006), o engenheiro de software deve antecipar problemas potenciais de interface e projetar caminhos de manipulação de erros que testem toda a informação que chega de outros elementos do sistema; conduzir uma série de testes que simule maus dados ou outros erros em potencial na interface do software, registrar os resultados dos testes para usá-los como evidência, se houver



dedo-duro, e participar do planejamento e projeto dos testes de sistema para garantir que o software seja adequadamente testado. Teste de sistema é uma série de diferentes testes com a finalidade de exercitar por completo o sistema baseado em computador. Apesar de cada teste ter uma finalidade distinta, todos trabalham para verificar se os elementos do sistema foram adequadamente integrados e executam as funções a eles alocadas.

✓ **Teste de Recuperação**

O teste de recuperação é um teste de sistema que força o software a falhar de diversos modos e verifica se a recuperação é realizada.

✓ **Teste de Segurança**

O teste de segurança tenta verificar se os mecanismos de proteção incorporados ao um sistema vão de fato protegê-lo de invasão imprópria. O papel do projetista do sistema é tornar o custo da invasão maior do que o valor da informação que vai ser obtida.

✓ **Teste de Estresse**

Os testes de estresse são projetados para submeter programas a situações anormais. O teste de estresse executa um sistema de um modo que demanda recursos em quantidade, frequência ou volumes anormais. Os testes de estresse são relevantes para sistemas distribuídos com base em uma rede de processadores. Esses sistemas frequentemente exibem uma degradação enorme quando são sobrecarregados. A rede torna-se cheia de dados de coordenação, que os diferentes processos devem trocar e assim os processos tornam-se cada vez mais lentos, à medida que eles esperam pelos dados solicitados a partir de outros processos.

✓ **Teste de Desempenho**

O teste de desempenho é projetado para testar o desempenho do software durante a execução, no contexto de um sistema integrado.

○ **Teste de Ambientes, Arquiteturas e Aplicações Especializadas**

Segundo Pressman (2006), a medida que o software para computador torna-se mais complexo, a necessidade de abordagens de testes especializadas tem crescido.

✓ **Teste de GUI**

Para o mesmo autor, os testes de interface ocorrem quando módulos ou subsistemas são integrados para criar sistemas maiores. O objetivo dos testes de interface é detectar erros que possam ter sido introduzidos no sistema, em razão de erros ou suposições inválidas sobre interfaces.

Interfaces gráficas com o usuário (*Graphical User Interfaces, GUI*) apresentam desafios interessantes para os engenheiros de software. Devido a componentes reusáveis fornecidos como parte de ambientes de desenvolvimento de GUI, a criação da interface com o usuário tornou-se menos demorada e mais precisa. Mas ao mesmo tempo, a complexidade das GUI cresceu, levando a mais dificuldade no projeto e execução de casos de teste.

Como muitas GUI modernas têm a mesma aparência e funcionamento, uma série de testes padrão pode ser derivada.

✓ **Teste de Arquitetura Cliente/Servidor**

Para o mesmo autor, a forma distribuída de ambientes cliente/servidor, os aspectos de desempenho, associados com o processamento de transações, a presença potencial de várias plataformas de *hardware* diferentes, as complexidades de redes de comunicação, a necessidade de atender muitos clientes por uma base de dados centralizadas e os requisitos de coordenação impostos ao servidor, tudo se combina para tornar o teste de arquiteturas C/S e do software que nelas reside consideravelmente mais difícil do que de aplicações isoladas.

Os testes, nesse caso, tornam-se mais complexos e mais propensos a erros. Devem-se considerar no plano de testes o tipo de arquitetura a ser utilizados e conduzidos os testes.

