Capítulo 10

Qualidade de Produtos de Software

Renata Bezerra e Silva de Araújo[1], *Virgínia Carvalho Chalegre*[2]

O objetivo deste capítulo é explicar as diversas formas de se melhorar a qualidade do software, através de normas de qualidade, de técnicas de inspeção e testes de software e de modelos de melhoria do processo de testes. O conteúdo principal deste capítulo está dividido nas seguintes seções:

Seção 10.1 – Introdução: uma breve introdução ao capítulo.

Seção 10.2 – Modelos de qualidade de produto: Serão apresentadas normas que objetivam garantir a qualidade do software.

Seção 10.3 – Teste de Software: Serão apresentados tipos, abordagens, estágios e processo de testes.

Seção 10.4 – Inspeção de Software: Nesta seção, serão apresentadas as etapas da inspeção de software e as ferramentas de apoio à inspeção.

Seção 10.5 – Modelos de Maturidade de Testes de Software: Serão abordados os modelos de maturidade de teste de software para avaliar e melhorar o nível de qualidade dos processos de testes aplicados numa organização desenvolvedora de software.

10.1. Introdução

Desde que a ABNT (Associação Brasileira de Normas Técnicas) foi criada, a preocupação com qualidade no Brasil vem se ampliando. A indústria busca continuamente aprimorar seus produtos e alinhar critérios com os padrões mais rigorosos em uso no mundo. A qualidade, atualmente, é percebida como um objetivo de negócio. Maior qualidade afinal significa cliente satisfeito.

Sob a perspectiva de software, o assunto qualidade é bastante extenso. Para cada aspecto diferente do ciclo de vida de um produto, há dezenas de técnicas e ferramentas, visando apoiar os desenvolvedores. Existem soluções para agilizar tarefas, guiar profissionais ao aplicarem uma metodologia ou mesmo analisar o produto e suas especificações em busca de falhas potenciais.

Um problema fundamental da qualidade de software é definir claramente os objetivos que se pretende atingir com um projeto. Para fazê-lo, é preciso enumerar características desejáveis do software, que idealmente devem incluir valores quantitativos e qualitativos a serem respeitados.

[1] rbsa@cin.ufoe.br
[2] vcc@cin.ufpe.br

Quando um produto é medido em relação a uma série de características, é possível realizar um diagnóstico mais preciso de sua qualidade. As medidas também podem ser usadas para uma definição mais precisa de requisitos, fixando-se no início do projeto, os valores desejados para o produto final.

10.2. Modelos de qualidade de produto

Os modelos de qualidade objetivam avaliar o produto de software, segundo diferentes aspectos baseados na visão do usuário. Para padronizar internacionalmente as características de implementação do software, foram criadas algumas normas que serão vistas a seguir.

10.2.1. ISO 9126

A norma 9126 é um conjunto de atributos que têm impacto na capacidade do software de manter o seu nível de desempenho dentro de condições estabelecidas por um dado período de tempo [Cortes 2009]. A ISO 9126 é dividida em quatro partes:

- ISO/IEC 9126-1:2001

  Esta parte da norma é referente ao modelo de qualidade e foca nas diretrizes de uso e características de qualidade de produto de software que serão apresentadas neste capítulo.

- ISO/IEC TR 9126-2:2003

  Esta parte da norma é referente às métricas externas que têm como propósito determinar a taxa de implementação das funções definidas na especificação de requisitos e o controle de ocorrência de falhas.

- ISO/IEC TR 9126-3:2003

  Esta parte da norma é referente às métricas internas que têm como objetivo verificar se as funções são adequadas ao que foi especificado, determinar o número de falhas previstas, estimar o tempo de resposta da aplicação e verificar se as alterações são registadas adequadamente.

- ISO/IEC TR 9126-4:2004

  Esta parte da norma é referente às métricas de qualidade em uso baseadas na eficácia, produtividade, segurança e satisfação do usuário.

Esta seção vai detalhar as características da ISO/IEC 9126-1 pelo fato desta ser focada em modelo de qualidade, tema deste livro.

Diretrizes para uso da norma NBR ISO/IEC 9126-1

Esta norma pode ser aplicada nas seguintes situações [Cortes 2009]:

- Definição dos requisitos de qualidade de um produto de software;
- Avaliação da especificação de software para verificar se ele irá satisfazer aos requisitos de qualidade durante o desenvolvimento;
- Descrição de particularidades e atributos do software implementados, por exemplo, em manuais de usuário;

- Avaliação do software desenvolvido antes da entrega ao cliente;
- Avaliação do software desenvolvido antes da aceitação pelo cliente.

Características e subcaracterísticas de qualidade de software

Os modelos de qualidade, geralmente, representam a totalidade dos atributos do software classificados em uma estrutura de árvore hierárquica de características e subcaracterísticas. O nível mais alto dessa estrutura é composto pelas características de qualidade e o nível mais baixo é composto pelos atributos de qualidade do software.

Esta norma fornece um modelo de propósito geral, o qual define seis amplas categorias de características de qualidade de software, observadas na Tabela 10.1, que podem ser divididas em subcaracterísticas, onde cada uma possui atributos mensuráveis.

O efeito combinado das características de qualidade, em uma situação particular de uso, é definido como qualidade em uso; também devem ser consideradas as qualidades internas e externas, detalhadas abaixo [Guerra & Colombo 2009].

- **Qualidade em Uso**: tem por objetivo satisfazer as necessidades reais de usuários ao utilizar o produto de software, para atingir metas em contextos de uso específicos, ou seja, o efeito da utilização do produto medido com relação às necessidades dos usuários.
- **Qualidade Externa**: tem por objetivo influenciar o comportamento do sistema para satisfazer necessidades explícitas e implícitas, quando o sistema for utilizado sob condições especificadas, ou seja, o efeito da execução das funções medido com relação aos requisitos externos.
- **Qualidade Interna:** atributos estáticos do produto de software que satisfazem as necessidades explícitas e implícitas, quando o sistema for utilizado em condições especificadas, ou seja, o efeito das propriedades de produtos intermediários, medidos com relação aos requisitos internos – projeto e código.

Tabela 10.1. Características de qualidade interna e externa (NBR ISO/IEC 9126-1, 2001)

Características	Definições
Funcionalidade	A capacidade de um produto de software prover funcionalidades que satisfaçam o usuário em suas necessidades declaradas, dentro de um determinado contexto de uso.
Confiabilidade	A capacidade que o produto tem de se manter no nível de desempenho nas condições estabelecidas.
Usabilidade	A capacidade do software ser compreendido, seu funcionamento aprendido, ser operado e ser atraente ao usuário.
Eficiência	Atributos que evidenciam o relacionamento entre nível de desempenho do software e a quantidade

	de recursos usados, sob condições estabelecidas.
Manutenibilidade	Atributos que evidenciam o esforço necessário para fazer modificações necessárias no software.
Portabilidade	Capacidade do software de ser transferido de um ambiente para outro sem ter danos.

Além de definir as características de qualidade de software e orientar quando estas devem ser utilizadas, a norma também apresenta um modelo de qualidade de produto de software. Sua publicação teve o grande mérito de estabelecer um modelo básico de qualidade, transformando em referência conhecida por grande parte da comunidade.

Na Tabela 10.2 são apresentadas as subcaracterísticas para cada característica e suas respectivas definições:

Tabela 10.2 - Características e subcaracterísticas de qualidade[Guerra & Colombo 2009]

Características	Subcaracterísticas
Funcionalidade (satisfação das necessidades)	Adequação - execução do que é apropriado Acurácia - execução de forma correta Interoperabilidade - interação com outros sistemas Conformidade - aderência às normas Segurança de acesso - bloqueio de uso não autorizado
Confiabilidade (imunidade a falhas)	Maturidade - frequência das falhas Tolerância a falhas - reação a falhas Recuperabilidade - capacidade de se recuperar das falhas
Usabilidade (facilidade de uso)	Intelegibilidade - facilidade de entendimento Apreensibilidade - facilidade de aprendizado Operacionalidade - facilidade de operação
Eficiência (rápido e "enxuto")	Tempo - tempo de resposta, velocidade de execução Recursos - quais recursos são utilizados
Manutenibilidade (facilidade de manutenção)	Analisabilidade - facilidade de encontrar falha Modificabilidade - facilidade de modificar Estabilidade - baixo risco de alterações Testabilidade - facilidade de testar
Portabilidade (uso em outros ambientes)	Adaptabilidade - facilidade de se adaptar a outros ambientes Capacidade para ser instalado - facilidade

	de instalar em outros ambientes Conformidade - aderência a padrões de portabilidade Capacidade para substituir - facilidade de ser substituído por outro

10.2.2. ISO 12119

Esta norma foi publicada em 1994 e trata da avaliação de pacotes de software, também conhecidos como "software de prateleira". Além de estabelecer os requisitos de qualidade para este tipo de software, ela também destaca a necessidade de instruções para teste deste pacote. Esta norma divide-se em itens como Escopo, Definições, Requisitos de Qualidade e Instruções para testes, apresentados na Tabela 10.3.

Tabela 10.3. Itens da Norma 12119[Guerra & Colombo 2009]

Requisitos de qualidade	
1. Descrição do Produto	Descreve o produto para ajudar o comprador e servir como base para testes. Deve incluir declarações sobre funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.
2. Documentação do usuário	Deve ser completa, correta, consistente, fácil de entender e capaz de dar uma visão geral do produto.
3. Programas e dados	Descreve em detalhes cada uma das funções do software.
Instruções para teste	
1. Pré-requisitos de teste	Lista de itens necessários para o teste, incluindo documentos, componentes do sistema e material de treinamento.
2. Atividades de teste	Instruções detalhadas sobre os procedimentos de teste, inclusive instalação e execução de cada uma das funções descritas.
3. Registro de teste	Informações sobre como os testes foram realizados. Deve incluir parâmetros utilizados, resultados associados, falhas ocorridas e até a identidade das pessoas envolvidas.

4. Relatório de teste	Relatório incluindo: identificação do produto, hardware e software utilizado, documentos utilizados, resultados dos testes, lista de não conformidade com os requisitos etc.

Esta norma inclui detalhes que devem estar presentes no produto, visando também facilitar o trabalho do avaliador, tais como:

- Documentação do usuário de fácil compreensão;
- Presença de um Manual de instalação com instruções detalhadas;
- Possibilidade de verificar se uma instalação foi bem sucedida;
- Especificação de valores limites para todos os dados de entrada, que deverão ser testados;
- Operação normal mesmo quando os dados informados estão fora dos limites especificados;
- Função de auxílio (help) com recursos de hipertexto;
- Mensagens de erro com informações necessárias para a solução da situação de erro;
- Diferenciação dos tipos de mensagem: confirmação, consulta, advertência e erro;
- Clareza nos formatos das telas de entrada e relatórios;
- Capacidade de reverter funções de efeito drástico;
- Alertas claros para as consequências de uma determinada confirmação;
- Identificação dos arquivos utilizados pelo programa;
- Identificação da função do programa que está sendo executada no momento;
- Capacidade de interromper um processamento demorado;

10.2.3. ISO 14598

É um guia para avaliação de produtos de software, baseado na utilização prática da norma ISO 9126, já que esta define as métricas, características e subcaracterísticas de qualidade de software [Koscianski & Soares, 2007].

