# “A utilização de uma metodologia de teste no processo da melhoria da qualidade do software”

*Marcus Valério de Queiroz Bruneli*

Trabalho Final de Mestrado Profissional em Computação

Instituto de Computação

Universidade Estadual de Campinas

# "A utilização de uma metodologia de teste no processo da melhoria da qualidade do software"

Marcus Valério de Queiroz Bruneli

22/02/2006

Banca Examinadora:

- **Profa. Dra. Eliane Martins (Orientadora)**
  Instituto de Computação – Unicamp

- **Prof. Dr. Adalberto Nobiato Crespo**
  Centro de Pesquisas Renato Archer – CenPRA

- **Prof. Dr. Mário Lúcio Côrtes**
  Instituto de Computação – Unicamp

- **Prof. Dr. Célio Cardoso Guimarães (Suplente)**
  Instituto de Computação – Unicamp

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP**
Bibliotecária: Miriam Cristina Alves – CRB8a / 5094

Bruneli, Marcus Valério de Queiroz

B835u A utilização de uma metodologia de teste no processo da melhoria da qualidade do software / Marcus Valério de Queiroz Bruneli -- Campinas, [S.P. :s.n.], 2006.

Orientadora: Eliane Martins

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Software - Testes. 2. Engenharia de software. 3. Software - Qualidade. I. Martins, Eliane. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: The utilization of a test methodology in the process of improvement of software quality

Palavras-chave em inglês (Keywords): 1. Software tests. 2. Software engineering. 3. Software quality.

Área de concentração: Engenharia de Software

Titulação: Mestre em Ciência da Computação

Banca examinadora: Profª. Drª. Eliane Martins (IC-UNICAMP)
Prof. Dr. Adalberto Nobiato Crespo (CenPRA)
Prof. Dr. Mário Lúcio Côrtes (IC-Unicamp)

Data da defesa: 22/02/2006

# "A utilização de uma metodologia de teste no processo da melhoria da qualidade do software"

Este exemplar corresponde à redação final do Trabalho Final devidamente corrigido e defendido por Marcus Valério de Queiroz Bruneli e aprovado pela Banca Examinadora.

Campinas, 22 de fevereiro de 2006.

Eliane Martins

Prof. Dra. Eliane Martins

(Orientadora)

PROF Dr. ALEXANDRE [illegible] FALCÃO
Coordenador [illegible] Profissional
Instituto de Computação - UNICAMP
Matr. 27.180-7

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Computação na Área de Engenharia de Software.

# TERMO DE APROVAÇÃO

Trabalho Final Escrito defendido e aprovado em 22 de fevereiro de 2006, pela Banca Examinadora composta pelos Professores Doutores:

**Prof. Dr. Mário Lúcio Côrtes**
**IC - UNICAMP**

**Prof. Dr. Adalberto Nobiato Crespo**
**CenPRA**

**Profa. Dra. Eliane Martins**
**IC - UNICAMP**

*A Deus, aos meus pais, à minha esposa e filho(s), que ainda por vir.*

*Em especial a memória de meu pai que sempre foi um exemplo de luta, superação e dedicação à família.*

# Agradecimentos

Aos meus pais, que me proporcionaram condições de estudar e me formar em uma universidade federal e bem conceituada, e irmãos, Leonardo e Andréa, que de uma forma tiveram paciência em respeitar meus momentos de dedicação ao estudo.

A minha querida esposa Ana Cláudia, que desde o início deste desafio esteve ao meu lado me incentivando nos momentos difíceis e apoiando com muito carinho e paciência.

Ao grande amigo Luiz Gustavo, companheiro de longas viagens e noites de estudo, que me apoiou não me permitindo que desanimasse, assim como sua esposa Veridiana.

Agradeço também aos amigos de curso Dinailton, Wenndel, Marcos Ganhoto, João Marcelo e Adão, os quais tive o prazer de conhecê-los e compartilhar grandes momentos de alegria e aprendizado.

Às secretárias do Instituto de Computação, em especial a Cláudia, que por diversas vezes me ajudaram, mesmo à distância.

À professora Eliane Martins que, não menos importante que os outros, me aceitou como seu orientando em meio a sua carga horária sempre cheia, a qual pude aprender um pouco mais e crescer profissionalmente.

*"Nenhuma técnica é boa em todos os lugares, mas uma técnica pode ser boa em pelo menos um lugar, a questão é saber quando ela irá funcionar ou não."*
[KANNER, 2003]

# Resumo

O processo de teste de software é normalmente postergado para o final do desenvolvimento, após a codificação concluída. Devido a isto, problemas como falta de tempo para planejamento, testes incompletos, falta de recursos e tempo adequado são fatos corriqueiros na atividade de testes das empresas que não possuem um processo de teste estruturado com documentos e regras bem definidas. Este trabalho apresenta um processo de teste de software, onde a atividade de teste começa tão logo a atividade de desenvolvimento se inicia e caminha em paralelo com o ciclo tradicional de desenvolvimento. Esta abordagem permite detectar e prevenir erros através do processo de desenvolvimento, conduzindo para uma maior confiança e qualidade do software. Um estudo de caso é apresentado para ilustrar a aplicabilidade do método e os resultados obtidos.

# Abstract

The process of software testing is usually postponed to the end of the development life cycle, after the coding phase. For this reason, problems like lack of time for test planning, incomplete tests, lack of resources and schedule are frequent issues in the activity of software testing in companies where there is not a structured software testing process with well defined documents and rules. This work present a software testing process where the test activity starts along with software development and continues during the traditional development cycle. This approach allows us to detect and prevent bugs through of the software development cycle, which leads to a higher confidence and quality of the software. A case study has been developed to illustrate the applicability of the method and show the results obtained.

# Sumário

1. INTRODUÇÃO ..... 1
2. TESTE DE SOFTWARE ..... 3
3. METODOLOGIAS DE TESTE ..... 14
4. ESTUDO DE CASO ..... 29
5. CONCLUSÃO E TRABALHOS FUTUROS ..... 44
6. REFERÊNCIAS ..... 45

# Índice

1. INTRODUÇÃO ..... 1
1.1. Contexto ..... 1
1.2. Objetivo ..... 2
1.3. Organização ..... 2
2. TESTE DE SOFTWARE ..... 3
2.1. Considerações ..... 3
2.2. Princípios e objetivos do teste ..... 5
2.3. Técnicas de teste ..... 6
2.3.1. Teste caixa-preta ..... 6
2.3.2. Teste caixa-branca ..... 8
2.4. Fases de testes ..... 10
2.5. Considerações Finais ..... 13
3. METODOLOGIAS DE TESTE ..... 14
3.1. Modelo 3P x 3E ..... 16
3.2. Norma IEEE 829 ..... 21
3.3. Metodologia do CenPRA ..... 24
3.4. Agedis ..... 25
3.5. Considerações Finais ..... 27
4. ESTUDO DE CASO ..... 29
4.1. Contexto ..... 29
4.2. A Realização dos Testes Antes da Proposta de Melhoria ..... 31
4.3. Proposta de Melhoria ..... 32
4.3.1. Documentos Utilizados ..... 32
4.3.2. Comunicação ..... 36
4.3.3. Planejando os Testes ..... 37
4.3.4. Especificando os Testes ..... 37
4.3.5. Execução e Entrega ..... 39
4.4. Resultados ..... 41
5. CONCLUSÃO E TRABALHOS FUTUROS ..... 44
6. REFERÊNCIAS ..... 45

APÊNDICE A. MODELOS ..... 47
A.1 Plano de Teste ..... 47
A.2 Casos de Teste ..... 48
A.3 Relatório de Incidente ..... 48
A.4 Relatório Resumo de Teste ..... 49

## Lista de Figuras

Figura 1 - Exemplo de grafo de fluxo derivado de um trecho de programa ..... 10
Figura 2 - Ciclo de vida do processo de teste ..... 17
Figura 3 - Etapas de desenvolvimento e a integração com os tipos de teste ..... 21
Figura 4 - Relacionamento entre os Documentos de Teste [CSBSAJ,2004] ..... 23
Figura 5 - Metodologia Agedis ..... 27
Figura 6 - Arquitetura tecnológica dos sistemas da empresa ..... 30
Figura 7 - Metodologia 3P x 3E - Documentos utilizados - Norma IEEE 829 ..... 36
Figura 8 - Fluxo do planejamento e especificação dos testes ..... 39
Figura 9 - Fluxo da execução dos testes ..... 41

## Lista de Tabelas

Tabela 1. Exemplo de retorno de investimento [BLACK,2005] ..... 15
Tabela 2. Etapas e subetapas modelo 3P x 3E [FR,2002] ..... 18
Tabela 3. Projetos realizados com a utilização da metodologia ..... 42
Tabela 4. Projetos realizados sem a utilização da metodologia ..... 43

# 1. INTRODUÇÃO

## 1.1. Contexto

No mundo atual existe uma demanda não satisfeita por softwares de qualidade. As empresas têm sinalizado uma preocupação com a qualidade de seus produtos através da garantia das funcionalidades especificadas no levantamento do projeto e o cumprimento de prazos de entregas.

Como ferramenta para aumento da qualidade dos produtos, o teste de software é uma atividade que merece destaque pelos resultados alcançados. Entretanto, através da atividade de teste por si só, pode-se obter relativas melhorias no software, mas não podendo tornar um software ruim em um de boa qualidade [JO,1995].

Através do uso de uma metodologia que possibilite a interação das atividades de teste a partir do início das especificações e que se estenda durante a construção do software até a entrega do sistema, pode-se obter melhorias drásticas na eficácia e eficiência dos testes, e consequentemente aumentar a qualidade do software [CSBSAJ,2004].

O processo de teste deve basear-se em uma metodologia aderente ao processo de desenvolvimento, em pessoal técnico qualificado, em ambiente e ferramentas. A metodologia de teste deve ser o documento básico para organizar a atividade de testar aplicações no contexto da empresa. Como é indesejável o desenvolvimento de sistemas sem uma metodologia adequada, também acontece o mesmo com a atividade de teste [FR,2002].

Um dos grandes problemas encontrados na implantação de uma metodologia de teste reside no fato dele ainda ser visto como um causador de aumento dos custos e prazos dos projetos, podendo, deste modo, vir a criar hostilidade entre os membros do projeto e os responsáveis pelos testes e, problemas com usuários [FR,2002].

Entretanto, esta visão está sendo substituída por uma outra que privilegia a qualidade e, como conseqüência, enfatiza o maior esforço no processo, entendendo que testar é um

investimento em qualidade e, que este produz um retorno positivo contribuindo decisivamente para o sucesso do projeto [BULLOCK,2005].

Este trabalho apresenta a utilização de uma metodologia de teste de software, a partir da definição inicial dos requisitos, passando pelas fases de análise, desenvolvimento ou codificação, até os testes finais e entrega do produto ao cliente.