✓ **Teste da Documentação e Dispositivos de Ajuda**

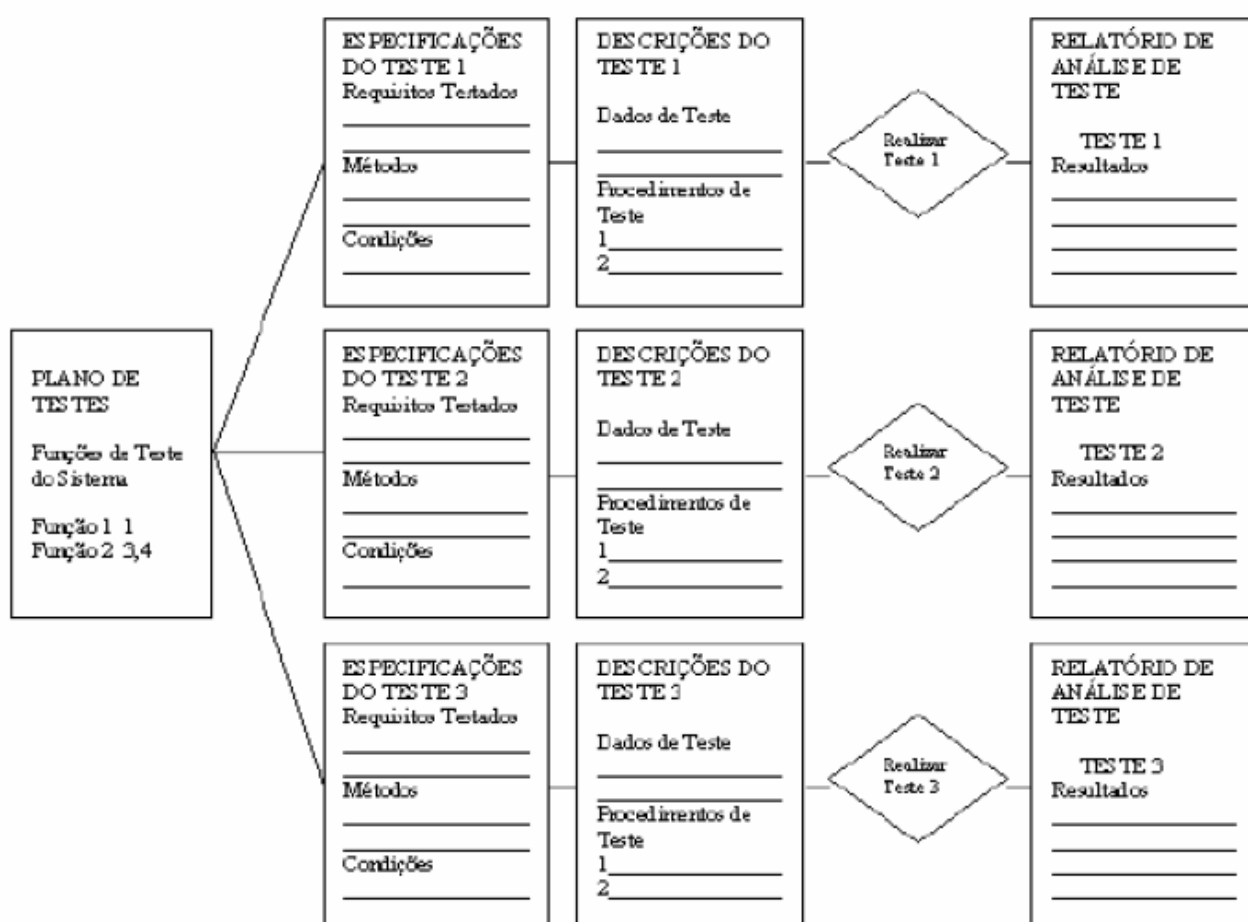
Para o mesmo autor, o teste de documentação deve ser uma parte significativa de todo o plano de testes de software, devido a sua importância. O teste de documentação pode ser visto em duas fases:

- Revisão e inspeção, onde examina o documento quanto à clareza editorial;
- Teste ao vivo, usa a documentação em conjunto com o uso do programa real.

## 2.6. Documentação de Testes

Para Pfleeger (2003), os testes podem ser complexos e difíceis. Quando os sistemas são grandes, o grande número de pessoas envolvidas no desenvolvimento e nos testes pode tornar a coordenação difícil. Para controlar a complexidade e a dificuldade dos testes, deve ser utilizada uma documentação completa e cuidadosa do teste projetado. Um plano de testes descreve o próprio sistema e o plano para executar todas as funções e características. A especificação dos testes e da avaliação detalha define os critérios para avaliar cada característica por ela tratada. Uma descrição do teste apresenta os seus dados e os procedimentos para os testes individuais, e um relatório de análise do teste descreve o resultado de cada um. A figura 10 mostra a relação dos documentos com o processo de testes.

Figura 10 - Detalhamento dos documentos produzidos durante o teste



Fonte: PFLEEGER (2003)

(A seguir, será descrito cada um dos documentos produzidos durante os testes).

## 2.7. Plano de Testes

Segundo Pfleeger (2003), o plano de testes deve começar estabelecendo os seus objetivos, que são:

- ✓ Orientar o gerenciamento dos testes;
- ✓ Orientar o esforço técnico requerido durante os testes;
- ✓ Estabelecer o planejamento e a programação da aplicação dos testes, incluindo a especificação dos equipamentos necessários, dos requisitos organizacionais, dos métodos de teste, dos resultados antecipados e da orientação do usuário;
- ✓ Explicar a natureza e extensão de cada teste;
- ✓ Documentar a entrada do teste, especificar procedimentos e resultados esperados.

Para o mesmo autor, os componentes de um plano de teste são: plano de testes, objetivos, referência a documentos, resumo do sistema, testes principais, cronograma e materiais necessários. O plano de testes se refere a outros documentos importantes, produzidos durante o desenvolvimento. O plano descreve as referências a documentos, explicam a relação entre os documentos de requisitos, os documentos, os componentes de código e os procedimentos de teste. No plano de teste, se descreve um resumo do sistema, pois como o leitor do plano de testes pode não ter se envolvido com os estágios anteriores do desenvolvimento, o resumo do sistema coloca o cronograma de testes e eventos em um contexto. O resumo não precisa ser detalhado, ele pode ser um esboço das principais entradas e saídas do sistema como uma descrição das transformações mais importantes. Após o resumo, o plano descreve os testes principais e as abordagens a serem utilizadas e o cronograma de eventos. O cronograma inclui a localização do teste, bem como o seu agendamento, como:

- Todo o período de testes;
- As subdivisões principais dos testes e seu início e término;
- Quaisquer requisitos antes dos testes;
- O tempo necessário para preparar e rever o relatório de testes.