Esta norma possui recursos interessantes aos avaliadores, pois trata o processo de avaliação em detalhe. Ela leva em consideração a existência de três grupos interessados em avaliar um software, ou seja, os três tipos básicos de certificação, mostrados na Tabela 10.4:

Tabela 10.4. Tipos de certificação

Certificação	Quem realiza	Finalidade
de 1ª Parte	Empresas que desenvolvem software	Melhorar a qualidade de seu próprio produto
de 2ª Parte	Empresas que adquirem software	Determinar a qualidade do produto que irão adquirir
de 3ª Parte	Empresas que fazem certificação	Emitir documento oficial sobre a qualidade de um software

Esta norma se constitui, na verdade, de seis documentos distintos, relacionados entre si, como mostra a Tabela 10.5:

Tabela 10.5. Documentos da ISO 14598

Norma	Nome	Finalidade
14598-1	Visão Geral	Ensina a utilizar as outras normas do grupo. Define os termos técnicos utilizados nas demais partes, contém requisitos gerais para especificação e avaliação de qualidade de software e esclarece os conceitos gerais.
14598-2	Planejamento e Gerenciamento	Como fazer uma avaliação de forma geral.
14598-3	Guia para Desenvolvedores	Como avaliar sob o ponto de vista de quem desenvolve.
14598-4	Guia para Aquisição	Como avaliar sob o ponto de vista de quem vai adquirir.
14598-5	Guia para Avaliação	Como avaliar sob o ponto de vista de quem certifica.
14598-6	Módulos de Avaliação	Detalhes sobre como avaliar cada característica.

As normas das séries 9126 e 14598 podem ser utilizadas conjuntamente, de acordo com o objetivo da avaliação. A norma NBR ISO/IEC 14598-1 contém conceitos de como avaliar a qualidade de software e define um modelo de processo de avaliação genérico. Juntas, as normas NBR ISO/IEC 14598-2 e NBR ISO/IEC 14598-6 estabelecem itens necessários para o suporte à avaliação. Por outro lado, as normas NBR ISO/IEC 14598-3, NBR ISO/IEC 14598-4 e NBR ISO/IEC 14598-5 estabelecem um processo de avaliação específico para desenvolvedores, adquirentes e avaliadores de software, respectivamente. O relacionamento entre elas pode ser visto na Figura 10.1.

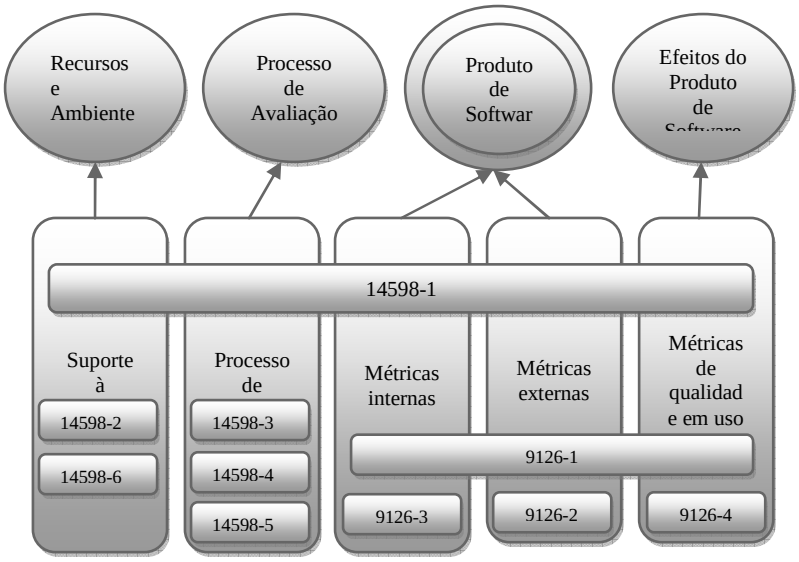

Figura 10.1. Relacionamento entre as séries 9126 e 14598 (Guerra & Colombo 2009)

Em resumo, esta nova norma complementa a ISO/IEC 9126, permitindo que haja uma avaliação padronizada das características de qualidade de um software. É importante notar que, ao contrário da 9126, a 14598 possui detalhes mínimos, incluindo modelos para relatórios de avaliação, técnicas para medição das características, documentos necessários para avaliação e fases da avaliação. Como um exemplo, observe a Tabela 10.6 - um modelo de relatório de avaliação, segundo um anexo da norma 14598-5.

Tabela 10.6. Modelo de relatório de avaliação, segundo um anexo da norma 14598-5

Seção	Itens
1 – Prefácio	Identificação do avaliador Identificação do relatório de avaliação Identificação do contratante e fornecedor
2 – Requisitos	Descrição geral do domínio de aplicação do produto Descrição geral dos objetivos do produto Lista dos requisitos de qualidade, incluindo: - Informações do produto a serem avaliadas - Referências às características de qualidade - Níveis de avaliação
3 - Especificação	Abrangência da avaliação Referência cruzada entre os requisitos de avaliação e os componentes do produto Especificação das medições e dos pontos de verificação Mapeamento entre a especificação das medições com os requisitos de avaliação
4 – Métodos	Métodos e componentes nos quais o método será aplicado
5 – Resultado	Resultados da avaliação propriamente ditos Resultados intermediários e decisões de interpretação Referência às ferramentas utilizadas

10.2.4. Projeto SQuaRE

O projeto SQuaRE (Software product Quality Requirements and Evaluation), traduzindo, Requisitos de Qualidade e Avaliação de Produtos de Software, surgiu a partir da necessidade de se criar um manual de utilização das normas ISO/IEC 9126 e ISO/IEC 14598. Foi feita uma reorganização do material existente sem maiores mudanças. O modelo hierárquico de qualidade proposto na 9126 continuou válido, assim como diversos aspectos organizacionais abordados na série 14598.

A norma SQuaRE surge de uma maneira muito sólida e por diversos motivos, pois abrange amplamente o assunto e estabelece uma base precisa, tanto para definir o modelo quanto para realizar a avaliação. Seus documentos contemplam um certo caráter didático. Houve, por exemplo, a preocupação em fornecer uma extensa lista de exemplos de métricas. Os documentos são resultados de um esforço e consenso de centenas de pesquisadores; representam, assim, uma soma de experiências única no assunto. Outros modelos de qualidade elaborados por pesquisadores de maneira pontual podem ser mapeados para o modelo SQuaRE. [Guerra & Colombo 2009]

Norma SQuaRE (ISO/IEC 25000)

Na reorganização das antigas normas 9126 e 14598, o projeto SQuaRE adotou uma divisão de assuntos em cinco tópicos ilustrados na Figura 10.2.

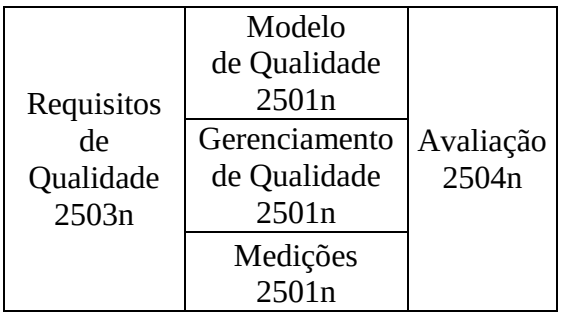

Figura 10.2. Partes componentes da norma SquaRE (Guerra & Colombo 2009)

Cada divisão é composta por um conjunto de documentos e trata de um assunto específico, como apresentadas abaixo:

- **Gerenciamento de Qualidade**: os documentos desta divisão são voltados a todos os possíveis usuários, tais como: gerentes, programadores, avaliadores ou compradores. São definidos os termos utilizados em todos os demais documentos e são feitas recomendações e sugestões de caráter geral sobre como utilizar o SQuaRE.
- **Modelo de Qualidade**: esta divisão corresponde principalmente à ISO/IEC 9126-1. São definidos os conceitos de qualidade externa, interna e em uso, que permitem orientar diferentes perspectivas de avaliação. Por exemplo, desenvolvedores e clientes têm visões e preocupações diferentes relacionadas à qualidade do mesmo produto. É definido um modelo hierárquico de características de qualidade, permitindo que se faça uma descrição extensa e precisa do que cada ator espera de um produto.

- **Medições**: dois pontos importantes fazem parte dessa divisão. Primeiro, definir o que é uma medição e descrever os diversos aspectos relacionados à realização dessa tarefa. Por exemplo: cuidados relacionados com a garantia da precisão dos resultados obtidos. O segundo ponto consiste na proposta de uma série de métricas que podem ser utilizadas ou adaptadas pelos usuários das normas às suas necessidades específicas.
- **Requisitos de Qualidade**: umas das noções importantes introduzidas pela 9126 e retomada pelo projeto SQuaRE é estabelecer objetivos de qualidade para um produto. Isto significa que para garantir a qualidade de um produto, apenas realizar medidas não basta. É preciso que metas tenham sido previamente especificadas. Tais valores fazem parte da especificação dos requisitos do software.
- **Avaliação**: a norma SQuaRE é concretizada na realização de uma avaliação de qualidade a partir de medições cujos resultados devem ser confrontados contra um modelo definido pelo usuário. A divisão de avaliação é direcionada aos diferentes públicos da norma, como desenvolvedores e compradores. São sugeridos procedimentos a serem adotados em cada caso para realizar uma avaliação.

Vale enfatizar que o projeto SQuaRE não surgiu para desvalorizar as normas 9126 e 14598, mas sim para organizá-las.

As características de qualidade das normas, citadas nesta seção, podem ser validadas e implementadas através de testes e inspeções de software que serão detalhados nas próximas seções.

10.3. Teste de Software

Quando um programa está sendo desenvolvido, atividades de verificação e análise devem ser realizadas durante todo o ciclo de vida do software como uma forma de acompanhar a evolução e corretude do produto. Verificação e Validação (V & V) é o processo utilizado para garantir que o produto atenda suas especificações e realizem a funcionalidade esperada pelo cliente. Verificação está relacionada à atividade de avaliar um produto e determinar se ele está de acordo com seus requisitos especificados, enquanto que validação possui um conceito de caráter mais abrangente: garantir que o produto atenda às necessidades dos clientes.

Dentro do processo de V & V existem duas abordagens para verificar e analisar um sistema: Teste de Software e Inspeção de Software [Sommerville 2004]. A primeira é uma técnica dinâmica de verificação e validação, uma vez que seu foco está em exercitar e observar o comportamento operacional do produto. Já a Inspeção de Software verifica os artefatos de um sistema, como o documento de requisitos, diagramas de análise e projeto, e código fonte. Sendo assim, é uma técnica estática de V & V, já que não precisa que o software seja executado. Esta técnica será apresentada na próxima seção.

De um modo simples, um dos principais propósitos dos testes de software é o de executar um sistema desenvolvido de forma a encontrar defeitos que representem qualquer tipo de comportamento não desejado do sistema. Esta definição contrasta

com o outro objetivo de testes: garantir que o sistema faz aquilo que é suposto fazer, ou seja, demonstrar ao cliente que o sistema atende aos seus requisitos funcionais e não funcionais.

O processo de teste de software é, portanto, importante para analisar se a implementação está em conformidade com os requisitos do sistema, para reduzir os custos associados à manutenção e retrabalho, verificar a integração correta entre todos os componentes do software, e especialmente, garantir o aumento da qualidade do software, resultando na satisfação do cliente.

A idéia de encontrar defeitos tem o intuito de reportá-los à equipe de desenvolvimento, de modo que eles possam ser corrigidos. Dessa forma, o produto final deve ter a menor quantidade de defeitos possível, garantindo assim sua qualidade e confiança. É importante ressaltar a impossibilidade de se encontrar todos os erros existentes em um programa através de testes. Sendo assim, é importante saber que os softwares sempre irão conter defeitos, e dessa forma, o objetivo é prover sistemas nos quais os defeitos remanescentes não sejam críticos a ponto de comprometer sua integridade.

