UNIVERSIDADE CATÓLICA DE SANTOS
UNISANTOS

Gestão de Qualidade para Testes de *Software* conforme NBR ISO/IEC 12207

IVAN CARLOS PAVÃO

Santos
2009

UNIVERSIDADE CATÓLICA DE SANTOS

UNISANTOS

IVAN CARLOS PAVÃO

Gestão de Qualidade para Testes de *Software* conforme ISO 12207

Relatório de defesa submetida à avaliação, como parte dos requisitos para a obtenção de título de Mestre em Informática na Universidade Católica de Santos – UNISANTOS, sob a orientação do Prof. Dr. Getulio Kazue Akabane.

Santos

2009

Dados Internacionais de Catalogação
Sistema de Bibliotecas da Universidade Católica de Santos – UNISANTOS
SibiU

P337g Pavão, Ivan Carlos
Gestão de Qualidade para Testes de Software conforme NBR ISO/IEC 12207 /
Ivan Carlos Pavão – Santos:
[s.n.] 2009.
101 f. ; 30 cm. (Dissertação de Mestrado – Universidade Católica de Santos, Programa em Informática)

I. Pavão, Ivan Carlos. II. Título.

CDU 681.3:004(043.3)

UNIVERSIDADE CATÓLICA DE SANTOS
UNISANTOS

IVAN CARLOS PAVÃO

Gestão de Qualidade para Testes de *Software* conforme NBR ISO/IEC 12207

Relatório de defesa submetida à avaliação, como parte dos requisitos para a obtenção de título de Mestre em Informática na Universidade Católica de Santos – UNISANTOS, sob a orientação do Prof. Dr. Getulio Kazue Akabane.

COMISSÃO EXAMINADORA

_____
Prof. Dr. Getulio Kazue Akabane

_____
Prof. Dr. Belmiro do Nascimento João

_____
Prof. Dr. Carlos Hideo Arima

Santos
2009

DEDICATÓRIA

Aos meus pais,
Laércio Pavão "*in memorian*"
e Encarnação Manoel Pavão.

AGRADECIMENTOS

Agradecimento especial aos meus pais, Laércio Pavão "*in memorian*" e Encarnação Manoel Pavão que contribuíram, emocionalmente e financeiramente, para a minha formação acadêmica. Pessoas que, com o uso de palavras, seria difícil homenagear, pois aprendi com eles que isso se faz com gestos e atitudes.

Ao Professor Orientador Dr. Getulio Kazue Akabane que gentilmente aceitou terminar a orientação deste trabalho.

Ao Ex-Orientador Professor Dr. Leandro Nunes de Castro Silva, que com sua paciência e bondade, orientou este trabalho até a qualificação.

A todos os professores e ex-professores do mestrado em informática da Unisantos, cujo ensinamento transmitido durante as aulas, desde Estrutura de Dados até Criptografia, me acompanhará por toda a vida.

A minha namorada Fabiana Cremonez da Silva, pelo constante apoio e pela paciência a mim dispensada, nas inúmeras vezes que deixamos de estar juntos devido ao trabalho que por mim esperava.

Ao meu amigo Paulo Marcotti que, muitas vezes, entre um pedaço de pizza e um pão com bolinho, incentivou e acreditou que esta dissertação seria concluída.

A todos os meus amigos que contribuíram com palavras de incentivo durante os períodos de dificuldades.

Por último, porém importante, Adilson Barros que ouviu todas as minhas dificuldades, tristezas, felicidades e desespero durante estes últimos três anos de terapia.

RESUMO

Com a exigência de apresentar um diferencial ao mercado e garantir a sua integração em um ambiente de negócios competitivos, muitas organizações buscam conquistar a alta qualidade em todos os seus processos, tanto nos produtos ou serviços, quanto nas atividades essenciais da empresa. Devido a essa busca, o termo "qualidade" está em crescente evidência e para obtê-la no processo de desenvolvimento de *software,* é necessário que as organizações mudem hábitos, quebrem os paradigmas e adotem processos que garantam melhorias não somente em uma fase, mas em todo o processo de desenvolvimento do *software*. Este trabalho aborda a qualidade de testes de *software*, mostrando que no planejamento de todos os processos do ciclo de vida de *software*, há a possibilidade de se obter bons resultados qualitativos necessários ao produto final. Para que isto aconteça, é necessário que se aplique a "qualidade" desde o processo inicial (levantamento de requisitos) até a conclusão e entrega do sistema de *software* ao cliente.

Palavras-chaves: Ciclo de vida, NBR ISO/IEC 12207, processo, qualidade, *software*, teste.

ABSTRACT

Trying to show the market differences and guarantee a business competition environment, lots of organizations are looking for quality in their processes; both in products or services, as in company activities. Becase of this search, the Word "quality" is becoming evident day by day. To get "quality" in the development of a software is necessary that the organizations let their old faiths, let down paradigms and look at quality not only as a fase, but as an integrator part of the process of software's development.

This assignment studies the software tests'quality, showing that in the software cicle of life, there is a possibility to get the necessary quality to the final software product. For this happening, it's necessary to apply "quality" since the beggining of the process till its conclusion and final destiny the client.

Keywords: *Cicle of life, NBR ISO/IEC 12207, process, qualitity, software, test.*

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

ABNT – Associação Brasileira de Normas Técnicas.

ANSI – American National Standards Institute

ASQC – American Society for Quality Control

CMM – Capability Maturity Model

CMMI – Capability Maturity Model Integration

ISO – International Standards Organization

IEC – International Electrotechnical Commission

IEEE – Institute of Eletrical and Eletronic Engineers

KPA – Key Process Área

NBR – Norma Brasileira

PMBOK – Project Management Body of Knowledge.

PMI – Project Management Institute.

SEI – Software Engineering Institute

SQA – Software Quality Assurance

SQUID – Software Quality in Development

TQC – Total Quality Control

SUMÁRIO

1. INTRODUÇÃO

Num ambiente cada vez mais competitivo, as organizações buscam eficiência em seus processos internos, para ampliar a sua forma de atuação no mercado. Com o objetivo de orientar suas decisões e apresentar um diferencial no mercado, as organizações estão progressivamente mais dependentes da tecnologia, em particular dos Sistemas de *Software*.

Neste cenário, os riscos de mau funcionamento dos sistemas aumentam, devido à grande complexidade dos processos internos das empresas, tornando-se mais difícil produzir *software* com um nível de qualidade desejado.

O processo de desenvolvimento de *software* envolve uma série de fatores que o tornam crítico e complexo. Entre esses fatores, o mais relevante é exigência de programas de alta qualidade em prazos cada vez menores.

Como conseqüência da redução desses prazos, uma série de etapas, consideradas essenciais por diversas metodologias, não recebem a devida atenção durante o processo de desenvolvimento.

Uma dessas etapas é a fase de testes, foco do presente trabalho de pesquisa, considerada atualmente essencial para garantir as funcionalidades requeridas e exigida pelo cliente.

Um grande equívoco é pensar que testes de *software* só se aplicam ao final da codificação. Desta forma, se por algum motivo há atraso nos processos anteriores ao de testes, o teste é pressionado e acaba sendo realizado sem nenhum planejamento ou qualidade. Este pensamento conduz os projetos de *software* ao fracasso.

A norma ISO/IEC 12207 – tecnologia da informação - processos de ciclo de vida de *software* tem como objetivo auxiliar os envolvidos na produção de *software* a definir seus papéis, por meio de processos bem definidos, e, assim, proporcionar às organizações melhor entendimento das atividades a serem executadas obtendo qualidade em todo o ciclo de vida do *software.*

As empresas devem perceber que implantar um processo de garantia da qualidade de *software* não é somente uma opção a ser estudada, mas parte de uma estratégia de sobrevivência em um mercado cada vez mais exigente e competitivo.

O presente trabalho apresenta diversas definições e teorias que atualmente começam a ser praticadas nas empresas, mas em um grau de importância abaixo do merecido. A consolidação dos conceitos deste trabalho pode auxiliar as pequenas e médias empresas, visualizar a importância da implantação de metodologias para a área de testes de *software*, utilizando como citado nesta dissertação a Norma NBR/ISO 12207.

1.1 Justificativa

Nota-se atualmente que muitas organizações tem suas atividades suportadas por aplicativos, como é o caso dos diversos sistemas de informação, que integram vários departamentos da empresa. Consequentemente, a ocorrência de falhas nesses sistemas pode acarretar perdas significativas para as organizações. A qualidade do *software* com que as organização trabalha torna-se, portanto, um fator imprescindível para o aumento de produtividade.

Justifica-se, dessa forma, a procura por produtos cada vez melhores, disseminando-se as práticas de testes para todas as fases do ciclo de vida do *software*. É nesse contexto que se insere a necessidade de um levantamento a respeito da gestão de qualidade para os testes de *software*, no Brasil.

1.2 Objetivo

O objetivo da pesquisa foi identificar os benefícios existentes na relação entre garantia de qualidade de *software* e as atividades de processo de teste durante o ciclo de vida de desenvolvimento do *software.*

Para tanto, pretende-se observar e elencar os principais passos que devem ser implementados durante todo o processo do ciclo de vida do *software,* enfatizando a fase de testes, de forma a garantir a qualidade dos *software* de acordo com as características estabelecidas e exigidas pela norma ISO/IEC 12207 (1998).1.2.1 Objetivos específicos

1 - Apresentar estudo sobre qualidade de *software* e determinar a fase ideal de inserção dos processos de testes de *software*.

2 – Detalhar as atividades de teste de *software* durante o ciclo de vida do desenvolvimento dos sistemas baseados em *software.*

3 – Detalhar os tipos de teste de *software*.

4 – Investigar como as empresas utilizam os testes de *software*.

1.3. Metodologia da pesquisa

Bryman (1989) afirma que a metodologia possui duas abordagens: a quantitativa e a qualitativa. Para Nakano e Fleury (1996), os métodos quantitativos são chamados de métodos tradicionais e os qualitativos de não-tradicionais.

A metodologia abordada foi qualitativa, Segundo Bryman (1989), a principal característica do método de pesquisa qualitativa é obter informações sobre o objeto em estudo segundo a perspectiva da pessoa pesquisada.

Para Malhotra (2005), o objetivo da pesquisa exploratória é explorar ou examinar um problema/situação para proporcionar esclarecimento e compreensão. Pelo método qualitativo, o resultado demonstra a preocupação com a qualidade das informações e respostas, proporcionando melhor visão e compreensão do problema/situação.

Esta pesquisa tem como objeto de estudo os processos de gestão de qualidades em empresas de consultoria especializadas na produção de *software* comercial de gestão empresarial. Na abordagem de tal objeto, a pesquisa tomou o estudo de caso, amparando-se por um levantamento bibliográfico dos principais aspectos referentes ao objeto de estudo.

Gil (2002) define a pesquisa ou estudo de caso como sendo semelhante ao levantamento bibliográfico, mas com várias vantagens; entre elas destaca-se o fato de a pesquisa de campo ter maior profundidade. Enquanto o levantamento procura ser representativo de universo definido e oferecer resultados caracterizados pela precisão estatística, a pesquisa de campo procura o aprofundamento das questões propostas.

Yin (2005) ressalta que existem diferentes estudos de caso e que em geral visam responder as questões quem, onde, como e porque, o fato em questão.

Gil (2002) destaca as vantagens do estudo de campo:

- é desenvolvido no próprio local onde ocorrem os fenômenos, tornando os resultados mais fidedignos.
- não requer equipamentos especiais para a coleta de dados, tornando a pesquisa mais econômica.
- o pesquisador apresenta maior participação, elevando a probabilidade dos sujeitos pesquisados responderem de forma mais confiável.

A parte teórica expõe as principais contribuições a que a pesquisa bibliográfica conduziu. Temas como qualidade, qualidade *de software* e testes de *software* foram explorados visando à obtenção de dados sobre:

- Real utilização dos tipos de teste de software.
- Visão das consultorias sobre a importância desses testes.
- Divisão do tempo das principais fases do ciclo de vida.
- Estruturação das equipes de testes.
- Certificação das empresas e utilização da NBR ISO\IEC 12207.

A Pesquisa baseou-se na aplicação de questionário estruturado com perguntas abertas e fechadas. Tal estruturação permite, ao mesmo tempo em que uma abordagem objetiva, uma eventual exploração de aspectos importantes não previstos na elaboração do questionário. Os questionários oferecem uma forma ágil de coleta de dados.

Foram distribuídos ao todo trinta questionários para profissionais das áreas de consultoria de *software*, para áreas relacionadas direta ou indiretamente a Testes de *software*. Tomou-se o cuidado de escolher profissionais de diferentes empresas de consultoria especializadas na produção de *software* comercial de gestão empresarial, na região metropolitana de São Paulo, uma vez que são os procedimentos destas empresas o foco de estudo. Também foram consultados profissionais do grupo DFTESTES, um grupo virtual do Yahoo para discussão do tema de teste de *software*.

Todos os questionários foram distribuídos por e-mail, mas duas empresas pesquisadas sendo uma do bairro Brooklin em São Paulo e outra de Itú no interior de São Paulo permitiram uma visita técnica na empresa.

Dos trinta questionários enviados, obteve-se quatorze questionários completamente respondidos e, portanto, passíveis de análise. Nos outros dezesseis questionários algumas das perguntas não foram respondidas ou não retornaram até o prazo de 15 dias estipulados para o início da análise dos dados.

Assim para definir a quantidade de consultorias a serem pesquisadas, basei-me em Einsenhart (1988) afirma que não existe um número ideal de casos, mas que entre quatro e dez costuma ter resultados eficientes. O autor afirma que menos de quatro casos dificulta uma avaliação adequada, pois não permite uma contextualização adequada e pode prejudicar a consistência da pesquisa, a não ser que cada caso contenha mini-casos dentro deles. Com mais de dez casos, fica muito difícil lidar com a complexidade e quantidade das informações coletadas. Portanto quatorze ficou próximo ao limite máximo por ele proposto.

Não se pretende aqui, evidentemente, um estudo com pretensões quantitativas, no qual seria pertinente e justificável uma preocupação com tamanho de amostras e com a validade estatística dos resultados. As pretensões metodológicas restringem-se a uma inicial exploração teórico-empírica do problema, definindo-se a pesquisa como qualitativa e exploratória.

2. REFERENCIAL TEÓRICO

2.1 QUALIDADE

Há várias acepções para o termo qualidade, tornando difícil a sua definição exata. Tomando este ou aquele aspecto do produto, várias definições são formuladas, levando autores como Weinberg (1994) a afirmar que a qualidade é relativa e, portanto, a definição também é relativa.

Na mesma direção, Deming (1990) afirma, que a qualidade tem muitas escalas. Numa gráfica, por exemplo, um livro de qualidade é determinado pela legibilidade, papel, tamanho, inexistência de erros. O leitor deseja encontrar no livro cultura e entretenimento, enquanto a editora deseja encontrar nele, boa oportunidade de venda. A avaliação do que é qualidade dependeria, portanto, das escalas de valor determinadas por cada entidade.

Shewhart explica que a maior dificuldade na definição de qualidade está em mensurar as necessidades futuras do cliente, introduzindo-as nos produtos ou serviços que serão consumidos, por um preço que o usuário aceite pagar. (Deming, 1990)

Já para Algarte e Quintanilha (2000) qualidade pode ser definida como a combinação adequada de segurança, desempenho e custo.

Para Chiavenato (2000) (se aproximando de uma definição mais ampla), a qualidade deve adequar a finalidade ao uso, ou a conformidade com as exigências e, sobretudo, equivale atender às necessidades dos clientes internos ou externos.

A ISO define a qualidade como a totalidade de características de uma entidade que lhe confere a capacidade de satisfazer às necessidades explícitas e implícitas. Cabendo ressaltar que qualidades explícitas são aquelas que o produtor de *software* garante, explicitamente, ao cliente; enquanto as qualidades implícitas são aquelas subjetivamente esperadas pelo usuário.

2.1.1 Qualidade de *software*

Adaptando a definição de qualidade da ISO mencionada acima, a NBR 13596 (1996) define qualidade de *software* como sendo *totalidade das características de um*

produto de software, que lhe confere a capacidade de satisfazer as necessidades explícitas e implícitas.

A qualidade de um *software,* dessa forma*,* somente será verificada quando houver:

- Alinhamento entre as especificações do produto e as necessidades/expectativas dos usuários.
- Alinhamento total entre as especificações aprovadas e o produto construído.
- A existência do menor número de erros no produto final.

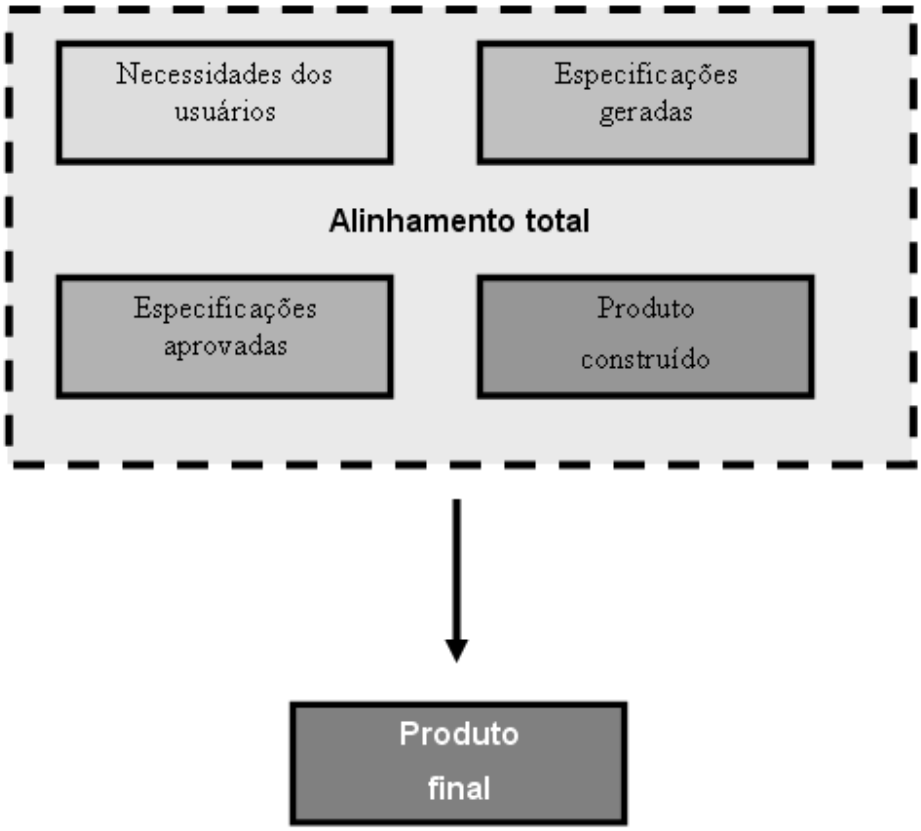
Reafirmando definição dada acima, Pressman (1995) especifica *qualidade de software* "Conformidade a *requisitos funcionais* de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas em todo *software* profissionalmente desenvolvido".

Essa definição contém três pontos importantes:

- Os *requisitos funcionais* de *software* são as bases pelas quais a qualidade do *software* é medida. A ausência de conformidade aos requisitos funcionais caracteriza um produto sem qualidade.
- Padrões especificados definem um conjunto de critérios de desenvolvimento que orientam a maneira segundo a qual o *software* passa pelo trabalho de engenharia.
- Conjunto de requisitos implícitos que freqüentemente não são mencionados, mas sem os quais a qualidade do *software* será comprometida*.*

A **Figura 1** resume o comentado.

Figura 1: Caracterização do qualidade em desenvolvimento de software



Fonte: Inthurn, 2001.

Para que o *software* tenha qualidade e vá ao encontro das necessidades do usuário, é importante que se disseminem os pressupostos de testes de qualidade por todo o ciclo de desenvolvimento de *software*.

Bartié (2002) destaca que qualquer decisão tomada durante o ciclo de desenvolvimento de *software* afetará sua qualidade final. O produto final do desenvolvimento é, exatamente, o somatório de todas as decisões e ações executadas durante todo o ciclo de desenvolvimento.

Daí que, a *qualidade do software* é um aspecto que deve estar presente em todo o ciclo de desenvolvimento e não somente no produto final. Deve-se, portanto, investir em qualidade em todos os pontos do processo de desenvolvimento.

2.1.2 Qualidade do processo de *software*

Segundo Humphrey e Paulk, “Processo é uma seqüência de etapas executadas para realizar um determinado objetivo. O processo de *software* envolve métodos, ferramentas e pessoas”. (Spinola, 2005).

Para um processo funcionar satisfatoriamente, ele deve possuir:

- Procedimentos e métodos que descrevam a relação entre tarefas.
- Ferramentas e equipamentos que deem suporte à realização das tarefas, simplificando e automatizando o trabalho.
- Pessoas com perfis adequados, treinadas nos métodos e nas ferramentas que lhes habilitem a realizar adequadamente os procedimentos.

O processo de *software* (que designa o desenvolvimento de um *software* desde o início) pode servir de modelo para a expansão e modificação de sistemas. Sommerville (2003) define que um processo de *software* “é um conjunto de atividades e resultados associados que levam à produção de um produto de *software*”. Este processo envolve o desenvolvimento de *software* desde o início, sendo possível ser utilizado este modelo para *software* já existentes ou para a expansão e a modificação de sistemas.

As normas existem para determinar padrões, que são essenciais para a qualidade dos produtos finais de um processo.

No **Quadro 1,** são apresentadas as principais normas da ISO para a área de desenvolvimento de *software* e seus respectivos propósitos.

Quadro 1: Principais normas da ISO para a área de desenvolvimento de *software*

Norma	Propósito
ISO/IEC 12207	Processo de *ciclo de vida* de *software*
ISO/IEC 12119:1994	Pacotes de *software* – Requisitos de qualidade e teste
ISO/IEC 14598-1:1999	Avaliação de qualidade de produtos de *software*
ISO/IEC 9126-1:2001	Modelo de qualidade – Características
ISO/IEC 25000:2005	Modelo de qualidade de *software*, nova versão das séries 14.598 e 9.126
ISO 9241:1998	Ergonomia de *software*
ISO/IEC 20926:2003	Medida de *software* por ponto de função
ISO/IEC 9000-3:2004	Diretivas para aplicação da ISO 9001 ao *software*
ISO 9001:2000	Requisitos para sistemas de gerenciamento de qualidade (aplicáveis a qualquer empresa, de *software* ou não)

Fonte: Adaptado de KOSCIANSKI (2006).

Os processos de gestão da qualidade abrangem todos os aspectos de construção do produto. As considerações práticas contêm observações quanto a recomendações gerais sobre como transcorre a execução das atividades relacionadas com qualidade. Isto é válido também para os *software*.

Koscianski (2006) subdivide as atividades relacionas com a qualidade de *software* nos seguintes tópicos:

- Requisitos de qualidade de *software* .
- Referentes à “fatores de influência” sobre os requisitos.
- Orçamento para realização; usuários envolvidos; ferramentas e métodos necessários etc.

Além disso, são considerados os aspectos relacionados com a segurança de funcionamento e as conseqüências que as falhas podem causar. Todas as atividades, cujo foco principal é garantir a qualidade de cada fase do processo da engenharia de *software*, devem ser consideradas dentro da dimensão da garantia da qualidade do processo da engenharia de *software*.

2.1.3 Qualidade de produto de *software*

Conforme Bartié (2002), cada organização possui uma abordagem específica para realizar testes nos produtos de *software* gerados durante o ciclo de desenvolvimento.

É comum que exista, nos cronogramas, a indicação da fase específica para os testes. Entretanto, na maioria das vezes, esse teste é substituído por atividades de correção e manutenção do *software*.

Qualidade do produto de *software* tem como principal objetivo garantir a qualidade tecnológica do produto desenvolvido. Todas as atividades que tenham por objetivo estressar as funcionalidades de um sistema informatizado podem ser categorizadas na dimensão da garantia da qualidade tecnológica do produto.