### 1.2. Objetivo

O objetivo deste trabalho é apresentar, através de estudos de casos, que a utilização de planos de testes elaborados, com técnicas existentes [MYERS,1979] [PRESSMAN,2002] associados ao uso da norma IEEE 829-1998 e das metodologias de testes 3P x 3E [FR,2002] e CenPRA [CSBSAJ,2004], possibilita produzir subsídios suficientes para a preparação e execução de testes, capaz de detectar erros e melhorar a qualidade do produto final.

### 1.3. Organização

A divisão deste trabalho encontra-se da seguinte forma: no capítulo dois é apresentado uma visão geral sobre a atividade de teste de software, visando a oferecer um melhor entendimento da área em que o trabalho está inserido. Para isso, são apresentados os conceitos básicos relacionados a essa atividade, as principais técnicas e as fases de execução do teste de software.

No capítulo 3 são apresentadas as metodologias 3P x 3E [FR,2002], metodologia do CenPRA [CSBSAJ,2004], e a norma IEEE 829-1998 que possuem os principais conceitos utilizados neste estudo. E, uma visão geral da metodologia AGEDIS [AGEDIS], que utiliza-se da automatização de etapas para aumentar a qualidade do software.

No capítulo 4 é apresentado um estudo de caso comparativo, desenvolvido em um empresa privada, com aplicações utilizando documentos padronizados e planos de testes desde o início do desenvolvimento do software e, aplicações sem o uso de um plano formal de teste.

No capítulo 5 é apresentado as contribuições deste trabalho e perspectivas para trabalhos futuros.

# 2.TESTE DE SOFTWARE

Os princípios e objetivos do teste de software são apresentados, e também as principais técnicas de software que podem ser utilizadas no processo de desenvolvimento. Destaca-se a forma com que o teste de software pode contribuir para revisão de requisitos, projeto e implementação ao longo de todo o processo.

## 2.1. Considerações

Nas décadas de 60 e 70, os desenvolvedores dedicavam a maior parcela dos seus esforços e tempos na parte de codificação e testes de unidade. Estima-se que 80% deste esforço eram despendidos nestas atividades.

Do ponto de vista psicológico, teste de software poderia ser visto como destrutivo em vez de construtivo, pois, em um primeiro momento, o engenheiro tenta construir um software a partir de um conceito até obter um produto tangível. Depois vem o teste, que é uma série de casos de testes destinados a destruir o software que acabara de ser construído [PRESSMAN,2002].

A partir da década de 90, e principalmente nos dias atuais, com a utilização da Internet para a realização de negócios, houve uma mudança na abrangência e complexidade das aplicações, onde fatores como segurança e desempenho passaram a ser relevantes, tornando a atividade de testar cada vez mais especializada. Um site fora de operação por defeitos de programa poderá comprometer seriamente o negócio e/ou a imagem da organização [FR,2002].

No propósito de melhorar a qualidade e desenvolvimento de seus softwares, as empresas têm voltado suas atenções para os processos de testes, proporcionando uma grande economia em correções de softwares mal produzidos. Quanto antes à presença do defeito (*fault,* deficiência algorítmica que pode levar a violação das especificações) é revelada, menor o custo de correção e maior a probabilidade de corrigi-lo corretamente. Os custos de descobrir e corrigir defeitos no software aumentam exponencialmente na proporção em que o trabalho evolui através das fases do projeto [FR,2002].

O termo programa designa qualquer objeto que possa ser conceitualmente ou realmente executado [PRESSMAN,2002]. É este o significado utilizado no presente trabalho. Programas podem ser vistos como uma função matemática.

Os componentes essenciais de um teste de programa são:

- Programas em sua forma executável;
- Uma descrição de seu comportamento esperado;
- A definição de um modo de observar o comportamento do programa;
- Uma descrição do domínio da função;
- Um método para determinar, se o comportamento observado está de acordo com o esperado.

Dentre os cincos componentes necessários no processo de testes, o mais difícil de se obter é geralmente a descrição do comportamento esperado. Consequentemente, métodos "ad hoc" devem ser usados incluindo cálculo manual, simulações e soluções alternadas para o mesmo problema. Pode-se também construir um oráculo, um programa que para qualquer descrição de entrada dada pode prover uma descrição completa do comportamento da saída esperada [PRESSMAN,2002].

Uma completa validação[1] do programa em qualquer estágio do ciclo de vida pode ser obtida através da execução do processo de teste para todas possíveis condições de valores de entrada. Este método é conhecido como teste exaustivo, sendo a única técnica de testes que garantiria a validade do programa. Lamentavelmente, esta técnica não é viável. Na maior parte dos casos, o domínio da função é infinito, ou quando finito, grande o bastante para fazer o número de testes requeridos inviável. Uma consideração fundamental em um programa de teste é questão econômica [MYERS,1979].

Com o objetivo de reduzir o número potencialmente infinito de testes do processo de testes exaustivos para um número possível, deve-se encontrar um critério para selecionar elementos representativos do domínio da função. Estes critérios devem refletir tanto a descrição funcional quanto a estrutura do programa.

Casos de teste são um conjunto de entradas, condições de execuções, e resultados esperados desenvolvidos para um objetivo particular, como executar caminhos de um programa ou verificar

---

[1] Validação é o processo designado para aumentar a confiança que o programa funciona conforme o pretendido. Geralmente validação envolve testes. [OLAN,2003]

a conformidade de acordo com as especificações requeridas [IEEE,1990]. No entanto, o problema é encontrar um conjunto de casos de teste adequado, que seja grande o suficiente para englobar o domínio e suficientemente pequeno para que se possa testar cada elemento do conjunto. Desenvolver bons casos de teste é uma tarefa complexa [KANNER,2003].

Embora seja difícil medir e definir um *software* como sendo de alta qualidade, é fácil identificar um *software* de baixa qualidade. Os erros freqüentes como paralisação do sistema ou a inadequação aos requisitos são sempre notadas.

A principal técnica de teste continua sendo executar o *software* usando casos de testes representativos e comparar a saída gerada com a esperada. As diferenças representam falhas que devem ser corrigidas.

## 2.2. Princípios e Objetivos do Teste

Adaptando a citação em [MYERS,1979], tem-se que:

- Todos os testes devem ser relacionados aos requisitos do cliente. O objetivo do teste de software é descobrir defeitos, e os defeitos mais indesejáveis são aqueles que levam o programa a deixar de satisfazer seus requisitos.
- O planejamento de teste pode começar tão logo o modelo de requisitos seja completado. A definição detalhada dos casos de teste pode iniciar, assim que o modelo de projeto tenha sido consolidado. Desta forma, todos os testes podem ser planejados e projetados antes que qualquer código tenha sido gerado.
- O princípio de Pareto se aplica ao teste de software. Implica que 80% de todos os defeitos descobertos durante o teste podem ser relacionados a 20% de todos os componentes do programa. O problema, sem dúvida, é isolar os componentes suspeitos e testá-los rigorosamente.
- O teste deve começar “no varejo” e progredir até o teste “no atacado”. Os primeiros testes planejados e executados geralmente concentram-se nos componentes individuais. À medida que o teste progride, o foco se desloca numa tentativa de encontrar defeitos em conjuntos integrados de componentes, e finalmente, em todo o sistema.
- Teste exaustivo não é possível. A quantidade de permutações de caminhos mesmo para um programa de tamanho moderado é excepcionalmente grande.

- Para ser mais efetivos, os testes deveriam ser conduzidos por equipes ou pessoas não envolvidas com a codificação. Por mais efetivo, entende-se que o teste tem a maior probabilidade de encontrar defeitos.

O principal objetivo do teste de software é descobrir defeitos, e em conseqüência é garantir que o processo de desenvolvimento de software atinja níveis de qualidade especificados e um produto de alta qualidade e baixo custo.

Segundo [MYERS,1979]:

- 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 probalidade de encontrar um erro ainda não descoberto.
- Um teste bem-sucedido é aquele que descobre um erro que ainda não foi descoberto.

## 2.3. Técnicas de Teste

O projeto de um teste de software pode ser tão desafiador quanto o desenvolvimento do próprio produto. Elaborar testes para encontrar todos os erros de um programa é uma atividade praticamente impossível, em se tratando de aplicações não triviais. Conforme citado em [MYERS,1979], encontrar todos os erros em um programa é impraticável e impossível. O problema fundamental esta relacionado com o aspecto econômico do teste, tais como os possíveis resultados do programa e também como deverão ser projetados os casos de testes.

De acordo com [PRESSMAN,2002], qualquer produto que passa por um processo de modificação pode ser testado por duas maneiras: (1) sabendo a função especificada que o produto foi projetado para realizar, podem ser realizados testes que demonstram que cada função está plenamente operacional, enquanto ao mesmo tempo procuram erros em cada função; (2) sabendo como é o trabalho interno de um produto, podem se realizados testes para garantir que "todas as engrenagens combinam", isto é, que as operações internas são realizadas de acordo com as especificações e que todos os componentes internos foram adequadamente exercitados. A primeira abordagem de teste é chamada teste caixa-preta e a segunda, teste caixa-branca.

### *2.3.1. Teste caixa-preta*

Neste tipo de técnica o programa é visto como uma caixa-preta, ou seja, não há nenhum interesse quanto à forma interna e estrutura do programa. Somente interessa-se em encontrar circunstâncias em que o programa não funciona conforme as especificações, [MYERS,1979].

Os testes caixa-preta são projetados para validar os requisitos funcionais sem se prender ao funcionamento interno do programa. As técnicas de teste caixa-preta focalizam o domínio da informação do software, originando casos pelo particionamento dos domínios de entrada e saída de um programa.

O primeiro passo no teste caixa-preta é entender os módulos ou coleção de comandos que estão modelados no software e as relações que se conectam. Uma vez que isto tenha sido conseguido, o passo seguinte é definir uma série de testes que verifica se "todos" os módulos ou comandos têm a relação esperada uns com os outros [BEIZER,1995] e [PRESSMAN,2002].

Através do método particionamento de equivalência é possível dividir o domínio de entrada de um programa em classes de dados, das quais casos de testes podem ser derivados. O particionamento de equivalência busca definir um caso de teste que descobre classe de erros, reduzindo assim o número total de casos de teste que precisam ser desenvolvidos [PRESSMAN,2002].

Classes de equivalência podem ser definidas de acordo com as seguintes diretrizes:

1. Se uma condição de entrada especifica um *intervalo*, uma classe de equivalência válida e duas inválidas são definidas.
2. Se uma condição de entrada exige um *valor* específico, uma classe de equivalência válida e duas inválidas são definidas.
3. Se uma condição de entrada especifica um membro de um *conjunto*, uma classe de equivalência válida e uma inválida são definidas.
4. Se uma condição de entrada é *booleana*, uma classe de equivalência válida e uma inválida são definidas.

A construção de casos de testes deve considerar situações que exercitam valores limites. Ou seja, por motivos que não estão completamente claros, um grande número de erros tende a ocorrer nas fronteiras do domínio de entrada ao invés de no "centro" [PRESSMAN,2002]. Este tipo de técnica é conhecido como *Análise de valor-limite*.