De modo a deixar o processo de teste mais robusto e adequá-lo a boas práticas é interessante adotar um modelo de melhoria. Para isto, surgiram modelos de maturidade de teste para avaliar e melhorar os processos de teste de software nas organizações. Na seção 10.4 serão apresentados os 3 principais modelos. Nas próximas subseções serão apresentados conceitos fundamentais inerentes ao processo de teste de software.

10.3.1. Abordagens de Testes

Existem duas abordagens principais de testes: abordagem funcional ("*black box*" ou "caixa preta") e abordagem estrutural ("*White box*" ou "caixa branca") [Sommerville 2004], [Pressman 2002].

- **Caixa preta:** como o próprio nome já sugere, nesta abordagem o testador visualiza o software como uma caixa preta, ou seja, não considera a estrutura interna do programa, a forma que o código foi implementado ou que tecnologia foi utilizada, por exemplo. Considerando os dados de entrada, o objetivo principal é observar as saídas geradas pelo sistema e verificar se estas estão de acordo com o esperado. A **Figura 10.**3 ilustra este tipo de abordagem.

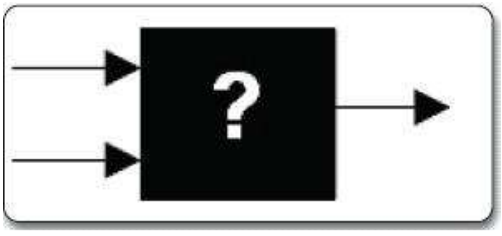

Figura 10.3 Abordagem funcional (Zeilinger 2003)

- **Caixa branca:** diferentemente da abordagem anterior, neste tipo de abordagem o testador está interessado no que está acontecendo "dentro da caixa". É caracterizada por avaliar as funcionalidades internas dos componentes do software, baseando-se no código fonte e procurando exercitar estruturas de controle e de dados do programa. Sendo assim, faz-se necessário

que o analista de testes tenha boa habilidade em programação de modo a entender todos os caminhos lógicos possíveis. A **Figura 10.**4 ilustra a abordagem estrutural.

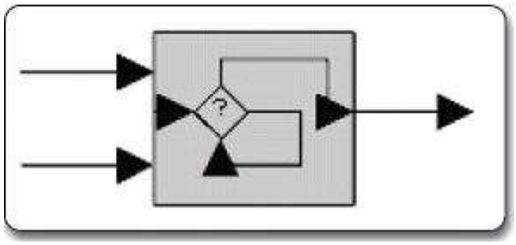

Figura 10.4 Abordagem estrutural (Zeilinger 2003)

10.3.2. Estágios de Testes

Os testes de software normalmente são executados em diferentes estágios durante o ciclo de vida do desenvolvimento do software. Dependendo do objetivo principal do teste, quatro estágios são conhecidos [Graham et. al 2007], [Myers 2004]:

- **Teste de unidade:** realiza testes em componentes individuais (módulos, programas, objetos, classes, etc.) de forma a determinar se cada um deles, separadamente, está sendo executado de maneira correta. Normalmente estes testes são de caixa branca, realizados pelos próprios desenvolvedores do componente. Geralmente utilizam ferramentas que proveem um suporte adicional para verificar a corretude do programa, como ferramenta de *debbuging* ou *framework* para teste unitário, por exemplo. Os defeitos encontrados neste estágio são normalmente corrigidos de imediato, sem a necessidade de documentá-los formalmente, e assim, reduzindo o custo, pois antecipa a correção de defeitos. Geralmente é necessária a utilização de *stubs* (módulos que substituem outros módulos subordinados) e *drivers* (um módulo que substitui outro módulo que seja responsável por controlar a chamada de um sistema), para serem utilizados no lugar dos softwares que estejam eventualmente faltando e para simular a interface entre os componentes de software.
- **Teste de integração:** nesta etapa, as unidades que foram testadas individualmente no estágio anterior são testadas de forma integrada, bem como as interfaces entre os componentes. A integração deve ser realizada adicionando-se os componentes um por um, e após cada passo, um teste é necessário (teste incremental). Esta técnica tem a vantagem de adiantar a detecção de defeitos no processo de testes e corrigi-los mais rapidamente, enquanto é mais fácil determinar as causas dos erros. Por outro lado, tem a desvantagem de ser uma prática bastante custosa, uma vez que sempre que um componente é adicionado, um novo teste é necessário. Sendo assim, a integração pode ser feita basicamente de duas formas: *Top-down* ou *Bottom-up*. Na primeira, os testes são realizados de cima para baixo (começando da GUI ou do menu principal); componentes ou sistemas são substituídos por *stubs*. Na segunda, os testes começam na parte mais básica do sistema até o nível mais alto; componentes ou sistemas são substituídos por *drivers*.
- **Teste de sistema:** neste estágio o propósito do teste está em verificar o funcionamento de todo o sistema, já integrado, e analisar se ele está de acordo com os requisitos que foram especificados. Neste momento, não só são

realizados os testes de integração dos componentes do software entre si, como também destes componentes com um ambiente de teste correspondente à produção final (hardware, software, outros sistemas), de modo a minimizar o risco de que falhas relacionadas com o ambiente operacional do produto não sejam encontradas. Geralmente a estratégia de caixa preta é utilizada neste estágio, mas testes de caixa branca também podem ser realizados.

- **Teste de aceitação:** o teste de aceitação corresponde ao teste realizado pelo usuário de fato do sistema, no momento em que todos ou quase todos os defeitos encontrados nas etapas anteriores já tenham sido corrigidos. O propósito deste teste é estabelecer a confiança do sistema; ele está mais relacionado com a validação do sistema, em que está se tentando determinar se o sistema está de acordo com os requisitos especificados. Normalmente os testes de aceitação podem ser de duas categorias: testes *alfa* e testes *beta*. Os primeiros são realizados nas instalações do desenvolvedor, que fica observando os usuários utilizarem o sistema, e anotam os problemas identificados. Já os testes *beta* são realizados no ambiente real de trabalho do usuário, que instala o sistema e testa, sem a presença do desenvolvedor. Em seguida, um documento contendo os registros dos problemas encontrados é enviado à organização desenvolvedora.

10.3.3. Tipos de Testes

Cada tipo de teste é focado em um grupo de atividades com um determinado objetivo. É necessário pensar em diferentes tipos de testes uma vez que testar a funcionalidade de um componente ou sistema pode não ser suficiente em cada um dos estágios envolvidos para se chegar aos objetivos dos testes. Um tipo de teste é focado em um objetivo particular de teste, que poderia ser um teste de uma função a ser executada pelo componente ou sistema; alguma característica não funcional; a estrutura ou arquitetura do componente ou sistema, etc. Existem vários tipos de testes, dependendo do objetivo de cada projeto e de cada organização. Abaixo serão apresentados alguns dos mais comuns [Graham et. al 2007]:

- **Teste funcional:** este tipo de teste está focado nas regras de negócio do sistema, ou seja, o fluxo de trabalho do programa é avaliado.
- **Teste de recuperação de falha:** o sistema é forçado a falhar de diversas maneiras, de modo a verificar seu comportamento diante destas falhas, e reparar de que formas ele se recupera.
- **Teste de interoperabilidade:** testa um produto de software de modo a determinar sua capacidade de interagir com um ou mais componentes ou sistemas.
- **Teste de segurança:** verifica se o sistema possui atributos para prevenir acessos não autorizados, acidentais ou propositais, a programas e dados.
- **Teste de carga:** um tipo de teste para medir o comportamento do sistema quando este é submetido a níveis altos de carga, diferente das condições normais. É importante determinar o quanto de carga o sistema consegue suportar sem falhar.
- **Teste de performance:** verifica o rendimento de um sistema, como o tempo de resposta e processamento, taxa de transferência de dados, para diferentes

condições (configurações, numero de usuários, etc) as quais o programa é submetido.

- **Teste de estresse:** teste conduzido para avaliar o comportamento do sistema diante de condições que ultrapassem o limite especificado nos requisitos.
- **Teste de configuração:** testa o funcionamento do sistema em diferentes configurações de hardware/software, testando compatibilidade, configuração do servidor, tipos de conexões com a Internet, etc.
- **Teste de usabilidade:** testes para determinar se um produto é facilmente entendível, fácil de aprender, fácil de operar e atrativo aos usuários, ou seja, se o produto tem uma interface amigável para os que utilizarão o sistema.
- **Teste de regressão:** teste de regressão é a atividade de testar uma nova versão de um sistema para validar esta versão, detectando se erros foram introduzidos devido às mudanças realizadas no software, e então, garantir a corretude das modificações. Uma vez que a re-execução de todos os testes é uma atividade bastante custosa, uma seleção de testes de regressão geralmente é realizada.

Processo de testes

O processo de testes pode ser dividido basicamente em cinco etapas: planejamento e controle, análise e projeto, implementação e execução, avaliação de critério de saída e reportagem e atividades de encerramento de testes [Graham et. al 2007]. Estas atividades são logicamente sequenciais, porém, em um projeto específico, podem se sobrepor, serem executadas em paralelo ou até mesmo serem repetidas. Cada uma destas atividades será detalha dos sub-tópicos abaixo.

Planejamento e Controle

Durante o planejamento de testes deve-se ter certeza de que os objetivos dos clientes e *stakeholders* foram entendidos de maneira correta [Graham et. al 2007]. Baseados neste entendimento, os propósitos da atividade de testes propriamente dita são estabelecidos, e assim, uma abordagem e plano para os testes é obtida incluindo especificação das atividades de teste. O planejamento de testes apresenta as seguintes atividades principais:

- **Determinar o escopo e riscos e identificar os objetivos de teste:** são determinados os softwares, componentes, sistemas ou outros produtos que devem ser testados; os riscos que devem ser levados em consideração; e qual o propósito do teste (encontrar defeitos, verificar se está de acordo com os requisitos ou dentro dos padrões de qualidade, etc.).
- **Determinar a estratégia de teste:** aqui serão estabelecidas as técnicas que serão utilizadas, o que precisa de fato ser testado (selecionar e priorizar os requisitos) e que nível de cobertura é necessário. Serão também analisadas quais pessoas precisarão se envolver e em que momento (desenvolvedores, usuários, etc.), incluindo a definição da equipe de teste.
- **Definir recursos:** são definidos todos os recursos necessários durante o ciclo de vida de testes, tanto recursos materiais (PCs, softwares, ferramentas, etc.) como recursos humanos (principais e de apoio).

- **Fazer um cronograma para análise e projeto, implementação, execução e avaliação de teste:** deverá ser elaborado um cronograma de todas as tarefas e atividades, para que seja possível terminar a fase de testes a tempo.
- **Estabelecer os critérios de saída:** critérios de saída, como critério de cobertura, por exemplo, deverão ser estabelecidos de modo a determinar quando a etapa de testes chegou ao fim.

Após planejar é necessária uma medida de controle para verificar se tudo está indo de acordo com o planejado. É preciso comparar o andamento real com o que foi estabelecido no plano de testes, e tomar medidas corretivas quando necessário.