Para o mesmo autor, se os testes ocorrem em diversos locais, o plano inclui um cronograma para cada local. Também são identificadas as necessidades de treinamento e manutenção. O plano identifica os materiais de teste em termos do que deve ser

providenciado e dos materiais supridos pelo local. Podemos verificar esses artefatos na figura 11.

Figura 11 - Detalhamento da composição de plano de testes.



Fonte: PFLEEGER (2003)

## 2.8. Especificação e Avaliação dos Testes

Segundo Pfleeger (2003), o plano de testes descreve sua divisão em testes individuais que abordam itens específicos. Para cada teste individual escreve-se a especificação e como avaliá-lo. A especificação começa ao relacionar os requisitos cuja satisfação deverá ser demonstrada pelo teste. Uma forma de observar a correspondência entre os requisitos e os testes é utilizar uma tabela ou gráfico, conforme o exemplo abaixo demonstrado pelo Quadro 6. Os requisitos relacionados trazem um número que faz referência ao documento de requisitos. A função à esquerda é exigida pelo requisito em cuja coluna é colocado um X.

Quadro 6 - Diagrama de Correspondência Requisitos-Teste

Diagrama de Correspondência Requisitos - Teste			
Teste	Requisito 2.4.1 – Gerar e Manter o Banco de Dados	Requisito 2.4.2 – Recuperar Dados Seletivamente	Requisito 2.4.3 – Produzir Relatórios Específicos
1.Adicionar Novo Registro	X		
2.Adicionar Campo	X		
3.Modificar Campo	X		
4.Excluir Registro	X		
5.Excluir Campo	X		
6.Criar Índice		X	
Recuperar Registro Com Solicitação			
7.Número da Célula		X	
8.Altura da Água		X	
9.Altura da Cobertura		X	
10.Cobertura do Piso		X	
11.Taxa c/ que a água flui		X	
12.Imprimir todo o Banco de dados X			X
13.Imprimir Relatório			X
14.Imprimir as Palavras Chaves			X
15.Imprimir Resumo da Simulação			X

Fonte: PFLEEGER (2003)

Para o mesmo autor, as funções do sistema envolvidas nos testes são enumeradas no quadro. Os testes de desempenho podem também ser descritos deste modo. Um teste individual é um conjunto de testes menores cuja somatória ilustra a conformidade com os requisitos. A especificação do teste mostra a relação entre os testes menores e os requisitos. Cada teste é conduzido por uma filosofia específica e adota um conjunto de métodos, mas que podem ser restringidos por outros requisitos e pela real situação de teste. A especificação esclarece essas condições que são:

- A forma de entrada de dados (manual ou automática – por um programa).
- Critérios de abrangência dos testes.
- Forma de registrar os dados.
- Especificação das limitações de sincronia, interface, equipamento, pessoal, banco de dados ou outras limitações nos testes.
- A ordenação dos testes menores, quando houver.

## **2.9. Descrição dos Testes**

Segundo Pfleeger (2003), uma descrição de testes é escrita para todo teste definido na respectiva especificação. Esses documentos devem ser detalhados e claros e precisam incluir:

- Os meios de controle
- Os dados
- Os procedimentos

Para o mesmo autor, o documento começa com uma descrição geral do teste, após descreve-se como o teste será iniciado e controlado automaticamente ou manualmente. Os dados de teste podem ser visualizados em partes diversas: dados de entrada, comandos de entrada, estados de entrada, dados de saída, estados de saída e mensagens produzidas pelo sistema. Um procedimento de testes é chamado de roteiro de teste porque ele dá uma descrição passo a passo de como realizar o teste. Um conjunto de etapas rigidamente definido estabelece o controle sobre o teste, de modo que se possa duplicar condições e recriar a falha, quando necessário, para tentar encontrar a causa do problema. Se o teste for interrompido por alguma razão, pode-se continuar os testes, sem precisar recomeçar. Por fim, a descrição dos testes explica a sequência de atividades requeridas para finalizar os testes. Essas atividades podem consistir na leitura ou na impressão dos dados mais importantes, na finalização de procedimentos automáticos ou no desligamento de partes dos equipamentos.

## **2.10. Relatório de Análise de Teste**

Segundo Pfleeger (2003), um relatório de análise de teste é necessário, pois:

- Ele documenta os resultados de um teste;

- Se uma falha ocorre, o relatório fornece as informações necessárias para duplicar a falha e localizar e solucionar a origem do problema;
- Ele fornece as informações necessárias para determinar se o projeto de desenvolvimento está completo;
- Ele dá confiança no desempenho do sistema.