Apesar de bastante empregada nas organizações, a eficiência dessas atividades é assustadoramente baixa, o que as incentiva a substituírem tais atividades pela correção de problemas.

São vários os motivos que levam a constatação desse fato. Os principais pontos são: falta de planejamento das atividades de testes, ausência de testes que validem funcionalidades antigas e a ausência de um processo de automação dos testes e conferências.

Conforme Rocha (2001), a avaliação de produtos de *software* requer planejamento, controle e uso de técnicas de avaliação adequadas, descreve um método, o *Software Quality in Development* (SQUID), no qual a organização de desenvolvimento de *software* pode usar medições para planejar e controlar a qualidade do produto durante sua elaboração; avaliar a qualidade do produto final e aprender sobre o processo de *software* e sobre o produto Dessa maneira, obtém-se um *feedback* para melhorar o projeto em andamento e os novos projetos em desenvolvimento (Rocha, 2001).

3.1.3.2 Normas e relatórios desenvolvidos para qualidade do produto

Segundo Morselli (2004) para produzir *software* com qualidade, uma das primeiras providências a ser tomada por uma organização é estabelecer um padrão, norma ou modelo de processo de desenvolvimento adequado. Conforme Scalet (2001),

o subcomitê de *software* do comitê técnico juntamente com a ISO e IEC vem trabalhando na elaboração de normas e relatórios técnicos que permitam especificar e avaliar qualidade dos produtos de *software*.

Essas normas e relatórios são definidos pelos seguintes documentos:

1. Qualidade do produto de *software*

- ISO/IEC 9126-1: Modelo de qualidade.
- ISO/IEC 9126-2: Métricas externas.
- ISO/IEC 9126-3: Métricas Internas.
- ISO/IEC 9126-4: Métricas da qualidade em uso.

2. Avaliação de produtos de *software*

- ISO/IEC 14598-1: Visão geral.
- ISO/IEC 14598-2: Planejamento e gestão.
- ISO/IEC 14598-3: Processo para desenvolvedores.
- ISO/IEC 14598-4: Processo para adquirentes.
- ISO/IEC 14598-5: Processo para avaliadores.
- ISO/IEC 14598-6: Documentação de módulos de avaliação.

3. Teste e requisitos de qualidade em produtos de *software*

- NBR ISO/IEC 12119: Pacotes de *software* - teste e requisitos de qualidade.

2.1.4 A busca dos defeitos

Os defeitos são conhecidos por vários nomes: erros, falhas, incidentes, inconsistências, não conformidades, ocorrências ou problemas. Também é comum a utilização de palavras inglesas, tais como: *abends, bugs, crasches*.

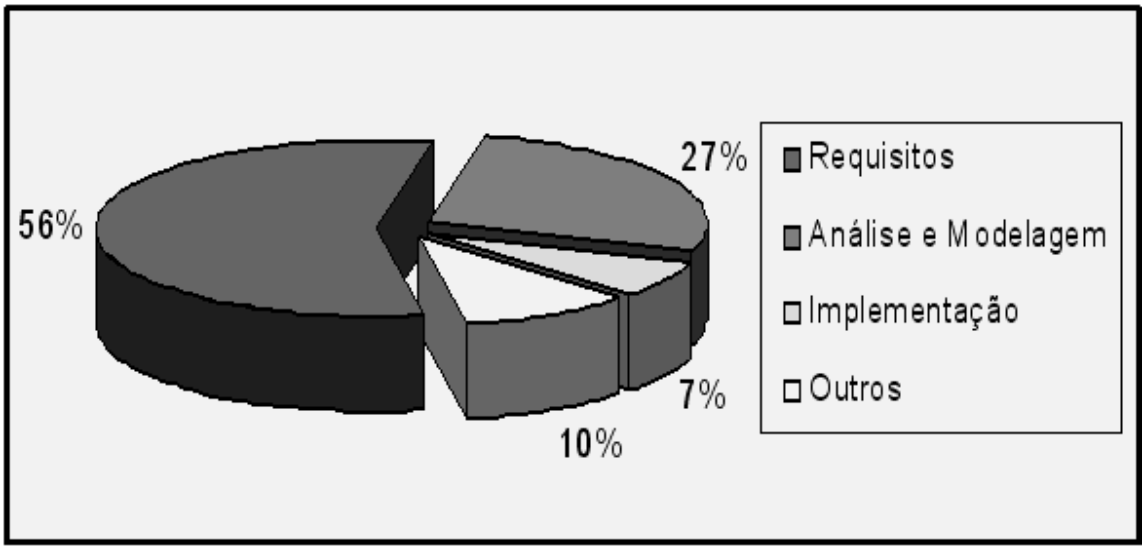
Entretanto, conforme afirma Bartié (2002), apesar destes nomes terem significados diferentes, todos eles demonstram um desvio de qualidade. São esses desvios de qualidade que produzem o retrabalho, aumentando os custos do projeto, dilatando os prazos para a entrega do *software,* diminuindo a produtividade e aumentando a insatisfação do cliente.

Conforme Molinari (2003), existem quatro situações evitadas pela indústria de *software,* caso uma ou mais dessas situações ocorrerem, se caracteriza um defeito:

- *Software* não faz algo que a especificação do produto diz que faz.
- *Software* faz algo que na especificação diz que não é para fazer.
- *Software* faz algo que a especificação do produto não menciona.
- *Software* é difícil de entender, difícil de usar, lento, ou simplesmente aos olhos do testador o usuário não gostará.

Segundo Bartié (2002), há uma visão muito comum de que os defeitos somente são provenientes do código-fonte do produto e que somente os profissionais de desenvolvimento, qualidade e testes são responsáveis por um *software* sem defeitos. Essa visão está equivocada, os erros podem ser gerados durante *todo* o processo de engenharia e desenvolvimento do *software* (por exemplo, são resultados de traduções imperfeitas de requisitos). A participação das diversas áreas reduz drasticamente o risco do insucesso do projeto. Isto significa que, as áreas de desenvolvimento e qualidade deverão interagir continuamente com todas as outras áreas.

Gráfico 1: Incidência de defeitos nas fases de desenvolvimento de um software



Fonte: Bartié, 2002.

Observa-se no **Gráfico 1** que os requisitos correspondem à maioria dos erros, sendo seguidos pela análise e modelagem.

Molinari (2003), afirma que a resposta é a falta de comunicação. Se a funcionalidade desejada não é especificada ou comunicada de forma correta, seria como especificar algo incerto ou até mesmo incorreto.

2.2 *Software*

Software designa programas de computação que existem na forma de conceitos, idéias, planos e modelos. Podem ser representados pela lógica do processamento na forma de conjuntos de instruções organizadas, reconhecidas e executadas pelo computador, com a finalidade de realizar ações do usuário. (SANTOS, 2003)

Software pode ser definido como programas de computador, procedimentos e possivelmente documentos e dados associados pertinentes à operação de um sistema de computador. (Peter, Pedrycz, 2001).

Sommerville (2003) afirma que *software* não é apenas o programa de computador, mas também toda a documentação associada e os dados de configuração necessários para que os programas operem corretamente.

A norma NBRISO/IEC 12207 define que *software* é uma parte fundamental da tecnologia de informação e de sistemas convencionais, tais como sistemas de transporte, militares, da área médica e financeira. (ABNT, 1998)

Pressman (2006) conceitua o *software* como:

> (1) Instruções (programas de computador) que, quando executadas, produzem a função e o desempenho desejados; (2) Estruturas de dados que possibilitam que os programas manipulem adequadamente a informação; (3) Documentos que descrevem a operação e o uso dos programas.

Outra definição de *software,* de seus componentes e processos é apresentada em normas de gestão da qualidade e garantia da qualidade. Segundo a norma NBR ISO 9000-3 (1993), que é uma interpretação da norma de garantia de qualidade ISO 9001 para aplicação aos produtos de *software*, as seguintes definições são apresentadas:

- Software: Criação intelectual compreendendo os programas, procedimentos, regras e qualquer documentação correlata à operação de um sistema de processamento de dados.

- Produto de software: Conjunto completo de programas de computador, procedimentos e documentação correlata, assim como dados designados para entrega a um usuário.
- Item de software: Qualquer parte identificável de um produto de software em etapa intermediária ou na etapa final de desenvolvimento.
- Desenvolvimento: Todas as atividades a serem realizadas para a criação de um produto de software.
- Fase: Segmento definido do trabalho.

O *software*, segundo Pressman (2006), é um elemento de sistema lógico e não físico. Desta forma, tem características diferentes do *hardware.* Segundo o autor, produto de *software* são programas desenvolvidos para serem vendidos a um ou mais clientes.

2.3 CMMI

O modelo CMMI (*Capability Maturity Model Integration*) foi desenvolvido pelo *Software Engineering Institute* (SEI), com o objetivo de melhoria nos processos de *software* através da avaliação das áreas-chaves do desenvolvimento de *software* para otimizar e garantir a qualidade em todo o projeto. Este modelo exige, mais que melhoria dos processos e procedimentos envolvidos, a melhoria nas etapas de desenvolvimento e execução.

O CMMI é um *framework* focado no processo de desenvolvimento de *software*. Foi desenvolvido através de observações das melhores práticas nas organizações de *software* e reflete uma coletânea de experiências e expectativas de muitas delas. Embora este modelo seja completo e abrangente, segundo o SEI / CMU - *Carnegie Mellon University*, sua preocupação está em descrever o quê uma organização em certo nível de maturidade deve praticar. O CMMI não orienta como a empresa deve implementar suas áreas de processo para atingir os níveis de excelência propostos pelo modelo. (Agnol e Herbert, 2004)

O objetivo do CMMI “é servir de guia de melhoria de processos na organização e também da habilidade dos profissionais em gerenciar o desenvolvimento, aquisição e manutenção de produtos ou serviços”.(Koscianski e Soares, 2006).

Este modelo pode ser representado de duas maneiras, por estágios e contínua. Por estágios, conforme o

Quadro 2, inicia com práticas básicas e a evolução dos níveis é gradual. Já "na representação contínua, é possível selecionar a seqüência de melhorias que convém aos objetivos dos negócios da organização e que diminui os riscos".(Koscianski e Soares, 2006), conforme mostra o Quadro 3.

Quadro 2 - Representação por estágios – adaptado de KOSCIANSKI e SOARES (2006)

Nível	Áreas do processo CMMI por estágios
Nível 1 Processos caóticos	▪ Sem subdivisões
Nível 2 Gerenciado	▪ Gerência de requisito ▪ Planejamento do projeto ▪ Gerência e controle do projeto ▪ Gerência de acordos com fornecedores ▪ Medição e análise ▪ Garantia da qualidade do processo e do produto ▪ Gerência de configuração
Nível 3 Definido	▪ Desenvolvimento de requisito ▪ Solução técnica ▪ Verificação ▪ Validação ▪ Foco no processo organizacional
Nível 4 Gerenciado quantitativamente	▪ Desempenho do processo organizacional ▪ Gerência quantitativa do projeto
Nível 5 Otimizado	▪ Inovação e implementação na organização ▪ Análise e resolução de causas

Quadro 3: Áreas de processo na representação contínua

Gerência de Processos	▪ Foco no processo ▪ Definição de processos ▪ Treinamento ▪ Desempenho de processo ▪ Inovação e implementação
Gerência do Projeto	▪ Planejamento de projeto ▪ Controle e monitoramento de projeto ▪ Gerência de acordos com fornecedores ▪ Gerência de projeto integrada ▪ Gerência de riscos ▪ Integração de equipe ▪ Integração de fornecedores ▪ Gerência quantitativa de projeto
Engenharia	▪ Gerência de Requisito ▪ Gerência de desenvolvimento ▪ Solução Técnica ▪ Integração de produto ▪ Verificação ▪ Validação
Suporte	▪ Gerência de configuração ▪ Garantia da qualidade de produto e processo ▪ Medida e análise ▪ Análise organizacional para integração ▪ Resolução e análise de causas

Fonte: adaptado de KOSCIANSKI e SOARES (2006)

Os níveis de capacitação da organização, de acordo com o CMMI estagiado ou contínuo, são descritos no quadro a seguir:

Quadro 4: Níveis de capacitação do CMMI, contínuo e estagiado

Contínua	Estagiada
Nível 0 – Incompleto	
Nível 1 – Realizado	Nível 1 - Processos caóticos
Nível 2 – Gerenciado	Nível 2 – Gerenciado
Nível 3 – Definido	Nível 3 – Definido
Nível 4 – Gerenciado Quantitativamente	Nível 4 - Gerenciado quantitativamente
Nível 5 – Otimizado	Nível 5 – Otimizado

Fonte: adaptado de KOSCIANSKI e SOARES (2006)

2.4 Engenharia de *software*

O conceito de engenharia de *software* não é recente. A primeira utilização do termo, segundo Pressman (2006), foi em 1968, na Alemanha, em uma conferência dedicada ao assunto para discutir a crise do *software*. Nesta conferência, proposta por Friz Bauer, o objetivo era o de estabelecer e colocar em prática os princípios de

engenharia para obter economicamente *software* confiável e funcional para máquinas reais. (Pressman, 2006)

Koscianski e Soares (2006) ressaltam que esta conferência foi realizada por uma entidade que, a rigor, não possuía nenhuma ligação com a área: o Comitê de Ciência da NATO (*North Atlantic Treaty Organisation* – Organização do Tratado do Atlântico Norte).

> "Curiosamente já havia instituições relacionadas com informática: a primeira delas foi a ACM (*Association for Computing Machinery*), criada em 1947 e que edita uma revista científica, *Communications of the ACM*, desde 1957."
>
> (KOSCIANSKI e SOARES, 2006)

Pressman (2006) define a engenharia de *software* como um rebento da engenharia de sistemas e de *hardware*, abrangendo três elementos fundamentais:

- Métodos.
- Ferramentas.
- Procedimentos.

Pressman (2006) afirma que estes elementos possibilitam aos gerentes o controle do processo de desenvolvimento do *software,* oferecendo aos profissionais uma base para a construção de *software* de alta qualidade.

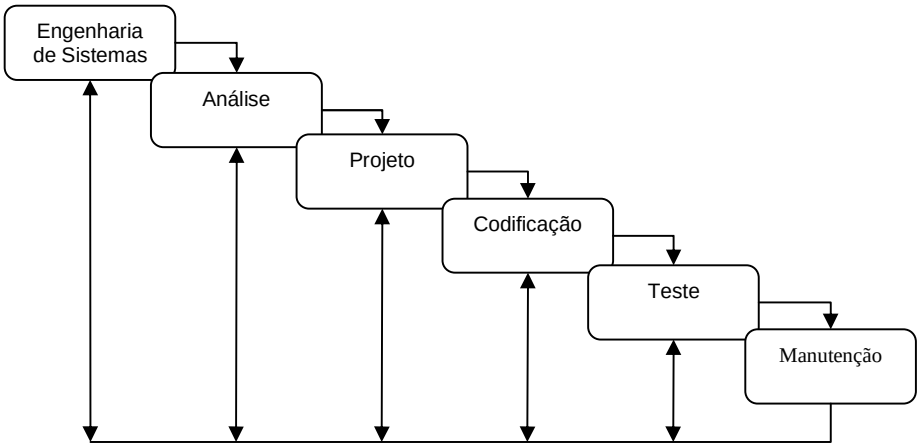
Neste conceito, métodos referem-se aos detalhes de como fazer para construir o *software.* Nesta etapa é definido o planejamento e a estimativa do projeto, análise de requisitos de *software* e de sistemas, o projeto da estrutura de dados, a arquitetura dos programas e algoritmo de processamento, os teste e a manutenção. As ferramentas proporcionam a automatização de todas estas etapas. Os procedimentos são os elos que mantêm juntos os métodos e as ferramentas e possibilitam o desenvolvimento do *software*. (Pressman, 2006).

Outro autor conceituado do tema engenharia de *software,* Sommerville (2003), define a engenharia de *software* como a disciplina da engenharia que se ocupa de todos os aspectos de produção de *software*, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema.

O ciclo de vida clássico (Pressman, 2006; Sommerville, 2003) ilustra este conceito, conforme mostrado na Figura 2. É também chamado de modelo cascata,

onde as etapas são seqüenciais. Nota-se que a manutenção é considerada a fase final do ciclo.

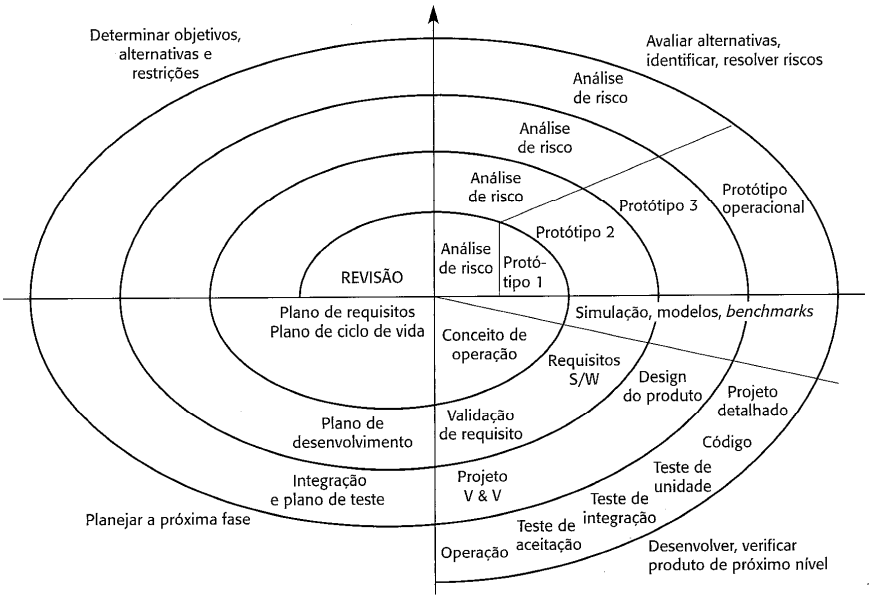
Figura 2: ciclo de vida clássico



Fonte: PRESSMAN, 2006; SOMMERVILLE 2003

Outra forma de desenvolvimento de *software* é a espiral, originalmente proposta por Boehm (1988), citado pelos autores Sommerville (2003) e Pressman (1995). Neste conceito o desenvolvimento é subdivido em etapas e cada etapa é reavaliada para ajustes e correções, como mostra a Figura 3.

Figura 3: Modelo Espiral - Boehm



Fonte: SOMMERVILLE (2003)

Neste modelo são consideradas etapas finais os teste de integração, aceitação e operação, ou seja o *software* em produção é validado e conferido. Nesta etapa final está o treinamento do usuário e a utilização da aplicação pelos envolvidos.

Pressman (2006) acredita que o modelo espiral para a engenharia de *software* atualmente é a abordagem mais adequada para o desenvolvimento de sistemas e de *software* em grande escala. Este modelo aborda a engenharia de *software* de forma evolucionária, capacitando o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa evolutiva. O modelo espiral usa a prototipação como um mecanismo de redução dos riscos, mas o é mais importante é possibilitar que o desenvolvedor aplique a abordagem de prototipação em qualquer etapa de evolução do produto. O modelo espiral mantém a abordagem de passos sistemáticos sugeridos pelo ciclo de vida clássico, mas incorporar uma estrutura iterativa que reflete o mundo real.

2.5 Ciclo de vida de *software*

Segundo a ISO 12207 (1998), um *software* possui um *ciclo de vida* relativamente curto. Caso ele não venha a sofrer implementações, ou seja, correções e melhorias, ele terá cerca de cinco anos.

2.5.1 Tecnologia da Informação - Processos de *ciclo de vida* de *software* NBR ISO\IEC 12207

A globalização da economia, internacionalizando o mercado, ampliou enormemente a competição, forçando as empresas produtoras e prestadoras de serviços de software a alcançar o patamar de qualidade e produtividade internacional. A norma internacional *NBR ISO/IEC 12207:1998 - Tecnologia da Informação - Processos de Ciclo de vida de Software* é usada como referência em muitos países, inclusive no Brasil, para alcançar esse diferencial competitivo. Machado (2001).

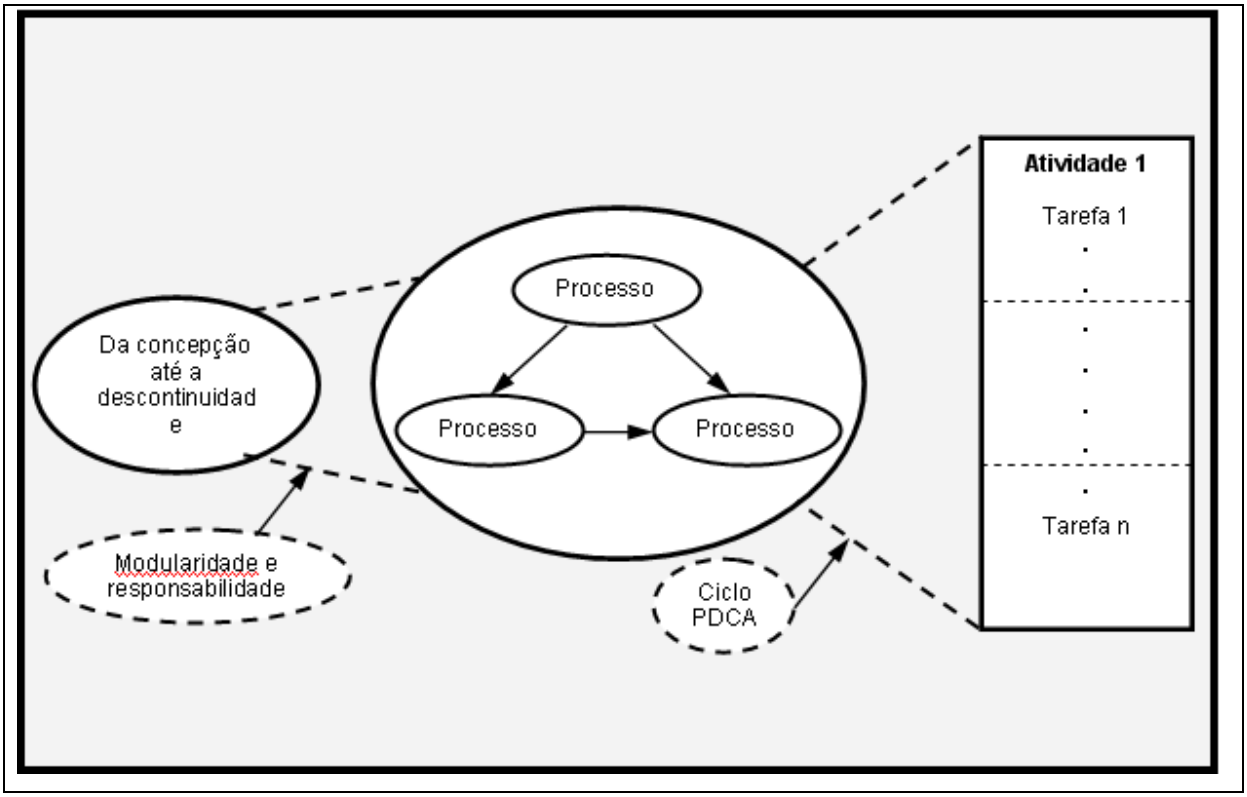
Ela auxilia os profissionais envolvidos na produção de *software* a definir, por meio de estabelecimento de processos bem definidos, a definir seus papéis num processo como todo. Dessa forma, a norma citada proporciona às organizações que a utilizam um melhor entendimento das atividades a serem executadas nas operações que envolvem, de alguma forma, o software.(Rocha *et al*., 2001).

Conforme Machado (2001), a arquitetura descrita na *NBR ISO/IEC 12207* é composta de processos, atividades e tarefas para aquisição, fornecimento, operação e desenvolvimento do *software*. A norma estabelece uma arquitetura que abrange desde a concepção até a descontinuidade do *software*. Essa arquitetura, baseada em processos-chave e no inter-relacionamento entre eles, segue dois princípios básicos:

Modularidade: Os processos tem alta coesão e baixo acoplamento, ou seja, todas as partes de um processo são fortemente relacionadas e o número de interfaces entre os processos é mantido ao mínimo.