Análise e Projeto

Esta é a atividade em que os objetivos gerais de testes são transformados em condições e projetos de teste tangíveis [Graham et. al 2007]. O propósito principal é identificar e descrever os casos de teste para cada versão de teste, e identificar e estruturar os procedimentos de teste, especificando como executar os casos de teste. As principais tarefas desta etapa podem ser destacadas em:

- **Revisar a base de testes (como a análise de risco do produto, requisitos, arquitetura, especificação de projeto e interfaces):** a base de testes é utilizada para criar os testes. É possível começar a projetar os testes de caixa preta antes da implementação, uma vez que a base de testes pode ser usada para compreender o que o sistema precisa fazer.
- **Identificar e descrever casos de teste:** um caso de teste é um cenário associado a um requisito; é um texto contendo: identificador, objetivo, pré-condições de execução, entradas, passos específicos do teste a ser executado e resultados esperados e/ou pós-condições de execução. Um caso de teste bem projetado tem muita chance de encontrar um erro ainda não conhecido.
- **Estruturar procedimentos de teste:** o passo a passo que descreve como os casos de teste devem ser executados. Inclui o estado inicial da aplicação; condições de funcionamento; como e quando fornecer os dados de entrada e obter os resultados; a forma de avaliar estes resultados, dentre outros.
- **Avaliar a capacidade de testar os requisitos:** a especificação de requisitos deve ser completamente clara, informando as condições necessárias para se definir os testes. Por exemplo, se a performance do software é algo crítico, deve ser claramente especificado o tempo de resposta mínimo em que o sistema deve responder.

Implementação e Execução

Uma vez que os casos e procedimentos de teste foram especificados em alto nível na etapa anterior, este é o momento em que o ambiente será preparado para que eles sejam executados e comparados com os resultados desejados [Graham et. al 2007]. Além disso, é a etapa em que os componentes necessários são implementados para que os testes sejam executados. As principais tarefas destas duas fases serão destacadas a seguir:

- **Implementação:**

o Implementar componentes: efetuar a implementação de novos componentes de apoio necessários à aplicação dos testes, ou modificação de componentes já existentes.

Ferramentas de automação podem ser utilizadas ou os componentes podem ser desenvolvidos explicitamente.

- Criar suítes de teste: baseado nos casos de teste, um conjunto de testes que naturalmente trabalham juntos representa uma suíte de teste a qual é utilizada para uma execução de teste eficiente.
- Implementar e verificar o ambiente: preparar e verificar se o ambiente de teste está funcionando corretamente.

- **Execução:**

- Executar as suítes de teste e casos de teste individuais, de acordo com os procedimentos de teste. Pode ser feito manualmente ou com o auxílio de ferramentas de execução de testes.
- Seguir as estratégias de teste definidas na etapa de planejamento.
- Criar um *log* com as saídas da execução dos testes e registrar os identificadores e versões do software que está sendo testado.
- Fazer a comparação dos resultados esperados e dos resultados obtidos.
- Quando houver diferenças entre os resultados esperados e os resultados obtidos, registrar os defeitos em um repositório centralizado. Não se deve registrá-los de forma aleatória.
- Realização de testes de regressão para confirmar que uma falha anteriormente registrada foi de fato consertada.

Avaliação do critério de saída e relatório

Esta é a fase em que se deseja observar se já foram executados testes suficientes para garantir a qualidade desejada do produto, sendo assim, critérios de saída são definidos com esta finalidade [Graham et. al 2007]. Estes critérios informam se uma dada atividade de testes pode ser considerada completa. As principais atividades são:

- **Verificar se os *logs* de testes batem com os critérios de saída especificados no plano de testes:** procura-se pelos testes que tenham sido executados e avaliados, e se defeitos foram encontrados, consertados ou re-testados.
- **Verificar se será necessária a inclusão de mais testes ou se os critérios de saída especificados devem ser mudados:** mais casos de testes podem precisar ser executados, se por acaso estes não tiverem sido todos executados conforme esperado, ou se for detectado que a cobertura de requisitos necessária ainda não foi atingida, ou até mesmo se aumentaram os riscos do projeto.
- **Escrever um relatório de resumo de testes para os *stakeholders*:** todos os *stakeholders* devem saber quais testes foram executados e quais os resultados destes testes, de modo a perceber que decisões precisam ainda ser tomadas visando a melhoria da qualidade do software.

Atividades de encerramento de teste

A atividade de encerramento de teste pode ser dada através de diversos fatores, como por exemplo, as informações necessárias do processo de testes já foram atingidas; o projeto é cancelado; quando um marco particular é alcançado; ou quando uma versão de manutenção ou atualização está concluída [Graham et. al 2007]. As atividades principais são:

- Verificar se as entregas programadas foram de fato entregues e garantir que todos os problemas reportados foram realmente resolvidos. Para os que permaneceram em aberto devem-se requisitar mudanças em uma futura versão.
- Finalizar e arquivar os artefatos produzidos durante o processo necessário para planejar, projetar e executar testes, como por exemplo, documentação, scripts, entradas, resultados esperados, etc. É importante reutilizar tudo que for possível destes artefatos, pois assim se consegue economizar tempo e esforço do projeto.
- Repassar os artefatos anteriormente citados para a equipe de manutenção, que irá prover suporte aos usuários do sistema e resolver qualquer problema encontrado depois de sua entrega.
- Avaliar como se deu o processo de testes e analisar as lições aprendidas, que serão de grande utilidade para futuras versões dos projetos. Este passo pode permitir não só melhorias no processo de testes, como também melhorias no processo de desenvolvimento do software como um todo.

10.3.4. Testes ao longo do ciclo de vida de Software

As atividades de teste não são atividades que são realizadas sozinhas, mas sim em paralelo com o ciclo de vida de desenvolvimento do software. Dessa forma, a escolha do ciclo de vida do projeto irá afetar diretamente as atividades de teste. O processo de desenvolvimento adotado depende muito dos objetivos e propósitos do projeto. Portanto, o modo como as atividades de teste são estruturadas deve se ajustar ao modelo de desenvolvimento de software, ou do contrario, não conseguirá obter o sucesso desejado.

Um modelo de desenvolvimento de software bastante conhecido é o modelo em cascata, que como o próprio nome já sugere, tem sua base voltada a um desenvolvimento sequencial das atividades. As primeiras atividades começam no topo da cascata, e então vão seguindo sequencialmente através das várias atividades de concepção do projeto, e finalmente terminando com a etapa de implementação. Após isso, é que as atividades de teste são introduzidas, e dessa forma os defeitos só podem ser detectados bem perto da fase de implementação [Graham et. al 2007]. A Figura 10.5 ilustra o modelo em cascata.

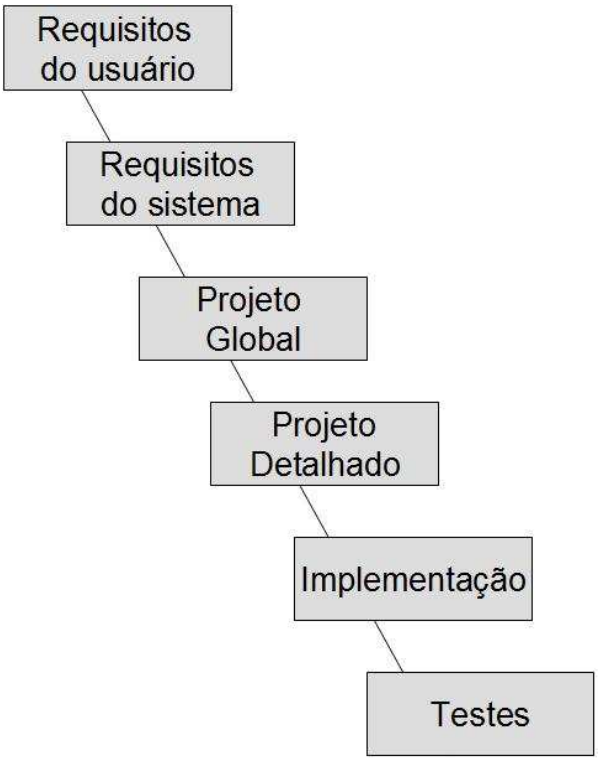

Figura 10.5 Modelo em cascata (adaptado de Graham et. al 2007)

Com o objetivo de tentar contornar os problemas do modelo em cascata, foi desenvolvido o modelo em V **[**Craig e Jaskiel 2002], que foca nos testes do produto durante todo o ciclo de desenvolvimento para conseguir uma detecção adiantada de defeitos. A idéia é que as atividades de testes não são simplesmente uma fase única, mas pelo contrário, como já foi visto na sessão anterior, se faz necessária toda uma preparação, passando por etapas de planejamento, análise, projeto, etc, que devem ser executadas em paralelo com as atividades de desenvolvimento.

Todos os artefatos gerados pelos desenvolvedores e analistas de negócio durante o desenvolvimento, proveem a base de testes em um ou mais níveis. Promovendo as atividades de teste mais cedo, defeitos podem ser geralmente encontrados nos documentos da base de testes. O modelo em V demonstra como as atividades de verificação e validação podem ser executadas em conjunto com cada fase do ciclo de vida de desenvolvimento do software.

Basicamente, o ciclo de desenvolvimento do modelo usa os quatro estágios de teste explicados neste capítulo: testes unitários, testes de integração, testes de sistema e testes de aceitação. Cada estágio pode ser ajustado com as etapas do clico de desenvolvimento de acordo com seus objetivos específicos. Na prática, os estágios de testes podem variar no modelo em V, dependendo dos propósitos do projeto, podendo ser combinados ou reorganizados. O modelo V pode ser visualizado na Figura 10.6.

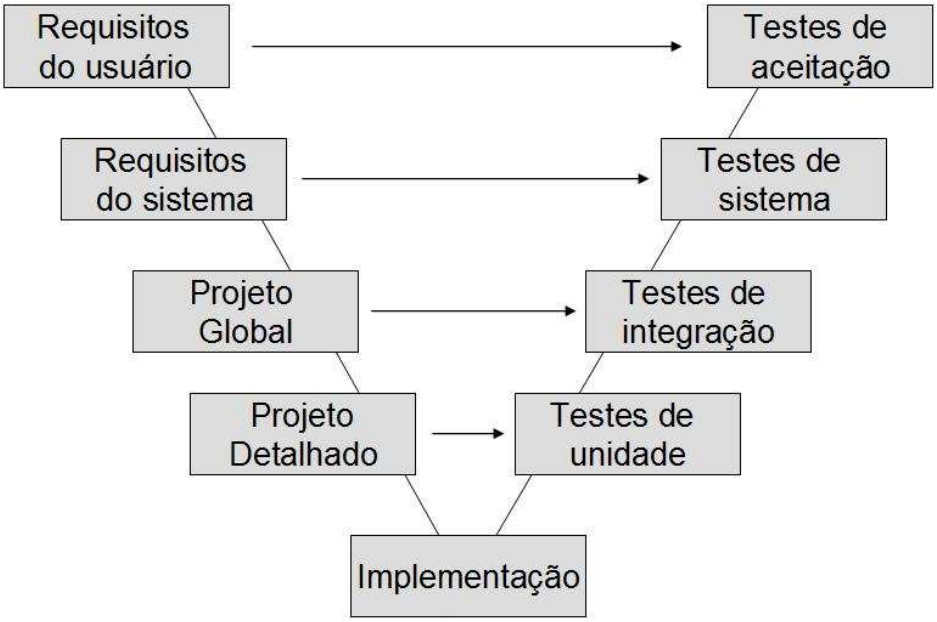

Figura 10.6 Modelo em V (adaptado de Graham et. al 2007)

10.4. Inspeção de Software

Como explicado na sessão anterior, a inspeção de software é uma técnica estática do processo de V & V, em que são efetuadas revisões no sistema com o objetivo de encontrar defeitos e então, corrigi-los. O objetivo principal das inspeções é garantir que defeitos sejam reparados o mais cedo possível no processo de desenvolvimento de software, uma vez que quanto mais evoluído o software estiver, mais difícil será para encontrar os erros e mais custoso ainda consertá-los. Qualquer artefato produzido no desenvolvimento do software pode ser utilizado no processo de inspeção, como requisitos, modelo de projeto ou código.