Para o mesmo autor, o relatório de testes inclui um resumo do projeto, seus objetivos e as referências relevante para esse teste, e também indica as partes do plano de teste e documentos de especificação que lidam com esse teste. O relatório de análise de teste relaciona as funções e características de desempenho que serão demonstradas e descreve os resultados reais. Se um defeito ou deficiência for encontrado, o relatório discute o seu impacto. Se um defeito que não interrompe o teste, então pode continuar os testes, mas se um defeito causar a interrupção do sistema ou a exclusão de um arquivo de dados, a equipe de testes pode decidir parar os testes até que o defeito seja corrigido.

## **2.11. Formulário de Relatório de Problemas**

Para Pfleeger (2003), ao executar os testes, obtêm-se dados sobre defeitos e falhas nos formulários de relatório de problemas. Um formulário de relatório de discrepâncias é um relatório de problemas que descreve as ocorrências de problemas, em que os comportamentos ou atributos do sistema real não são o que se esperava. O formulário de relatório de defeito explica como um defeito foi detectado e resolvido, frequentemente, em resposta ao preenchimento de um formulário de relatório de discrepâncias.

O relatório de discrepâncias deve conter:

- Local: identificador dentro do sistema;
- Tempo: fase do desenvolvimento durante as quais o defeito foi criado, detectado e corrigido;
- Sintoma: tipo de mensagem de erro relatada ou atividade que revelou o erro;
- Resultado final: falha causada pelo defeito;
- Mecanismo: como a origem foi criada, detectada e corrigida;
- Causa: tipo de erro humano que levou ao defeito;
- Gravidade: refere-se à gravidade das falhas resultantes ou potenciais;



- Custo: tempo ou esforço necessário para localizar e corrigir; isso pode incluir a análise de custo se o defeito tivesse sido encontrado anteriormente ao processo de desenvolvimento.

Para o mesmo autor, o ideal seria que esse formulário fizesse referência a um ou mais relatórios de defeitos de modo que pudesse dizer quais defeitos causaram quais falhas. Tornam-se necessárias informações mais completas nos formulários de relatórios de problemas para poder avaliar a efetividade e a eficiência das práticas de testes e de desenvolvimento e remodelá-los de modo a diminuir as falhas.

### 3. Metodologia de Pesquisa

Quanto à metodologia, foi feita a opção pelo método Estudo de Caso,, porque esse método permite a comparação dos dados levantados no com o conteúdo utilizado de autores que são referência no assunto. Enquanto procedimento, este trabalho foi realizado por meio de Observação Direta, porque os dados levantados são vivenciados no seu cotidiano pelo autor dessa obra. A pesquisa realizada neste trabalho pode ser classificada como Exploratória, porque envolveu livros, artigos de revistas especializadas, e publicações na *Internet* de instituições de renome para o meio comercial e acadêmico, limitando-se em alguns autores de maior relevância em virtude do vasto número de conteúdo disponível.

#### 3.1. Protocolo da Pesquisa

O primeiro passo foi verificar e entender como é o processo de liberação dos sistemas desenvolvidos para a entrega final. Posteriormente, foi necessário identificar quais eram os processos de testes empregados para determinados tipos de projetos de desenvolvimento e quem eram os agentes responsáveis pela execução e divulgação dos resultados. A maior parte do tempo dedicado nesse trabalho,foi utilizado na pesquisa bibliográfica para identificar os métodos e as normas existentes e quais eram os mais praticados no mercado. Como o conteúdo literário era vasto, a pesquisa foi limitada aos autores de maior relevância no assunto Gestão de Qualidade e Gestão de Projetos de Software.

Após fichar as obras dos autores elaborei um questionário que foi apresentado a dois coordenadores de desenvolvimento (anônimos), que continha as seguintes perguntas:

- 1) – Todas as etapas do plano de testes são respeitadas nos projetos sob sua coordenação ?
- 2) – É possível negligenciar as etapas do plano de testes para que o prazo do projeto não seja comprometido?
- 3) - Os membros da equipe de projetos estão preparados para aplicar os planos de testes exigidos pela empresa?

Após receber as respostas, junto com o cotidiano observado pelo autor desta obra, foi possível elaborar o Estudo de Caso que é apresentado nesse trabalho onde é mostrado as práticas de Qualidade de Software atuais e quais suas vantagens e desvantagens.

## **4. Estudo de Caso**

### **4.1. Breve Histórico da Empresa “Banco X S.A.”**

Fundado em 1943, no Estado de São Paulo, o Banco X S.A. (nome fictício) é uma grande instituição financeira reconhecida mundialmente que opera em mais de 20 países e desenvolve 70% dos seus Sistemas de apoio utilizando conceitos amplamente difundidos no mercado financeiro mundial. Possui mais de 4.000 agências espalhadas em todo o território nacional e mais de 33.000 postos de atendimento. Foi um dos pioneiros no emprego maciço de tecnologia voltado para bancos. Possui departamento de desenvolvimento de softwares é composto por mais de 5.000 profissionais divididos por categorias, área de atuação e diretorias. Forma parcerias com grandes empresas de tecnologia como as norte-americanas Oracle, líder mundial em banco de dados e a Microsoft líder mundial fornecedora de sistemas operacionais básicos e interfaces de desenvolvimento.