A análise de valor-limite é uma técnica de projeto de casos de teste que vem completar o particionamento de equivalência, aumentando as chances de encontrar erros.

De acordo com [MYERS,1979] citado em [PRESSMAN,2002], em vez de focalizar somente as condições de entrada, a análise de valor-limite deriva casos de teste também para o domínio de saída.

De acordo com [PRESSMAN,2002], as diretrizes para esta técnica são:

1. Se uma condição de entrada especifica um intervalo limitado pelos valores *a* e *b*, casos de testes devem ser projetados com os valores *a* e *b* imediatamente acima e imediatamente abaixo de *a* e *b*.
2. Se uma condição de entrada especifica vários valores, casos de teste devem ser desenvolvidos para exercitar os números mínimo e máximo. Valores imediatamente acima e imediatamente abaixo do mínimo e do máximo também são testados.
3. Aplica as diretrizes 1 e 2 às condições de saída. Por exemplo, considere que uma tabela de temperatura versus pressão é esperada como saída de um programa de análise de engenharia. Casos de teste devem ser projetados para criar um relatório de saída que produza o número máximo (e mínimo) admissível de entradas na tabela.
4. Se as estruturas de dados internas do programa têm limites prescritos (p.ex., um vetor tem um limite definido de 100 entradas), certifique-se de projetar um caso de teste para exercitar a estrutura de dados no seu limite.

Uma outra técnica funcional utilizada é o teste baseado em modelos, que a partir de informações sobre o comportamento funcional do software, pode-se determinar como será realizado o teste. Através dos requisitos funcionais do software determina-se quais ações são possíveis em sua execução e quais saídas são esperadas [AD,1997].

Como é possível modelar várias formas do comportamento do software, esta técnica torna-se bastante ampla, implicando para cada técnica de modelagem diferentes critérios associados. Exemplos de critérios dessa técnica são todos_os_estados, todas_as_transições e todos_os_caminhos, os quais exigem, respectivamente, que cada estado, cada transição e cada caminho do modelo comportamental do programa seja executado pelo menos uma vez durante o teste.

### *2.3.2. Teste caixa-branca*

O teste caixa-branca, ao contrário do caixa-preta, focaliza a estrutura do programa, como funções, módulos ou rotinas, para a elaboração de casos de teste. Neste trabalho não será tratado

com detalhes esta técnica por não se adequar ao método utilizado na empresa para a construção dos casos de testes. Apenas será citado superficialmente o conceito de caixa-branca. Informações mais detalhadas poderão ser encontradas nas citações mencionadas.

De acordo com [PRESSMAN,2002], usando técnicas de teste caixa-branca, o engenheiro de software pode derivar casos de teste que (1) garantam que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez, (2) exercitam todas as decisões lógicas em seus lados verdadeiro e falso, (3) executam todos os ciclos nos seus limites e dentro de seus intervalos operacionais, e (4) exercitam as estruturas de dados internas para garantir sua validade.

A natureza dos defeitos de software, tais como *erros lógicos*, *caminhos que acreditamos não ser percorridos* e *erros tipográficos*, é um forte motivo para a realização de testes caixa-branca [JONES,1981] e [PRESSMAN,2002].

O teste de caminho básico, proposto por [McCABE,1976], possibilita que o projetista do caso de teste derive uma medida da complexidade lógica de um projeto procedimental e use essa medida como guia para definir um conjunto básico de caminhos de execução. Os casos de teste derivados para exercitarem o conjunto básico têm a garantia de executar cada instrução do programa pelo menos uma vez durante a atividade de teste.

Os caminhos que devem ser testados podem ser identificados através da técnica Complexidade Ciclomática. Através desta é possível identificar o número máximo de testes que devem ser realizados para garantir que todas as instruções sejam executadas pelo menos uma vez.

Um caminho independente é qualquer caminho através do programa que introduza um novo conjunto de instruções de processamento ou uma condição nova.

A complexidade ciclomática tem suas bases na teoria dos grafos, figura 1. Neste método as construções estruturadas do programa são representadas na forma de grafos de fluxo [PRESSMAN,2002]. Uma ou mais instruções procedimentais denomina-se nó. Os arcos do grafo de fluxo, que indicam o fluxo lógico do processamento, são denominados ramos. Se um nó contém uma condição ele é denominado nó predicativo.

```
1     início
1       leia nro
2       enquanto nro ≠ 0
3               se nro > 0
4                       raiz = raiz-quadrada(nro)
4                       escreva raiz
5               senão
5                       escreva mensagem de erro
6               fim-se
6       leia nro
7       fim-enquanto
```

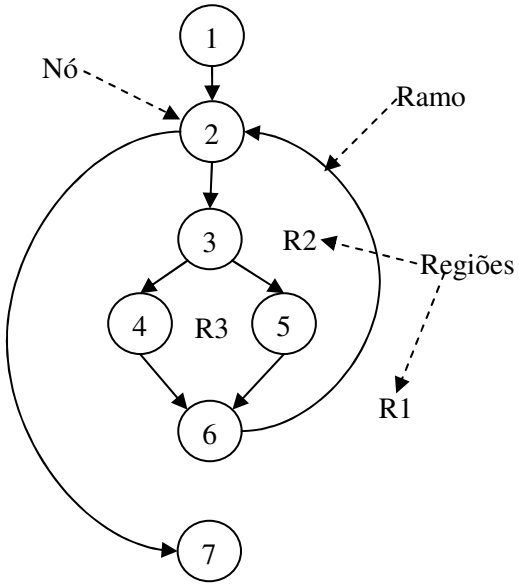


**Figura 1 -** Exemplo de grafo de fluxo derivado de um trecho de programa

A forma de cálculo da complexidade e os métodos utilizados para projetar casos de testes podem ser conseguidos em [BEIZER,1990] e [PRESSMAN,2002].

### 2.4. Fases de testes

As principais fases de teste podem ser classificados em teste de unidade, teste de integração, teste de sistema e teste de validação. A estratégia começa pelo teste de unidade onde focam-se os componentes individualmente, garantindo que funcionem adequadamente, então, passa-se para o teste de integração que é a montagem dos componentes e construção do programa. Neste momento, testes podem ser refeitos para assegurar que não há defeitos, testes de regressão. Depois que o software foi integrado, passa-se ao teste de sistema que é a verificação se todos os elementos como, por exemplo, hardware e bases de dados combinam adequadamente, e se a função/desempenho global do sistema é conseguida. O último passo é o teste de validação, o qual engloba o teste de aceitação do usuário, que fornecerá a garantia final que o software satisfaz todos os requisitos funcionais, comportamentais e desempenho.

- Testes de unidade:

O teste de unidade focaliza o esforço de verificação de uma unidade funcional de projeto do software que foi implementada. Uma unidade funcional é um componente de software que não pode ser subdividido [IEEE,1990]. Em programas, refere-se a uma sub-rotina ou a um procedimento que é a menor parte funcional de um programa que pode ser executada.

O teste é orientado para caixa branca e pode ser conduzido em paralelo para diversos componentes [PRESSMAN,2002].

- Testes de integração:

O teste de integração é uma técnica sistemática para construir uma estrutura de programa enquanto, ao mesmo tempo, conduz testes para descobrir erros associados às interfaces. O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura do programa que foi determinada pelo projeto.

- Teste de regressão

Toda vez que um programa sofre algum tipo de alteração existe uma possibilidade considerada que algo deixe de funcionar conforme o esperado ou ainda, durante a fase de desenvolvimento onde nas estratégias *bottom-up* e *top-down* novos módulos são acrescidos ao programa principal erros inesperados podem ocorrer.

O teste de regressão é importante quando o software sofre manutenções, pois é a realização de algum subconjunto de testes que já havia sido executado em uma outra etapa, com o objetivo de garantir que as manutenções introduzidas não causaram erros ou comportamento indesejável.

O subconjunto de casos de teste a ser executado no teste de regressão apresenta três diferentes classes de teste [PRESSMAN,2002]:

- Uma mostra representativa de testes que vai exercitar todas as funções do software.
- Testes adicionais que focalizam funções do software, que serão provavelmente afetadas pela modificação.
- Testes que focalizam os componentes de software que foram modificadas.

Durante o teste de integração muitos outros casos de testes devem surgir e com isso o número de casos de teste de regressão aumenta significativamente. Desta forma, não é prático e nem eficiente executar todos os testes toda vez que ocorrer uma modificação. Assim sendo, o subconjunto de casos de testes de regressão deve-se limitar a incluir apenas testes que cuidam de uma ou mais classes de erros em cada uma das principais funções do programa.

- Testes de sistema:

Teste de sistema é uma série de diferentes testes cuja finalidade principal é exercitar por completo o sistema baseado em computador. Cada teste tem uma finalidade distinta e todos trabalham para verificar se os elementos do sistema foram adequadamente integrados e executam as funções a eles alocadas [PRESSMAN,2002].

Os testes de sistemas podem ser classificados em: teste de recuperação, teste de segurança, teste de estresse e teste de desempenho, onde:

- Teste de recuperação é um teste que tem como objetivo forçar o sistema falhar de diversos modos e verificar se a recuperação automática ou pelo próprio sistema, é adequadamente realizada sem causar uma parada global do sistema. Ou ainda, se a recuperação for necessária através de intervenção humana o tempo de correção está dentro dos limites aceitáveis. Falhas que não forem corrigidas em certo tempo poderá acarretar prejuízo econômico, como por exemplo, sites de internet, aplicativos de call center, etc.
- Teste de segurança é o teste que se tenta de todas as formas invadir o sistema e verificar se os mecanismos incorporados ao software irão protegê-lo de acessos indevidos. Um bom teste de segurança vai acabar invadindo o sistema, mas o papel do projetista do sistema é tornar o custo da invasão maior do que o valor da informação que vai ser obtida [PRESSMAN,2002].
- Teste de estresse é o teste onde o programa é submetido a uma série de condições anormais de funcionamento, como um grande número de usuários faz diversos tipos e formas de acesso ao banco de dados ao mesmo tempo ou durante um longo período. Casos de testes que exigem um elevado processamento e recursos de máquina são projetados com o objetivo de destruir o programa.
- Teste de desempenho é o teste realizado dentro de um sistema integrado com o objetivo de avaliar o tempo de execução do software. O teste de desempenho é realizado ao longo de todo o processo de teste a partir da etapa de unidade até a etapa em que todos os elementos estejam acoplados. Freqüentemente são acoplados ao teste de estresse e usualmente requerem instrumentação tanto de hardware quanto de software para medir a utilização precisa dos recursos. Fazendo isto, o usuário poderá descobrir situações que levam à degradação e possíveis falhas do sistema.
- Testes de validação:

Após os testes de unidade e integração o software já está praticamente montado e estruturado, as interfaces estão funcionando e as funções estão de acordo com os requisitos de testes.

A realização dos testes de validação é feita através de uma série de testes caixa-preta que procuram demonstrar conformidade com os requisitos. Após a realização de cada caso de teste é possível conhecer se as características de função ou desempenho satisfazem à especificação e se são aceitas e, havendo um desvio da especificação, uma lista de deficiências é criada.