Responsabilidade: Segundo a norma, cada processo está sob a responsabilidade de uma “parte envolvida”, que pode ser uma organização ou parte dela. As partes envolvidas podem ser da mesma organização ou de organizações diferentes.

Figura 4: Arquitetura da NBR ISO/IEC 12207

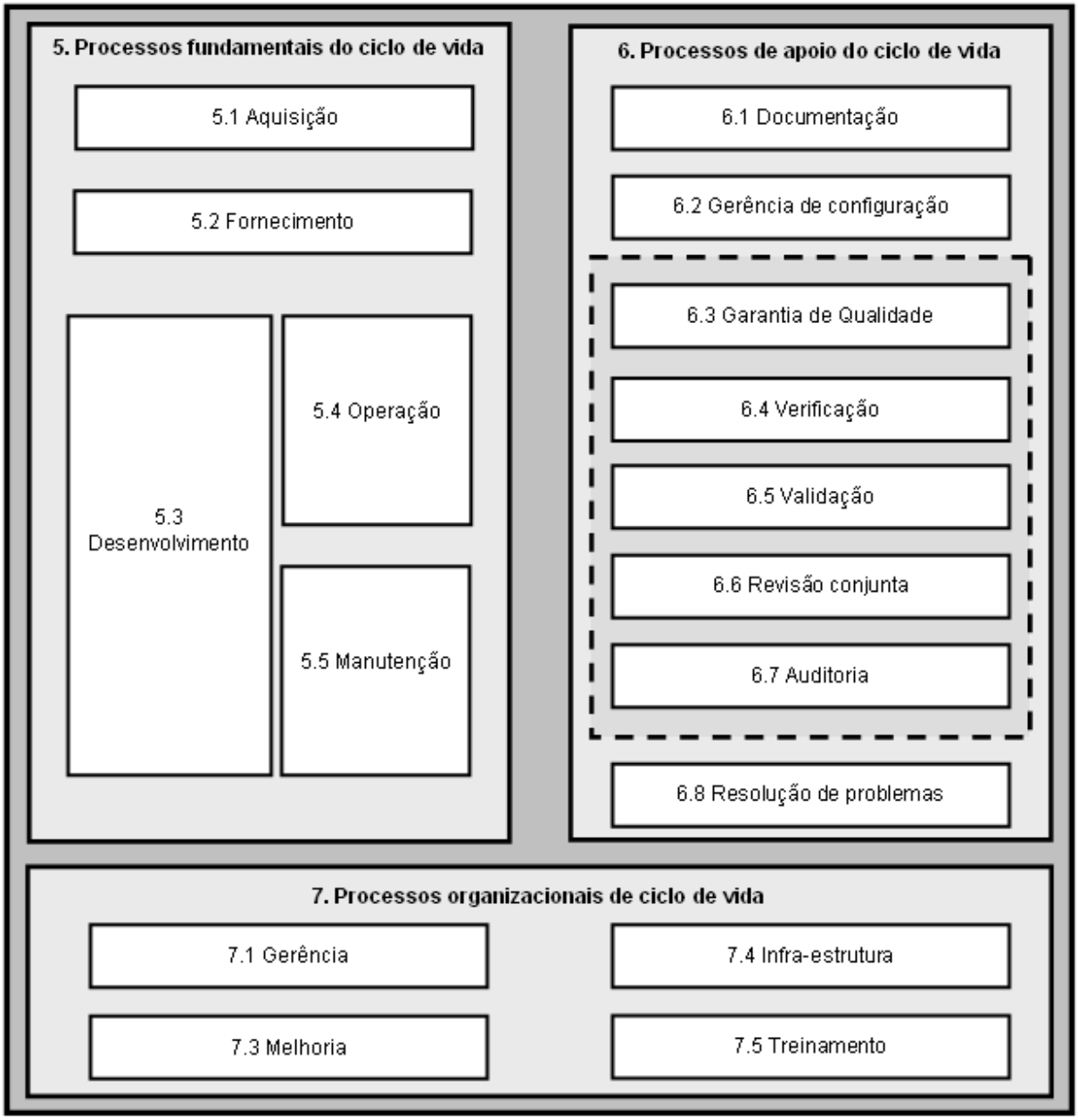


Fonte: Rocha, 2001

2.5.2 Processos de ciclo de vida de software

Esta norma agrupa as atividades que podem ser executadas durante o *ciclo de vida* do *software* em cinco ”processos fundamentais“, oito “processos de apoio” e quatro “processos organizacionais”. Cada processo do *ciclo de vida* é dividido em um conjunto de atividades. Cada atividade é então dividida em um conjunto de tarefas, como exibido na Figura 5. (ISO/IEC 12207,1998).

Figura 5: Estrutura da norma ISO/IEC 12207:1998



Fonte: ISO/IEC 12207, 1998.

2.5.2.1 Processos fundamentais

Segundo Machado (2001), os *processos fundamentais* atendem no início, à contratação entre o adquirente e o fornecedor e à execução do desenvolvimento da operação ou da manutenção de produtos de *software* durante o *ciclo de vida*.

Segundo a norma ISO/IEC 12207(1998), os *processos fundamentais* são de:

- Aquisição: Contém as atividades do adquirente. Inicia-se com a definição da necessidade de adquirir um sistema um produto ou serviço de software, e continua com a preparação e emissão de pedido de proposta, seleção do fornecedor e gerência do processo de aquisição por meio da aceitação do sistema, produto de software ou serviço de software.
- Fornecimento: Contém as atividades do fornecedor. Inicia-se pela decisão de preparar uma proposta e/ou assinar um contrato com o adquirente para fornecer o sistema, produto de *software* ou serviço de *software*. Neste processo há a determinação dos recursos necessários para gerenciar e garantir o projeto, até a entrega do sistema, produto de *software* ou serviço de *software* para o adquirente.
- Desenvolvimento: Contém as atividades do desenvolvedor. Incluem as atividades de análise de requisitos, projeto, codificação, integração testes, instalação e aceitação relacionadas aos produtos de software. Processo de operação: Contém as atividades do operador. O processo cobre a operação do produto de software e o suporte operacional aos usuários.
- Operação: Contém as atividades do operador. O processo cobre a operação do produtode *software* e o suporte operacional aos usuários.
- Manutenção: Contém as atividades do mantenedor. O objetivo é fazer alterações em um software existente, preservando sua integridade e de sua documentação.

2.5.2.2 Processos de apoio

Conforme Machado (2001), os *processos de apoio* auxiliam e contribuem para o sucesso e a qualidade do projeto de *software, e serão descritos com ênfase mais adiante*. Segundo a norma ISO/IEC 12207(1998), os *processos de apoio* são de:

- Documentação: Registros de informações produzidas por um processo ou atividade do *ciclo de vida*. Esse processo contém um conjunto de atividades que planeja, projeta, desenvolve, produz, edita, distribui e mantém aqueles documentos necessários a todos os interessados.

- Gerência de configuração: Destinado a identificar e definir os itens do *software*; controlar as modificações e liberações dos itens; registrar e apresentar a situação dos itens e dos pedidos de modificação; garantir a completeza à consistência e a correção dos itens; e controlar o armazenamento a manipulação e a distribuição dos itens de *software*.
- Garantia de qualidade: Garante que os produtos de *software* estejam em conformidade com os requisitos e planos estabelecidos.
- Verificação: Determina se os produtos de *software* de uma atividade atendem os requisitos ou condições impostas a eles nas atividades anteriores.
- Validação: Determina se os requisitos e o produto final atendem ao uso específico pretendido.
- Revisão conjunta: Define as atividades para avaliar a situação e os produtos de uma atividade de um projeto.
- Auditoria: Determina a adequação do produto aos requisitos, aos planos e ao contrato quando apropriado.
- Resolução de problemas: Define um processo para analisar e resolver os problemas (incluindo as não-conformidades) de qualquer natureza ou fonte detectados durante o desenvolvimento, a operação, a manutenção ou a realização de outros processos.

2.5.2.3 Processos organizacionais

São empregados por uma organização para estabelecer e implementar uma estrutura constituída pelos processos do *ciclo de vida* e pelo pessoal envolvido no desenvolvimento do *software*. Eles são geralmente empregados fora do domínio de projetos e contratos específicos; entretanto, os ensinamentos desses projetos e contratos contribuem para a melhoria da organização. Machado (2001).Segundo a norma ISO/IEC 12207(1998), os *processos organizacionais* são de:

- Gerência: Define as atividades básicas da gerência, incluindo gerência de projeto, durante um processo do *ciclo de vida*.

- Melhoria: Define as atividades básicas que uma organização executa para estabelecer, medir, controlar e melhorar o seu processo de *ciclo de vida*.
- Infraestrutura: Define as atividades básicas para o estabelecimento da estrutura de apoio de um processo de *ciclo de vida*.
- Treinamento: Define as atividades para promover pessoal adequadamente treinado.

2.5.3 Relatórios técnicos complementares da Norma ISO/IEC 12207

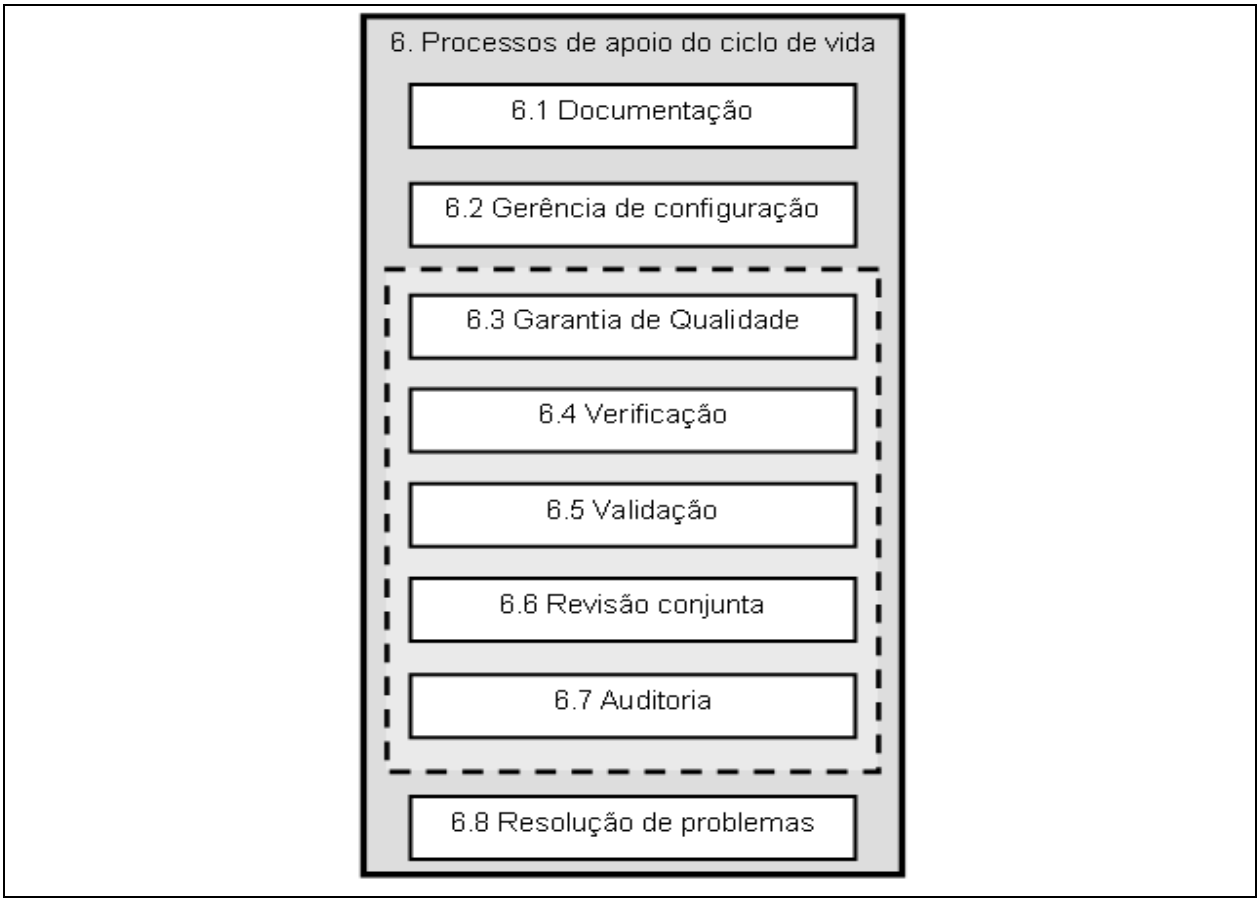
Como demonstra Machado (2001), a ISO/IEC elaborou um conjunto de guias, padrões e relatórios técnicos para apoiar a aplicação da norma ISO/IEC 12207:

- ISO/IEC TR 15271 - Guia para ISO/IEC 12207.
- ISO/IEC 14764 - Norma de manutenção de *software*.
- ISO/IEC TR 14759 - Relatório técnico – Maquete.
- ISO/IEC TR 15846 - Relatório técnico - Gerência de Configuração.
- ISO/IEC TR 16326 - Relatório técnico - Gerência de Projetos.

2.5.4 Principais processos que envolvem testes de *Software*

A qualidade é requerida em todo o *ciclo de vida* do *software*. A norma ISO/IEC 12207 (1998) apresenta os processos e as atividades que necessárias para que se atinja tal objetivo. Segundo esta norma, as atividades e as tarefas do processo de apoio são de inteira responsabilidade da organização que as executa. “Revisões conjuntas, auditorias, verificação e validação podem ser utilizadas como técnicas de garantia de qualidade”. ISO/IEC 12207 (1998).

Figura 6: Processos de apoio do ciclo de vida de *software*



.Fonte: Adaptado da norma ISO/IEC 12207, 1988.

2.5.4.1 Processo de garantia de qualidade

O processo de garantia da qualidade da norma ISO/IEC 12207 serve para garantir que os processos e produtos de software, no ciclo de vida do projeto, estejam em conformidade com os requisitos especificados e referentes aos planos estabelecidos.

As atividades descritas a seguir fazem parte do processo de garantia de qualidade.

1) Atividades de garantia do produto:

- Todos os planos exigidos pelo contrato estejam de acordo com o mesmo, sejam documentados, estejam mutuamente consistentes e

sejam executados quando exigidos. Que os produtos de *software* e sua documentação estejam de acordo com o contrato e os planos.

- Os produtos de *software* satisfaçam totalmente os requisitos contratuais e sejam aceitáveis pelo adquirente.

2) Atividades de garantia do processo:

- Os processos do *ciclo de vida* do *software*, utilizados no projeto estejam de acordo com o contrato e com planos.
- As práticas de engenharia de *software*, os ambientes de desenvolvimento e de teste, e as bibliotecas estejam de acordo com o estabelecido no contrato.
- Em casos de subcontratação, os requisitos aplicáveis sejam passados ao subcontratado e que seus produtos de *software* satisfaçam os requisitos do contrato original.
- O adquirente e outras partes envolvidas tenham o apoio e a cooperação necessários.
- As medições do produto e do processo de *software* estejam de acordo com os padrões e procedimentos estabelecidos;
- A equipe tenha a qualificação e o conhecimento necessários para o projeto e receba todo o treinamento necessário.

3) Atividades de Implementação do processo:

Conforme Rocha (2001), para cada projeto de desenvolvimento de software, deve-se definir um processo de garantia da qualidade adequado às necessidades em questão.

Os objetivos do processo devem ser determinados de maneira a assegurar que os produtos e processos de software estejam em conformidade com os requisitos e planos estabelecidos. Também é necessário definir um Plano de Garantia da Qualidade que deve ser documentado implementado, mantido e incluir:

- Padrões de qualidade, metodologias, procedimentos e ferramentas para a execução das atividades de garantia da qualidade.

- Procedimentos para revisão do contrato e sua coordenação.
- Procedimentos para identificação, coleta, arquivamento, manutenção e disponibilização dos requisitos de qualidade.
- Recursos, cronograma e responsabilidades para conduzir as atividades de garantia da qualidade.
- Atividades e tarefas selecionadas dos processos de apoio, tais como verificação, validação, revisão conjunta, auditoria e resolução de problemas.

4) Atividades adicionais de gerência da qualidade

- Segundo a norma ISO/IEC 12207 (1998), as atividades adicionais de gerência da qualidade devem ser garantidas de acordo com a série de normas ISO 9000 e conforme especificado no contrato.

2.5.4.2 Processo de verificação

Segundo a ISO/IEC 12207 (1998), a verificação “é um processo que visa determinar se os produtos de software de uma atividade atendem aos requisitos ou condições impostas a eles nas atividades anteriores”.

O processo de verificação consiste nas atividades descritas a seguir:

1) Atividades de verificação envolve:

- O contrato.
- O processo.
- Os requisitos.
- O Projeto.
- O código.
- A integração.
- A documentação.

2) Atividades de implementação do processo de verificação:

- Determinar se o projeto justifica um esforço de verificação e o grau de independência organizacional.
- Um processo de verificação deve ser estabelecido para verificar o produto de software.
- Na seleção de uma organização qualificada e responsável para conduzir a verificação.
- Na determinação das atividades do ciclo de vida e dos produtos que necessitam de verificação.
- Num plano de verificação devidamente desenvolvido e documentado, baseado nas tarefas de verificação determinadas anteriormente.
- Que o plano de verificação deve ser implementado. ISO/IEC 12207 (1998)

2.5.4.3 Processo de validação

Conforme a norma ISO/IEC 12207 (1998), o processo de validação é um processo que visa determinar se os requisitos e o produto final, sistema ou produto de *software* construído atendem ao uso específico pretendido.

O processo de verificação consiste nas atividades descritas a seguir:

1) Atividades de validação do processo:

- Preparar os requisitos, casos e especificações de testes selecionados para a análise dos resultados dos testes.
- Assegurar que esses requisitos, casos e especificações de testes reflitam os requisitos particulares para cada uso específico.
- Conduzir os testes incluindo os de estresse, do produto de software e teste de usuário.Validar que o produto de software satisfaça seu uso pretendido.
- Testar o produto de software, quando apropriado, nas áreas selecionadas do ambiente-alvo.

2) Atividade de implementação do processo de validação:

- Determinado se o projeto justifica o esforço de validação e o grau de independência organizacional.
- Estabelecido para validar o sistema ou produto de software.
- Selecionada uma organização qualificada e responsável para conduzir o processo de validação.
- Desenvolvido um plano de validação.
- O plano de validação deve ser implementado.

2.5.4.4 Processo de revisão conjunta

O processo de revisão conjunta é um processo para avaliar se a situação e produtos de uma atividade de um projeto é apropriada. As revisões conjuntas são feitas tanto nos níveis de gerenciamento do projeto como nos níveis técnicos e são executados na vigência do projeto (ISO/IEC 12207, 1998).

O processo de revisão conjunta consiste nas atividades descritas a seguir:

1) Atividades de revisão conjunta:

- Revisões periódicas devam ser promovidas em marcos predeterminados, como especificado nos planos do projeto.Todos os recursos requeridos para conduzir as revisões devam ser acordados pelas partes.
- Todos os problemas detectados durante as revisões devam ser registrados e incluídos no processo de resolução de problemas.Os resultados de revisão devam ser documentados;
- As partes devem concordar pelos resultados da revisão.

Segundo a ISO/IEC 12207 (1998), a situação do projeto deve ser avaliada em relação aos planos, cronogramas, padrões e diretrizes aplicados ao projeto. O resultado da revisão deveria ser discutido entre as duas partes e deveria fornecer subsídios para:

- Conduzir a progressão das atividades de acordo com o plano, baseado em uma avaliação da situação da atividade ou do produto de *software*.
- Manter o controle geral do projeto mediante alocação adequada de recursos.
- Redirecionar o projeto ou determinar a necessidade de um planejamento alternativo.
- Avaliar e gerenciar as situações de risco que possam comprometer o sucesso do projeto.

Conforme ISO/IEC 12207 (1998), as revisões técnicas devem ser promovidas para avaliar os produtos ou serviços de *software* em consideração e prover evidência de que eles estão completos, aderentes aos seus padrões e especificações, e que suas alterações estão implementadas adequadamente, tendo afetado somente as áreas identificadas pelo processo de gerência da configuração.

2.5.4.5 Processo de auditoria

Segundo a ISO/IEC 12207 (1998), o processo de auditoria é um processo para determinar adequação aos requisitos, planos e contrato, quando apropriado.

1) Atividades de auditorias devem ser conduzidas para assegurar que:

- Produtos de *software* codificados sejam o reflexo da documentação do projeto.
- A revisão de aceitação e requisitos de testes prescritos pela documentação esteja adequada para a aceitação dos produtos de *software.*
- Dados de teste estejam aderentes às especificações.
- Os produtos de *software* sejam testados com sucesso e atendam às suas especificações.
- Os relatórios de testes estejam corretos.
- A documentação do usuário esteja aderente aos padrões.

- As atividades sejam conduzidas de acordo com os requisitos, planos e contratos aplicáveis.
- Os custos e cronogramas tenham sido cumpridos.

2) Atividades de implementação do processo de auditoria:

- As auditorias devam ser promovidas em marcos predeterminadas, conforme especificado no plano do projeto.
- O pessoal da auditoria não deva ter nenhuma responsabilidade direta pelos produtos de software e atividades que auditam;
- Todos os recursos requeridos para a auditoria devem ser acordados pelas partes.
- Problemas detectados durante as auditorias devam ser registrados e incluídos no processo de resolução de problemas.
- Após a conclusão de uma auditoria, os resultados devam ser documentados e entregues à parte auditada;
- As partes devam concordar com o resultado da auditoria.

2.5.4.6 Processo de resolução de problemas

Resolução de problemas

Segundo a ISO/IEC 12207 (1998), o processo de resolução de problemas é um processo que analisa e resolve os problemas (incluindo não-conformidades) de qualquer natureza ou fonte, que são descobertos durante a execução do desenvolvimento, operação, manutenção ou outros processos. O objetivo é prover os meios, em tempo adequado, e de forma responsável e documentada para garantir que todos os problemas encontrados sejam analisados e resolvidos e tendências sejam identificadas.

O processo de resolução de problemas consiste nas atividades descritas a seguir.

1) Atividades de implementação do processo de resolução de problemas:

- Ser de ciclo fechado.
- Conter um esquema para categorizar e priorizar os problemas.
- A análise deve ser executada para detectar tendências nos problemas relatados.
- As resoluções de problemas e suas aplicações devem ser avaliadas, para garantir que tudo sairá como o desejado.

3. TESTES DE SOFTWARE

3.1 Histórico

No início do desenvolvimento dos *software,* a atividade de testes era encarada como uma simples tarefa de navegar pelo código e corrigir problemas já conhecidos. Estas tarefas eram realizadas pelos próprios desenvolvedores, e não havia dedicação à atividade de testes, que era realizada *tardiamente*, somente quando o produto estava quase pronto. Bartié (2002)

Na computação, o termo "*bug"* (inseto, em inglês), foi introduzido em setembro de 1947, por uma das pioneiras da computação, Grace Murray Hopper. Ela operava um computador baseado em relés, o Mark II, da Universidade de Harvard, quando notou um comportamento inesperado. (Teixeira, 2007).

Ao verificar encontram um inseto preso no relé de número 70 do painel F. Por isso, no livro de registro, fixaram com uma fita a causa do problema e o incidente ocorrido. Entretanto, a invenção do termo "*bug*" pode ser considerada mais antiga, pois este termo já era aplicado durante a II Guerra Mundial em problemas eletrônicos nos radares. E a utilização do termo pode ser considerada ainda mais antiga, pois já indicava problemas de defeito industrial desde os tempos de Thomas Edison. (Teixeira, 2007).