O modelo CMMI [SEI 2006], abordado no capítulo 8 deste livro, exige a realização de revisões como uma prática específica do processo de verificação, demonstrando assim sua importância na garantia da qualidade do produto. Segundo Fagan, a utilização de inspeções informais de software captura em torno de 60% dos erros em um programa [Fagan 1986]. Mills et al. sugere que uma aplicação mais formal de inspeção de software pode detectar até mais de 90% dos erros de um programa [Mills et al. 1987]. Selby e Basili (1987) comparam empiricamente a efetividade de inspeções e testes. Eles perceberam que a revisão de código estática se mostrava mais efetiva e menos cara do que a procura por erros utilizando testes.

Boehm e Basili ressaltam ainda que inspeções e testes capturam diferentes tipos de defeito e em diferentes momentos do processo de desenvolvimento de software [Boehm e Basili 2001]. Dessa forma, é interessante aplicar tanto inspeções como testes para detectar defeitos em artefatos de software. Começando o processo de V & V com inspeções, defeitos podem ser encontrados e corrigidos nas etapas iniciais do processo de desenvolvimento do software, e uma vez que o sistema esteja integrado, é necessária a introdução de testes para verificar as propriedades do sistema e ver se este está de acordo com o desejo do cliente.

10.4.1. A Equipe de Inspeção (Participantes)

A equipe de inspeção é composta por um pequeno grupo de pessoas que possuam interesse e conhecimento do produto. Geralmente o tamanho da equipe varia de quatro a sete participantes, e o número mínimo é de três pessoas. Equipes maiores são normalmente utilizadas para analisar documentos de mais alto nível do produto, enquanto que times menores são preferíveis ao se inspecionar detalhes mais técnicos.

É bastante interessante para o processo de inspeção que exista uma boa variedade de inspetores, pertencentes a diferentes áreas de conhecimento. O papel de cada participante será explicado abaixo.

- **Autor:** é o criador (desenvolvedor) do artefato que será inspecionado. Suas principais responsabilidades são: corrigir os problemas detectados durante o processo de inspeção, prover uma visão geral do produto aos demais participantes e tirar quaisquer dúvidas que surgirem com relação ao artefato desenvolvido.
- **Inspetor:** examina o produto antes e durante a reunião de inspeção (fase de preparação) de modo a tentar encontrar defeitos. Pode também identificar problemas amplos que estão fora do escopo da equipe de inspeção, como também sugerir melhorias.
- **Leitor:** pessoa responsável por apresentar o artefato aos demais participantes do processo de inspeção durante a reunião. Uma pessoa que usará o produto numa próxima etapa do seu ciclo de vida é um candidato forte pare esta tarefa, uma vez que a atividade de ler sobre o produto irá permitir a este potencial usuário se tornar bastante familiar com o produto.
- **Escritor:** tem o papel de registrar as informações sobre cada defeito encontrado durante a reunião, que incluem: a localização do defeito, um resumo do problema, sua classificação e uma identificação do inspetor que o encontrou. Todas as decisões e recomendações feitas também são registradas.
- **Moderador:** o moderador tem o papel mais crítico no processo de inspeção e por este motivo faz-se necessário um treinamento mais aprofundado do que os outros membros da equipe. Ele é a pessoa que lidera toda a equipe e participa ativamente de todas as etapas. Dentre suas principais responsabilidades podemos destacar: selecionar e liderar a equipe de inspeção, distribuir o material a ser inspecionado, agendar as reuniões, atuar como moderador nos encontros, supervisionar a correção dos defeitos, e emitir relatório de inspeção. Uma outra responsabilidade muito importante do moderador é garantir que o foco da reunião se mantenha com o objetivo de encontrar falhas no produto, e não de acusar o autor dos problemas encontrados.

10.4.2. O Processo de Inspeção de Software (Etapas)

O processo tradicional de inspeção de software [Fagan 1976] é definido por seis estágios, cada um representado por seu principal responsável. A Figura 10.7 ilustra esta sequencia de etapas e em seguida cada uma das etapas será explicada detalhadamente.

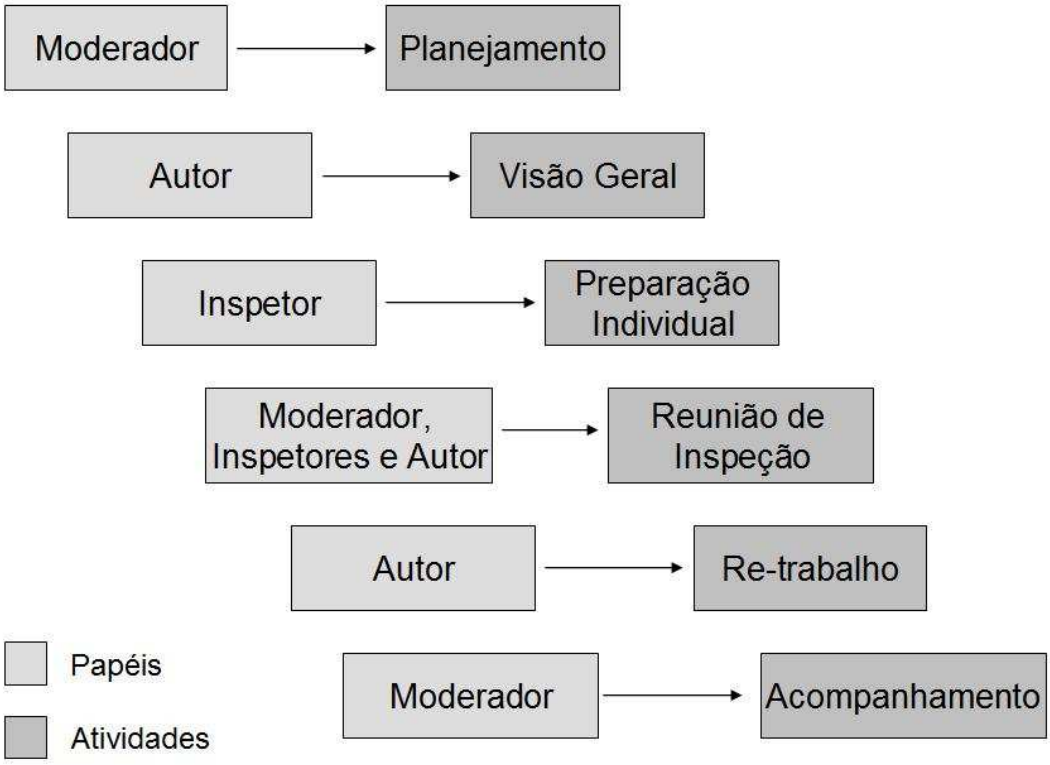

Figura 10.7 Processo de Inspeção de Software (adaptado de Fagan 1976)

- **Planejamento:** o moderador é a pessoa responsável por esta etapa. O planejamento envolve selecionar a equipe, verificar se o produto está pronto para inspeção, organizar a reunião, delegar as atividades de cada membro e garantir a completude dos materiais a serem inspecionados. Nesta etapa o moderador também deve verificar se o material a ser inspecionado possui um tamanho adequado para uma única reunião. Caso contrário, o material deverá ser dividido em tamanhos menores, com inspeções a serem realizadas para cada uma destas partes.
- **Visão geral:** nesta etapa o autor apresenta o produto aos demais membros da equipe, descrevendo o que o programa é suposto fazer. O moderador é responsável por decidir se esta etapa se faz realmente necessária, pois se a equipe já for bem familiarizada com o material a ser inspecionado ou novas técnicas não estejam sendo aplicadas, este estágio é dispensável.
- **Preparação:** este é o momento em que cada membro do time de inspeção estuda individualmente a especificação e o programa a ser inspecionado, e procura por defeitos no material. Todos os possíveis defeitos devem ser registrados num *log* de preparação, assim como o tempo que foi gasto na preparação. O moderador é encarregado de analisar os *logs* antes da reunião de inspeção para determinar se a equipe está preparada para suas tarefas, e caso contrário, ele pode remarcar a reunião.

- **Reunião:** nesta etapa, o passo a passo principal consiste na leitura e interpretação do produto, pelo leitor; em seguida o autor tira quaisquer dúvidas que eventualmente surgirem com relação ao material, e a equipe de inspetores então identificam os possíveis defeitos. Esta reunião deve ser curta, não podendo ultrapassar o limite de duas horas, e deve ser focada na detecção de defeitos, não conformidade com o padrão e programação de má qualidade. O time de inspeção não deve discutir como estes defeitos poderiam ser corrigidos e nem sugerir mudanças em outros componentes.
- **Re-trabalho:** o propósito do re-trabalho é corrigir os defeitos identificados durante a reunião de inspeção. O autor é a pessoa responsável por essas correções, devendo corrigir em primeiro lugar os defeitos considerados mais relevantes e graves, e corrigindo os de menor importância apenas se o tempo permitir.
- **Acompanhamento:** aqui o moderador deve decidir se uma nova inspeção é necessária ou não. Ele deve analisar o material corrigido pelos autores e verificar se os defeitos foram corrigidos com sucesso. O moderador pode incluir revisores adicionais nesta etapa se forem necessários conhecimentos técnicos extras. Se todos os problemas mais relevantes forem resolvidos, os problemas em aberto solucionados, e o produto satisfizer aos critérios de saída, o moderador aprova o *release* do produto. Se as condições não foram atingidas, ainda será necessário mais tempo na etapa de re-trabalho.

10.4.3. Ferramentas de Apoio ao Processo de Inspeção

Baseado na classificação de *groupware* (softwares voltados para o apoio a atividades de trabalho em grupo) e nas constantes mudanças tecnológicas, [Hedberg 2004] identificou quatro gerações de ferramentas de inspeção de software:

1. Primeiras Ferramentas (*Early tools*)
2. Ferramentas Distribuídas (*Distributed tools*)
3. Ferramentas Assíncronas (*Asynchrounous tools*)
4. Ferramentas baseadas em WEB (*Web-based tools*)

As Primeiras Ferramentas, foram criadas no início da década de 90 e logo em seguida vieram as Ferramentas Distribuídas. No final da década de 90 surgiram as ferramentas para Internet.

As ferramentas da primeira geração são aquelas que apenas permitem o trabalho de toda a equipe no mesmo ambiente e ao mesmo tempo (inspeções síncronas). A segunda já permite que a equipe possa trabalhar de forma distribuída, ou seja, em lugares diferentes, porém ainda é preciso que seja ao mesmo tempo (inspeções distribuídas). A total independência de tempo e lugar foi introduzida na terceira geração, com as ferramentas assíncronas. As ferramentas da quarta geração também são assíncronas, diferenciando-se das demais devido a sua base tecnológica.

A seguir será apresentada uma ferramenta representante de cada geração introduzida anteriormente [Wong 2006]. A ferramenta ICICLE representará a geração de Primeiras Ferramentas. Em seguida a ferramenta Scrutiny exemplificará as Ferramentas Distribuídas. Assist ilustrará as Ferramentas Assíncronas, e finalmente, IBIS será a representante das Ferramentas baseadas em WEB.