Para ajudar nestes testes a participação do usuário poderá ser de grande valia, visto que, até neste momento, boa parte do programa ou ele todo já está terminado. Os testes que envolvem muitos usuários são chamados de testes *Alfa* e *Beta*.

Teste *alfa* é o teste realizado nas dependências do desenvolvedor em um ambiente controlado e com acompanhamento do desenvolvedor fazendo apenas as devidas anotações dos erros e problemas de uso. Teste *beta* é realizado fora das dependências do desenvolvedor, podendo ser realizado por mais de um cliente em seus respectivos ambientes reais de uso. Sem a presença do desenvolvedor os erros são reportados para que posteriormente as correções sejam feitas e o software seja liberado para venda. A Microsoft é um bom exemplo de empresa que utiliza desta estratégia para lançar seus produtos.

## 2.5. Considerações Finais

Neste capítulo são apresentados os princípios e objetivos do teste de software como uma atividade fundamental à qualidade do software. Os testes são classificados como caixa-preta quando focam nas funcionalidades pretendidas do software; e em caixa-branca quando focam no código fonte do programa para a escolha dos teste a serem executados.

Através de uma técnica de teste de software é possível planejar as atividades antecipadamente em cada uma das fases do desenvolvimento do software, aumentando as chances de se reduzir os defeitos.

# 3.METODOLOGIAS DE TESTE

A metodologia de teste deve seguir métodos que possibilitem organizar a atividade de testar aplicações no contexto da empresa. Como é indesejável o desenvolvimento de sistemas sem uma metodologia adequada, também acontece o mesmo com a atividade de teste [FR,2002].

A implantação de uma metodologia de testes pode ser vista por muitos como um causador de aumento dos custos e prazos dos projetos, podendo, deste modo, vir a criar hostilidade entre os membros do projeto e os responsáveis pelos testes e problemas com usuários [FR,2002]. Entretanto, esta visão está sendo substituída por uma outra que privilegia a qualidade e, como conseqüência, enfatiza o maior esforço no processo, entendendo que testar é um investimento em qualidade e que este produz um retorno positivo e contribui decisivamente para o sucesso do projeto [BULLOCK,2005].

Conforme sugere [BLACK,2005], tem-se um exemplo simples para justificar a montagem de uma estrutura formal para testes.

Para o exemplo é assumido que, em um projeto de 10.000 horas, foram identificados 1000 defeitos. Os custos para correção nas fases de desenvolvimento, testes e após a implantação foram de R$10,00, R$100,00 e R$1.000,00, respectivamente.

Conforme é mostrado na tabela 1, tem-se na primeira coluna a identificação dos itens de custo. Na segunda coluna tem-se a representação das estruturas de testes normalmente encontrada nas organizações, ou seja, processos sem uma estrutura formal realizados pelos desenvolvedores e usuários. Neste caso, durante o desenvolvimento, foram identificados e corrigidos 250 defeitos e o restante (750) foram descobertos e corrigidos após a implantação (na manutenção).

Na terceira coluna há a representação de uma organização que possui uma estrutura dedicada para execução dos testes. E neste caso, durante o desenvolvimento, foram identificados e corrigidos, pela equipe de desenvolvimento, os mesmos 250 defeitos, 350 pela equipe de teste e 450 após a implantação (pelo usuário ou na produção).

Tabela 1. Exemplo de retorno de investimento [BLACK,2005]

| | **Processo sem uma estrutura formal de teste (P1)** | **Processo com uma estrutura formal de teste (P2)** |
|---|---|---|
| Investimentos em teste com pessoal | R$0,00 | R$90.000,00 |
| Investimentos em teste com infra-estrutura | R$0,00 | R$16.000,00 |
| ***Total do investimento*** | ***R$0,00*** | ***R$106.000,00*** |
| Defeitos encontrados durante o desenvolvimento | 250 | 250 |
| Custo dos erros encontrados durante o desenvolvimento | R$2.500,00 | R$2.500,00 |
| Defeitos encontrados pela equipe de teste | 0 | 350 |
| Custo dos erros encontrados pela equipe de teste | R$0,00 | R$35.000,00 |
| Defeitos encontrados após a implantação em produção | 750 | 400 |
| Custo dos erros encontrados após a implantação em produção | R$750.000,00 | R$400.000,00 |
| ***Custo da qualidade*** | ***R$752.500,00*** | ***R$543.500,00*** |
| ***Redução do custo (P1 – P2)*** | ***R$209.000,00*** | |
| ***% Redução Custo / Investimento*** | ***197%*** | |

Os custos considerados da equipe de teste foram de 30% das horas do projeto a R$30,00/hora e os custos de infra-estrutura, correspondente a 2 equipamentos para uso da equipe de testes por 8 meses a R$1000,00 cada.

Com base nessas premissas, a redução de custos com o processo formal de testes foi de R$209.000,00, que representa 197% (R$209 / R$106) em relação custo de investimento em uma equipe de teste.

Uma inadequada infra-estrutura (pessoal, equipamentos, metodologia e ferramentas) para testar softwares leva à situação onde os defeitos são identificados mais tarde no processo de desenvolvimento, requerendo mais recursos para localizar e corrigir a fonte dos problemas [FR,2002]. As conseqüências afetam os custos de desenvolvimento nos seguintes aspectos;

- Mão de obra: pessoal adicional para garantir as correções;

- Hardware: equipamentos adicionais para os novos técnicos ou mais potentes para permitir rodar ferramentas de apoio contratadas emergencialmente;
- Software: para os equipamentos adicionais e/ou ferramentas de apoio que precisem ser contratados para apoio em situação de emergência;
- Atrasos: multas, moras etc. Previstas em contratos (se for o caso), suspensão de pagamento etc.;
- Imagem: perda de credibilidade junto ao usuário que podem ter impactos em futuros negócios.

Neste trabalho é proposto um modelo de ciclo de vida do projeto interagindo com as atividades de teste em cada estágio do ciclo de desenvolvimento do software. Esta integração permite que erros sejam detectados logo no início das atividades e que não sejam propagadas para outras fases do desenvolvimento do software.

A metodologia de teste utilizada neste estudo é a 3P x 3E [FR,2002] com a utilização de documentos sugeridos pela norma IEEE 829-1998 [IEEE,1998]. A justificativa para a utilização dessas está diretamente relacionado ao fato de atender sistemas comerciais, serem de fácil implementação e também por não exigirem utilização de metodologias OO para modelar sistemas, tais como o UML.

Entretanto, até mesmo pela importante contribuição que as metodologias OO trazem para o processo de desenvolvimento de software, será também apresentado neste capítulo a metodologia AGEDIS que se utiliza deste conceito, para estudo de comparações e trabalhos futuros.

## 3.1. Modelo 3P x 3E

A metodologia de teste proposta em [FR,2002] foi confeccionada para servir de modelo básico para a elaboração e acompanhamento/monitoração do processo de testes, assim como, deve ser aderente à Metodologia de Desenvolvimento de Sistemas utilizada pela organização.

O processo de teste deve ser inserido nas primeiras fases do processo de desenvolvimento e serem realizadas simultaneamente, até a implantação dos softwares, para que sejam atingidos os resultados esperados.

O processo de teste, conforme figura 2, é composto por diversas etapas ou fases, sendo quatro delas seqüenciais ou em cascata e duas paralelas.

O modelo recebe o nome 3P x 3E, pois têm relação com as atividades, produtos e documentos. O primeiro P (Procedimentos Iniciais) é uma fase pequena, na qual é traçado, em linhas gerais, um pequeno esboço do processo de teste e assinado um acordo de nível de serviço. Os outros dois P's (Planejamento e Preparação) são atividades complementares que dão suporte ao processo e que acompanham todo o processo de teste. Os três E´s representam o âmago de todo processo de teste e consomem em torno de 80% a 85% de todo o processo.

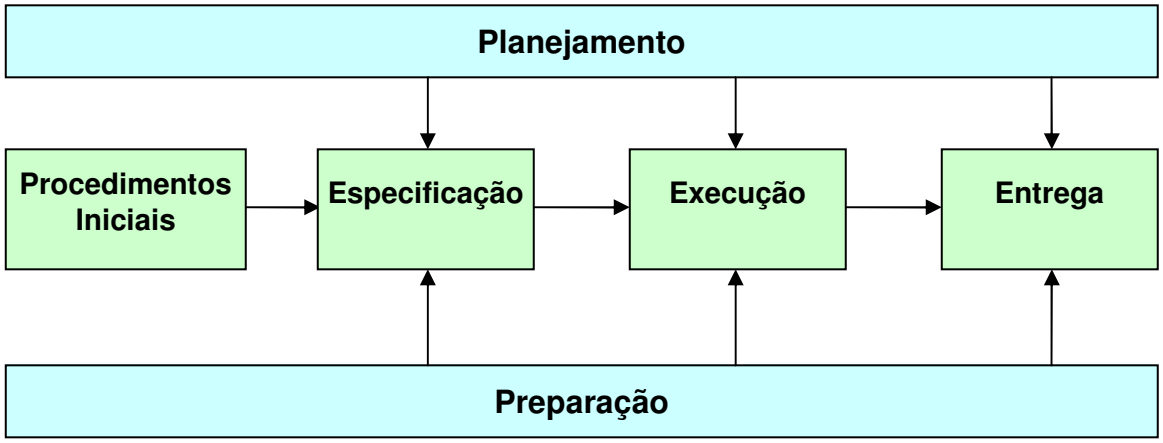


**Figura 2 -** Ciclo de vida do processo de teste

A seqüência de etapas da execução dos testes é a seguinte:

1. Procedimentos iniciais
2. Planejamento
3. Preparação
4. Especificação
5. Execução
6. Entrega

Cada etapa do modelo é composta de subetapas, insumos e produtos, conforme mostrado na tabela 2. A subetapa representa, em linhas gerais, o que vai ser executado e criado dentro do item. Uma etapa poderá contar com uma ou mais subetapas. O insumo contém os objetos necessários para início da etapa ou subetapa. E o produto são os produtos propriamente ditos criados pelas etapas e subetapas.