Em 1957, o conceito de "teste de *software*" ampliou seus valores e se tornou um "processo de detecção de erros". Mas ainda continuava sendo aplicado erroneamente, ou seja, somente no *final* do processo de desenvolvimento. (Bartié, 2002).

No início da década de 1970, esse processo passou a ter uma abordagem mais profunda com a chegada de "engenharia de *software*"*,* porém havia pouco consenso sobre o que viria a ser "teste de *software.*

Em 1972 foi realizada a primeira conferência formal sobre testes na Universidade da Carolina do Norte. Ocasião em que foi definido que o termo testar

deveria abranger um grande número de atividades, a fim de se garantir a confiança num produto de desempenho esperado.(Ishida,2002).

Em 1979, Myers produziu um dos primeiros trabalhos mais completos e profundos acerca dos ”testes de *software*”. Nesse trabalho, Myers definia testes como “processo de trabalho com a intenção de encontrar erros”.

Myers foi um dos estudiosos precursores do processo de testes. Seus trabalhos tinham como premissa que o objetivo do teste seja apenas provar a boa funcionalidade do aplicativo. O que pode ser considerado como revolução é o fato de que uma vez que um problema era encontrado, os profissionais da qualidade buscavam vários cenários para avaliar o comportamento do *software*. Entretanto, os testes continuavam a ser executados tardiamente. (MYERS apud Bartié, 2002)

No início da década de 80, surgiram os primeiros conceitos de *qualidade de software*. Na abordagem de *qualidade de software* os desenvolvedores e os testadores trabalhavam juntos desde o início do processo de desenvolvimento. Nessa mesma época, foram formados muitos padrões e organizações que utilizamos até hoje. (Bartié, 2002)

Finalmente, na década de 90, as ferramentas de testes começaram a ser produzidas. Estas ferramentas tratariam de alta produtividade e qualidade no processo de teste. (Bartié, 2002).

Na computação, o termo “*bug”* (inseto, em inglês), foi introduzido em setembro de 1947, por uma das pioneiras da computação, Grace Murray Hopper. Ela operava um computador baseado em relés, o Mark II, da Universidade de Harvard, quando notou um comportamento inesperado.

Ao verificar encontram um inseto preso no relé de número 70 do painel F. Por isso, no livro de registro, fixaram com uma fita a causa do problema e o incidente ocorrido. Entretanto, a invenção do termo “*bug*” pode ser considerada mais antiga, pois este termo já era aplicado durante a II Guerra Mundial em problemas eletrônicos nos radares. E a utilização do termo pode ser considerada ainda mais antiga, pois já indicava problemas de defeito industrial desde os tempos de Thomas Edison. (Teixeira, 2007).

3.2 Definições de testes de *software*

Segundo Pressman (1995), foi Myers quem enumerou algumas das características que descrevem o teste, quais sejam:

- 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.
- Um teste bem-sucedido é aquele que descobre um erro ainda não descoberto.

Segundo Bartié (2002), de forma geral, todas as equipes de desenvolvimento aplicam testes em seus *software,* independentemente da qualidade dos mesmos. Porém tais testes não são suficientes para detectar os erros que estão inseridos em uma aplicação. Um dos motivos básicos para que isso ocorra é a forma que estes profissionais encaram a fase de testes.

Abaixo algumas definições de diversas equipes de desenvolvimento:

- Teste é o processo de demonstrar que os defeitos não estão presentes.
- Teste é o processo de demonstrar que algo funciona corretamente.
- Teste é o processo de provar que determinadas coisas fazem o que deveriam fazer.

Muitos enxergam testes como uma forma de provar de que tudo está bem e funcionando conforme o estabelecido. Nas definições mencionadas acima, percebe-se uma visão positiva sobre a questão, qual seja: os testes são compreendidos sob o prisma de que servem para provar que tudo está funcionado bem. Ora, se o objetivo é provar que tudo está funcionando corretamente, os erros não serão percebidos, ou dificilmente serão percebidos.

Percebendo essa falha, Pressman (2006) argumenta que o objetivo fundamental dos testes deve ser o de encontrar erros, sendo que um bom teste é aquele que tem alta probabilidade de encontrar erros, e não apenas de provar que algo funciona.

3.3 Cenário atual do desenvolvimento de software

O cenário de competitividade do mundo globalizado vem sendo observado pelos estudiosos da área de software como o mais importante fator de estímulo à busca de eficiência nos processos internos das empresas. Segundo Bartié: “O que se percebe é que de forma rápida e constante, as organizações estão aumentando a sua dependência tecnológica, isto significa que suas operações internas estão sendo conduzidas e direcionadas por um conjunto cada vez maior de sistemas informatizados”. (Bartié, 2002).

No mundo empresarial, portanto, as organizações estão buscando maior eficiência como uma estratégia de sobrevivência, uma vez que depende desse aprimoramento, a redução de custos e a ampliação de mercado. Observa-se, ainda, que os riscos de mau funcionamento aumentam proporcionalmente à complexidade desse ambiente, tornando-se mais difícil produzir *software* com um “nível de qualidade” desejado. Nos últimos quarenta anos as empresas tiveram um grande avanço no desenvolvimento de *software,* porém muitas empresas estão presas a antigos paradigmas, o que impede seu amadurecimento no processo de desenvolvimento. Não perceberam, ainda, que implantar um processo de garantia da qualidade de software não é uma opção a ser estudada, mas exigência de um mercado cada vez mais exigente e competitivo.

Segundo Bartié (2002), apesar de todas as organizações concordarem em apontar a Tecnologia da Informação como um dos aspectos estratégicos para se ter inovação e melhoria em seus produtos e serviços, percebe-se a ineficiência e o despreparo no processo de desenvolvimento de *software*, ou então, há a necessidade de mudanças para a adaptação à novas tecnologias do mercado.

O mercado está em uma rápida e constante mudança, enquanto as empresas desenvolvedoras de *software* não estão conseguindo acompanhar o este ritmo. O motivo é claro: simplesmente não investiram no aperfeiçoamento de seus processos internos. (Molinari, 2003).

Neste cenário é perceptível a falta de qualidade com que os produtos de *software* estão sendo feitos. Estudos americanos (Standish, 2001) descrevem uma triste realidade para os projetos de desenvolvimento de *software*:

- Mais de 23% dos projetos são cancelados antes de serem finalizados.
- Mais de 41% dos projetos falham nas entregas das funcionalidades esperadas.
- Os custos extrapolam em até 45% os valores originalmente previstos.
- Os prazos excedem em mais de 63% os cronogramas originais.

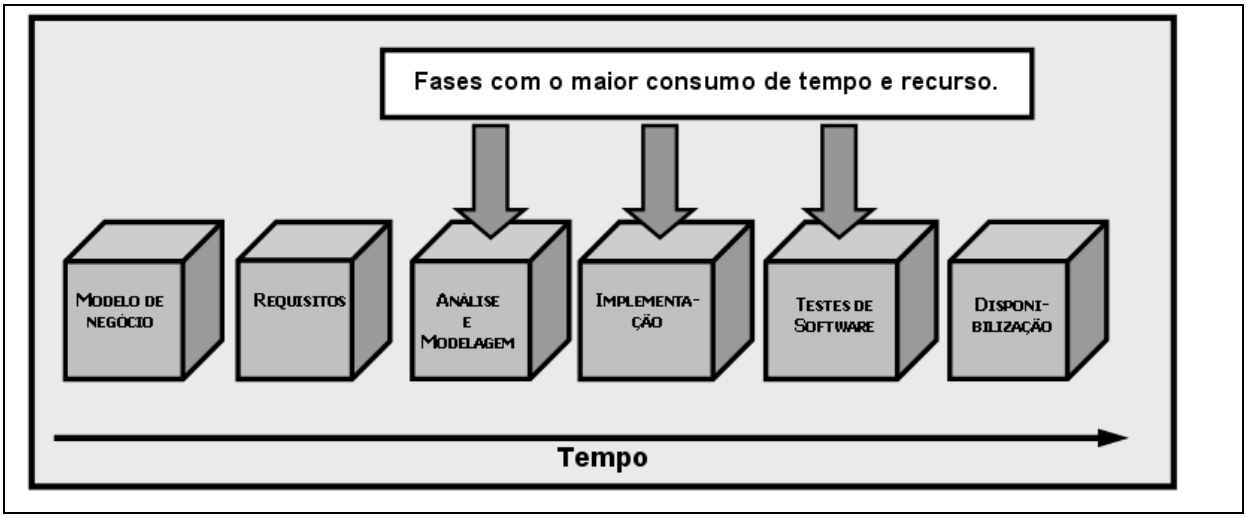
Devido a esses acontecimentos as empresas brasileiras estão contratando empresas internacionais para desenvolver os projetos de tecnologia de *software*. Ou seja, o mercado brasileiro está perdendo espaço para empresas estrangeiras que investem no aperfeiçoamento de seus profissionais e de seus processos de *software*.

3.3.1 A fase de teste no ciclo de vida

Nos projetos de desenvolvimento de *software* é comum se encontrar uma fase dedicada à atividade de teste de *software*. Entretanto, como também já apontamos, geralmente no cronograma do projeto de *software* a etapa de testes é descrita como uma fase posterior à codificação, ou seja, no final, já considerado como produto acabado.

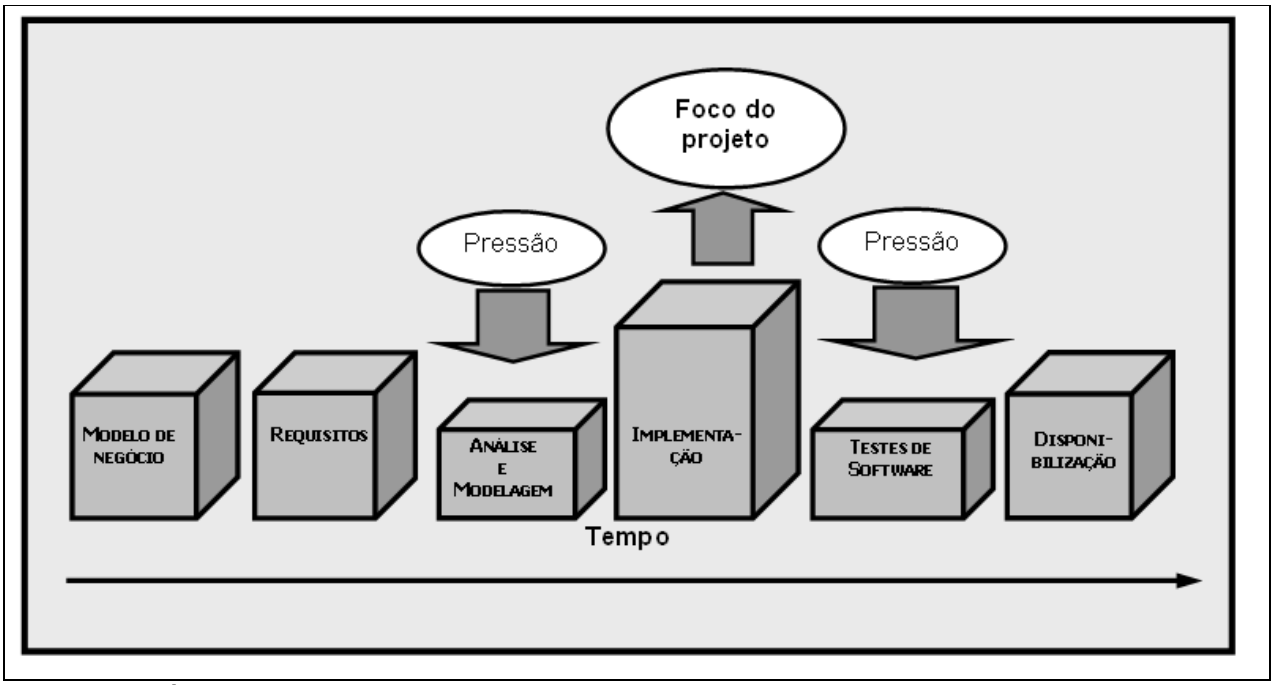
Vários fatores concorrem para que os prazos de entrega do *software sejam* sempre maiores do que o planejado. Por isso é comum a área de desenvolvimento buscar atalhos, na tentativa de compensar tais atrasos. Entretanto, conforme demonstra Molinari (2003), estes atalhos prejudicam outras fases, principalmente a fase de testes.

Figura 7: Fase que consomem mais tempo e recursos no planejamento de *software*



Fonte: Bartié, 2002.

Figura 8: Fases que consomem maior tempo e recursos no ciclo de desenvolvimento real



Fonte: Bartié, 2002.

3.3.2 Ausência de um ambiente de testes isolados

Segundo Bartié (2002), o processo de validação de um *software* necessita de um ambiente exclusivo de testes, isolado de qualquer interferência externa (demais

ambientes). Qualquer interferência pode comprometer a qualidade do trabalho de testes, bem como a confiabilidade das informações e dos processos executados, diminuindo o nível de eficiência e qualidade dos trabalhos.

O ambiente deverá ter equipamentos dedicados, ferramentas de testes e sistemas parcialmente instalados em seus respectivos *software* de apoio. Esse ambiente separado proporciona para a área de testes uma autonomia, que garante a qualidade do desenvolvimento do trabalho de testes. .

3.3.3 Testes que garantem a qualidade do processo

A qualidade dos processos pode ser medida por meio de testes aplicados em documentos gerados em cada fase do desenvolvimento. Como cada etapa deve produzir um conjunto de documentos, é possível estabelecer a qualidade nas várias fases do ciclo de desenvolvimento do software mediante a qualidade dos documentos produzidos. Se esses documentos apresentarem um alto nível de defeitos e não atingirem um nível mínimo de qualidade, é possível reconstruir o documento ou até mesmo re-executar a fase inteira. (Molinari, 2003).

3.3.4 Testes que garantem a qualidade do produto

Bartié (2002) aponta que a qualidade dos produtos de *software* pode ser garantida por intermédio de sistemáticas aplicações de testes nos vários estágios do desenvolvimento da aplicação. Estes testes são conhecidos como testes de validação porque, quando se constrói uma unidade de *software*, validam-se sua estrutura interna e sua conformidade aos requisitos estabelecidos.

Avalia-se a sua integração com as demais unidades de *software* existentes, validando as interfaces de comunicação existentes entre os componentes de *software*. Ao final de tudo, valida-se a solução tecnológica como um todo, submetendo-a a testes de todas as categorias. (Bartié, 2002).

3.3.5 Limitação dos testes

Autores como Bartié (2002) argumentam que todo o processo de desenvolvimento de *software* deve mostrar a todos elevado nível de confiança em

relação aos processos de testes. Entretanto, estes processos só terão qualidade quando forem executados com grande planejamento e a formalização das atividades (por exemplo, massas de dados não controladas). Procedendo-se, assim, obter-se-á um alto nível de eficiência desse processo. Porém nunca se poderá afirmar que um *software* está totalmente livre de erros, mesmo que se possuísse todos os recursos profissionais e tecnológicos disponíveis.

Deve-se a Myers, o relacionamento de *testes* com o compromisso de *identificação de erros.* O autor demonstra que "zero-defeito" é algo inatingível, ou seja, pela complexidade envolvida nos processos de testes de *software*, torna-se impossível obter um produto de *software* livre de erros.

Bartié (2002) compartilha de tal perspectiva ao lembrar que a qualidade de *software* trabalha com o conceito "zero-defeito", que representa a não tolerância a erros. Entretanto, ainda segundo o autor, erros vão sempre existir, sempre existindo mais um erro a ser descoberto. O desafio de um processo de qualidade de *software* é, contudo, tornar esse risco o mais próximo possível de zero.

Como já mencionado anteriormente, a maioria das organizações ainda insiste em acreditar que os testes devem ocorrer somente após a produção do *software* ou quando estiver parcialmente pronto (Bartié, 2002).

Isto que dizer que a qualidade será obtida somente na fase posterior do desenvolvimento do *software*. Entretanto, tal crença conduz a um grande erro, pois a maioria dos erros se encontra na fase inicial do desenvolvimento, principalmente na fase de requisitos, como demonstrado no **Gráfico 1**.

Devem-se ampliar as etapas de aplicação dos conceitos de qualidade, utilizando-os em todo o processo de criação e desenvolvimento de um *software*. Desta forma, o processo será mais controlado e confiável, possibilitando o completo gerenciamento do projeto de desenvolvimento do *software*. (Bartié, 2002).

3.4 Processo de testes

O processo de teste caracteriza-se por um conjunto de atividades executadas durante o ciclo de desenvolvimento de *software*. Segundo Boas (2003), as

atividades são divididas em planejamento, projeto, implementação e execução, como podem ser definidas a seguir.

3.4.1 Planejamento

Planejar é decidir antecipadamente o que e quando deve ser feito. Nesta fase são determinados os requisitos a serem testados: descrever o escopo, identificar métodos empregados, itens que serão ou não serão testados, recursos e pessoal necessários, cronograma de atividades e responsabilidades. O planejamento é a distribuição racional no tempo dos recursos disponíveis para a realização de alguma atividade.

Conforme Molinari (2003), são sete os conceitos de testes envolvidos no planejamento. São, segundo ele, mágicos, uma vez que simples, mas extremamente estratégicos:

1) *Planejamento de testes:* é o processo em si de planejamento.
2) *Plano de teste*: artefato principal é o fruto do planejamento
3) *Requerimentos de teste*: é a meta ou aquilo que se quer testar de forma clara na aplicação.
4) *Casos de teste*: são as situações de teste ou o teste em si.
5) *Procedimento de teste:* são os passos necessários para realizar um teste. Também conhecidos como passos de teste.
6) *Script de teste:* é a implementação do procedimento de teste através de uma ferramenta de teste automatizado.
7) *Ponto de verificação: é um check que se faz dentro do script automatizado para saber se o teste foi bem sucedido.*

3.4.2 Projeto

Projetar significa especificar a forma de se realizar algo detalhadamente. Nesta etapa, é gerado um projeto de testes que contém os requisitos e as estratégias para se efetuar o teste, será escolhido um grupo específico de características,

descrevendo os métodos utilizados e os tipos de testes que serão executados. Inthurn (2001) complementa que o projeto de teste objetiva mensuração e gerenciamento das ações que serão executadas durante toda a fase de teste.

3.4.3 Implementação

A implementação visa o fornecimento de subsídios importantes na execução de alguma tarefa. Nesta etapa serão descritas detalhadamente os itens que serão testados, as sequências de tarefas tem como objetivo avaliar o conjunto de características de um determinado item do *software*. Como resultado, obtêm-se os casos de testes e o procedimento de teste.

O caso de teste reflete os requisitos que serão verificados durante o teste de *software*. Trata-se de um documento que contém a identificação, objetivos, condições de entrada, seqüência de passos e resultados esperados, e que determinam se a característica testada está executando corretamente ou não. A identificação dos casos de testes também tem como finalidade servir como base para o projeto e desenvolvimento dos procedimentos de testes, explica Inthurn (2001).

Inthurn (2001) complementa que o procedimento de testes é um conjunto de instruções detalhadas que fazem o levantamento, execução e avaliação dos resultados de um determinado caso de teste. Este procedimento trata das precondições para o teste, condições de entrada, ações a serem tomadas, resultados esperados e os métodos de validação destes resultados.

3.4.4 Execução

Nesta etapa ocorre efetivamente a prática de teste a partir dos procedimentos de teste e dos casos de teste escritos. Durante a fase de execução, o testador deve registrar toda a atividade de teste, identificar data e hora, autor, procedimento seguido, pessoas envolvidas, resultados obtidos, condições do ambiente e os eventos não esperados, se ocorrerem. Os testes serão executados inúmeras vezes pois serão aplicados os testes por unidade e em seguida, integrados a fim de verificar se a funcionalidade foi atingida ou não, como afirma Inthurn (2001).

3.5 Tipos de testes de *software*

No Quadro 5, apresentado a seguir, elencamos os principais tipos de testes de *software.*

Quadro 5: Principais tipos de testes

TIPOS DE TESTES	DESCRIÇÃO
Teste de unidade	**Teste de nível ou classe. É o teste cujo objetivo é um "pedaço do código".**
Teste de integração	**Garante que um ou mais componentes combinados (ou unidades) funcionam corretamente.**
Teste de sistema	**A aplicação tem que funcionar como um todo. Neste momento a aplicação tem que "fazer aquilo que diz que faz".**
Teste operacional	**Garante que a aplicação pode "rodar" muito tempo sem falhar.**
Teste negativo-positivo	**Garante que a aplicação vai funcionar no "caminho feliz" de sua execução e vai funcionar no seu fluxo de exceção.**
Teste de regressão	**Um dos mais importantes testes. "Para ir para o futuro, tem-se de voltar ao passado, sempre". Toda vez que se for inserir uma característica nova na aplicação, deve-se testar** toda **a aplicação. Afinal, pode-se, ao "consertar algo, quebrar outro".**
Teste de caixa-preta ou *Black-box test*	**Testar todas as entradas e saídas desejadas. Não se está preocupado com o "código".**
Teste de caixa-branca ou *White-box test*	**O objetivo é testar o código. Às vezes existem partes do código que nunca foram testadas.**
Teste *beta*	**O objetivo é testar a aplicação em produção.**
Teste de verificação de versão	**Toda vez que se libera uma nova versão de aplicação existem, condições mínimas que validem se a versão liberada está *Ok*. Este teste é usado durante o processo de construção da aplicação. Pode querer testar, às vezes, apenas uma parte da aplicação.**
Teste funcional	**Testar se as funcionalidades presentes na documentação funcionam como especificadas. Incluem-se as regras de negócio.**
Teste de interface	**Verificar se a navegabilidade dos objetos de telas funciona corretamente, em conformidade com os padrões vigentes (em nível de interface).**
Teste de *performance*	**Verifica se o tempo de resposta é o desejado para "momento" de utilização da aplicação e suas respectivas telas envolvidas.**

Teste de carga	**Verifica se a aplicação suporta a quantidade de usuários simultâneos requeridos.**
Teste de aceitação do usuário ou UAT (*User Acceptance Test*)	**A meta é clara. É um teste exploratório voltado para validar aquilo que o usuário deseja, tendo um objetivo claro: dar o aceite ou não.**
Teste de estresse	**Testar a aplicação em situações inesperadas.**
Teste de volume	**Testar a quantidade de dados envolvidos (pode ser pouca, normal, grande ou além de grande).**
Teste de configuração	**Testa se a aplicação funciona corretamente em diferentes ambientes de *hardware* e de *software*.**
Teste de instalação	**Verificar se a instalação da aplicação (*hardware e software*) foi *Ok*.**
Teste de documentação	**A documentação existe? Mostra o que o *software* faz efetivamente? Falta algo na documentação?**
Teste de integridade	**O objetivo é testara integridade dos dados armazenados.**
Teste de segurança	**Testar a segurança da aplicação nas mais diversas formas.**
Teste de aplicações *Mainframe*	**Teste de aplicações *mainframe* requer um formalismo e um forte planejamento de testes.**
Teste de aplicações *Client*	**Neste momento se está preocupado mais com a funcionalidade, com a interface e a *performance* do que outras coisas.**
Teste de aplicações *Server*	**Neste momento se esta preocupado mais com desempenho dos processos que com a integridade dos dados.**
Teste de aplicações *Network*	**Análise estatística do desempenho das aplicações é enfoque pouco utilizada, daí o fato de certas aplicações serem testadas em produção.**
Teste de aplicações *Web*	**Na verdade desfez-se um mito, que não somente a interface é fundamental, mas o desempenho e a adequação das necessidades aliadas a uma alta flexibilidade das ferramentas envolvidas são fator-chave de sucesso. Um planejamento estratégico dos testes passou a ser fundamental.**
Teste de monitoração	**São, na realidade, testes funcionais, que visam verificar o *status* e disponibilidade de diversas funcionalidades e da aplicação em si.**

Teste de ameaça ou *Thread*	**Semelhante ao de segurança, mas em escala mais em nível de falhas.**
Monkey test	**Testa o aplicativo de forma aleatória e inesperada. O "teste do macaco" é antes de tudo "sem planejamento".**
Teste de módulo	**Teste de um módulo, porém em nível menor. Semelhante ao de unidade, porém é mais abrangente que este.**

Fonte: Molinari, 2003.