Essa parceria proporciona a base para o sistema ERP (Enterprise Resource Planning) denominado Sistema de Gestão Empresarial FQ. O ERP-FQ é um sistema de informações composto por módulos e sistemas integrados que aperfeiçoam todos os processos administrativos, possibilitando o controle da estrutura organizacional, proporcionando aos gestores tomadas de decisões com base em dados seguros e confiáveis.

### **4.2. Gerenciamento da Qualidade Aplicados no Desenvolvimento de Software**

#### **4.2.1. Práticas Atuais do Processo de Desenvolvimento**

A Empresa classifica o desenvolvimento de software em manutenção dos programas existentes e desenvolvimento de novos programas para que o ERP acompanhe a evolução tecnológica e as novas demandas do mercado financeiro.

Na manutenção dos programas, as principais atividades são do tipo corretiva, adaptativa e de aperfeiçoamento para melhorar o desempenho e funcionalidades dos programas. Esse processo inicia com a abertura da solicitação de manutenção. Na solicitação são registrados os dados da área do solicitante, o responsável pela abertura da solicitação, a data, o descritivo detalhado dos requisitos e a sua severidade. A próxima etapa do processo é a abertura da Ordem de Serviço (OS) com os dados da solicitação

classificado com o tipo de manutenção, implementação (sem custo para a área) ou customização (com custo para a área). As OS's classificadas como implementação e customização são manutenções adaptativas ou de aperfeiçoamento. Em seguida a equipe responsável pelo desenvolvimento determina o prazo para entrega com base na capacidade de atendimento. Todas essas informações são registradas na ordem de serviço que é cadastrada no sistema gerenciador de chamados para que todas as áreas da empresa tenham acesso aos dados atualizado sobre a posição do chamado a qualquer momento.

Chamados classificados como customizações dependem da assinatura do gestor da área, autorizando o desenvolvimento de acordo com o custo e prazo de entrega estabelecidos. O desenvolvimento de novos programas inicia pelo processo de abertura da solicitação nos quais são registrados os dados do solicitante, do analista responsável e o descritivo detalhado dos requisitos do software. Em seguida, a solicitação é encaminhada para um comitê responsável que faz a avaliação da solicitação. Esse processo inclui a decisão para aceitar ou rejeitar a solicitação. Quando a solicitação é aceita, é realizado o cadastro da solicitação como chamado no gerenciador e depois encaminhado para a área responsável para o levantamento de requisitos técnicos. Na sequência, é determinado o prazo de entrega baseado na capacidade de desenvolvimento, registrando as informações no sistema gerenciador de pedidos para que todas tenham acesso aos dados atualizados da posição do pedido.

A próxima etapa é o processo de desenvolvimento pelos analistas e programadores, baseado na Análise de Negócio definida pelo analista responsável. O processo de desenvolvimento inclui a análise detalhada dos requisitos, a modelagem dos dados e da aplicação, a codificação do programa, a documentação do programa e os testes na área de desenvolvimento.

Na conclusão do processo de desenvolvimento o profissional responsável pela etapa aplica os testes previstos no plano de testes e evidencia a aplicação do testes registrando os resultados no fluxo de desenvolvimento, os testes são aplicados em um ambiente exclusivo que simula o ambiente de produção, esse ambiente é denominado Homologação. Todas as etapas de testes tem seu agente previamente definido como responsável. Na conclusão dos testes é realizado a disponibilização no ambiente de produção onde somente o solicitante tem acesso.

#### 4.2.2. Práticas Atuais de Controle de Qualidade

Dentro das práticas atuais de qualidade, a empresa desenvolve atividades preventivas em relação aos problemas da qualidade que surgem durante o processo de desenvolvimento. Estão inseridos no processo de desenvolvimento uma série de mecanismos preventivos que contribuem para diminuir o número de erros durante o desenvolvimento e que facilitam a manutenção do software durante a sua vida útil. Os padrões estão definidos no guia de procedimentos da empresa para área de desenvolvimento. Pode-se destacar algumas atividades importantes:

- Definição de padrões de interface;
- Definição de padrões operacionais dos programas;
- Convenções de nomenclatura para tabelas, colunas, funções, procedimentos e variáveis;
- Definição de padrões de comandos para o acesso de dados;
- Padronização no formato das mensagens geradas pelo sistema;
- Modularização das rotinas e programas;
- Ambiente de Homologação idêntico ao de Produção.