Tabela 2. Etapas e subetapas modelo 3P x 3E [FR,2002]

| **Etapa** | **Subetapa** | **Insumo** | **Produto** |
|---|---|---|---|
| 1) Procedimentos Iniciais | 1.1) Elaborar o Guia Operacional de Teste | Requisitos do Negócio<br>Modelos de dados<br>Diagrama de fluxo de dados.<br>Outros documentos de desenvolvimento | Guia Operacional de Teste - GOT |
| 2) Planejamento | 2.1) Estabelecer Estratégia de Testes | Requisitos do Negócio<br>Modelos de dados<br>Diagrama de fluxo de dados.<br>Outros documentos de desenvolvimento<br>Guia Operacional de Teste | Estratégia de Teste<br>Análise de Riscos do Projeto de Teste |
| | 2.2) Estabelecer Plano de Teste | Estratégia de Teste<br>Análise de Riscos do Projeto de Teste<br>Necessidades de dados de testes<br>Planejamento do sistema que está desenvolvendo | Plano de testes - nova versão<br>Análise de Riscos do Projeto de Testes |
| | 2.3) Revisar Estratégia de Testes | Requisitos de negócio do sistema<br>Estratégia de Teste | Estratégia de Testes revisada |
| | 2.4) Revisar Plano de Testes | Estratégia de Testes<br>Plano de Testes | Plano de Testes revisado |
| 3) Preparação | 3.1) Adequar o projeto de testes à Gerência de Configuração e/ou de controle de mudanças | Arquitetura do ambiente de desenvolvimento<br>Arquitetura do ambiente de produção<br>Ferramentas e procedimentos de Gerência de Configuração e de mudança | Registro e controle das diversas versões do produto: funcional, desenvolvimento, produto e operacional |
| | 3.2) Disponibilizar infra-estrutura e ferramentas de teste | Estratégia de testes<br>Arquitetura básica do ambiente de desenvolvimentoArquitetura básica do ambiente de produçãoFerramentas de teste | Infra-estrutura e ferramentas de teste disponíveis para a equipe de teste |
| | 3.3) Disponibilizar pessoal | Estratégia de Testes<br>Plano de Testes<br>Ferramentas de testes<br>Definição do ambiente de teste | Equipe de testes definida e capacitada |

| Etapa | Subetapa | Insumo | Produto |
|---|---|---|---|
| 4) Especificação | 4.1) Elaborar casos de teste | Estratégia de testes<br>Plano de Testes<br>Documentação técnica do sistema<br>Necessidades de dados de teste<br>Posição quanto aos testes já realizados | Casos de Testes<br>"Scripts" de testes (se usar ferramentas)<br>Especificação das necessidades de dados de testes |
| | 4.2) Elaborar Roteiros de Teste | Casos de Testes<br>Planos de Teste<br>Fluxo de execução dos programas previsto pela equipe de desenvolvimento | Roteiros de Testes |
| 5) Execução | 5.1) Preparar dados de testes | Casos de Testes<br>"Scripts" de Testes<br>Roteiros de Testes<br>Documentação do Sistema<br>Especificação das necessidades de dados de testes<br>Processos de criação de bases e/ou arquivos de teste | Bases/Arquivos de teste disponíveis |
| | 5.2) Executar testes | Roteiros de Testes<br>Casos de Teste<br>"Scripts" de Testes<br>Resultados Esperados | Resultados dos testes<br>Relatórios de defeitos encontrados<br>Ajustes no material de testes |
| | 5.3) Solucionar ocorrência de testes | Relatórios de defeitos com status a resolver<br>Resultados dos testes | Relatórios de defeitos encontrados com status resolvido ou a avaliar |
| | 5.4) Acompanhar a execução dos Casos de Testes | Relatórios de defeitos (resumo)<br>Resultados dos testes (resumo)<br>Estratégia de teste<br>Plano de Testes<br>Casos de Testes<br>Roteiros de Testes | Análise do andamento dos Casos de Testes |
| | 5.5) Elaborar Relatório Final | Análise dos resultados de teste<br>Estratégia de Testes<br>Resultados de testes<br>Relatórios de defeitos - resumo<br>Plano de testes | Relatório final do testes |
| 6) Entrega | 6.1) Avaliação e Arquivamento da documentação | Documentos de testes | Relatórios de não conformidade<br>Relatório final de testes<br>Documentação arquivada |

Os principais documentos produzidos nesta metodologia são: GOT (guia operacional de testes), estratégias de testes, plano de testes, casos de testes e roteiros de testes.

O GOT é um documento que não necessariamente precisa fazer parte do projeto, visto que é um documento com intuito de formalizar entre as partes envolvidas (desenvolvedores, usuários e equipe de teste), o início do projeto de testes, a responsabilidade de cada um e uma análise inicial dos riscos envolvidos.

O documento de estratégia de testes tem por finalidade fornecer uma visão geral do projeto de teste e as diretrizes para execução do processo. De acordo com as necessidades da utilização nas organizações estes documentos poderão ser alterados.

Dentre as principais informações contidas na estratégia de teste, deve estar claramente definido o objetivo e as metas a serem atingidas, o ambiente de teste, tais como equipamentos, softwares, pessoal e ferramentas, e a abordagem dos testes a se considerar (unidades, integração, sistema e aceitação).

O plano de teste contém o projeto ou desenho lógico do processo de teste e está alinhado com a estratégia de teste. Define os objetivos gerais esperados e as expectativas do projeto de teste.

O documento caso de teste contém os testes propriamente ditos, detalhando campos de formulários, arquivos, telas e outros. As principais informações neste documento são: entradas e resultados esperados.

No roteiro de teste deve ser registrada a seqüência lógica de passos a serem executados dos casos de testes.

Em correspondência entre as etapas de desenvolvimento do sistema e a integração como os tipos de testes, a execução do teste não segue uma relação unívoca com as etapas de desenvolvimento, pois cada teste poderá corresponder a mais de uma etapa, figura 3. Esse modelo acaba sendo uma abstração do modelo em V [SOMMERVILLE,1999], onde o processo de teste ocorre em paralelo com o processo de desenvolvimento desde o seu início.

O objetivo maior de se executar os testes cada vez mais próximos do início do desenvolvimento é a redução de custos [FR,2002]. O custo da correção dos defeitos cresce exponencialmente à medida que o projeto de desenvolvimento avança dentro do ciclo de vida do sistema [MYERS,1979].

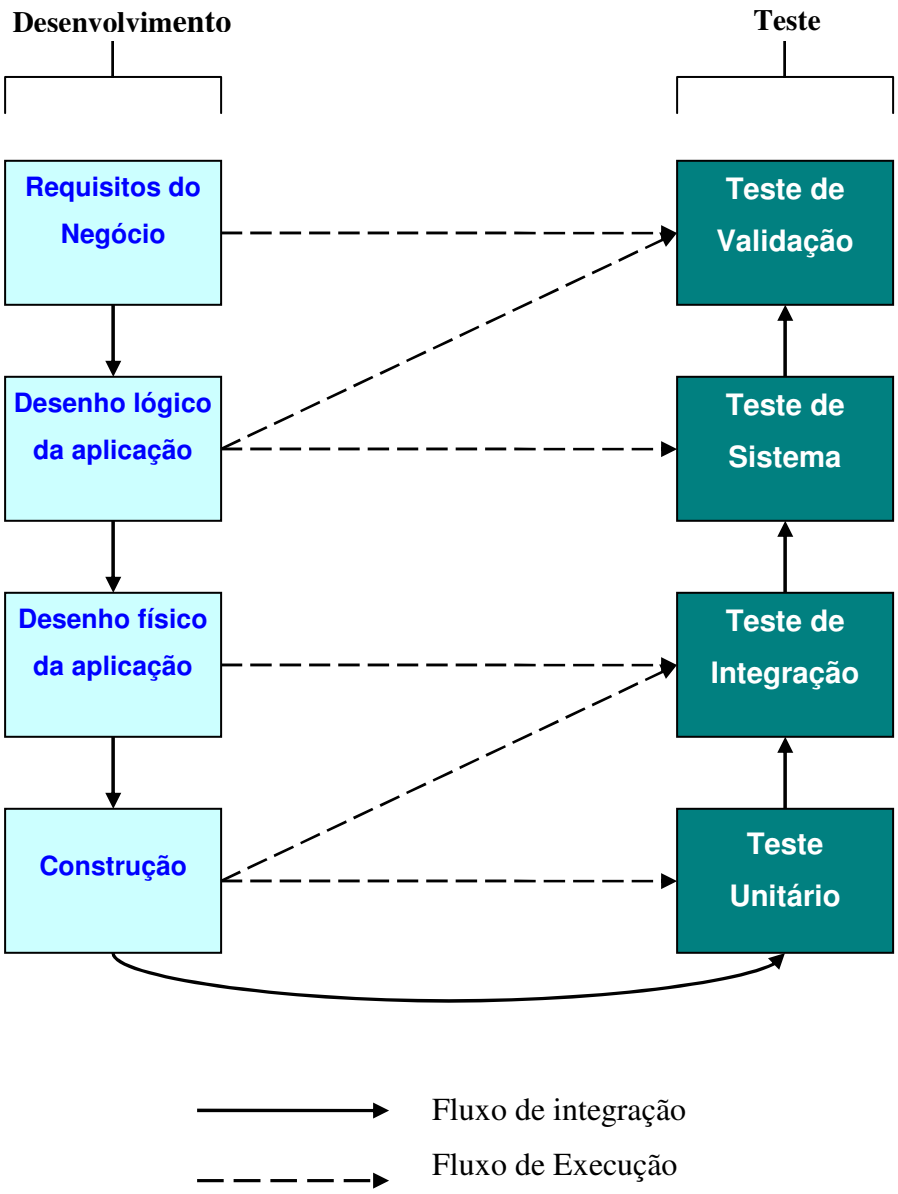


**Figura 3** - Etapas de desenvolvimento e a integração com os tipos de teste

### 3.2. Norma IEEE 829

A Norma IEEE 829, [IEEE,1998] descreve um conjunto de documentos para as atividades de teste de um produto de software.

Os oito documentos definidos pela norma, que cobrem as tarefas de planejamento, especificação e relato de testes, são apresentados a seguir:

1. **Plano de Teste** – Apresenta o planejamento para execução do teste, incluindo a abrangência, abordagem, recursos e cronograma das atividades de teste. Identifica também, os itens e as funcionalidades a serem testados bem como as tarefas a serem realizadas e os riscos associados com a atividade de teste.

A tarefa de especificação de testes é coberta por 3 documentos:

2. **Especificação de Projeto de Teste** – Refina a abordagem apresentada no Plano de Teste e identifica as funcionalidades e características a serem testadas pelo projeto e por seus testes associados. Este documento também identifica os casos e os procedimentos de teste, se existirem, e apresenta os critérios de aprovação.
3. **Especificação de Caso de Teste** – Define os casos de teste, incluindo dados de entrada, resultados esperados, ações e condições gerais para a execução do teste.
4. **Especificação de Procedimento de Teste** – Especifica os passos para executar um conjunto de casos de teste.

Os relatórios de teste são cobertos por 4 documentos:

5. **Diário de Teste** - Apresenta registros cronológicos dos detalhes relevantes relacionados com a execução dos testes.
6. **Relatório de Incidente de Teste** - Documenta qualquer evento que ocorra durante a atividade de teste e que requeira análise posterior.
7. **Relatório-Resumo de Teste** – Apresenta de forma resumida os resultados das atividades de teste associadas com uma ou mais especificações de projeto de teste e provê avaliações baseadas nesses resultados
8. **Relatório de Encaminhamento de Item de Teste** – Identifica os itens encaminhados para teste no caso de equipes distintas serem responsáveis pelas tarefas de desenvolvimento e de teste.