3.5.1 Escolha do tipo de teste

Segundo Molinari (2003), cada caso deve ser tomado em sua especificidade.

No caso de um teste entre uma caixa branca e uma caixa preta, por exemplo, faz-se as seguintes constatações:

- Se o enfoque da aplicação é o negócio, então o que deve ser testado é o negócio em si. Usa-se o teste de caixa-preta primeiro.
- Se for a interface de aplicação o que se deseja testar, usa-se caixa-preta.
- Se for um componente, ou aplicação "invisível" que fica na rede, o objeto de teste, use caixa-branca.
- Se o ambiente em que o código e o negócio devem ter alta qualidade, comece pelo de caixa-preta e depois vá para o de caixa-branca.
- Tudo depende também do *budget* disponível para comprar uma ferramenta de testes automatizados.
- Na maioria dos casos, o teste de caixa-preta é eficiente.

3.6 Principais tipos de teste de software

A seguir será descrito com mais detalhes alguns dos principais testes de *software*.

3.6.1 Verificação e Validação do *Software*

Conforme Molinari (2003) afirma, o teste de *software* é tradicionalmente considerado apenas como um processo de avaliação (uma fase do ciclo de

desenvolvimento do produto), ou seja, é comum acreditar que os testes de verificação e validação sejam atividades redundantes. Ao contrário, entretanto, possuem funções complementares, de natureza e objetivos distintos, a fim de fortalecer o processo de detecção de erros e aumentar a qualidade final do produto.

Os testes de verificação visam garantir a qualidade dentro do processo de engenharia do *software*, ao mesmo tempo em que os testes de validação enfocam a garantia da qualidade do produto de *software* (Bartié, 2002). Uma maneira simples de distinguir as duas atividades é refletir nas seguintes perguntas, como sugere a autora Inthurn (2001):

- Validação: Estamos construindo o produto certo?
- Verificação: Estamos construindo certo o produto?

3.6.1.1 Verificação

Durante os testes de verificação ocorre a coleta de informações de negócios e planejamento da arquitetura do *software*. Com isso, pretende-se estabelecer o perfeito entendimento entre o negócio a ser atendido e o *software* a ser construído. Durante esse período, devem ser observados os documentos que especificam o comportamento do produto, que funcionam como uma maquete do projeto de *software*. Trata-se de um processo de auditoria das atividades, e a avaliação de documentos, gráficos, manuais, códigos-fontes produzidas em cada etapa do processo. Com tudo isso, tem-se o propósito de evitar que dúvidas e problemas não resolvidos prossigam na próxima fase. (Bartié, 2002).

A principal característica dos testes de verificação é o fato de não envolver nenhum processamento de *software*, pois antecede a criação do aplicativo. Isso garante que todas as decisões e definições sejam entendidas e aceitas pelos integrantes do grupo que participarão do desenvolvimento (Bartié, 2002).

A verificação é definida por Molinari (2003) como um processo de melhoria contínua sem fim e que deve ser usada para garantir a operação e manutenção do sistema. O autor segue afirmando que o custo geral do *software* é reduzido nesta etapa de desenvolvimento (como prevenção): se forem reduzidos os erros de quatro para um,

os defeitos encontrados antes da fase de produção custariam de vinte a cem vezes menos do que se tais erros fossem encontrado no produto pronto.

Para Bartié (2002) os testes de verificação devem respeitar cada estágio do desenvolvimento das seguintes etapas:

- Definição das necessidades e características de negócio a serem atendidas pela solução.
- Identificação dos requisitos funcionais e não funcionais que a solução tecnológica deverá contemplar
- Definição do modelo de arquitetura da solução tecnológica.
- Construção de *software* que atenderão às definições da arquitetura estabelecida.

3.6.1.2 Validação

Esta fase tem como característica a existência do componente computacional, no qual um conjunto de testes avalia a qualidade do produto de *software* desenvolvido conforme os requisitos e especificações identificados, analisados e revisados durante a etapa inicial do projeto. É o processo formal de avaliação dos produtos, aplicados em componentes isolados, módulos ou sistemas como um todo. (Bartié, 2002).

Nesta fase, a presença física do *software*, funcionando em um ambiente tecnologicamente preparado, verifica seu comportamento segundo as diversas condições simuladas. Esses testes são importantes, pois mostram mais os sintomas do que propriamente erros. Os defeitos não ocorrem de maneira explícita (interrupção do processamento ou não-execução de uma funcionalidade, por exemplo), mas de forma implícita e difícil de ser diagnosticada (o ajuste financeiro errado ou cálculo de imposto irregular). Os resultados do processamento serão os principais pontos de validação.

Segundo Bartié (2002), as atividades de teste (planejamento, modelagem, execução e conferência) devem ocorrer paralelamente às atividades de construção dos componentes computacionais. O autor afirma que as validações deverão ser aplicadas respeitando-se os estágios de desenvolvimento:

- Testes em componentes executáveis isolados recém-criados e alterados.
- Testes de interface entre componentes à medida que eles vão sendo integrados.
- Testes em sistema ou módulos completos.
- Aceite de um sistema ou módulos pelos clientes e usuários.

3.6.2 Testes de caixa branca

Os testes de caixa branca ou estruturais verificam a estrutura interna da unidade, percorrendo todos os caminhos internos possíveis, procurando ter certeza de que todo o código-fonte tenha sido executado durante o conjunto de testes realizados, conforme explica Inthurn (2001). Bartié (2002) complementa que é necessário o conhecimento da arquitetura interna do produto, em outras palavras, a pessoa encarregada de fazer esses testes deverá ter acesso a fontes, estruturas dos bancos de dados e realizar todos os testes previstos no processo de validação.

Segundo Molinari (2003), os casos devem:

- Garantir que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez.
- Exercitar todas as decisões lógicas para valores falsos ou verdadeiros.
- Executar todos os laços em suas fronteiras e dentro de seus limites operacionais.
- Exercitar as estruturas de dados internas para garantir a sua validade.

3.6.3 Testes de caixa preta

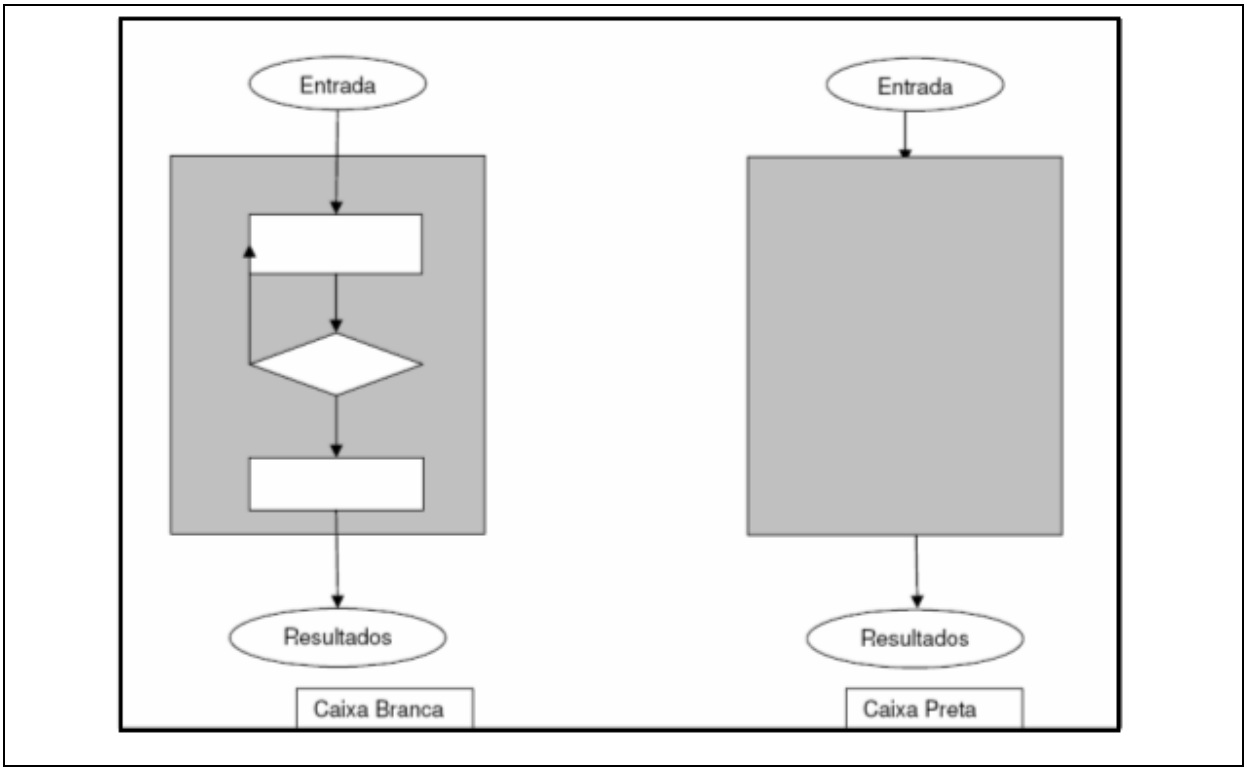
Os testes de caixa preta concentram-se nos requisitos funcionais do software. Por isso podem ser também chamados de *testes funcionais*. Por meio deles é possível verificar as entradas e saídas de cada unidade, preocupando-se apenas com o que está sendo executado e não com a maneira como a unidade está sendo executada.

Esses testes tem como objetivo descobrir funções incorretas ou ausentes; erros de interface nas estruturas de dados ou no acesso a banco de dados externos; desempenho e inicialização, e término, como explica Inthurn (2001). Segundo Bartié

(2002), trata-se de testes que visam garantir que as regras de negócio (requisitos do sistema) serão plenamente atendidas pelo *software*. Enquanto os *testes de caixa branca* são aplicados no início do processo de teste, os *testes de caixa preta* são executados nas últimas etapas da atividade de teste, pois desconsidera a estrutura do controle (código do programa), concentrando-se apenas na funcionalidade do *software*, afirma Turchetti (2003).

Esses testes não requerem o conhecimento da tecnologia utilizada ou dos conceitos da implementação, mas exigem um planejamento apurado e transparente para que todos os cenários sejam simulados e executados, explica Bartié (2002).

Figura 9: Abordagens fundamentais do testes



Fonte: Bartié, 2002.

3.6.4 Testes progressivos e regressivos

Segundo Bartié (2002), os testes de progressão são criados de acordo com a evolução do *software*, ou seja, assim que o programa recebe novas funcionalidades,

um novo conjunto de testes deve ser elaborado. Serão testadas somente as inovações do *software*, no qual se assume que não houve nenhum erro após esse processo de desenvolvimento.

O autor prossegue afirmando que uma vez que as funcionalidades novas são testadas, passam a ser testes de regressão, ou seja, é a necessidade de realizar novamente os planos de testes, a fim de assegurar que as alterações e inserções de funcionalidades ou correções não irão afetar o resto do programa.

3.6.5 Testes de unidade

Os testes de unidade tem como finalidade verificar a menor do projeto de *software* – o módulo ou função, garantindo a qualidade deste determinado componente tecnológico, avaliando sua estrutura interna e a perfeita adequação com seus requisitos, explica Bartié (2002). Segundo Inthurn (2001) dentre as verificações destacam-se:

- A interface com o módulo. As informações que entram e saem devem ser consistentes.
- A estrutura de dados local.
- Os dados armazenados temporariamente devem manter sua integridade durante todos os passos de execução correta da unidade.
- Os caminhos básicos. Todas as instruções de uma unidade devem ser executadas pelo menos uma vez, a fim de verificar se os resultados obtidos estão corretos e;
- Os caminhos de tratamento de erros. Da mesma forma que todos os caminhos básicos devem ser testados, os caminhos para tratamento de erros devem ser executados a fim de verificar se valores não verdadeiros sejam devidamente tratados.

Humphrey apud Boas (2003) ressalta que é importante executar os testes unitários, pois a partir deles é possível corrigir eventuais problemas que poderiam acarretar em grandes atrasos durante a próxima fase de testes.

3.6.6 Testes de integração

O objetivo dos testes de integração é concentrar-se na arquitetura do *software*. Estes testes devem montar e integrar os componentes (unidades), validando a sua compatibilidade para fornecer um pacote de *software*, explica BARTIÉ (2002). UKUMA (2002) ressalta a importância destes conjuntos de testes, pois mesmos se cada módulo já foi testado isoladamente, pode apresentar algum problema quando houver a interação das unidades.

Turchetti (2003) apresenta uma das estratégias incrementais para a integração de todos os módulos que é o *bottom-up*, que testa fase por fase os módulos de baixo para cima, separadamente e, em seguida, em conjunto. Todas as unidades serão testadas uma a uma até o topo da hierarquia. A partir do último módulo, pode-se adotar a estratégia *top-down* que percorre a direção oposta, testando os níveis mais altos até o nível baixo.

Ukuma (2002) afirma que também é possível combinar as duas estratégias citadas: o *sandwch* que integra de baixo para cima e de cima para baixo as unidades e eventualmente a integração ocorre em algum momento no meio da estrutura. Já o *Big-Bang* aplica os testes após combinar todos os módulos do *software*.

3.6.7 Testes de sistema

Para Bartié (2002), a validação de sistema é o estágio do processo mais difícil de realizar, pois é validada a solução como um todo. Os testes de sistema são executados quando o *software* está completo, antes da entrega ao cliente. Deve garantir os requisitos estabelecidos pelo cliente, verificando as interfaces externas e mostrando que o *software* satisfaz os requisitos funcionais e de desempenho especificados como explica Boas (2003). Turchetti (2003) complementa que esses testes tem como objetivo encontrar erros de funções e características de desempenho. Um conjunto de especificações são analisadas e combinadas com elementos de sistemas (hardware, pessoas, banco de dados e outros), para verificar se suas funções e características de desempenho são atingidas.

3.6.8 Categorias de testes

Em determinados momentos, ao tentar realizar todos os testes, haverá um grande esforço desnecessário da equipe para realizá-los. Isto ocorre quando se deseja procurar determinados erros, Bartié (2002) explica que se deve planejar uma estratégia em que priorize os tipos de testes que irá detectá-lo.

3.6.9 Funcionalidade

Estes testes tem como objetivo simular os cenários de negócio, afim de garantir que os requisitos funcionais sejam implementados. Para simular todas as variações possíveis, é importante conhecer profundamente as regras de negócio. Os testes funcionais são baseados nos documentos de especificação funcional que descrevem o comportamento do *software*. Bartié (2002) ainda ressalta que o importante é garantir que não existam diferenças entre os requisitos funcionais e o comportamento esperado do programa. Os testes de funcionalidade podem ser:

- As pré-condições de uma transação de negócios
- O fluxo de dados de uma transação de negócios
- Os cenários alternativos de uma transação de negócios. Os cenários de exceção de uma transação de negócios
- As pós-condições de uma transação de negócios.

3.6.10 Usabilidade

São testes que simulam as condições de uso do programa sob a visão do usuário para verificar se o *software* é intuitivo e simples. Verifica-se neste caso a facilidade de navegação na aplicação, clareza de textos, acesso simplificado de mecanismos de apoio ao usuário, o volume reduzido de ações para realizar determinada tarefa, entre outros. As mensagens de alerta ao usuário a fim de evitar que este faça alguma ação que danifique suas tarefas também podem ser consideradas como testes de usabilidade, explica Bartié (2002). Como exemplos de testes, podem-se destacar:

- Entrar em cada tela e avaliar a facilidade de navegação entre elas.

- Realizar “n” operações e depois desfazê-las.
- Realizar procedimentos críticos e avaliar mensagem de alerta.
- Avaliar números de passos para realizar as principais operações.
- Avaliar a existência de ajuda em todas as telas.
- Realizar “n” buscas no manual de ajuda e validar os procedimentos sugeridos.

3.6.11 Segurança

Os testes de segurança visam descobrir erros de segurança, impedindo o acesso indevido ao sistema, explica Turchetti (2003). O teste de segurança verifica se os mecanismos de proteção realmente executam as suas tarefas.

Recomenda-se que estes sejam executados por pessoas que não participaram do desenvolvimento do *software*, pois já são de seu conhecimento os mecanismos implementados. Os testadores deverão tentar entrar no sistema de diversas formas: obtenção ilegal de senha, desarme do sistema, tentativa de acesso durante a recuperação do sistema ou após inserção de falha intencional garantindo que o sistema se comporte de maneira adequada diante tais tentativas ilegais de acesso como afirma Inthurn (2001). Bartié (2002) afirma que estes testes podem ser:

- Validar todos os requisitos de segurança identificados
- Acessar funcionalidades e informações que requerem perfil avançado.
- Invadir/derrubar o servidor de dados.
- Extrair backups de informações sigilosas.
- Descobrir senhas e quebrar protocolos de segurança
- Processar transações geradas de fontes inexistentes
- Simular comportamento/infecção de vírus

3.6.12 Carga (Estresse)

Os testes de carga ou estresse simulam condições atípicas do *software*, como aumentos e reduções sucessivas de transações gerando momentos de picos a fim de avaliar como a solução lida com as variações de processamento, explica Bartié (2002). Para verificar estes testes, pode ser executados situações como:

- Elevar e reduzir sucessivamente o número de transações simultâneas
- Aumentando e reduzindo o tráfego de rede
- Aumentando o número de usuários simultâneos
- Combinando todos esses elementos.

3.6.13 Volume

Alguns autores como Pressman (2006) afirmam que os testes de Estresse é aquele em que há apenas aumento da carga na aplicação, entretanto, seguindo a linha de pensamento de Bartié (2002), esta definição se aplica aos testes de volume, os quais consistem em avaliar o *software* em situações limites (cadastrar valores ou informações com grande carga de dados, procura excessiva no disco, abertura de muitas janelas, aumento excessivo de índices de dados) com a intenção de pará-lo, a fim de conhecer o limite da aplicação. Estes testes podem ser realizados aumentando sucessivamente:

- O volume de transações.
- Consultas.
- Tamanho de arquivos a serem processados.

3.6.14 Configuração

Bartié (2002) explica que nestes testes, executa-se o *software* em diversas configurações de *software* e hardwares diferentes, visando garantir que a aplicação seja compatíveis com os mais variados ambientes. São possíveis testes:

- Variar os sistemas operacionais (e suas versões).
- Variar os navegadores.
- Variar os hardwares que irão interagir com a solução.
- Combinar todos os elementos.

3.6.15 Compatibilidade

É muito comum a aplicação não ser compatível com determinada versão. Estes testes verificam se o *software* não apresenta problemas se forem executados interagindo-se com versões anteriores de outras aplicações ou dispositivos (*software* ou

hardwares), afirma Bartié (2002). Os testes de compatibilidade podem ser:Importar os dados gerados pela solução anterior.

Comunicar com todas as versões de layout (atual e anteriores).

3.6.16 Desempenho (ou *Performance*)

Estes testes podem ser realizados combinando os testes de estresse, utilizando instrumentação de *hardware* e *software* para medir a utilização dos recursos do sistema, explica Turchetti (2003). Ao instrumentar um programa possibilita descobrir situações que levam à degradação e possível falhas do sistema, pois permite monitorar intervalos de execução, registrar eventos quando ocorrerem e mostrar estados de máquina em base regular, complementa Inthurn (2001). O autor salienta que é importante desenvolver a atividade de testes de desempenho durante o estágio do planejamento de testes, visando garantir que os objetivos sejam atingidos. Para isto, muitas vezes é necessário desenvolver técnicas de simulação de situações (como replicar centenas de vezes entradas para obter o alto volume necessário para o teste ou reduzindo a quantidade de *software* disponível), modelagem e medida de capacidade do sistema. Segundo Bartié (2002),

Entre os testes possíveis estão:

- Validar todos os requisitos de desempenho identificados.
- Simular “n” usuários acessando a mesma informação simultaneamente.
- Simular “n” usuários processando a mesma transação simultaneamente.
- Simular grande tráfego de rede.
- Combinar todos estes elementos.

3.6.17 Instalação

Estes testes validam os procedimentos de instalação de uma aplicação, na qual, além do procedimento padrão, Bartié (2002) complementa que é necessário verificar as variações que a instalação pode ter. Os testes de instalação podem variar entre:

- A primeira instalação do *software.*
- A instalação do *software* já instalado.
- A instalação de atualização de um *software.*
- A instalação de *software* em vários ambientes distintos.

3.6.18 Confiabilidade e Disponibilidade.

Estes testes monitoram o programa por um determinado período e avaliam o nível de confiabilidade da arquitetura do *software*. Enquanto estiver executando, e há um problema, os testes identificam se ocorreu por falha da infra-estrutura (confiabilidade) e o tempo para resolver este problema (disponibilidade), assim explica BARTIÉ (2002). Os testes podem ser realizados da seguinte forma:

- Monitorar permanentemente o ambiente de aceite
- Identificar todas as interrupções do ambiente (confiabilidade).
- Identificar o tempo de interrupção do ambiente (disponibilidade).

3.6.19 Recuperação

Os testes de recuperação visam verificar como os *software* reagem frente à falhas que podem vir a ocorrer, ou seja, o sistema é forçado a falhar de diversas maneiras para se verificar a correção do problema será resolvida no período previamente especificado, executada adequadamente. A recuperação pode ser automática, realizada pelo próprio sistema: a reinicialização, recuperação de dados e o reinício são avaliados a partir da concretização das respectivas atividades. Se a recuperação exigir intervenção humana, o tempo médio de reparo é avaliado dentro dos limites aceitáveis afirma Inthurn (2001).

O acompanhamento do sistema deve ser realizado através de um *log* que registra as atividades realizadas, inclusive aquelas que antecedem às falhas que poderão ocorrer. As falhas podem ser de disco, *interface*, memória insuficiente, entre outras. O *software* ideal é aquele capaz de se recuperar rapidamente e com o mínimo de intervenção humana explica Inthurn (2001). Segundo o autor Bartié (2002) pode se realizar os seguintes testes:

- Interrompendo o acesso à rede (por alguns instantes ou longos períodos)
- Interrompendo o processamento (desligando o computador ou o servidor)
- Gerando arquivos, cancelar o processamento e avaliar se existe arquivos gerados.

3.6.20 Contingência

Bartié (2002) afirma que esta categoria de testes valida os procedimentos de contingência a serem aplicados à determinada situação prevista no planejamento de *software*, simulando os cenários de contingência, avaliando a precisão dos procedimentos. Os testes podem ser aplicados com:

- A instalação emergencial de uma aplicação.
- Recuperação da perda de conexão da filial com sua matriz.

O quadro 6 ilustra as categorias de testes utilizando como exemplo uma operação de saque. Os cenários de teste são:

Quadro 6: Levantamento dos cenários aplicando-se os conceitos de categorização

FUNCIONAL	**SEGURANÇA**	**USABILIDADE**	**PERFORMANCE**
• Simular saques • acima do saldo disponível • Simular saque na conta poupança; • Simular saque acima do valor do limite da conta • Simular saques com valores não múltiplos das notas	• Simular saques com cartão vencido • Avaliar se a senha do cartão está sendo requisitada antes e depois da transação • Avaliar se a senha adicional e randômica está sendo requisitada no início da operação • Simular saque noturno acima do valor permitido	• Avaliar se todas as telas possuem ajuda. • Avaliar se as mensagens são claras e objetivas • Avaliar se o padrão visual é mantido em todos os momentos • Avaliar se todas operações possuem caminhos de fuga.	• Avaliar se a duração do saque dura até 30 segundos em um universo de 5 milhões de correntistas e 100 milhões de movimentação bancária • Garantir que a manipulação com dispositivos físicos no saque não ultrapasse dez segundos da operação
CARGA E CONCORRÊNCIA	**CONFIGURAÇÃO**	**RECUPERAÇÃO**	**CONTINGÊNCIA**

Fonte: Bartié, 2002.