Além desses processos preventivos também existe uma política definida para os procedimentos de testes de software. O plano de testes da empresa está dividido em tipos de testes e cada tipo tem o seu responsável pela execução definido no plano de testes que está dividido da seguinte maneira:

- **Teste de Unidade**

Testa um pedaço do código, um nível ou componente, responsável pelo teste é o programador.

- **Teste Caixa Branca**

Testa se a execução funciona de acordo com o código definido, responsável pelo teste é o programador.

- **Teste Interface**

Testa se os objetos de tela e sua navegabilidade funcionam de acordo com os padrões definidos, responsável é o programador.

- **Teste Integração**

Testa um ou mais componentes combinados, responsável é o Analista Sênior da equipe.

- **Teste de Sistema**

Testa a aplicação como um todo, responsável é o Analista Sênior da equipe.

- **Teste Funcional**

Testa se as funcionalidades e regras do negócio que estão documentadas funcionam de acordo com as especificações, o responsável é Analista de Negócio solicitante.

- **Teste da Documentação**

Verifica se a documentação está completa e se mostra o que o software faz efetivamente, o responsável é o Analista Sênior.

- **Teste Negativo-Positivo**

Testa se aplicação vai funcionar no seu fluxo de execução, responsável setor de testes.

- **Teste de Regressão**

Testa toda aplicação toda vez que for inserida ou alterada uma característica na aplicação, o responsável é o Analista de Testes;

- **Teste Caixa Preta**

Testa todas as entradas e saídas desejadas sem se preocupar com o código, o responsável é o Analista de Testes.

- **Teste de Integridade**

Testa a integridade dos dados gerados e armazenados, o responsável é o Analista de Testes.

De acordo com as normas estabelecidas pela empresa, nenhum dos testes aplicados deveria ter inconsistências no resultado. Não há margens ou percentuais de tolerância. Porém não assegura para que o sistema seja entregue sem que todas as rotinas de testes tenham sido aplicadas.

#### **4.2.3. Comparação entre Literatura e Prática**

De acordo com a pesquisa, foi possível identificar os tipos de teste que estão sendo praticados pela a empresa em desenvolvimento de softwares. É importante salientar que nem todos os testes são aplicados em todos os softwares desenvolvidos, o plano de testes é desenvolvido em relação ao tamanho e complexidade do projeto de desenvolvimento. No estudo de caso foram pesquisados apenas projetos considerados de grande porte onde o plano de teste é o mais completo, como podemos ver no quadro 7.

No processo de desenvolvimento, percebeu-se que o modelo de desenvolvimento utilizado é incompleto e não segue todos os padrões citados na pesquisa bibliográfica. Na garantia da qualidade, foram identificados alguns processos preventivos, mas notou-se que a auditoria dos resultados é feita pela própria equipe de desenvolvimento e não por um setor ou grupo independente focado na qualidade do software, como sugerem os autores. Também foi possível identificar que as atividades preventivas em uso estão mais relacionadas à definição de padrões de desenvolvimento do que os Testes propriamente ditos, desta forma, identifica-se a necessidade de ampliar as atividades de auditoria dos resultados dos testes antes de entrega do sistema.

Baseado na pesquisa bibliográfica apresentada, conclui-se que a gestão de processo e qualidade de desenvolvimento de software na empresa está de acordo com os conceitos e práticas pesquisadas, mas foram identificadas algumas importantes atividades no processo que necessitam melhorias. A primeira delas é em relação à especificação dos requisitos de sistema e negócio, que apresenta um baixo nível de detalhamento e estruturação, gerando como resultado, um elevado número de não conformidades com os requisitos especificados, muitos erros de interpretação pela área de desenvolvimento, e por consequência, muitas alterações no sistema mesmo antes da entrega para a fase de testes. A auditoria, para assegurar que as atividades planejadas estejam sendo executadas, não é realizada de maneira eficiente e, assim, informações importantes sobre a situação do processo de desenvolvimento não são fornecidas a tempo aos gestores, gerando desvios e muitas vezes atrasos no prazo de entrega. O plano de testes da empresa segue padrões recomendados na literatura, porém existem brechas no processo para que os resultados sejam negligenciados e/ou não realizados. A documentação dos testes também não é registrada pelos responsáveis, gerando um baixo nível de qualidade e comprometimento.

Quadro 7 – Testes Aplicados e seus Responsáveis

Tipo Teste	Responsável	Satisfatório?	Motivos
Teste de Unidade	Programador	Quase sempre	O profissional manipula o dado de forma que o resultado dos testes seja satisfatório.
Teste Caixa Branca	Programador	Quase sempre	O profissional testa a entrada da informação somente no ponto alterado.
Teste Interface	Programador	Quase sempre	O profissional testa a entrada da informação somente no ponto alterado usando apenas um dado em específico
Teste Integração	Analista Sênior	Às vezes	O profissional não testa todas as integrações com outros sistemas
Teste de Sistema	Analista Sênior	Às vezes	O profissional não tem conhecimento de todas as funcionalidades do sistema
Teste Funcional	Analista Negócio	Às vezes	O profissional não tem total conhecimento das regras de negócio solicitadas
Teste da Documentação	Analista Sênior	Nunca	A qualidade da documentação é péssima suprimindo informações importantes do negócio e do sistema
Teste Negativo-Positivo	Analista de Testes	Às vezes	Todas as falhas nos testes de software a cargo dos Analistas de Testes remetem as seguintes condições: A) Plano de Testes mau definido; B) Profissionais desqualificados; C) Ambiente de Testes desatualizados; D) Documentação de sistema mal detalhada.
Teste de Regressão	Analista de Testes	Às vezes	
Teste Caixa Preta	Analista de Testes	Às vezes	
Teste de Integridade	Analista de Testes	Às vezes	