A norma separa as atividades de teste em três etapas: preparação do teste, execução do teste e registro do teste. A Figura 4 mostra os documentos que são produtos da execução de cada uma das fases e os relacionamentos entre eles.

Embora a norma possa ser utilizada para o teste de produtos de software de qualquer tamanho ou complexidade, projetos pequenos ou de baixa complexidade podem agrupar alguns documentos propostos, diminuindo o gerenciamento e os custos de produção dos documentos. Além disso, o conteúdo dos documentos também pode ser abreviado.

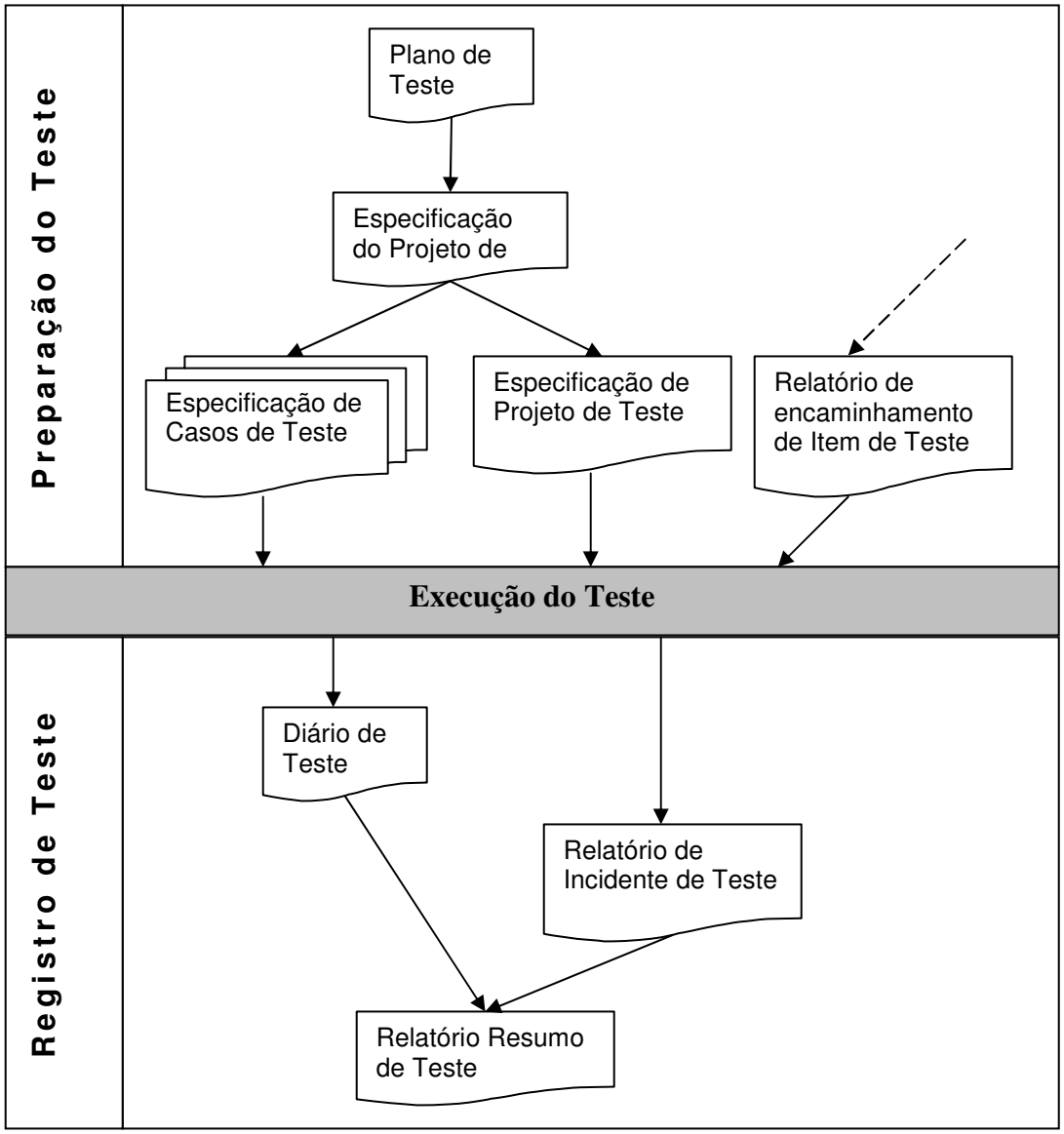


**Figura 4 -** Relacionamento entre os Documentos de Teste [CSBSAJ,2004]

Adicionalmente, a equipe responsável pelo teste deverá tomar outras decisões em relação à aplicação da norma em projetos específicos, decidindo, por exemplo, se é mais conveniente elaborar um único plano que englobe os testes de unidade, integração e aceitação, ou um plano para cada uma das fases de teste citadas.

Mais do que apresentar um conjunto de documentos, a serem utilizados ou adaptados para determinadas empresas ou projetos, a norma apresenta um conjunto de informações necessárias

para o teste de produtos de software. Sua correta utilização auxiliará a gerência a se concentrar tanto com as fases de planejamento e projeto quanto com a fase de realização de testes propriamente dita, evitando a perigosa armadilha de só iniciar a pensar no teste de um produto de software após a conclusão da fase de codificação.

### 3.3. Metodologia do CenPRA

A metodologia desenvolvida pelo CenPRA[2] é fundamentada na adoção de um processo de teste utilizando modelos sugeridos pela Norma IEEE 829, e tem como objetivo melhorar o processo de teste de softwares nas empresas através de técnicas, procedimentos e ferramentas. Desta forma, as empresas seriam capazes de desenvolver produtos de melhor qualidade [CSBSAJ,2004].

A metodologia é aplicada a qualquer tipo de software, seja ele sistema de informações ou software científico, e também podendo ser empregada tanto em empresas que desenvolvem quanto empresas que adquirem software.

A implantação do processo de teste envolve um conjunto de atividades que vai desde o levantamento das necessidades da empresa, passa pela realização de treinamentos da equipe técnica e vai até ao acompanhamento dos trabalhos realizados, constituindo assim, um completo ciclo de implantação da atividade de teste dentro da empresa.

O processo de teste proposto na metodologia está baseado em alguns pressupostos básicos:

- Os testes de sistema e aceitação são projetados e executados sob a responsabilidade da equipe de teste.
- Os testes de sistema, e eventualmente também o de aceitação, são realizados de forma iterativa, havendo, antes do início de cada ciclo de teste, uma avaliação rápida do produto.

Os dois documentos que formam a base da metodologia são: Guia para Elaboração de Documentos de Teste de Software [CenPRA,2001a] e Processos para Elaboração de Documentos de Teste de Software [CenPRA,2001b]. O primeiro tem o propósito de servir de referência para a criação dos documentos de teste tanto na fase de preparação para a atividade de teste quanto na

[2] Centro de Pesquisas Renato Archer – Campinas – SP – Brasil

fase de registro dos resultados do teste. O segundo documento, apresenta os processos relativos à preparação, execução e o registro dos testes.

Esses processos estabelecem uma orientação geral e, se necessário, podem ser modificados para adequar-se às situações particulares de organizações envolvidas nas atividades de teste. Um processo é definido para cada documento da Norma, segundo a seguinte estrutura [CSBSAJ,2004]:

1. Funções e responsabilidades no processo – participantes na execução das tarefas;
2. Critérios para o início do processo – elementos e/ou condições necessários para iniciar a execução das tarefas
3. Entradas do processo – dados, recursos ou ferramentas necessários para a execução das Tarefas
4. Tarefas do processo – ações necessárias para produzir as saídas do processo. Para cada tarefa são identificadas suas entradas, com indicação de possíveis fontes, e as saídas produzidas. A ordem de apresentação das tarefas não reflete necessariamente a seqüência em que devem ser executadas
5. Saídas do processo – dados ou produtos gerados pela execução das tarefas
6. Critérios para término do processo – elementos e/ou condições necessários para encerrar a execução das tarefas; e
7. Medições do processo – medidas a serem coletadas como parte da execução das tarefas.

### 3.4. Agedis

A metodologia AGEDIS (Automated Generation and Execution of Test Suites for Distributed Component-based Software) [AGEDIS], é o resultado da pesquisa de um grupo de empresas européias lideradas pela IBM com o propósito de promover melhorias na qualidade dos softwares e reduzir despesas na fase de teste.

Seguindo a mesma idéia onde defeitos detectados antes do desenvolvimento são mais baratos para corrigir [BEIZER,1995], a metodologia AGEDIS recomenda que o teste inicia-se assim que especificações do software estejam definidas. Desta forma, a preparação dos testes irá caminhar junto com o desenvolvimento do software facilitando na detecção de erros antes do código propriamente dito.

Conforme mostrado na figura 5, a metodologia é dividida em seis passos, os quais são:

1. Criação de um modelo abstrato do software a ser testado.
    a. A partir dos requisitos é criado um modelo abstrato do software contendo a descrição dos tipos de dado (classes), as estruturas de dados (objetos) e a transição entre eles.
2. Levantamento das informações do teste contendo:
    a. Descrição dos critérios de cobertura, especificação do propósito de teste e as restrições dos testes
    b. Descrição das interfaces de teste e os pontos de controles
3. Geração automática dos casos de testes levantados.
4. Revisão dos casos de teste junto aos desenvolvedores, equipe de teste e os prováveis usuários.
5. Execução dos casos de testes e registro dos resultados.
6. Análise dos resultados e defeitos encontrados e refazer as etapas de 2 a 5 até atingir os objetivos e metas estabelecidas.

Nessa metodologia, a produção de alguns artefatos necessários às etapas seguintes como, por exemplo, a criação dos casos de testes, é obtida através da utilização de algumas ferramentas disponibilizadas pela própria AGEDIS.