3.6.21 Testes de Aceitação

Turchetti (2003) explica que estes tem como objetivo de verificar se a implementação está conforme com os requisitos estabelecidos do *software*, evitando "surpresas desagradáveis". Segundo Inthurn (2001), os testes de validação são realizados por meio de testes de caixa preta, e visam garantir que todos os requisitos funcionais do sistema e desempenho sejam alcançados, documentação correta e outros requisitos com portabilidade e manutenbilidade sejam cumpridos. Os testes de validação propõem a demonstração que o produto executa aquilo que é esperado dentro do ambiente do usuário final e com o banco de dados real, complementa Boas (2003).

Entretanto, é complicado reproduzir o modo como o cliente realmente usará o programa, pois é possível que haja a má interpretação das instruções de uso, combinação não usual de dados ou até mesmo o não entendimento de saídas que podem ser claras para o desenvolvedor, mas ininteligíveis para o usuário, afirma PRESSMAN (2006). Para que o aceite do cliente seja válido, os desenvolvedores utilizam os processos de teste *alfa* e *beta* que descubram erros os quais apenas o usuário final será capaz de descobrir.

Os testes *alfa* são controlados, executado no próprio ambiente de desenvolvimento de *software*, onde o cliente utiliza o produto, acompanhado pelo desenvolvedor, a fim de registrar erros e eventuais problemas de utilização. Segundo Inthurn (2001), um dos problemas encontrados é a falta de naturalidade no comportamento do cliente perante a observação do desenvolvedor, o que dificulta nos resultados do testes.

Os testes *beta* são conduzidos nas instalações de um ou mais clientes, sem a supervisão do desenvolvedor. A partir deste momento, quem registra os problemas encontrados é o próprio usuário final, que, periodicamente deve repassar à equipe de desenvolvimento, para as devidas correções e finalmente ser possível a disponibilidade do *software* final para toda a base de clientes, explica Inthurn (2001). A seguir, Quadro 7 faz uma análise das fases do teste de validação.

Quadro 7: Análise das fases dos testes de validação de *software*

	FASE DA VALIDAÇÃO	**CATEGORIAS DE TESTES APLICADAS**	**CARACTERÍSTICA DA FASE DE VALIDAÇÃO**
TESTE DE BAIXO NÍVEL	**Teste de Unidade**	• **Estrutura Interna** • **Funcionalidade** • **Usabilidade** • **Segurança**	• Estratégia caixa branca e caixa preta • Testa parte do *software* • Requer conhecimento da estrutura interna • Executada pelo desenvolvedor ou profissional de teste.
	Teste de Integração	• **Interface** • **Dependências entre Componentes**	• Estratégia caixa branca e caixa preta • Testa integrações entre partes do *software* • Requer conhecimento da arquitetura interna do *software* Executada pelo desenvolvedor ou profissional de teste.
TESTE DE ALTO NÍVEL	Teste de Sistema	• Funcionais • Não Funcionais - Performance - Instalação - Recuperação - Carga	• Estratégia caixa preta • Os testes são aplicados no *software* como um todo. • Não requer conhecimento da arquitetura interna do *software* Deve ser executada por um grupo de teste independente.

Fonte: Bartié, 2002

4. PESQUISA DE CAMPO

O questionário apresentado no apêndice foi o instrumento de coleta de dados da presente pesquisa. Sendo o questionário composto de questões abertas e fechadas, estas foram separadas para a seqüência da avaliação e comparação de dados.

Como mencionado na metodologia, foram avaliados 14 questionários de empresas de consultoria. Tais questionários foram apresentados via e-mail ou mediante algumas visitas técnicas.

A pesquisa esbarrou na recusa de algumas empresas em participar da pesquisa, fato que reduziu o número de amostras disponíveis para análise. No entanto, considera-se que a amostra obtida seja suficiente para um razoável levantamento de dados visando os objetivos do trabalho.

Busca-se, aqui um melhor conhecimento do fenômeno estudado e não uma avaliação estatística. Desta forma, se por um lado, não possam fornecer inferências rigorosas para todo o universo de empresas do setor, por outro, acredita-se que os resultados obtidos tenha um valor heurístico importante, explorando uma área ainda pouco estudada.

4.1 Resultados

A disposição das questões no questionário considerou, entre outros fatores, a facilidade do posterior trabalho de tabulação. A análise dos resultados obtidos será apresentada a seguir.

Questão 1 – Tipo de sistema oferecido?

Esta questão tem por objetivo compreender quais são as soluções desenvolvidas pela empresa, pois, de acordo com os produtos oferecidos por ela pode-se especificar qual é sua área de atuação.

Observou-se que uma das empresas estudadas não possui foco em apenas um produto, prestando consultoria, suporte, monitoração, administração de redes e *help*

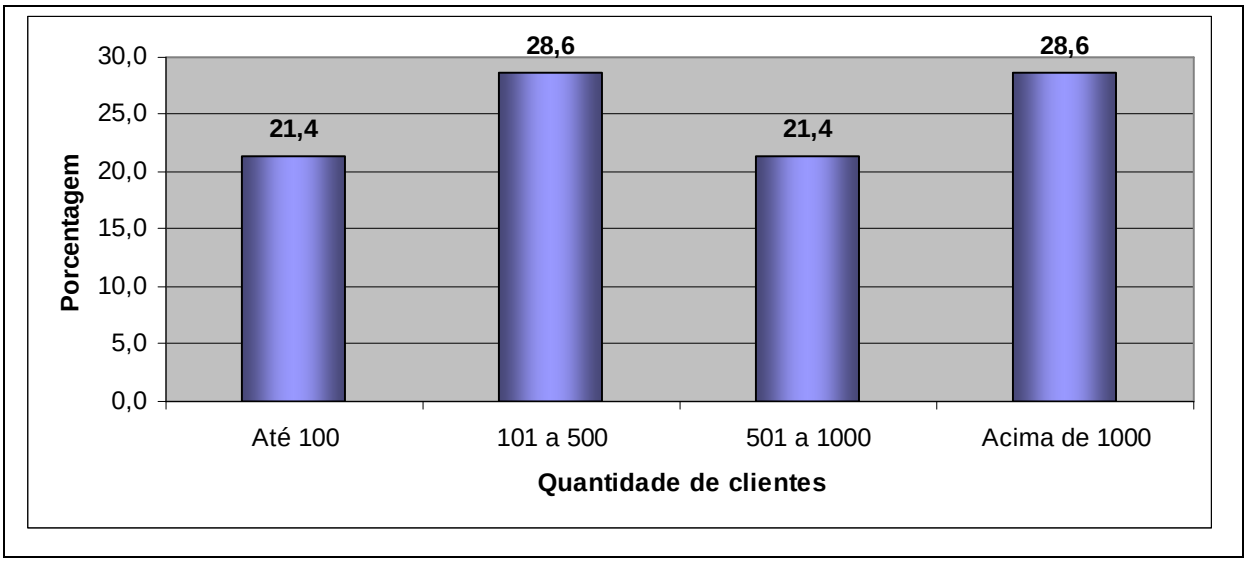
desk. Em contrapartida todas as demais desenvolvem *software* comerciais para gestão empresarial, tanto para o setor público quanto para o privado.

Questão 2 – Natureza

A região de abrangência das empresas pesquisadas se estende no território nacional, sendo que 21% delas também atendem ao exterior.

A quantidade de clientes em carteira é exposta no **Gráfico 1**..

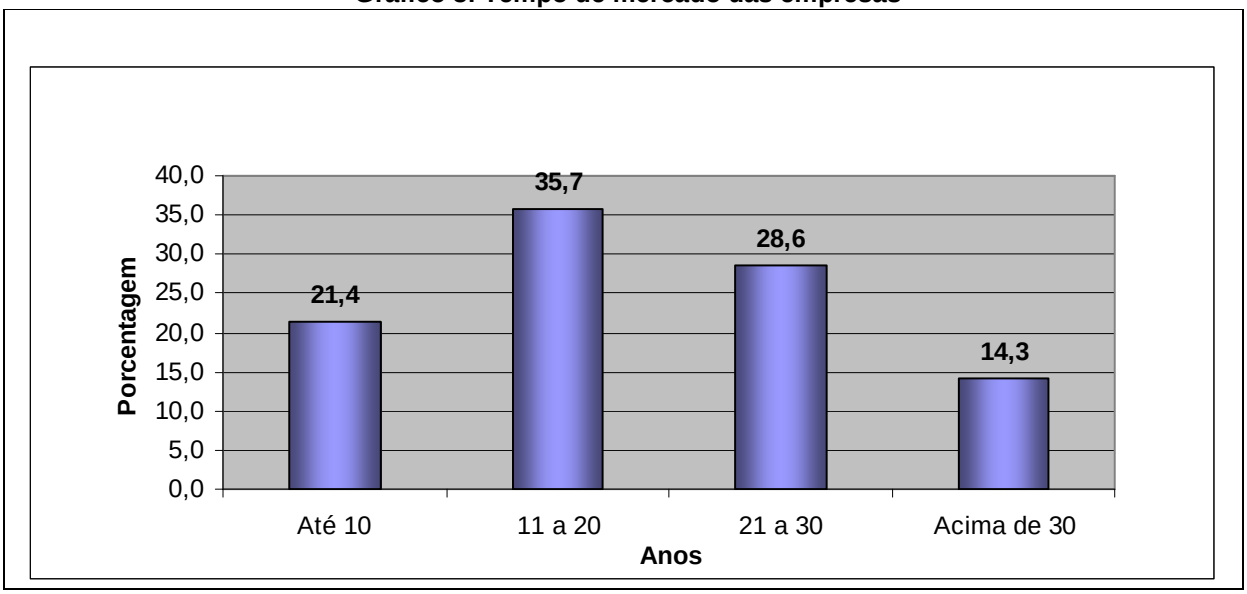
Gráfico 2: Clientes em carteira



Fonte: Pesquisa de campo, 2009

O tempo de mercado das empresas pesquisadas está representado no **Gráfico 3**:

Gráfico 3: Tempo de mercado das empresas



Fonte: Pesquisa de campo, 2009

Questão 3 – Quantidade de funcionários.

Esta questão tem por objetivo visualizar a quantidade de funcionários que atuam nas empresas pesquisadas, bem como a distribuição destes nas áreas de atuação: Área comercial, Desenvolvedores, Suporte, Equipe de teste.

Quadro 8: Quantidade de funcionários por setor

	Área comercial	Desenvolvedores	Suporte	Equipe de teste	Gerente de projetos	Outros
Emp 1	8	19	25	10	7	14
Emp 2	15	80	40	20	20	150
Emp 3	10	10	10	2	1	
Emp 4	20	15	50	2	1	50
Emp 5	56600	647	432	60	vários	35000
Emp 6	2	60	10	10	7	3
Emp 7		1400	10		100	
Emp 8	20	300	100	12	5	
Emp 9	2	10	4	3	1	15
Emp 10						
Emp 11	19	350	73	5	35	485
Emp 12	14	35	30	7	6	66
Emp 13	3	4	1		1	11
Emp 14	4	10	3	3	1	

Fonte: Pesquisa de campo, 2009

O quadro acima foi disposto exatamente como as pessoas responderam na pesquisa.

A discrepância dos valores apresentados na empresa 5 se justifica por ser um grande banco internacional, que atua em todas as regiões do país, com sua cede de *Centro de Processamento de Dados* e *Desenvolvimento de Software* em São Paulo.

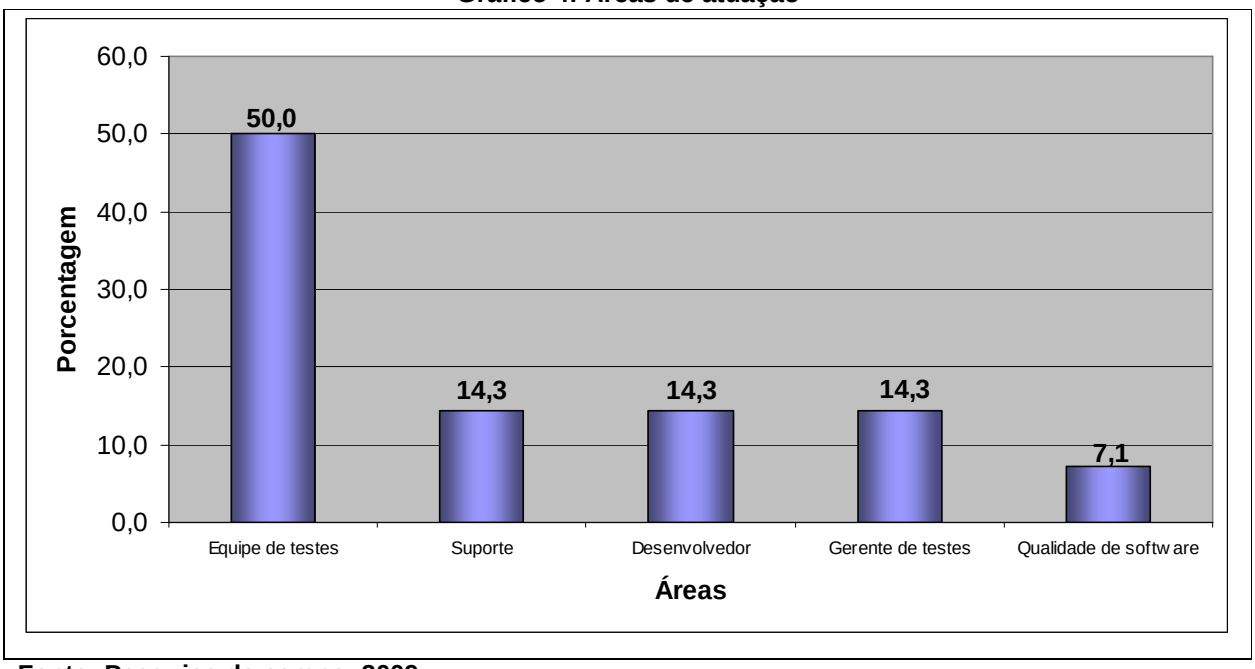
Questão 4 – Em qual das áreas acima você atua?

Esta questão tem por objetivo identificar a área de atuação dos profissionais das empresas que responderam a pesquisa.

Pode-se observar que a soma das áreas diretamente relacionadas a testes como Equipe de testes, Gerente de testes, Qualidade de *software* é de 71,4%.

Os 28,6% restantes atuam como Suporte e Desenvolvedores, embora não tenham demonstrado relação direta com a função de testes, por conta da nomenclatura do cargo, responderam a pesquisa a contento, demonstrando conhecimento técnico e teórico das normas de qualidade. Isso sugere que eles podem, na prática, executar determinados tipos de testes de *software*.

Gráfico 4: Áreas de atuação



Fonte: Pesquisa de campo, 2009

Questão 5 – O que você entende por qualidade de *software*?

Em resumo todos os profissionais da área de *software* (que aqui representam as empresas pesquisadas) responderam que a qualidade de *software* é essencial para a produção de tecnologia da informação, e que todas as fases do desenvolvimento de software devem ser contempladas com testes de qualidade, de forma estruturada, desde a concepção (a idéia do produto de *software*) até a utilização final pelo cliente. Segundo os entrevistados, tal processo reduziria custos, minimizaria e controlaria erros, melhorando a usabilidade e funcionalidade. Consequentemente, otimizando o ciclo da qualidade, seria alcançado o objetivo de satisfação e ampliação do leque de clientes.

Questão 6 – Quais os passos de desenvolvimento de *software* da empresa?

Esta questão tem por objetivo observar quais são as fases do *ciclo de vida* do *software* nas empresas que responderam a pesquisa.

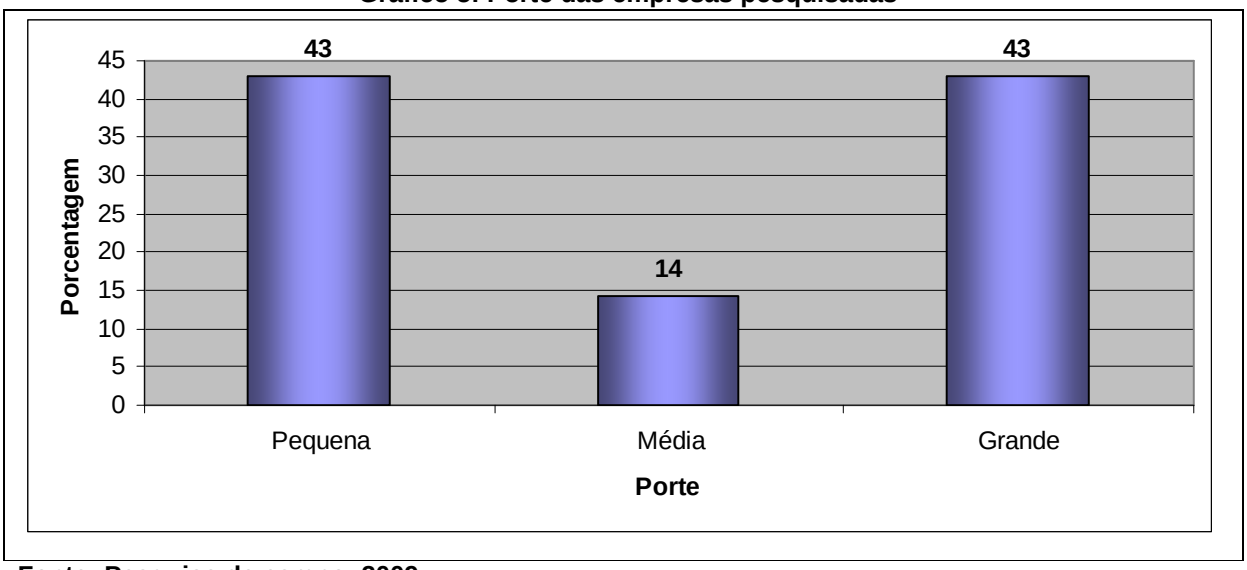
As respostas indicam que as empresas seguem, com algumas variações de denominação, as fases comumente utilizadas no processo de *software*: modelo de

negócios, análise de requisitos, análise e modelagem, implementação, testes e implantação.

Questão 7 - Qual o porte da empresa onde trabalha?

Essa questão possibilitou averiguar que a pesquisa foi realizada levando em consideração empresas de pequeno, médio e grande porte. Sendo que as empresas de médio porte foram menos presentes, enquanto as de pequeno e grande porte, apareceram com a mesma percentagem. O **Gráfico 5**, representa as proporções:

Gráfico 5: Porte das empresas pesquisadas



Fonte: Pesquisa de campo, 2009.

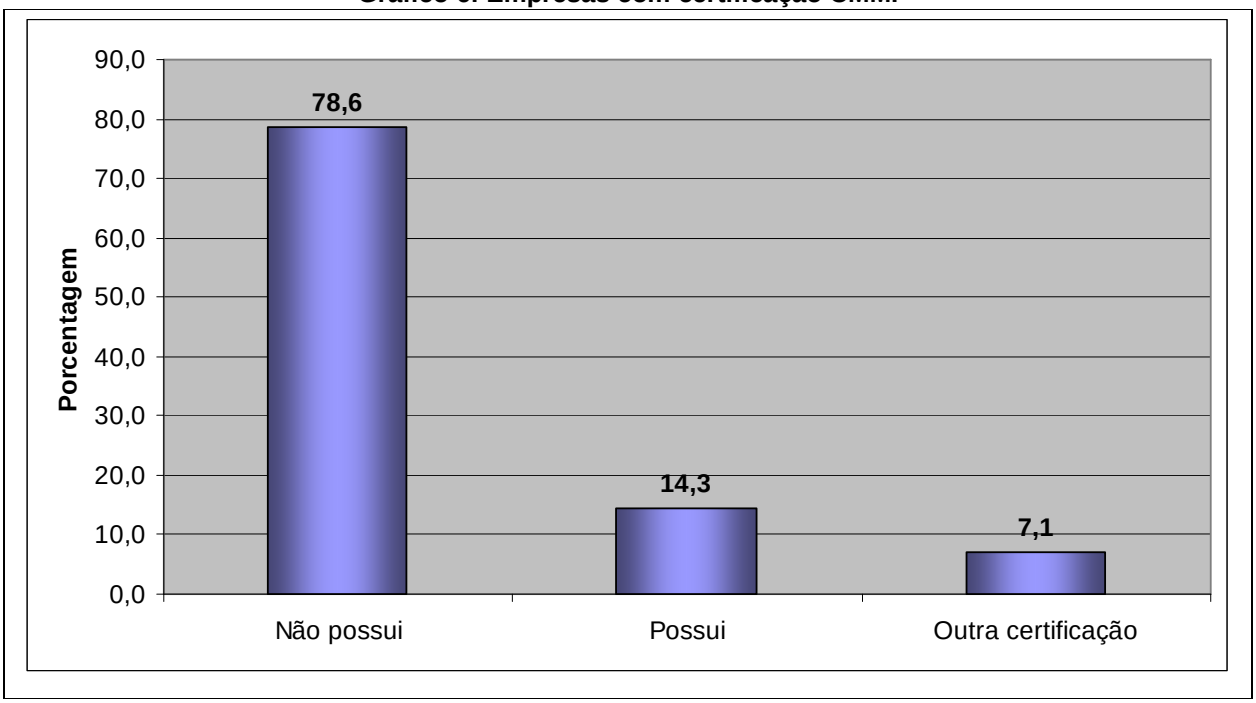
Questão 8 - Possui certificação CMMI?

Ao observar as respostas, pode-se constatar que há um número elevado entre as empresas pesquisadas que não possuí certificação CMMI.

Observou-se, além disso, que uma das empresas possui a certificação MPS-Br, que é brasileira e semelhante ao CMMI.

Os dados são representados no **Gráfico 6:**

Gráfico 6: Empresas com certificação CMMI



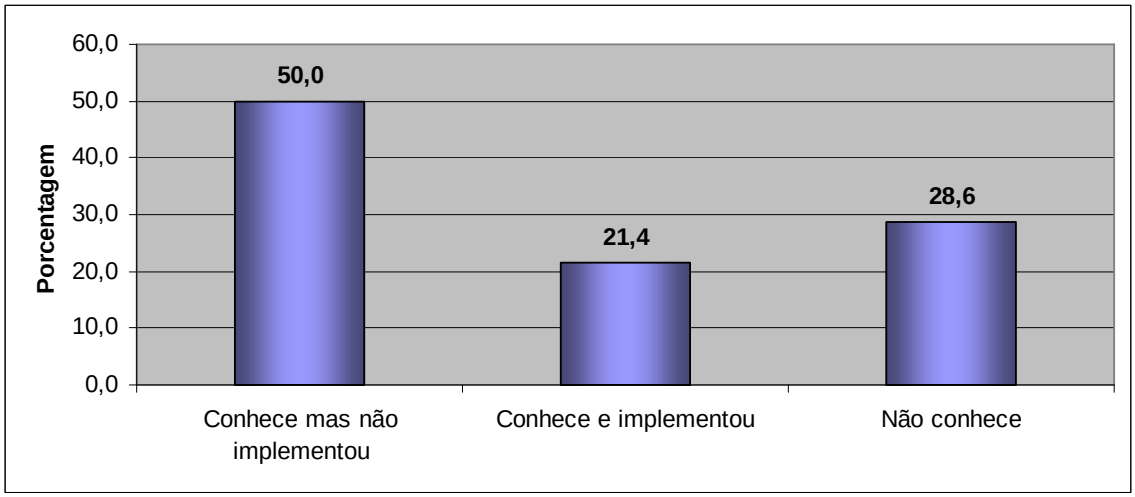
Fonte: Pesquisa de campo, 2009.

Questão 9 – Conhece e utiliza a NBR ISO/IEC 12207 - Tecnologia de informação – Processos de *ciclo de vida* de *software*?

Observa-se no **Gráfico 7** os resultados desta questão, dela pode-se constatar que uma pequena parte das empresas conhece e utiliza a norma, enquanto a grande maioria não a conhece ou a conhece mas não a utiliza.