- ICICLE – O ICICLE (*Intelligent Code Inspection Environment in a C Language Environment*) é o primeiro software de revisão publicado e visa apoiar o processo tradicional de inspeção de software. Como o próprio nome já sugere, ele foi desenvolvido para o contexto específico de inspeção de código C e C++, podendo ser usado para o auxílio da inspeção do código, tanto nas fases de preparação individual como nas reuniões em grupo. A reunião de inspeção em grupo deve ser realizada no mesmo local e a inspeção individual permite entrar com comentários em cada linha de código. A ferramenta não se aplica a inspeções mais genéricas, limitando o tipo de artefato a ser inspecionado e a técnica de detecção de defeitos, mas pode, entretanto, ser utilizada para inspecionar linhas de texto numa análise inicial. Um dos principais objetivos desta ferramenta é o de ajudar os inspetores de código a encontrarem defeitos óbvios.
- Scrutiny – O Scrutiny é uma ferramenta colaborativa *online*, sendo a primeira a permitir que os membros do time de inspeção se encontrassem dispersos geograficamente, podendo ser usada tanto de forma síncrona como assíncrona. Ela pode ser integrada com outras ferramentas e customizada para apoiar diferentes processos de desenvolvimento. Atualmente apenas suporta inspeções de textos. A ferramenta é baseada num processo de inspeção dividido em quatro etapas. No primeiro estágio, de iniciação, o moderador disponibiliza o documento a ser inspecionado na ferramenta. No próximo estágio, preparação, os inspetores inserem seus comentários a serem discutidos na reunião. Depois, na fase de resolução, o moderador guia os inspetores através dos documentos e dos defeitos coletados. Finalmente, no estágio de finalização, após as discussões e acordos referentes aos defeitos levantados, a ferramenta fornece um resumo dos defeitos que foram discutidos.
- Assist – *Asynchronous/ Synchronous Software Inspection Support Tool* foi desenvolvida para prover inspeções individuais e em grupo. Como o nome sugere, permite inspeções síncronas e assíncronas, com reuniões tanto em locais diferentes como no mesmo ambiente. Utiliza uma linguagem de definição de processo de inspeção (IPDL) e um sistema flexível para o tipo de documento inspecionado, permitindo o suporte a qualquer tipo de processo de inspeção de software. Inspeção de código, coletas de dados para métricas e cálculos para apoio as inspeções também estão presentes nesta ferramenta. É baseada numa arquitetura cliente/servidor, em que o servidor é usado como um repositório central de documentos e outros tipos de dados. Um *browser C++* pode automaticamente apresentar itens relevantes de *checklist*[3] para a sessão de código inspecionado.
- IBIS – *Internet-Based Inspection System* é uma ferramenta baseada em WEB com notificações por *email* que auxilia no processo de inspeção desenvolvido por Fagan. Permite que as inspeções sejam realizadas entre pessoas geograficamente distribuídas e possui uma interface bastante leve e amigável, tendo toda sua estrutura e dados armazenados em arquivos XML. Ela não limita o tipo de artefato a ser inspecionado e provê suporte a decisões, apoio a anotações e *checklists*. As principais vantagens desta ferramenta são: permite que os inspetores acessem a aplicação de seus próprios computadores; admite

[3] No contexto de inspeção de software, é uma lista de perguntas que guiam o inspetor na detecção de defeitos.

que a inspeção seja realizada com integrantes da equipe distribuídos em locais diferentes, até mesmo em países diferentes; permite que especialistas diferentes participem da reunião, podendo ser especialistas de outro departamento ou mesmo fora na organização.

10.5. Modelos de Maturidade de Testes de Software

Para se construir software com qualidade, é necessário que se tenha um processo de testes bem definido e que ele esteja alinhado ao processo de desenvolvimento. Nesta seção serão vistos três modelos de maturidade de teste de software, os quais indicam como criar e/ou melhorar o processo de testes.

10.5.1. Processo de melhoria de testes – TPI

O modelo *Test Process Improvement* - TPI foca na melhoria do processo de testes e ajuda a definir gradualmente os passos para sua evolução, levando em consideração o tempo, o custo e a qualidade. Foi desenvolvido porque a IQUIP (*Intelligent & Quick Information Processing*) e seus clientes necessitavam de um modelo que suportasse as constantes mudanças do processo de testes.

O processo de teste é parte do processo total de desenvolvimento, demonstrado na Figura 10.8, por isso é de extrema importância analisar os problemas relacionados às atividades de teste e os impactos que eles causam no processo geral [JACOBS & Sotokes 2000].

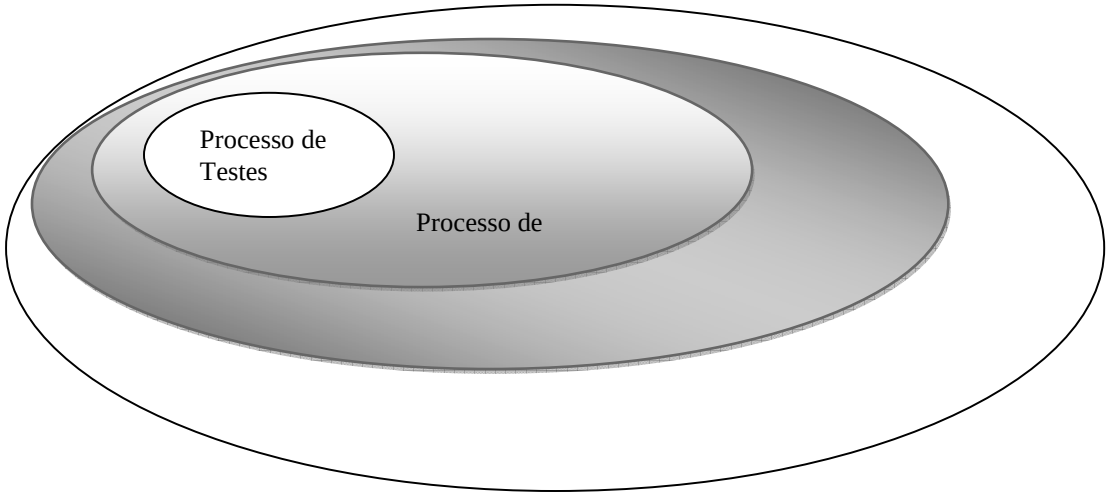

Figura 10.8. Processo Total de Desenvolvimento (adaptado de JACOBS & Sotokes 2000)

10.5.1.1. Escopo do TPI

O TPI possui vinte áreas chave dentro do processo de testes, em que cada área tem níveis, *checkpoints* e sugestões de melhoria que representam, respectivamente: a maturidade do processo de testes utilizado na empresa, os pontos verificados para definir onde o processo se encaixa e as dicas que explicam quais são os passos a seguir para se alcançar o nível desejado. A estrutura do TPI pode ser observada na Figura 10.9.

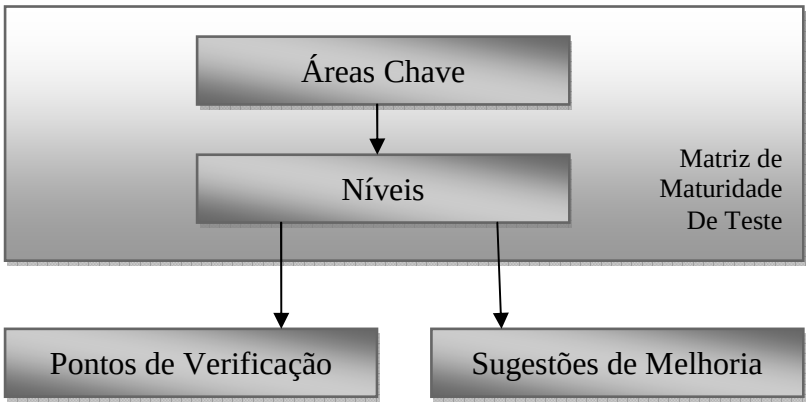

Figura 10.9. Estrutura do TPI (adaptado de JACOBS & Sotokes 2000)

10.5.1.2. Áreas Chave

O TPI possui vinte áreas chave, descritas abaixo, divididas em ciclo de vida do desenvolvimento do software, técnicas de planejamento e testes, infraestrutura do ambiente de testes e fatores organizacionais [JACOBS & Sotokes 2000]. São elas:

- **Estratégia de teste:** É focada na detecção de defeitos importantes o mais cedo possível. A estratégia define que requisitos e riscos serão cobertos por quais testes.
- **Modelo de ciclo de vida:** Definição das fases, como planejamento, preparação, especificação, execução e finalização. Em cada fase, várias atividades são requeridas.
- **Quando se envolver?** O envolvimento tardio dos testes é um risco para o projeto, porque os defeitos são mais caros e mais difíceis de serem consertados. Por esta razão, o TPI defende que os engenheiros de teste devem se envolver no início do projeto.
- **Estimativa e planejamento:** Indica que atividades terão que ser feitas e que recursos serão necessários para um certo período de tempo.
- **Técnicas de especificação de testes:** Padronização do caminho para especificar casos de testes da fonte de informação. Aplicando estas técnicas, obtém-se uma melhor qualidade e reusabilidade dos casos de testes.
- **Técnicas de teste estático:** Nem tudo pode ser testado dinamicamente, ou seja, rodando programas. Inspeção de produtos, avaliação de medidas ou *checklists* são exemplos de técnica de teste estático.
- **Métricas:** Para o processo de testes, as métricas de progresso e de qualidade do sistema testado são muito importantes. Especificamente, para melhoria do processo de testes, as métricas devem avaliar as consequências de certas ações de melhoria.
- **Automação de testes:** Quando possível, a automação serve para diminuir as horas de execução e ter um *status* dos resultados, o quanto antes.

- **Ambiente de testes:** Hardware, software, conexão ao banco de dados etc. O ambiente de testes tem uma grande influência na qualidade do software.
- **Ambiente de trabalho:** Um desorganizado ambiente de trabalho pode desmotivar as pessoas e, consequentemente, impactar o resultado dos testes.
- **Comprometimento e motivação:** A área de testes deve ter suficiente tempo, dinheiro e recursos (quantidade e qualidade) para que os profissionais realizem bons testes.
- **Funções de testes e treinamento:** Para a área de testes, uma mistura de diferentes conhecimentos, funções e perfis é requerida. É necessário ter experiência em testes, boa comunicação, conhecimento sobre o projeto e sobre a organização em geral.
- **Escopo da metodologia:** A empresa deve usar uma metodologia suficientemente genérica para ser aplicada em diversas situações e não tenha a necessidade de ser remodelada a cada novo projeto.
- **Comunicação:** No processo de teste, a comunicação é feita em diversos níveis, dependendo das pessoas envolvidas (desenvolvedor, usuário, cliente, gerente, etc).
- **Reportagem:** Teste não é só detecção de defeitos, mas sim a principal maneira de melhorar a qualidade de software. O relatório de resultados deve, não só conter as causas dos defeitos, como também o melhor caminho para consertá-lo.
- **Gerenciamento dos defeitos:** Deve ser capaz de acompanhar o ciclo de vida do defeito, além de dar suporte à análise e a resolução dos mesmos.
- **Gerenciamento dos documentos de testes:** Os produtos (plano de testes, especificações, banco de dados e arquivos) de teste devem ser mantidos, reusáveis e gerenciáveis.
- **Gerenciamento do processo de testes:** Para gerenciar as atividades de teste, é essencial seguir esses quatro passos: planejar, fazer, verificar e atuar. O gerenciamento é vital para a realização dos testes.
- **Avaliação:** Começa na inspeção de produtos intermediários de acordo com requisitos. Os defeitos são encontrados em estágios iniciais e o re-trabalho tem um custo baixo.
- **Testes de baixo nível:** Testes unitários e de integração. Erros encontrados na fase de desenvolvimento.