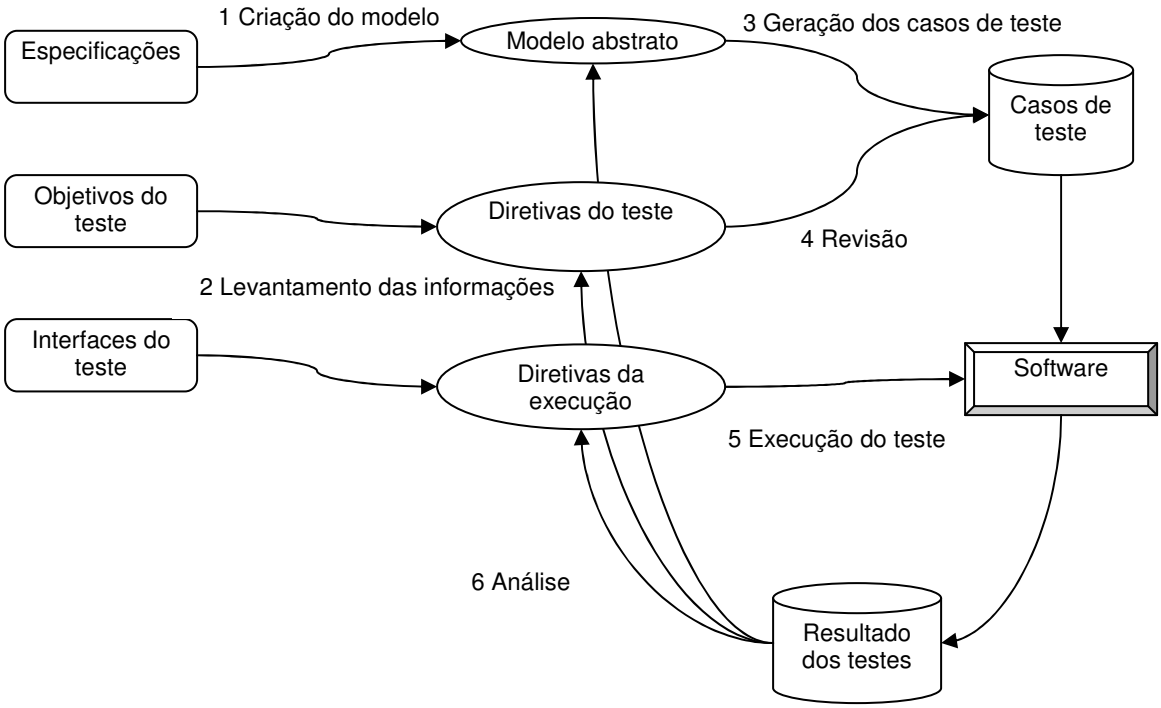


**Figura 5 -** Metodologia Agedis

Os principais benefícios destacados nesta metodologia são:

- Iniciando o processo a partir da especificação há um envolvimento dos usuários que farão os testes com o acompanhamento da equipe de teste desde o inicio do processo de desenvolvimento.
- Construindo o modelo e a interface de teste antes do código propriamente dito pode-se detectar erros de especificação e ajustar o modelo e plano de teste.
- Geração automática dos casos de teste possibilita uma maior garantia na cobertura dos casos de teste.
- Execução automatizada dos casos de testes possibilita encontrar erros de codificação e interfaces e reduzir custos na execução dos testes.

Algumas outras ferramentas ainda estão sendo desenvolvidas, conforme informação do próprio portal da AGEDIS. Para este estudo não será necessário detalhar tais ferramentas.

## 3.5. Considerações Finais

Neste capítulo são apresentadas metodologias existentes que permitem um melhor gerenciamento das atividades de teste e desenvolvimento do software.

Na metodologia proposta em 3P X 3E e na utilização de artefatos da norma IEEE 829, realiza-se uma estruturação das atividades de testes com a criação de documentos a partir da fase inicial do desenvolvimento do software, passando pela fase de codificação e execução dos testes de forma controlada, até a fase pós-implantação, onde será consolidado os resultados das atividades realizadas.

Através dos resultados obtidos pela utilização da metodologia CenPRA, conclui-se que a utilização de um método de testes estruturado em documentos que acompanham o ciclo de desenvolvimento do software é viável nas empresas, pois há um aumento na qualidade dos produtos entregues. Esta metodologia demonstra que os clientes, diante da melhoria verificada, estão dispostos a esperar mais pelo produto final e exigindo que os próximos projetos tenham o mesmo comportamento [CSBSAJ,2004].

No que se refere à melhoria da qualidade do software, a metodologia AGEDIS apresenta em comum, com este estudo, conceitos da aplicação de uma metodologia de teste a partir do início do desenvolvimento do software, através de um modelo que utiliza de ferramentas para a automatização dos testes, aumentando a qualidade do produto final.

# 4.ESTUDO DE CASO

## 4.1. Contexto

A empresa estudada é uma grande multinacional que tem como atividade principal a administração de cartões de crédito e prestação de serviços financeiros como empréstimos e vendas de cheques de viagem.

O foco da empresa não é o desenvolvimento de software e sim, a qualidade e diferenciação dos serviços prestados a seus clientes, que de uma forma direta está relacionada com a qualidade de seus softwares para dispor aos seus clientes informações corretas e rápidas.

Não existe na empresa equipe de desenvolvimento para a construção e manutenção das aplicações. O que existe é uma equipe de líderes de projetos composta por profissionais com formação e experiência em tecnologia da informação (TI), cujas responsabilidades principiais são: análise e levantamentos de requisitos, modelagem de dados, definição da arquitetura, elaboração de proposta técnica, suporte ao desenvolvimento, testes, implantação, acompanhamento em produção e gerencia do projeto no cumprimento de cada uma das etapas do desenvolvimento.

Todo e qualquer tipo de codificação é feito por empresas contratadas, devidamente cadastradas e capacitadas para a realização das tarefas, que através da cotação de menor preço são escolhidas para a implementação do projeto.

A arquitetura tecnológica da empresa é composta de um misto de aplicações que vai desde o mainframe até aplicações em Microsoft visual basic 6.0 e ASP. Sendo que, as informações cadastrais de clientes e faturamento estão centralizadas em aplicações mainframe, que também é conhecida como plataforma alta e tem como ponto positivo a capacidade de processamento, a segurança no armazenamento de dados, além do que a gestão de custos e investimentos, por sua natureza ser de forma estruturada, ser mais eficaz e precisa do que as plataformas cliente/servidor [MLQ,1999].

Entretanto, o modo de interação com usuários e a dificuldade na evolução dos sistemas, ora por limitação técnica da plataforma ora por custo elevado no desenvolvimento e manutenções, são fatores determinantes a serem considerados no desenvolvimento de novos produtos nesta plataforma.

Como forma de minimizar estes problemas a empresa adotou uma solução técnica de construir aplicações para processamento em ambiente Windows e interações com o mainframe através de um emulador de terminal, como mostra a figura 6. Estas aplicações são denominadas como *Automators* e *Navigators*.

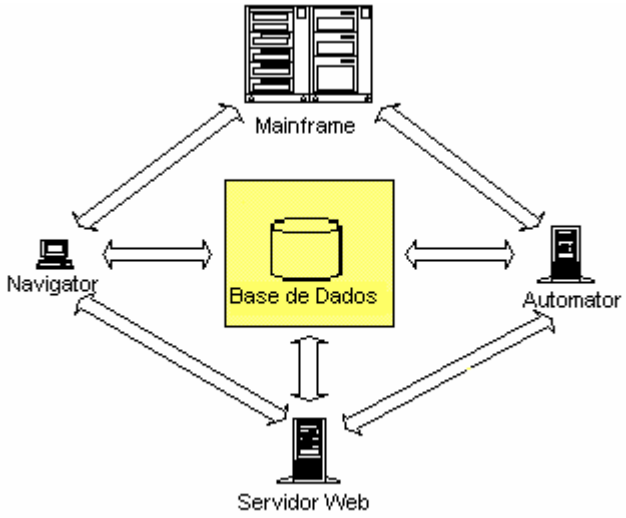


**Figura 6 -** Arquitetura tecnológica dos sistemas da empresa

Os líderes de projeto possuem como responsabilidade apenas o desenvolvimento e manutenção das aplicações em ambiente Windows, ou seja, Automators e Navigators. Qualquer tipo de manutenção necessária no ambiente mainframe é feito por equipes fora do Brasil seguindo metodologias próprias e que não fazem parte do escopo deste estudo.

Os Automators são aplicações desenvolvidas para executar funções repetidas que eram feitas antes por pessoas. O objetivo é automatizar processos repetidos que não exigem qualquer tipo de análise de decisão do usuário e assim, minimizar erros e aumentar o volume de processamento. Possuem pouca, ou quase nenhuma interface gráfica com usuários.

Estas aplicações geralmente são desenvolvidas na linguagem de programação visual basic 6.0 (VB) com banco de dados relacional Sql Server 6.0, podendo ou não ter interação com o mainframe.

Os Navigators são aplicações mais complexas que os Automators, pois, além da automatização de rotinas, possuem regras de negócios implementadas que exigem análise do usuário e acesso simultâneo. Em geral, são aplicações desenvolvidas tanto em VB quanto em ASP com banco de dados SQL Server 6.0 e interagindo com o mainframe.

## 4.2. A Realização dos Testes Antes da Proposta de Melhoria

A idéia de teste de sistemas deve ser estruturada como um projeto que envolve técnicas e planejamento, profissionais de todos os níveis e áreas, arquiteturas de hardware e software que simulem o ambiente de produção. [AMARO,1999]

O maior problema na empresa era não considerar a fase de teste como uma etapa essencial no processo de desenvolvimento do software.

Os testes não possuíam nenhum tipo de preparação, por exemplo: como seriam elaborados casos de testes e como seriam testados, quais as pessoas que deveriam ser envolvidas e quais as datas de início e término. A estratégia de teste usada era algo parecido com a estratégia big-bang, ou seja, após o software codificado testa-se tudo de uma única vez.

O teste era visto apenas como uma forma de validar se o desenvolvedor fez as codificações solicitadas de modo a assinar a carta de aceite para o pagamento da mesma, ou seja um processo *ad-hoc*.

As validações dos sistemas ou testes na maioria das situações, eram feitas apenas pelos líderes de projetos e desenvolvedores. Os usuários da aplicação só tinham conhecimento das alterações após a implantação em produção.

Um ponto muito importante a ser destacado é a entrega dos produtos pelas empresas responsáveis pelo desenvolvimento. Como existiam as datas de entregas dos produtos, e temendo punições quanto ao não cumprimento das mesmas, os desenvolvedores faziam entregas incompletas dos produtos quanto às funcionalidades exigidas na especificação. Isso gerava um desgaste entre o líder de projeto e desenvolvedores, pois os testes deveriam ser adiados e as datas de implantação também.

As justificativas mais comuns para essas entregas era o não entendimento do funcionamento do processo, cronogramas apertados e que não contemplavam testes, falta de uma base de testes, dentre outras.

A falta de padronização na documentação dos testes também era crítica. Cada líder de projeto possuía seu "modo" de construir, documentar, conduzir e executar os testes. As documentações eram feitas de forma desorganizada, incompleta e sem nenhuma preocupação em armazenar os registros de teste em uma estrutura para futuras consultas ou mesmo para os testes de regressão.

## 4.3. Proposta de Melhoria

O processo de teste tem maior probabilidade de ser bem sucedido para organizações que possuam uma equipe de teste bem definida e com um padrão de documentação seguido. [ZALLAR,2001]

Seguindo esta diretriz, o primeiro passo adotado na empresa para a melhoria no processo de desenvolvimento de software foi a criação de uma equipe dedicada exclusivamente ao teste de software.

Os principais objetivos desta equipe é a elaboração do plano de teste, garantir que o plano seja cumprido e os testes executados e documentados. Como premissa, foi adotado que nenhum projeto que envolva alteração ou criação de um sistema poderia ser colocado em produção sem a aprovação da equipe de teste quanto à qualidade do software.

Em conseqüência da equipe de teste ser reduzida em relação ao número de projetos da empresa, algumas exceções eram permitidas. Projetos pequenos que não envolvessem alterações em regras de negócio poderiam ser desenvolvidos sem o acompanhamento da equipe de teste ou a realização formal do processo de teste.