Cruzando-se esta questão com a questão oito, constata-se que as empresas que possuem a certificação CMMI (ou outras certificações) implementaram e fazem uso da norma.

Gráfico 7: Conhecimento da NBR ISO/IEC 12207 pelas empresas



Fonte: Pesquisa de campo, 2009.

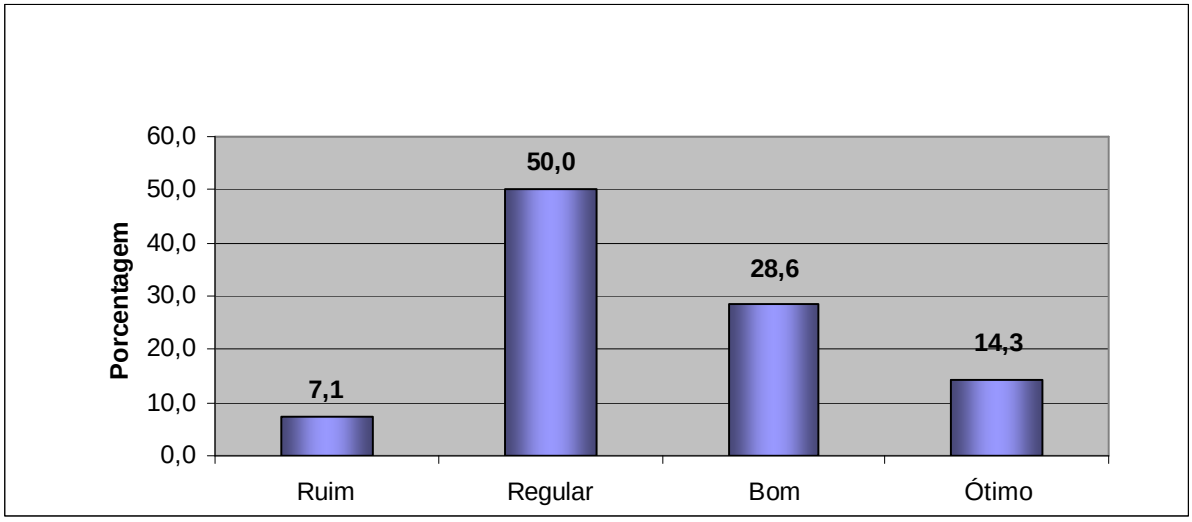
Questão 10 – Como você avalia o teste de *software* da empresa?

Observando os resultados no **Gráfico 8**, verifica-se insatisfação dos profissionais quanto ao teste de software da empresa, pois somando os resultados "ruim"e "regular", obtém-se 57,1%; enquanto a somatória das respostas "bom" e "ótimo" totaliza 42,9%.

Buscando observar esses resultados de outra perspectiva, atribuiu-se pesos um, dois, três e quatro para "ruim", "regular", bom" e "ótimo", respectivamente obtendo-se a média de 2,5. Tal resultado situa-se entre regular e bom.

Das duas análises efetuadas acima, constata-se que, seja em porcentagem , seja em peso (ou em ambos, simultaneamente), os resultados oferecem um indicativo significativo de insatisfação por parte de mais da metade dos profissionais pesquisados quanto à qualidade do teste de *software*.

Gráfico 8: Avaliação dos testes pelos usuários pesquisados



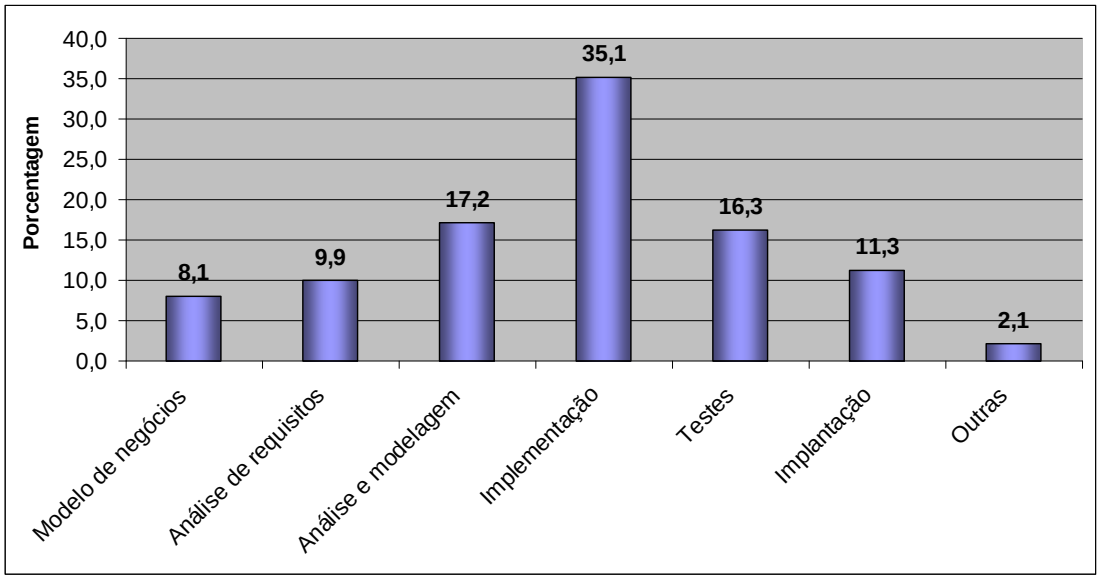
Fonte: Pesquisa de campo, 2009.

Questão 11 – Qual a porcentagem de tempo e recurso que cada fase do *ciclo de vida* consome? (A soma deve ser 100%).

Esta questão tem por objetivo observar o tempo médio gasto em cada uma das fases do *ciclo de vida* do *software,* nas empresas que responderam a pesquisa.

Pode-se observar no **Gráfico 9** que a maior parte do esforço de trabalho concentra-se na fase de *implementação* – 35,1%, e em segundo lugar, quase empatados, as fases de *análise e modelagem de dados* e de *testes de software*.

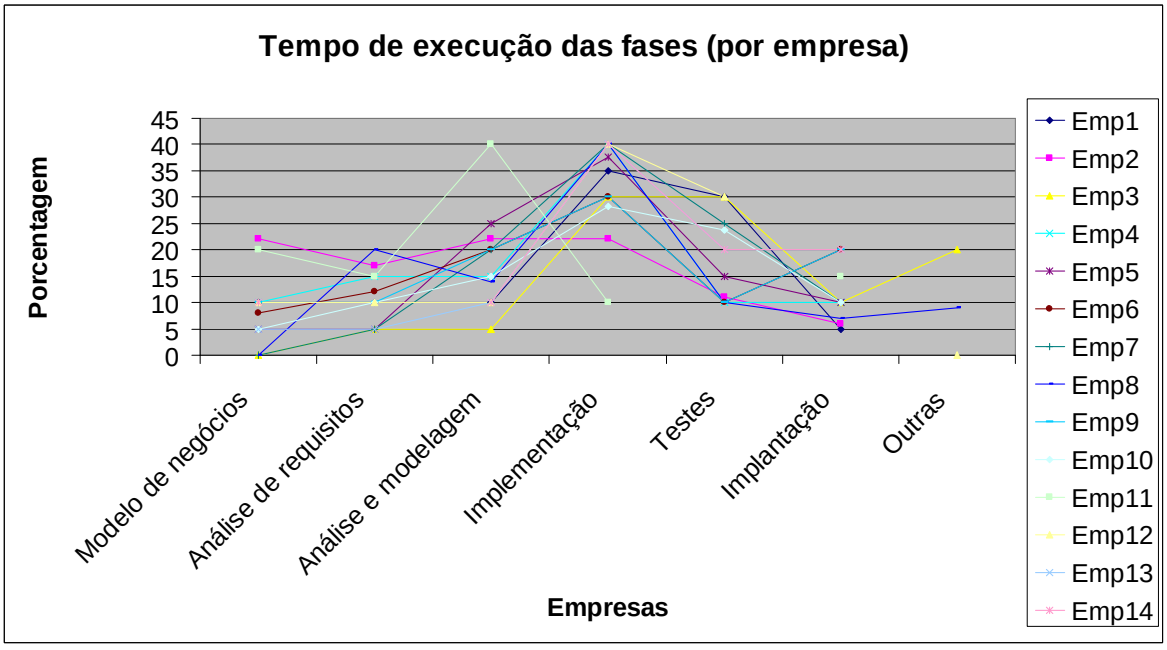
Gráfico 9: Tempo médio de duração das fases do ciclo de vida



Fonte: Pesquisa de campo, 2009.

No gráfico 10, pode-se observar que as empresas que gastam mais tempo nas fases iniciais como *modelo de negócios*, *análise de requisitos*, *análise* e *modelagem de dados* gastam menos tempo na implementação e no teste de *software*.

Gráfico 10: Tempo de duração das fases do ciclo de vida



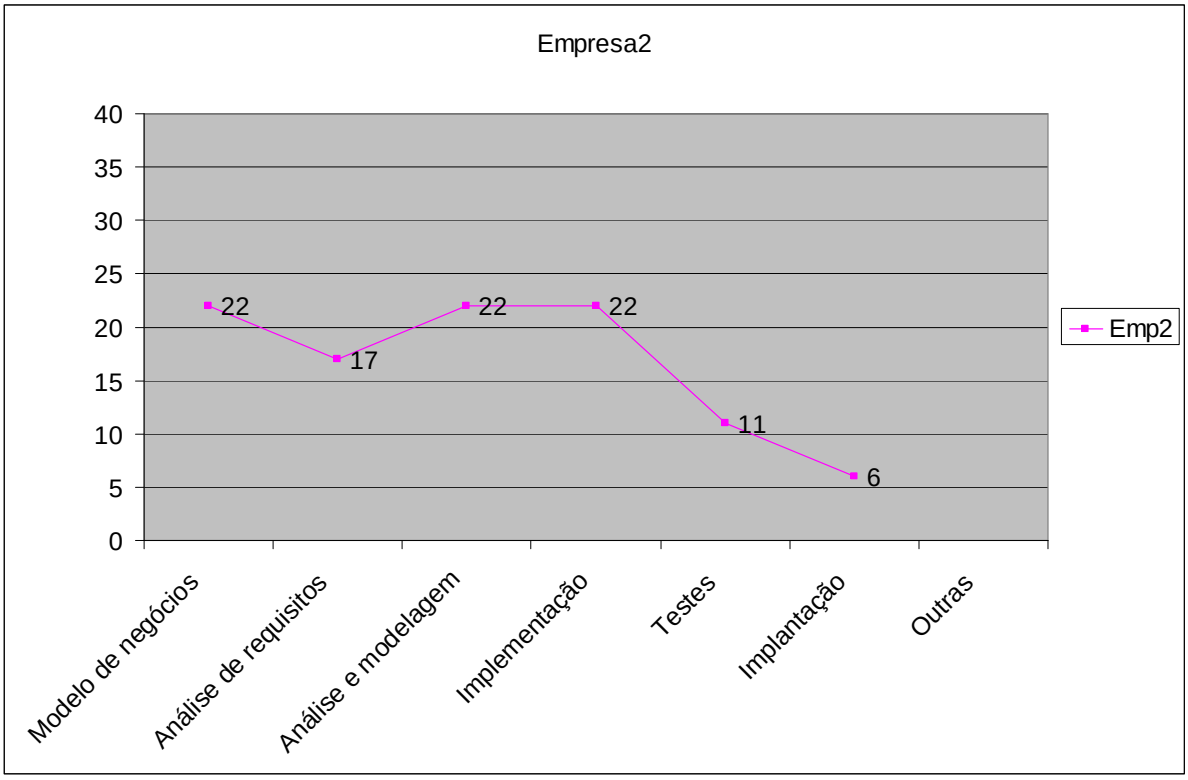
Fonte: Pesquisa de campo, 2009.

Para apresentar melhor o **Gráfico 11** apresenta o tempo gasto em cada fase do ciclo de vida da empresa 2, que possui certificação CMMI e utiliza a Norma ISO/IEC 12207.

Comparando com o gráfico que apresentou a média de cada fase do ciclo de vida, pode-se observar que as fases iniciais como *Modelo de negócios*, *Análise de requisitos* e *Análise* e *modelagem de dados os valores estão acima da média e as fases seguintes estão abaixo.*

Isto se deve ao fato de que a empresa2 possui processos de testes bem definidos e que são executados a cada fase.

Gráfico 11: Tempo de duração das fases do ciclo de vida na empresa 2

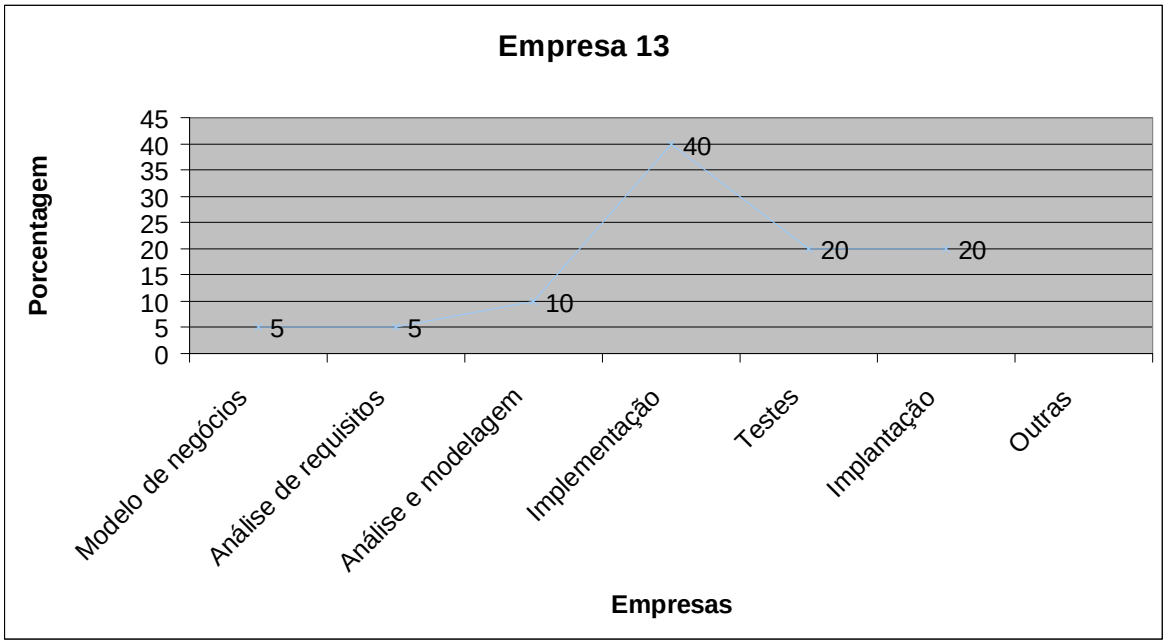


Fonte: Pesquisa de campo, 2009.

Em contraste o **Gráfico 12** apresenta o tempo gasto em cada fase do ciclo de vida da empresa 13, que não possui certificação CMMI e não conhece a Norma ISO/IEC 12207.

Comparando com o gráfico que apresentou a média de cada fase do ciclo de vida, pode-se observar que as fases iniciais como *Modelo de negócios*, *Análise de requisitos* e *Análise* e *modelagem de dados os valores estão abaixo da média e as fases seguintes estão acima.*

Gráfico 12: Tempo de duração das fases do ciclo de vida na empresa 13



Fonte: Pesquisa de campo, 2009.

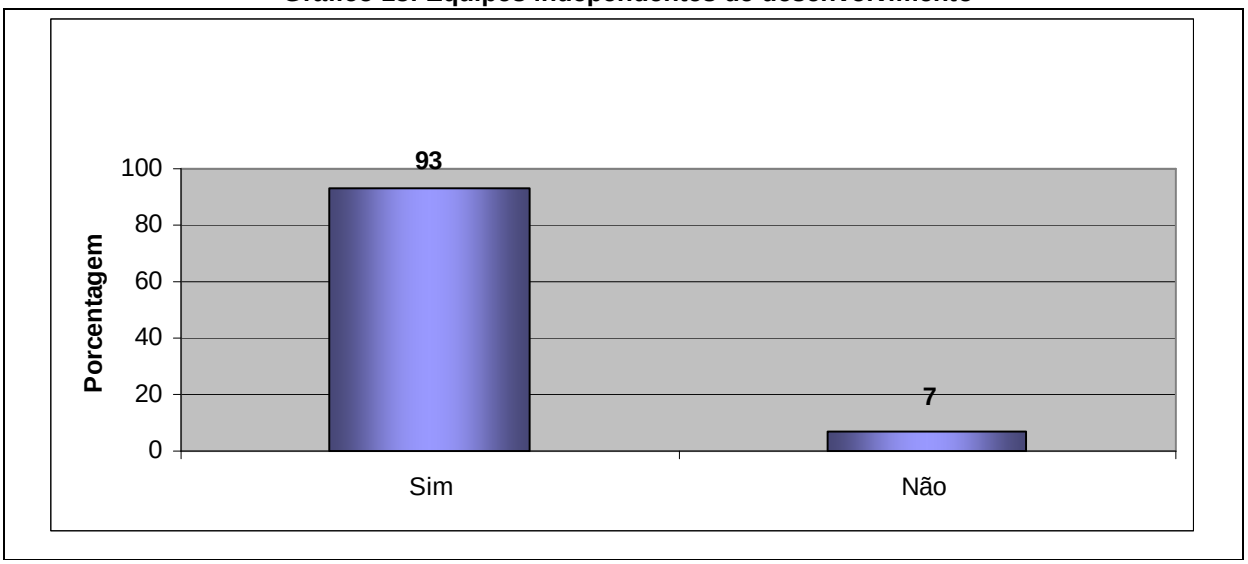
Isto se deve ao fato de que a empresa 13 não possui processos de testes estruturados e que não são executados a cada fase, assim a fase de teste fica sobrecarregada e retorna os erros encontrados para a fase de implementação sobrecarregando esta também .

Questão 12 – Possui equipe de teste independente da equipe de desenvolvimento?

Esta questão tem por objetivo identificar se a equipe de teste é independente da equipe de desenvolvimento, nas empresas que responderam a pesquisa.

Pode-se observar no que a maioria das empresas pesquisadas respondeu possuir equipes de *desenvolvimento* e *teste de software* independente.

Gráfico 13: Equipes independentes de desenvolvimento



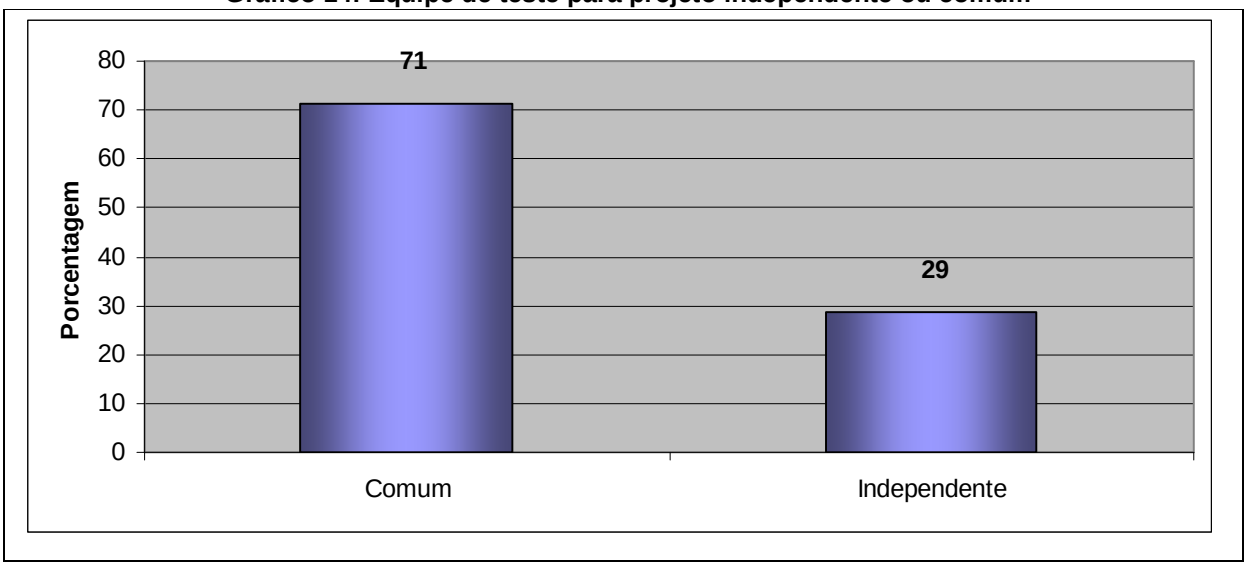
Fonte : Pesquisa de campo, 2009

Questão 13 – Possui uma equipe de testes: comum ou independente?

Esta questão tem por objetivo identificar se as empresas possuem equipes de *teste de software* independentes para cada projeto desenvolvido na empresa, ou equipes de *teste de software* em comum para todos os projetos desenvolvidos na empresa.

Pode-se observar no **Gráfico 14** que a maioria das empresas pesquisadas responderam possuir equipes de *teste de software* independentes para cada projeto.

Gráfico 14: Equipe de teste para projeto independente ou comum



Fonte: Pesquisa de campo, 2009.

Questão 14 – Utiliza *software* de teste automatizado?

Observa-se que a maioria das empresas pesquisadas respondeu possuir ferramenta de teste automatizado, sendo citadas: Jmeter, *Quick Test Professional* (QTP), *Rational Robot*, HP *Mercury Winrunner*, *Tester Director*, IBM *Rational Functional Tester*, *TestComplete*, *X-Zone Solutions Framework*.

Cita-se que algumas empresas responderam utilizar mais de uma ferramenta. A título de exemplo, uma delas respondeu utilizar o *Test Complete* para testes de funcionalidade em *desktop* juntamente com o *X-Zone Solutions Framework* para gerenciar os testes.

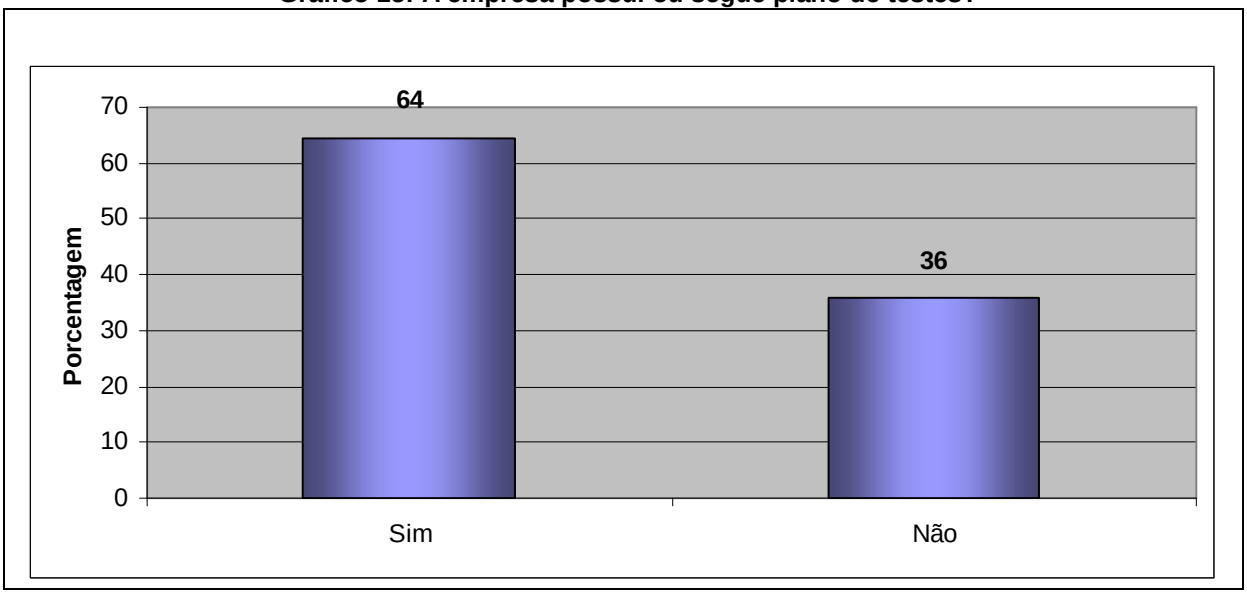
Uma das empresas respondeu usar todas as ferramentas possíveis e justificou:

> "Levar em consideração que a empresa fornece entre dezenas de outros serviços, o teste de *software* como serviço a ser adquirido pelas empresas, ou seja, nós possuímos uma Fábrica de Testes, da qual sou Diretora.
>
> Portanto, quando especificamos todas as ferramentas, é porque nossa Carteira de Clientes exige que a equipe tenha conhecimentos amplos sob todas as ferramentas de mercado."