10.5.1.3. Passos para implantar a melhoria

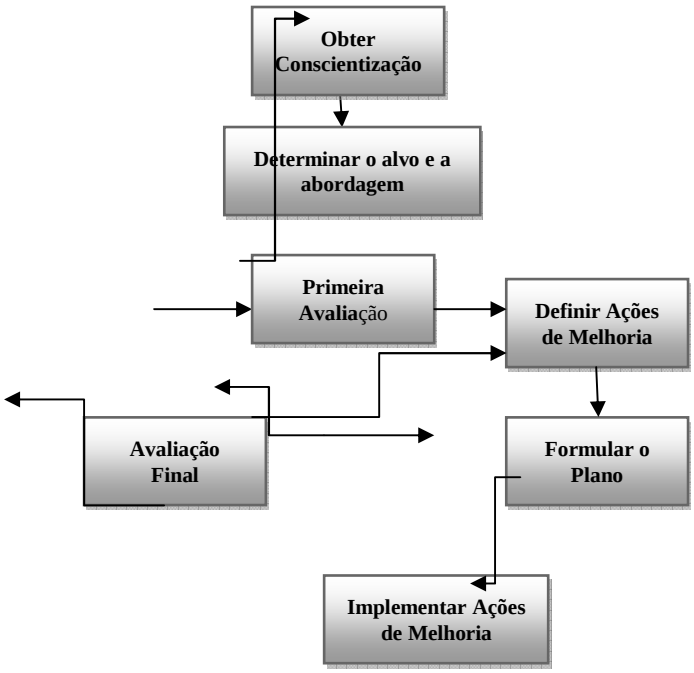

Figura 10.10. Modelo para Implantar Melhorias (adaptado de JACOBS & Sotokes 2000)

Como pode ser observado na Figura 10.10, há uma série de passos, detalhados a seguir, para se implantar a melhoria do processo de testes. É preciso fazer uma análise de como está o processo atual, para verificar qual é o objetivo e estruturar a empresa para alcançá-lo [Koomen & Pol 1999]. Os passos são:

- **Obter conscientização:** saber que o processo precisa melhorar e ter compromisso, não só no início das mudanças, mas sim ao longo de todo o projeto.
- **Determinar o alvo e a abordagem:** que áreas serão atacadas e quais as metas de melhoria?
- **Primeira avaliação:** conhecimento da situação atual. A partir da matriz são verificados os pontos fortes e fracos do processo de teste. Com base em entrevistas e documentação, os níveis das áreas-chave do TPI são definidos.
- **Definir ações de melhoria:** baseadas nas metas e no resultado da avaliação. Os níveis das áreas-chave e a Matriz de Maturidade dão várias possibilidades para definir a melhoria gradual das etapas.
- **Formular plano:** o plano aborda as atividades necessárias para orientar o processo de mudança em uma determinada direção.
- **Implementar ações de melhoria:** execução do plano. São analisadas as ações executadas e bem sucedidas.
- **Avaliação final:** qual foi o rendimento das ações implementadas? Nesta fase, o objetivo é mensurar as ações que foram executadas com sucesso, bem como avaliar se as metas iniciais foram cumpridas. Com base nestas observações, a decisão sobre a continuação do processo de mudança será tomada.

10.5.2. *Test Maturity Model* - TMM

O *Testing Maturity Model* – TMM [Burnstein, et al 1998] foi desenvolvido pelo *Illinois Instituty of Tecnology* como um guia para melhoria de processos de testes. A estrutura do TMM foi baseada no CMM, e está aderente ao CMMI, consistindo de cinco níveis que avaliam o grau de maturidade de um processo de testes. Para cada nível de maturidade, áreas de processo são definidas. Uma área de processo é um conjunto de atividades que, quando executadas de forma adequada, contribuem para a melhoria do processo de testes. Na Figura 10.11 pode-se observar a estrutura do TMM.

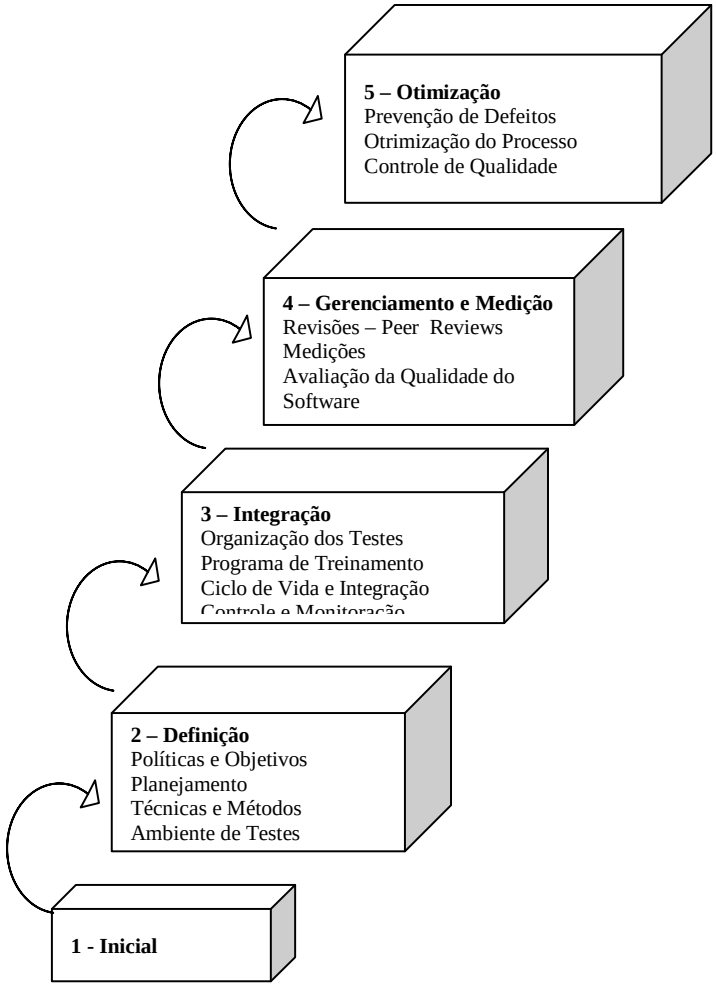

Figura 10.11. Estrutura do TMM

10.5.2.1. Níveis de Maturidade do TMM

Os cinco de níveis de maturidade do TMM mostram uma evolução de um processo caótico e indefinido para um processo de testes controlado e otimizado. Nesta seção, será visto o detalhamento de todos os níveis de maturidade do TMM.

- **Nível 1 – *Initial* (Nível Inicial)**

  Os testes são caóticos, não existe processo definido. O objetivo dos testes é mostrar que o software funciona sem maiores falhas.

Áreas do processo: Nenhuma área do processo é identificada nesse nível;

- **Nível 2 – *Phase Definition* (Nível Definido)**

  Ocorre a definição de um processo de teste. No contexto de estruturação do processo de testes, planos de teste são estabelecidos contendo estratégias de teste. Os testes ainda acontecem tardiamente dentro do ciclo de vida do desenvolvimento.

  Áreas do processo: Políticas e objetivos, planejamentos dos testes, técnicas, métodos e ambiente de testes;

- **Nível 3 – *Integration* (Nível Integrado)**

  Os testes são completamente integrados ao ciclo de vida do software, sendo reconhecido em todos os níveis do processo de testes. O planejamento acontece no estágio inicial dos projetos, através de um plano de testes máster. A estratégia de testes é determinada através de técnicas de gerenciamento de riscos e baseada em requisitos. Programas de treinamento e revisões fazem parte do processo.

  Áreas de processo: Organização dos testes, programa de treinamento, ciclo de vida, integração, controle e monitoramento;

- **Nível 4 – *Management and Measurement* (Nível Gerenciado e Medido)**

  Os testes são completamente definidos, bem fundamentados e medidos. Revisões e inspeções são incorporadas ao ciclo de vida do desenvolvimento e considerados parte dos testes. Os produtos de software são avaliados a partir de critérios de qualidade por características de qualidade, como reusabilidade, usabilidade e mantenabilidade. Casos de testes são armazenados e gerenciados em uma base de dados central para reuso e testes de regressão.

  Áreas de processo: Revisões, medição dos testes e avaliação da qualidade dos testes;

- **Nível 5 – *Optimization* (Nível Otimizado)**

  Resultados são arquivados com o objetivo de melhoramentos em estágios anteriores, a área de testes é completamente definida através de seu processo e capaz de controlar seus custos, sendo efetivos. No nível 5 métodos e técnicas são otimizados e estão em melhoramento contínuo. A prevenção de defeitos e o controle de qualidade são introduzidos em outras áreas do processo. Há procedimentos para escolha e avaliação de ferramentas de testes. Testar é um processo com o objetivo de prevenir defeitos.

  Áreas do processo: Prevenção de defeitos, controle de qualidade e otimização do processo de teste.

Os cinco de níveis de maturidade mostram uma evolução de um caótico e indefinido para um controlado e otimizado processo de testes.

10.5.3. *Test Improvement Model* - TIM

Desenvolvido pela Ericson, Subotic e Ursing o TIM [Koomen & Pol 1999] foi concebido pelos desenvolvedores que sentiam a necessidade de melhorar o processo de testes. O TIM se propõe a identificar o estado atual das práticas das áreas chaves e serve como um guia na implementação dos pontos fortes e na remoção dos pontos fracos.

O TIM é composto de um Modelo de Maturidade e um Processo de Avaliação.

10.5.3.1. Modelo de Maturidade

Consiste em quatro níveis, o primeiro foi denominado de nível 0, considerado um nível de não conformidades, no entanto os outros níveis, de 1 a 4, possuem nomes os quais representam seus objetivos gerais e também sub-objetivos. Um objetivo só poderá ser atendido se seus sub-objetivos forem atendidos também. Outro componente do TIM são as áreas chaves. Existem cinco ao todo e cada uma delas cobre partes importantes da disciplina de teste. Outro fator importante é que as Áreas Chaves possuem os mesmos nomes dos Níveis.

Os Níveis e seus sub-objetivos estão listados abaixo[Koomen & Pol 1999]:

- **Nível 1 – *Baselining* (Base)**

  - Padronização dos documentos, métodos e políticas.

  - Análise e classificação dos problemas.

- **Nível 2 – *Cost-efectiveness* (Efetividade de Custo)**

  - Detecção de bugs desde o início do projeto

  - Automação de tarefas de teste

  - Treinamento

  - Reuso

- **Nivel 3 – *Risk-lowering* (Redução de Custo)**

  - Envolvimento desde o início do projeto

  - Análise de custo x benefício para justificar os gastos

  - Análise de problemas nos produtos e processos

  - Métricas de produtos, processos e recursos

  - Análise e gerenciamento de riscos

  - Comunicação com todas as partes envolvidas nos projetos

- **Nivel 4 – *Optimizing* (Otimizado)**

  - Conhecimento e entendimento através de experimentação e modelagem

  - Cooperação com todas as partes envolvidas nos projetos em todas as fases do desenvolvimento

  - Análise das causas raízes para os principais problemas

- Melhoria contínua

Áreas Chave

As Áreas Chaves do TIM [Koomen & Pol, 1999] estão listadas abaixo e para cada nível de maturidade são apresentados os principais aspectos da disciplina de teste, são eles:

- **Aspecto: Organização**
    - No nível *Baselining* deve-se organizar um conjunto mínimo de papéis para executar as atividades básicas de teste.
    - No nível *Cost-efectiveness* a principal mudança em relação ao modelo organizacional é a independência da equipe de teste.
    - No nível *Risk-lowering* aumenta a interação entre as equipes de desenvolvimento e a equipe de teste. A equipe de teste necessita conhecer mais sobre o desenvolvimento do produto, de forma a aumentar a qualidade do produto e o conhecimento das regras de negócio.
    - No nível *Optmizing* os testadores fazem parte do time de desenvolvimento e possuem conhecimento em várias disciplinas. São estabelecidos grupos com o objetivo de avaliar continuamente o processo.
- **Aspecto: Planejamento e Rastreabilidade**
    - No nível *Baselining* o projeto de teste possui um planejamento básico, nele são estabelecidos critérios de entrada e saída, os resultados dos testes são documentados, processados e distribuídos.
    - No nível *Cost-efectiveness*, o planejamento e a rastreabilidade são auxiliados por ferramentas, alguns planos genéricos são utilizados. A escolha dos estágios e métodos de testes é alinhada de acordo com os objetos e os objetivos dos testes.
    - No nível *Risk lowerin,g* a análise dos riscos é realizada e sua influência é bastante elevada no planejamento, além de afetar partes do plano, mais precisamente o os objetivos dos testes.
    - No nível *Optimizing*, atividades de planejamento e a rastreabilidade é continuamente melhorada baseada na análise de métricas. Reuniões de *post-mortem*[4] são realizadas e os resultados armazenados e distribuídos.
- **Aspecto: Casos de Teste**
    - No nível *Baseling* os casos de testes são elaborados baseados nos requisitos de sistemas e segundo as instruções das políticas.