## 5. Conclusão

A metodologia do teste de software se reflete atualmente no comportamento das empresas na busca de implantar, ou mesmo melhorar, o processo de teste utilizado. Ainda que as técnicas de teste de software mais utilizadas foram criadas por volta dos anos 70, as empresa têm grande dificuldade com essa atividade devido a diversidade de tipos de sistemas e ambientes que possui.

A dificuldade de testar um software é caracterizada por alguns pontos importantes como:

1. O teste de software é um processo caro;
2. Existe uma falta de relação custo/benefício do teste;
3. Há a falta de profissionais especializados na área de teste;
4. Existem dificuldades em implantar um processo de teste;
5. Há o desconhecimento de um procedimento de teste adequado;
6. Há o desconhecimento de técnicas de teste adequadas;

No cenário atual, cada vez mais os clientes - usuários dos sistemas - estão menos dispostos a gastar com erros ocasionados pelos sistemas de software. Decisões podem ser tomadas em cima de dados incorretos, ocasionados por erros de programas.

O processo de testes, adequadamente definido, tem impacto positivo nos resultados de diversas outras atividades de desenvolvimento, podendo colaborar para estabelecer uma cultura de qualidade nos processos procurando diminuir o retrabalho gerado por falhas no desenvolvimento. Logo, torna-se necessário uma política de qualidade em todos os processos no desenvolvimento do software, aliada a uma equipe de testes bem organizada e treinada, de modo a auxiliar e minimizar os erros que possam passar despercebidos pela equipe de desenvolvimento.

É necessário salientar que, as organizações procuram por sistemas que atendam as suas necessidades, efetuem suas funções com a máxima qualidade e acrescentem valor ao negócio. A qualidade de um sistema é determinada a partir do início de seu desenvolvimento, devendo ser tratada como uma questão estratégica da organização. Uma criteriosa análise, feita logo no início do projeto, visa encontrar erros, identificar inconsistências e averiguar quão correto e completo é o entendimento do problema e

adequada é a solução trabalhada. A adoção da disciplina da boa gestão seguindo as diretrizes do PMI® tem resultado na melhoria da prática da engenharia de software uma vez que o desenvolvimento de sistemas ou produtos de software é realizado de forma sistemática e com adequada monitoração dos recursos alocados.

O PMI® frisa a formação de equipes de melhoria, medição e comunicação dos resultados, avaliação dos custos da qualidade, estabelecimento dos objetivos, comprometendo e envolvendo todos os níveis da organização com a qualidade. O papel dos gerentes é incentivar e instituir a qualidade como uma atitude na organização. Os esforços em direção à qualidade total devem ser constantes e ininterruptos e precisam de uma estrutura organizacional adequada e de uma estratégia direcionada para a qualidade. O aprimoramento dos processos, o treinamento da equipe, uma estrutura adequada, a aplicação dos princípios da gestão de projetos e a gestão pela qualidade são atividades essenciais à execução de projetos e sucesso de produtos.

Diante dos dados levantados e visualizados no Estudo de caso, o autor desse trabalho propõe um sistema de Dupla Custódia nos itens de Teste de Qualidade aplicados nos softwares desenvolvidos, como poderemos ver no Quadro 8. A aplicação desse método terá um impacto direto em dois fatores fundamentais no projeto de desenvolvimento aumentando o prazo necessário para a aplicação dos testes e aumentando o custo com ferramentas e profissionais. Porém é importante que a organização tenha a visão que todo o investimento feito em melhoria da Qualidade acarreta na diminuição significativa de atividades pós-implantação de consertos e ajustes que poderiam ter sido identificados na fase de testes.

Futuras pesquisas podem ser elaboradas baseadas nos resultados da aplicação rígida das técnicas de testes de software no método de Dupla Custódia proposto pelo autor desse trabalho.