Questão 15 – Possui ou segue algum plano de testes?

Pode-se observar no **Gráfico 15** que a maioria das empresas pesquisadas respondeu possuir e/ou seguir plano de testes.

Gráfico 15: A empresa possui ou segue plano de testes?



Fonte: Pesquisa de campo, 2009.

As empresas que possuem certificação CMMI ou MPS-Br e implementaram a Norma ISO/IEC 12207 possui plano de teste bem definidos.

A empresa com certificação MPS-Br respondeu a pergunta acrescentando:

> "Após realizar a análise de requisitos do sistema, é feito um teste de verificação do documento pelos arquitetos de software e outra verificação pelos analistas de testes.
>
> Logo após o desenvolvimento (codificação) é feito a verificação de código pelos desenvolvedores.
>
> Depois do fechamento do desenvolvimento é realizado o teste de funcionalidade de cada alteração.
>
> Para finalizar, são realizados os testes de sistema para garantir o conjunto das funcionalidades.
>
> Atualmente com a implantação de ferramentas de automação de testes estamos realizando os testes de regressão constantemente para garantir que o que já testamos um dia continue funcionando."

Enquanto que uma das empresas que possui certificação CMMI respondeu:

> "1- Após a análise de negócio produzida e aprovada pela equipe de analistas, faz-se uma documentação técnica detalhada com todos os processos e requisitos descritos para que seja passado à equipe de desenvolvedores.
>
> 2- Esta documentação é analisada novamente pelos desenvolvedores para garantir que tudo que está descrito está correto e se é possível de ser feito. Caso nenhum erro seja encontrado, esta documentação é passada para uma espécie de "fábrica de desenvolvimento", chamada off-shore (todo e qualquer desenvolvimento customizado é feito na Índia).
>
> 3- Feito isso, o off-shore entra em contato com a equipe de desenvolvedores para que eles testem o que foi desenvolvido de acordo com a documentação (é feito um teste unitário).
>
> 4- Se estiver tudo OK, o próximo passo é passar o que foi desenvolvido para a equipe de testes, que testa funcionalmente o programa de acordo com o teste script (documentação de teste desenvolvida pela equipe de testes).
>
> 5- Se nenhum problema for encontrado a implantação em produção é feita.
>
> PS. Todos os passos foram descritos considerando que nenhum erro foi encontrado, casa isso ocorra, deve-se voltar ao passo anterior e repeti-lo até que esteja OK para seguir o próximo passo."

Questão 16 – Responda "sim" ou "não" aos tipos de teste utilizados na empresa. E indique em que fase ocorre.

Esta questão tem por objetivo identificar quais tipos de teste que as empresas utilizam. Pode-se observar no **Gráfico 16** que, com maior ou menor frequência de uso, todas as empresas responderam utilizar todos os tipos de testes propostos no questionário.

Os tipos de teste *funcional, caixa preta* e *integração* destacaram-se, tendo sua utilização sido reconhecida por 100% das empresas. Em segundo lugar aparece os testes de *Positivo/Negativo* e *Sistema*.

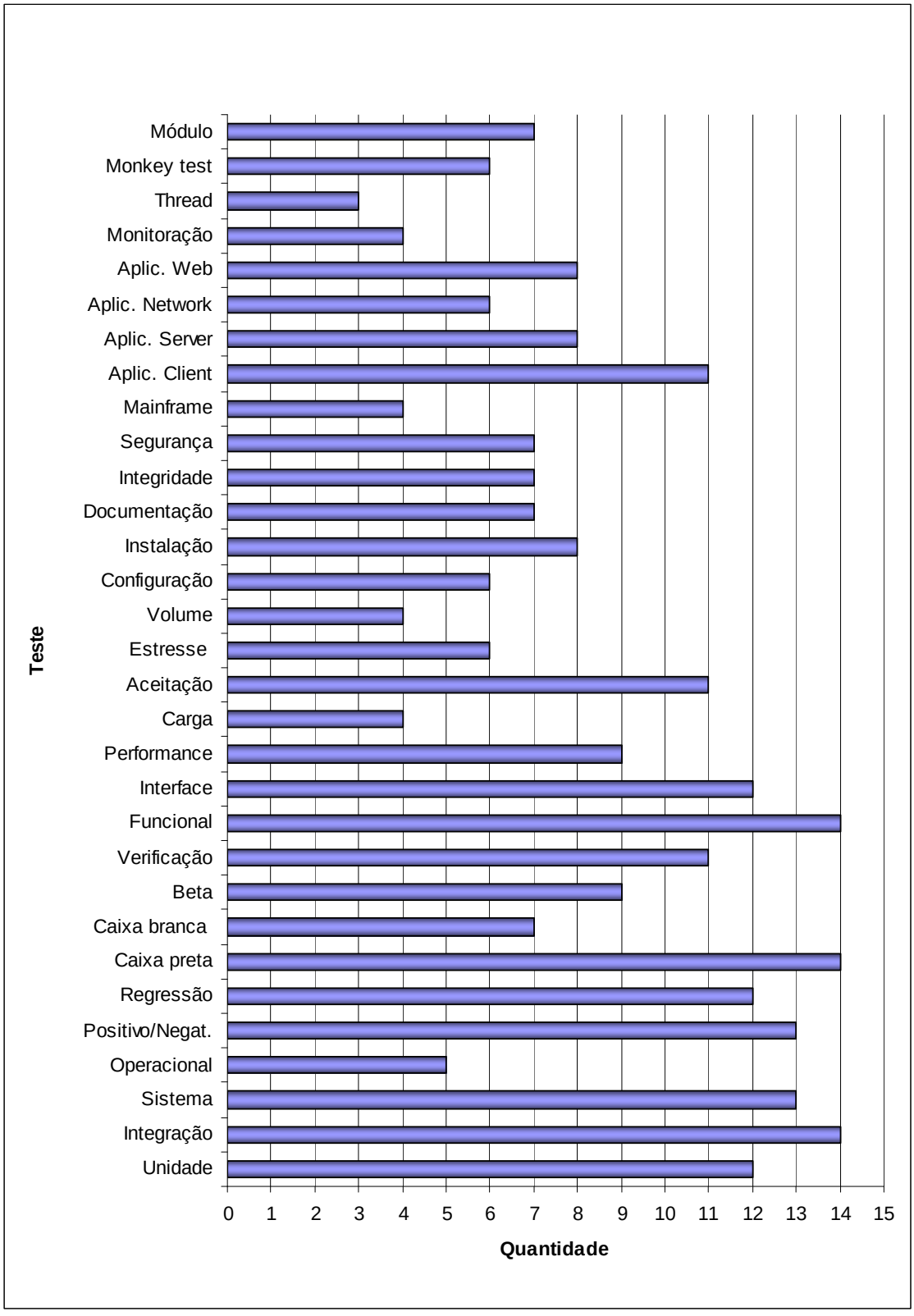
Os testes menos executados, como *Thread, Monitoração, Mainframe, Volume, Carga e Operacional* são importantes, mas por serem específicos, muitas vezes não há necessidade de executá-los.

Um caso em particular desperta a atenção, o teste de *Aplic. Network* é aplicado apenas por seis empresas, tendo em vista a realidade atual de que os sistemas são executados em redes, por mais simples que esta seja.

Mas a explicação deve-se ao fato de que este teste é executado quando os recursos de redes são fundamentais para o desempenho do software, como velocidade de tráfego da informação, Integridade, segurança entre outros.

Assim o mesmo ocorre com outros testes que possuem um baixo índice de aplicação.

Gráfico 16: Tipos de testes executados



Fonte: Pesquisa de campo, 2009.

5. CONCLUSÃO

O presente estudo pôde oferecer, após revisão da literatura sobre o tema e pesquisa de campo, uma amostra significativa do que ocorre no mercado. Para tanto, baseou-se nas informações indispensáveis, sondadas em pequenas, médias e grandes empresas: *quantidade de clientes em carteira*, *tempo de mercado*, *abrangência de mercado* em escala nacional e internacional.

Validou-se por meio desta pesquisa que todas as empresas reconhecem a necessidade de se produzir software com qualidade e que, na elaboração do *software* subdividem as etapas de forma a dispensar a maior parte do tempo na *implementação* do *software*. Sendo que a análise e modelagem do sistema aparecem como a segunda etapa em consumo de tempo e a etapa de teste do *software,* em terceiro. Em algumas empresas o teste do *software* é distribuído ao longo de todas as etapas do *desenvolvimento do software*. A maioria das empresas pesquisadas possui equipe de *desenvolvimento* e *de teste* de *software* independentes uma da outra, mas a *mesma* equipe de *teste de software* em comum para os diversos projetos. As empresas seguem um plano de testes, sendo que grande parte delas faz uso de ferramentas de teste automatizado. Destacam-se quatro tipos de testes utilizados pelas empresas*:* funcional, caixa preta, sistema e integração*.*

Embora se observe que a maioria das empresas não possuem certificação CMMI, conhecem e fazem uso da NBR ISO\IEC 12207, dando origem a metodologias, próprias de testes de *software*. Esse procedimento sugere que a maioria dos profissionais pesquisados reconhece a necessidade de sistematizar a aplicação de modelos de testes para assegurar a qualidade de *software*. Tal sistematização é pressentida como condição de sobrevivência no mercado, tanto para profissionais, quanto para as empresas.

Um dos equívocos mais difundidos sobre qualidade de *software* refere-se à etapa em que ela está inserida no processo do *ciclo de vida*. Pratica-se a “qualidade” apenas no final do processo ou quando o software está parcialmente pronto. Há a necessidade de mudança dessa visão, pois deve-se garantir qualidade em todos os pontos do processo de desenvolvimento do *software,* com vistas a tornar o processo

mais controlado e confiável, possibilitando o completo gerenciamento do projeto de desenvolvimento.

Uma forma de garantir qualidade é elaborar um planejamento adequado de todos os processos. A norma ISO/IEC 12207:1998 mitiga esta tarefa, pois tem por objetivo auxiliar os envolvidos na produção de *software* a definir seus papéis, por meio de processos, tarefas e atividades bem definidos. Desta forma, proporciona às organizações melhor entendimento dos processos a serem desenvolvidos.

No planejamento das empresas de desenvolvimento de *software* é comum que conste, nos cronogramas, a fase específica para os testes. Entretanto, na maior partes das vezes, essa fase é substituída por atividades de correção e manutenção do *software*, ou, então, acaba sendo, por diversos fatores – tais como os atrasos – preterida para dar lugar a outras fases. Apesar de empregada nas organizações, a eficiência das atividades de teste é baixa, o que as incentiva a substituírem tais atividades pela correção de problemas.

Constata-se, por fim, tendo em vista o exposto nas páginas precedentes, a necessidade dos profissionais e das empresas atualizarem-se e utilizarem as normas e procedimentos de qualidade de produção exigidas pelo mercado.

Como sugestões para pesquisas futuras, a área de testes é muito abrangente tendo ainda muito a ser estudado, principalmente o novo conceito que está surgindo de Fábrica de testes.

REFERÊNCIA

ABNT (Associação Brasileira de Normas Técnicas). **NBR ISO/IEC 12207:1998 - Tecnologia de informação** - Processos de ciclo de vida de software. Rio de Janeiro: ABNT,1998.

_____. **NBR 13596:1996 - Tecnologia de informação - Avaliação de produto de software -** Características de qualidade e diretrizes para o seu uso. Rio de Janeiro: ABNT, 1996.

AGNOL, S. D., HERBERT, J. S. **Utilizacao do TSP Para gerencia de equipes Nível 2 do CMMI**. Artigo publicado no VI Simpósio Internacional de Melhoria de Processos de Software – SIMPROS. 2004.

ALGARTE, Waldir; QUINTANILHA, Delma. **A história da Qualidade e o Programa Brasileiro da Qualidade e da Produtividade**. Rio de Janeiro: Inmetro/Senai, 2000.

BARTIÉ, Alexandre. **Garantia da qualidade de software**. Rio de Janeiro: Editora Campus, 2002.

BOAS, André Luiz de Castro Villas. **Gestão de configuração para teste de software**. Dissertação de mestrado. Universidade Estadual de Campinas, 2003.

BOEHM, B. **A Spiral Model of Software Development and Enhancement**. IEEE Computer. Pp 61-72, 1988.

BRYMAN, A. **Research methods and organization studies**. London: Unwin Hyman Ltd, 1989.

CHIAVENATO, Idalberto. **Introdução à teoria geral da administração**: edição compacta. 2ª Edição revista e atualizada. Rio de Janeiro: Campus, 2000.

COLOMBO, Regina Maria Thienne. **Processo de avaliação da qualidade de pacotes de software**. Monografia de Pós Graduação. Universidade Estadual de Campinas, 2004

COSTA NETO. P L. O. **Decisões com Qualidade**. In Qualidade e competência nas decisões. Coord. Costa Neto. P L. O. São Paulo: Blücher, 2007.

DEMING, William E. **Qualidade:** a revolução da administração. Rio de Janeiro: Marques – Saraiva, 1990.

EISENHART. M. **The ethnographic research tradition and mathematics education research**. Journal for Research in Mathematics Education. 19, 99-114. 1988.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. São Paulo: Atlas. 2002

INTHURN, Cândida. **Qualidade e teste de software**. Florianópolis: Visual Books, 2001.

ISHIDA, César Augusto Silvestrini. **Aplicações de técnicas de teste de *software* em diferentes arquiteturas de sistemas.** Dissertação (MBA) – Escola Politécnica da Universidade de São Paulo, 2002.

KOSCIANSKI, A., SOARES, M.S.. **Qualidade de software**: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. São Paulo: Novatec. 2006

LAUDON, K C.; LAUDON, J P. **Sistemas de informação**: com internet. Rio de Janeiro: LTC, 1999.

MALHOTRA, N K. et al **Introdução à pesquisa de marketing**. São Paulo: Prentice Hall, 2005.

MACHADO, Cristina A. F. **Normas e modelos de maturidade em qualidade de software**. São Paulo, Pearson Education, 2001.

MOLINARI, Leonardo. **Testes de software** - produzindo sistemas melhores e mais confiáveis. São Paulo: Editora Érica Ltda, 2003.

MORSELLI, Osmil A. **Contribuição ao esclarecimento dos motivos que levam as empresas de pequeno porte a não adotarem padrões de qualidade em desenvolvimento de software.** Dissertação de Mestrado, Instituto de Pesquisas Tecnológicas do Estado de São Paulo, 2004

NAKANO, D. N.; FLEURY, A. C. C. **Métodos de Pesquisa na Engenharia de Produção**. In;: XVI ENEGEP – Encontro Nacional de Engenharia de Produção, Anais. Piracicaba: UNIMEP/ABEPRO. 1996

PESSÔA, Marcelo; SPINOLA, Mauro M. **Normas e modelos de maturidade em qualidade de software**. São Paulo: Pearson Education, 2001.

PETERS, J. F., PETDYCZ, W. **Engenharia de software: teoria e prática.** Tradução Ana Patrícia Garcia. Rio de Janeiro: Campus, 2001

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Makron Books Ltda, 1995.

_____. **Software Engineering**: A Practioner's Approach, New York: McGraw-Hill, 2006.

ROCHA, ANA. R. C et al. **Qualidade de software** - Teoria e prática. São Paulo: Pearson Education, 2001.

SANTOS, Aldemar de Araújo. **Informática na empresa**. São Paulo: Atlas 2003.

SCALET, Danilo. **Normas de qualidade dos produtos de software in qualidade de software**. São Paulo: Pearson Education, 2001.

SOMMERVILLE, Ian. **Engenharia de software**. Addison-Wesley, São Paulo,2003.

SPÍNOLA, Mauro de Mesquita. **ISO 9000 para software**. Lavras. UFLA/FAEPE, 2005

STANDISH, Group International. **Extreme Chaos**. USA: The Standish Group, 2001.

TURCHETTI, Rogério. **Estudo sobre a atividade de teste de software**. Trabalho final de Graduação. Centro Universitário Franciscano. 2003

UKUMA, Luciano Hayato. **Uma estratégia para o desenvolvimento de componentes de sofware autotestáveis.** Dissertação de mestrado, Universidade Estadual de Campinas, 2002.

WEINBERG, Gerald M. **Software com qualidade** – Volume 2: Medidas de primeira ordem. São Paulo: Makron Books, 1994.

WEBER, K C, NASCIMENTO, C J do, MARINHO, D S. **Programa brasileiro da qualidade e produtividade em software**: treze anos acompanhando e disseminando a cultura da qualidade. Revista ProQualiti – Qualidade na Produção de Software Volume 2 — Número 1 — Maio 2006. Lavras. Editora UFLA. 2006

YIN, Robert K. **Estudo de caso. Planejamento e métodos.** 3ª edição. Porto Alegre: Bookman. 2005.

APÊNDICE

QUESTIONÁRIO UTILIZADO PARA PESQUISA

1. Tipo de sistema oferecido?

2. Natureza

Número estimado de clientes em carteira	
Número estimado de usuários do sistema	
Tempo de mercado	
Região de atendimento	

3. Quantidade de Funcionários:

Área comercial	
Desenvolvedores	
Suporte	
Equipe de teste	
Gerente de projetos	
Outros	
Total	

4. Em qual das áreas acima você atua?

5. O que você entende por qualidade de software?

6. Quais os passos de desenvolvimento de *software* da empresa?

7. Qual o porte da empresa onde trabalha?

( ) Pequeno	**( ) Médio**	**( ) Grande**

1. Possui certificação CMMI?

( ) Sim Nível: _____
( ) Não

8. **Conhece a norma NBR ISO/IEC 12207 - Tecnologia de informação – Processos de *ciclo de vida* de *software*?**

( ) Sim
( ) Não

9. **Como você avalia o teste de *software* da empresa?**

( ) Ruim	**( )** Regular	**( )** Bom	**( )** Ótimo

10. **Qual a porcentagem de tempo e recurso que cada fase do *ciclo de vida* consome? (A soma deve ser 100%).**

Modelo de negócios	
Análise de requisitos	
Análise e modelagem	
Implementação	
Testes	
Implantação	
Outra Qual?_____	

11. **Possui equipe de teste independente da equipe de desenvolvimento?**

( ) Sim
( ) Não

12. **Possui uma equipe de testes:**

( ) Independente para cada projeto
( ) Em comum para todos os projetos

13. **Utiliza *software* de teste automatizado:**

( ) Sim Qual: _____
( ) Não

14. **Possui ou segue algum plano de testes?**

( ) Não
( ) Sim

15. **Responda sim ou não nos tipos de teste utilizados na empresa. E indique em que fase ocorre.**

Sim / Não	**Fase que ocorre?**	**Tipos de testes**	**Descrição**
Primeira dimensão: Estado do teste ("o momento") Fase?			
( )		Teste de unidade	Teste de nível ou classe. É o teste que tem por objetivo verificar um "pedaço do código".
( )		Teste de integração	Garante que um ou mais componentes combinados (ou unidades) funcionam. corretamente.
( )		Teste de sistema	A aplicação tem que funcionar como um todo. A

			aplicação tem que "fazer aquilo a qual foi projetada para fazer".
Segunda dimensão: Técnica do teste ("como testar")			
( )		Teste operacional	Garante que a aplicação pode "rodar" muito tempo sem falhar.
( )		Teste negativo-positivo	Garante que a aplicação vai funcionar no "caminho feliz" de sua execução e vai funcionar no seu fluxo de exceção.
( )		Teste de regressão	Um dos mais importantes testes. "Para ir para o futuro, tem-se de voltar ao passado, sempre". Toda vez que se for inserir uma característica nova na aplicação, deve-se testar toda a aplicação. Afinal, pode-se, ao "consertar algo, quebrar outro".
( )		Teste de caixa-preta ou *Black-box test*	Testar todas as entradas e saídas desejadas. Não se está preocupado com o "código".
( )		Teste de caixa-branca ou *White-box test*	O objetivo é testar o código. Às vezes existem partes do código que nunca foram testadas.
( )		Teste *beta*	O objetivo é testar a aplicação em produção.
( )		Teste de verificação de versão	Toda vez que se libera uma nova versão de aplicação existem, condições mínimas que validem se a versão liberada está Ok. Este teste é usado durante o processo de construção da aplicação. Pode querer testar, às vezes, apenas uma parte da aplicação.
Terceira dimensão: Metas do teste ("o que testar")			
( )		Teste funcional	Testar se as funcionalidades presentes na documentação funcionam como especificadas. Incluem-se as regras de negócio.
( )		Teste de interface	Verificar se a navegabilidade dos objetos de telas funciona corretamente, em conformidade com os padrões vigentes (em nível de interface).
( )		Teste de performance	Verifica se o tempo de resposta é o desejado para "momento" de utilização da aplicação e suas respectivas telas envolvidas.
( )		Teste de carga	Verifica se a aplicação suporta a quantidade de usuários simultâneos requeridos.
( )		Teste de aceitação do usuário ou UAT (*User Acceptance Test*)	A meta é clara. É um teste exploratório voltado para validar aquilo que o usuário deseja, tendo um objetivo claro: dar o aceite ou não.
( )		Teste de estresse	Testar a aplicação em situações inesperadas.
( )		Teste de volume	Testar a quantidade de dados envolvidos (pode ser pouca, normal, grande ou além de grande).
( )		Teste de configuração	Testa se a aplicação funciona corretamente em diferentes ambientes de hardware e de *software*.
( )		Teste de instalação	Verificar se a instalação da aplicação (hardware e *software*) foi Ok.
( )		Teste de documentação	A documentação existe? Mostra o que o *software* faz efetivamente? Falta algo na documentação?
( )		Teste de	O objetivo é testara integridade dos dados

		integridade	armazenados.
( )		Teste de segurança	Testar a segurança da aplicação nas mais diversas formas.
Quarta dimensão: Onde ocorrerá o teste ("O ambiente")			
( )		Teste de aplicações Mainframe	Teste de aplicações mainframe requer um formalismo e um forte planejamento de testes.
( )		Teste de aplicações Client	Neste momento se está preocupado mais com a funcionalidade, com a interface e a performance do que outras coisas.
( )		Teste de aplicações Server	Neste momento se esta preocupado mais com desempenho dos processos que com a integridade dos dados.
( )		Teste de aplicações Network	Análise estatística do desempenho das aplicações é enfoque pouco utilizada, daí o fato de certas aplicações serem testadas em produção.
( )		Teste de aplicações Web	Na verdade desfez-se um mito, que não somente a interface é fundamental, mas o desempenho e a adequação das necessidades aliadas a uma alta flexibilidade das ferramentas envolvidas são fator-chave de sucesso. Um planejamento estratégico dos testes passou a ser fundamental.
( )		Teste de monitoração	São, na realidade, testes funcionais, que visam verificar o status e disponibilidade de diversas funcionalidades e da aplicação em si.
( )		Teste de ameaça ou *Threat*	Semelhante ao de segurança, mas em escala mais em nível de falhas.
( )		*Monkey test*	Testa o aplicativo de forma aleatória e inesperada. O "teste do macaco" é antes de tudo "sem planejamento".
( )		Teste de módulo	Teste de um módulo, porém em nível menor. Semelhante ao de unidade, porém é mais abrangente que este.