[4] *Reunião de Post-mortem: é uma reunião com o objetivo de coletar de experiências eficazes e de baixo custo que pode contribuir de forma significativa para a melhoria dos processos de software.*

- No nível *Cost-efectiveness*, os casos de testes são projetados de acordo com técnicas documentadas. Com armazenamento dos casos de testes a reusabilidade se torna possível.
- No nível *Risk-lowering*, com o armazenamento dos casos de testes no nível anterior é possível selecioná-los de acordo com a criticidade.
- No nível *Optmizing*, medições, revisões e melhorias são realizadas sobre os casos de testes.

- **Aspecto: *Testware* (artefatos de teste)**
  - No nível *Baselining* os problemas são reportados e computados.
  - O nível *Cost-efectiveness* se caracteriza pelo uso de ferramentas de cobertura, banco de dados para gerenciar o *testware.*
  - No nível *Risk-lowering*, são realizados testes de regressão quando o código sofre alteração. A análise de risco é realizada com uso de ferramentas.
  - No nível *Optimizing* Ambiente de Teste é Integrado.
- **Aspecto: Revisões**
  - No nível *Baselining*, Padrões de revisões de documentos são utilizados.
  - No nível *Cost-efectiveness*, os projetos e códigos são documentados e revisados através de técnicas de revisão escolhidas pela organização.
  - Técnicas de revisão e inspeção são constantemente evoluídas. Todo o *testware*, o processo e produto são revisados e medidos no nível de *Risk lowering.*
  - No nível *Optmizing*, técnicas e time são selecionados baseados em fatos.

Através dos níveis de maturidade e áreas chaves do TIM é possível avaliar um processo de teste.

10.6. Considerações Finais

O desenvolvimento de software engloba um mercado de extrema competitividade. Tendo em vista que os sistemas que apresentam melhor qualidade garantem seu espaço no mercado, as empresas que os desenvolvem têm investido bastante para assegurar a qualidade de seus produtos e garantir a satisfação dos clientes. A qualidade de um produto pode ser definida como sua capacidade de cumprir os requisitos inicialmente estipulados pelos clientes, e sendo assim, está diretamente relacionada à qualidade do processo de desenvolvimento. Por este motivo, tem surgido uma grande demanda ao incentivo de pesquisas que levem em consideração a procura por formas de melhoria da qualidade dos produtos.

Este capítulo procurou introduzir ao leitor boas práticas no que diz respeito à qualidade dos produtos, apresentando um conjunto de normas que representam a padronização mundial para avaliação da qualidade de produtos de software. As atividades de teste e inspeção também foram destacadas como forma de encontrar defeitos no software e corrigi-los, antes de entregar o produto a seus clientes, e analisar se o sistema faz o que é suposto fazer. Finalmente, modelos de maturidade de testes foram apresentados como mais uma tentativa de alcançar melhorias na qualidade do processo de teste de software, que afeta diretamente a qualidade do produto.

10.7. Tópicos de Pesquisa

Existem vários estudos atualmente na academia no que diz respeito à seleção de testes de regressão, uma vez que executar todos os casos de teste novamente sempre que uma nova versão do sistema for liberada é uma prática inviável. Dessa forma, várias pesquisas e propostas de soluções e técnicas para realizar uma quantidade suficiente de testes que atinja a cobertura necessária para garantir a corretude do software podem ser encontradas na literatura, como por exemplo, o artigo *Analyzing Regression Test Selection Techniques*, de *Gregg Rothermel* e *Mary Jean Harrold*, publicado em *IEEE Transactions on Software Engineering* [Rothermel & Harrold 1996].

Outra área de pesquisa bastante desafiadora na área de teste de software é a geração automática de casos de teste, considerando que a elaboração de casos testes manualmente é um processo que consome muito tempo e esforço. Sendo assim, diversas propostas são elaboradas dia após dia com o objetivo de tornar o processo de teste mais ágil, menos susceptível a erros e dependente da interação humana. Um exemplo de pesquisa nesta área é o projeto *Mulsaw: Automated Checking of Code Conformance*, que pode ser visualizado em http://projects.csail.mit.edu/mulsaw/.

Na área de inspeção de software, grandes desafios podem ser observados com o objetivo de encontrar estratégias para diminuir a quantidade de defeitos de um software. Na literatura, podem ser encontradas diversas pesquisas e artigos com estudos focados neste objetivo, como por exemplo, o artigo *Software Inspection: Eliminating Software Defects*, de *Bill Brykczynski* et al., puclicado em *Proceedings of the Sixth Annual Software Technology Conference* [Brykczynski 1994].

Na área de modelo de maturidade de testes, há uma organização, chamada *TMMi Foundation* (www.tmmifoundation.org), sem fins lucrativos, em Dublin – Irlanda, que foi fundada para tentar transformar o modelo TMM em uma norma e, consequentemente, promover a sua aceitação como um padrão da indústria internacional de avaliação e de organizações de teste de software. A Fundação TMMi tem como objetivo: a criação e gestão de uma organização independente, imparcial com repositório central de dados e prestação de serviços, métodos de avaliação com base no modelo padrão, definição e manutenção de avaliadores independentes e prestação de um fórum público das partes interessadas para facilitar a livre troca de informação, educação, idéias e uso da norma pública.

Em relação ao TPI, há pesquisas na academia que objetivam melhorar a produtividade do time de testes, utilizando as práticas definidas pela melhoria gradual do processo de testes. Na Europa, há uma empresa especializada na melhoria do processo de testes que provê consultorias na área. A página *web* da empresa é **www.sogeti.com**.

10.8. Sugestões de Leitura

Para conhecer mais sobre normas de qualidade de produto de software, é recomendada a leitura do livro Tecnologia da Informação: Qualidade de Produto de Software [Guerra & Colombo 2009].

Para ampliar o entendimento sobre o assunto de teste de software é recomendada a leitura do livro *Foundations of software testing*, Graham, D., Veenendaal, E. v., Evans, I. and Black, R., 2007 [Graham et. al 2007]. Este livro é utilizado por pessoas que desejam tirar o certificado ISTQB (*International Software Testing Qualifications Board)*, portanto, é muito interessante para adquirir melhores conhecimentos sobre este conteúdo.

Para um melhor conhecimento sobre os conceitos e o processo de inspeção de software é sugerida a leitura de *Design and Code Inspection to Reduce Errors in Program Development*, Fagan, M.E.,1986 [Fagan 1986].

Para se aprofundar mais sobre as ferramentas de inspeção de software é recomendada a leitura de *Modern Software Review Techniques and Technologies,* Wong, Y. K., 2006 [Wong 2006].

Para melhor conhecimento sobre o TPI (*Test Process Improvement*) é recomendada a leitura do livro *Test Process Improvement A practical step-by-step guide to structured testing*, Koomen & Pol, 1999 [Koomen & Pol 1999].

Para aprofundar a leitura sobre TMM (*Test Maturity Model*), é sugerida a leitura do livro *A Model to Assess Testing Process Maturity*, Burnstein & Grom, 1998 [Burnstein, et al 1998].

10.9. Exercícios

1. Quais são as diretrizes para uso da norma NBR ISO/IEC 9126-1?
2. A que se propõe a norma ISO 12119?
3. Que subdivisões da norma ISO 14598 estabelecem itens necessários para o suporte à avaliação?
4. Quais são os componentes do projeto SQuaRE? Defina-os.
5. Qual a diferença entre testes e inspeções de software?
6. Cite 5 tipos de testes e explique cada um deles.
7. Quais os estágios de testes possíveis e quais as características de cada um deles?
8. O que são testes beta?
9. O que são testes de regressão?
10. Qual a diferença entre a abordagem de caixa preta e a abordagem de caixa branca?
11. Quais são os papéis existentes na equipe de inspeção de software e quais suas responsabilidades?
12. Quais são as etapas do processo de inspeção de software? Explique cada uma delas.
13. Explique como é feita a implantação da melhoria no TPI.
14. Defina os níveis de maturidade do TMM.
15. No aspecto Organização, como são caracterizados os níveis de maturidade do TIM?

10.10. Referências

Guerra, A., C e Colombo, R., M., T. “Tecnologia da Informação: Qualidade de Produto de Software”, PBQP Software, 2009

Burnstein, L. “Practical software testing: a process-oriented approach”, New York, Springer, 2003

Cortes, M. L. (2009). *Qualidade.* Acessado em 03 de Setembro de 2009, disponível em http://www.ic.unicamp.br/~cortes/mc726/cap3.pdf

Burnstein, I., Homyen, A., Grom, R. and Carlson C.R. “A Model to Assess Testing Process Maturity”, Crosstalk, 1998

Koomen, T., e Pol, M. Test Process Improvement A practical step-by-step guide to structured testing. ACM Press, 1999

Boehm, B. W. e BASILI, V.R. (2001) “Software Defect Reduction Top 10 List.”, IEEE Computer 34 (1), p. 135-137.

Pressman, R. S., Engenharia de Software, McGraw Hill, 2002.

Sommerville, I., Software Engineering, 7th Edition, Addison Wesley, 2004.

Fagan, M.E. (1976) “Design and Code Inspection to Reduce Errors in Program Development”, IBM Systems Journal, vol. 15, no. 3, p. 182-211.

Hedberg, H. (2004) “Introducing the Next Generation of Software Inspection Tools”, In: International Conference of product focused software process improvement, 5, Kansai. Lecture notes in computer science, Berlin: Springer, p. 234-247.

Graham, D., Veenendaal, E. v., Evans, I. and Black, R. Foundations of software testing, ISTQB Certification, Thomson Learning, 2007.

Wong, Y. K. “Modern Software Review Techniques and Technologies”, IRM Press, 2006.

Selby, R. W. e Basili, V. R., et al. (1987). “Cleanroom software development: an empirical evaluation”, IEEE Trans on Software Engineering , SE-13(9), 102-37. (Chs. 4,22).

SEI - Software Engineering Institute. CMMI® para Desenvolvimento – Versão 1.2”, Carnegie Mellon University, 2006.

Mills, H. D. e Dyer, M., et al. (1987). “Cleanroom software engineering”, IEEE Software, 4(5), 19-25. (Chs. 3,4,22).

Zeilinger, C., (2003). “Robustness of Software”, Seminar Software-Development (programming styles).

Myers, G. J., The Art of Software Testing, 2nd Edition, John Wiley & sons, Inc, 2004.

Craig, R. D. e Jaskiel, S. P., Systematic Software Testing, Artech House Publishers, 2002.

Rothermel, G., Harrold, M.J. (1996) “Analyzing Regression Test Selection Techniques”, IEEE Transactions on Software Engineering.

Brykczynski, B. (1994) “Software Inspection: Eliminating Software Defects”, Proceedings of the Sixth Annual Software Technology Conference.