Quadro 8 - Método de Dupla Custódia de Testes

Tipo Teste	Primeiro Responsável	Segundo Responsável	Necessidades Básicas	Ações
Teste de Unidade	Programador 1	Programador 2	Ambiente de desenvolvimento; Definição de sistema melhor detalhado; Roteiro de Teste de Unidade melhor definido;	Documentar Testes; Realizar ajustes antes da próxima etapa de Testes;
Teste Caixa Branca	Programador	Analista Sênior	Acesso ao resultado dos testes anteriores; Definição de sistema melhor detalhada; Roteiro de Teste Caixa Branca melhor definido;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência erro; Informar analistas responsáveis
Teste de Interface	Programador	Analista Sênior	Acesso ao resultado dos testes anteriores; Definição de sistema melhor detalhada; Roteiro de Teste de Interface melhor definido; Mapeamento de todas as interfaces do sistema;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência de erro; Informar analistas responsáveis
Teste de Integração	Analista Sênior 1	Analista Sênior 2	Acesso ao resultado dos testes anteriores; Definição de sistema melhor detalhada; Roteiro de Teste de Integração melhor definido; Mapeamento de todas as interfaces do sistema;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência de erro; Informar analistas responsáveis; Informar áreas impactadas pelos testes;

<b>Tipo Teste</b>	<b>Primeiro Responsável</b>	<b>Segundo Responsável</b>	<b>Necessidades Básicas</b>	<b>Ações</b>
Teste de Sistema	Analista Sênior	Analista Negócio	Acesso ao resultado dos testes anteriores; Definição de sistema melhor detalhada; Roteiro de Teste de Sistema melhor definido; Mapeamento de todas as funções do sistema;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência de erro; Informar analistas responsáveis; Informar áreas impactadas pelos testes;
Teste Funcional	Analista Sênior	Analista Negócio	Acesso ao resultado dos testes anteriores; Definição de Negócio melhor detalhada; Roteiro de Teste Funcional melhor definido; Mapeamento de todas as funções do sistema;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência de erro; Informar analistas responsáveis; Informar áreas impactadas pelos testes;
Teste da Documentação	Analista Sênior	Analista Negócio	Definição de Negócio melhor detalhada; Definição de sistema melhor detalhada; Definição dos tipos de documento pertinente ao sistema	Publicar documentos; Informar analistas responsáveis; Informar equipe externa de testes;
Teste Negativo-Positivo	Analista de Testes 1	Analista de Testes 2	Acesso ao resultado dos testes anteriores; Acesso a documentação do sistema; Definição de Negócio melhor detalhada; Definição de sistema melhor detalhada;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência de erro; Informar analistas responsáveis;

<b>Tipo Teste</b>	<b>Primeiro Responsável</b>	<b>Segundo Responsável</b>	<b>Necessidades Básicas</b>	<b>Ações</b>
Teste de Regressão	Analista de Testes 1	Analista de Testes 2	Acesso ao resultado dos testes anteriores; Acesso a documentação do sistema; Definição de Negócio melhor detalhada; Definição de sistema melhor detalhada;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência de erro; Informar analistas responsáveis;
Teste Caixa Preta	Analista de Testes 1	Analista de Testes 2	Acesso ao resultado dos testes anteriores; Acesso a documentação do sistema; Definição de Negócio melhor detalhada; Definição de sistema melhor detalhada;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência de erro; Informar analistas responsáveis;
Teste de Integridade	Analista de Testes 1	Analista de Testes 2	Acesso ao resultado dos testes anteriores; Acesso a documentação do sistema; Definição de Negócio melhor detalhada; Definição de sistema melhor detalhada;	Documentar Testes; Devolver o sistema para desenvolvimento caso haja ocorrência de erro; Informar analistas responsáveis;

## REFERENCIAS BIBLIOGRÁFICAS

ISO/IEC 9126-1, International Standard. **Information Technology – Software Quality Characteristics and Metrics** – Part 1: Quality Characteristics and Sub-Characteristics; Part 2: External Metrics; Part 3: Internal Metrics; 2001 (Working Draft).

ISO/IEC 12119 International Standard. **Information Technology-Software Packages – Quality requirements and testing**; 1994.

ISO/IEC 14598 Technical Report., Parts 1-9: **Information Technology – Software Process Assessment**, 1997.

ISO/IEC 15504 Technical Report. Parts 1-9: **Information Technology – Software Process Assessment**, 1998.

PAULA FILHO, W. P. **Engenharia de software: fundamentos, métodos e padrões**. 2. ed. São Paulo: LTC, 2003.

PETERS, J. F.; PEDRYCZ, W. **Engenharia de Software Teoria e Prática**. Rio de Janeiro: Campus, 2001.

PFLEEGER, S. L. **Engenharia de Software – Teoria e Prática**. 2ª edição. São Paulo: Pearson Prentice Hall, 2003.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. São Paulo: Pearson Makron Books, 2006.

PROJECT MANAGEMENT INSTITUTE **Um guia do Conjunto de Conhecimentos em Gerenciamento de Projetos – Guia PMBOK**. 4 ed. Estados Unidos: Project Management Institute, 2008.

SOMMERVILLE, I. **Engenharia de Software**. 6. ed. São Paulo: Pearson Education do Brasil, 2003.

WEBER, K. C.; ROCHA, A. R. C.; NASCIMENTO, C. J. **Qualidade e Produtividade em Software**, 4ª edição - Renovada, Makron Books, São Paulo, 2001.