A metodologia de testes adotada está fundamentada nos princípios da metodologia 3P x 3E e nos artefatos sugeridos pela norma IEEE-829, ressaltando que alguns documentos, permitidos tanto pela metodologia quanto pela norma, foram agrupados em um único ou até mesmo retirados de acordo com a necessidade da empresa, conforme demonstrado na figura 7.

### *4.3.1. Documentos Utilizados*

O passo seguinte após a criação da equipe de teste foi a elaboração e definição dos documentos a serem utilizados de acordo com a seqüência de acontecimentos do projeto. Os documentos criados foram:

1. Plano de Teste (PTe)

2. Casos de Teste (CTe)
3. Relatório de Incidente (RIn)
4. Relatório Resumo de Teste (RRT)

Sendo que:

Plano de Teste (PTe)

O PTe é o primeiro documento criado no processo de teste. Sua importância principal é estabelecer um projeto de teste para o produto destacando os principais passos, as datas de início e término, pessoas envolvidas, riscos envolvidos e estabelecer um compromisso formal com todos envolvidos de forma a viabilizar a sua execução. Comparando-se com a metodologia 3P x 3E, eqüivale às etapas de procedimentos inicias, planejamento e preparação. E em relação à norma IEEE 829, eqüivale ao documento Plano de Teste.

As informações que constam no PTe são:

- Nome e número do projeto
- Nome das pessoas que participarão dos testes e suas respectivas responsabilidades
- Cronograma com datas e horários para os testes
- Local dos testes
- Critérios para considerar o teste como finalizado
- Funcionalidades e módulos que serão testados
- Equipamentos e/ou softwares necessários
- Riscos e contingências
- Forma de escalar problemas: através de email direto aos envolvidos de cada área ou aos superiores acima
- Nomes das pessoas que decidirão o término do teste
- Os documentos que serão criados durante os testes e as pessoas responsáveis pela sua elaboração

A elaboração do PTe é de responsabilidade da equipe de teste.

Casos de Teste (CTe)

É o documento contendo os casos de teste que deverão ser executados. Com relação à metodologia corresponde a etapa de Especificação, e em relação à norma, é a união dos

documentos Especificação do Projeto de Teste, Especificação dos Casos de Teste e Especificação dos Procedimentos de Teste.

As informações nestes documentos são:

- Nome e número do projeto
- Número identificador do teste, que deverá ser único
- O módulo ou rotina a ser testado
- Descrição sucinta do teste propriamente dito
- Roteiro de teste, contendo a descrição do teste e passos a serem seguidos
- O resultado esperado
- O resultado final do teste obtido pelo desenvolvedor constando data e nome de quem testou
- O resultado final do teste obtido pelo usuário, constando data e nome de quem testou

A responsabilidade pela documentação destas atividades era da equipe de teste, mas a elaboração dos casos de testes contava com a participação tanto da equipe de teste quanto dos líderes de projetos e usuários.

Relatório de Incidente (RIn)

Durante a execução dos testes é comum o surgimento de erros que podem ser causados tanto pela falha na construção do testes quanto por problemas de programação. Estes erros devem ser registrados no RIn para as devidas análises da equipe de teste e encaminhamento aos responsáveis pela correção. Comparando-se com a norma 3P x 3E eqüivale a etapa de Execução, e em relação à norma IEEE 829 é a união dos documentos Diário de Teste e Relatório de Incidente.

As informações contidas neste documento são:

- Número do caso de teste
- Status
- Nome da pessoa responsável pela correção
- Prioridade para correção do erro pelo desenvolvedor
- Descrição detalhada do erro, podendo conter o *print* da tela
- Nome e data de quem fez a correção

O RIn é o documento de interface entre a equipe de teste e a equipe de desenvolvimento, mas a responsabilidade de atualização é da equipe de teste.

Relatório Resumo de Teste (RRT)

O RRT é o documento final a ser preenchido pela equipe de teste. Além do nome e número do projeto, esse documento possui informações de início e fim, status final do teste, descrição detalhada de como foi o teste, pessoas envolvidas e os principais números dos testes, como por exemplo, número de casos de teste criados antes e durante a execução dos testes, e o número de execuções com sucesso.

Este documento corresponde na metodologia 3P x 3E a etapa de Entrega, e em relação à norma IEEE 829, corresponde ao documento Relatório Resumo de Teste.

Os principais tipos de status de um projeto são: completado com sucesso, completado com restrições, e não completado. Entretanto, outros tipos de status podem surgir de acordo com os problemas encontrados.

No RRT constam também informações estatísticas de como foi o teste, tais como, percentual de casos de teste executados, percentual de casos com sucesso e erros e o percentual de casos de testes corrigidos pelo desenvolvedor. Estas informações poderão ser usadas para uma apresentação final do projeto a todos envolvidos.

Na figura 7 tem-se esquema de como os documentos utilizados, a metodologia 3P x 3E e a norma IEEE 829 estão relacionados.

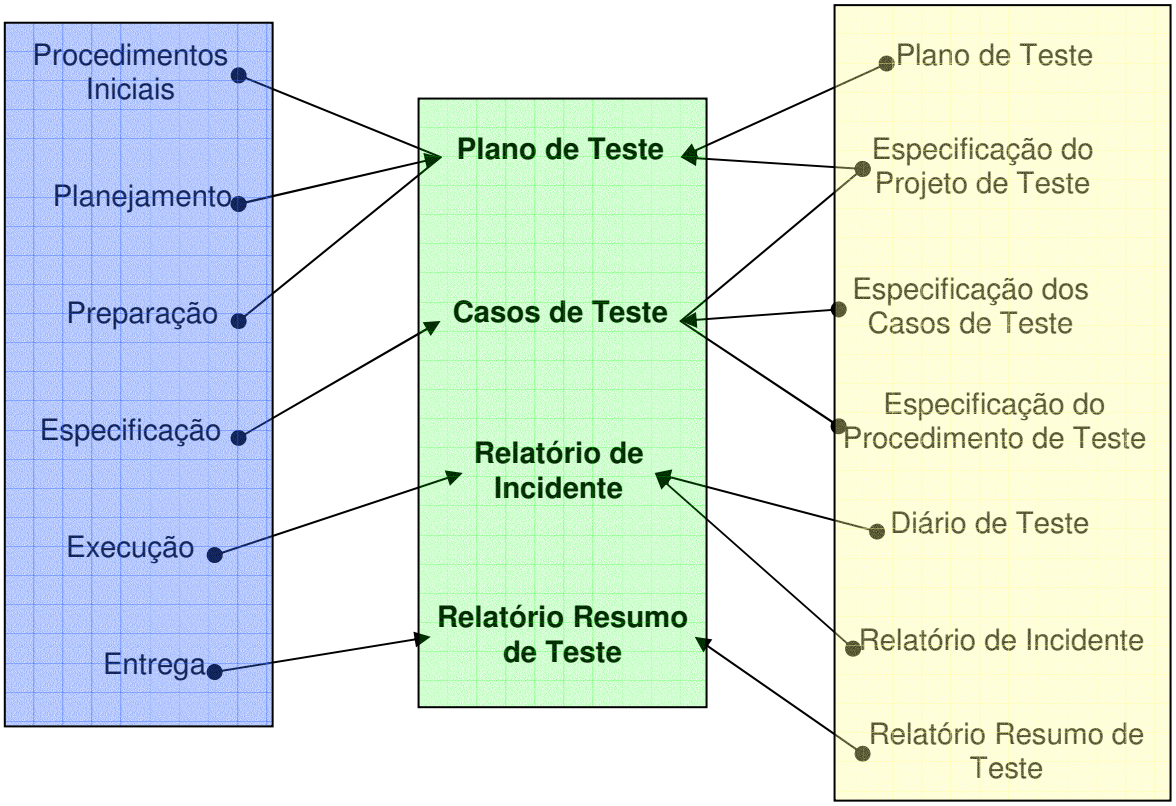


**Figura 7** - Metodologia 3P x 3E - Documentos utilizados - Norma IEEE 829

No apêndice A é apresentado um modelo de cada um dos documentos utilizados.

### *4.3.2. Comunicação*

Com a equipe de teste definida e os documentos criados, o passo seguinte foi a divulgação da metodologia para todos os líderes de projetos. A forma utilizada para a divulgação foi através de reuniões com a participação de todos os líderes de projetos, onde foi discutido o propósito de cada documento, sua importância, a forma e quando preenchê-los.

Os líderes de projetos foram orientados que o envolvimento da equipe de teste já a partir da fase de análise das especificações seria obrigatório. Entretanto, como a equipe de teste inicialmente era reduzida e o número de projetos era grande, alguns projetos poderiam ser analisados quanto à criticidade e risco para o negócio da empresa. Não havendo recursos humanos disponíveis por parte da equipe de teste, estes projetos poderiam ser desenvolvidos e implantados sem um plano formal de teste.

Como as empresas desenvolvedoras seriam afetadas diretamente com a implantação da metodologia, também foi feita uma apresentação dos documentos a serem exigidos e a forma de preenchê-los a partir daquele momento.

### *4.3.3. Planejando os Testes*

Assim que o líder de projetos recebe a solicitação do usuário com os requisitos é feito um estudo dos requisitos com análise das implementações necessárias e um plano do projeto é construído. A partir do plano do projeto, a equipe de teste é envolvida no processo de desenvolvimento do software. Em um projeto sempre existia a participação de no mínimo um líder de projeto de TI, um líder da área de solicitante e um representante da equipe de teste.

A partir das reuniões que se sucedem para análise dos requisitos e construção do modelo conceitual do projeto, o LP e equipe de teste fazem à identificação das principais características e funcionalidades do sistema, os recursos de hardware e softwares necessários, a complexidade do sistema, dentre outras. Essas informações serão importantes para uma idéia geral do projeto e assim, ter uma idéia da dimensão da atividade de teste e também identificar as pessoas e as responsabilidades de cada um no projeto.

No planejamento, são tomadas decisões do tipo [JOHN,2001]: quem executará os testes? Que partes serão testadas? Quando os testes serão executados? Que tipos de teste serão executados? O quanto cada parte será testada?

Com as informações reunidas, a equipe de teste elabora o PTe e encaminha para a assinatura dos responsáveis e obter a formalização do processo de teste.

### *4.3.4. Especificando os Testes*

Dado o aceite pelos responsáveis no documento PTe, tem-se o início da construção dos casos de teste determinando os valores das entradas e saídas esperadas, e a ordem de execução, figura 8.

A criação e documentação dos casos de testes no CTe envolvem a participação da equipe de testes, o líder do projeto e os usuários, onde todos poderão sugerir situações de testes e formas de executá-los. Neste momento a equipe de teste está atenta quanto aos recursos necessários para a realização do teste de forma a viabilizar sua execução dos mesmos, tais como uma base de dados com características particulares, um novo software, uma interface, etc.

A técnica de seleção dos casos de teste utilizada é a caixa-preta ou técnica funcional. A justificativa para a utilização desta técnica está relacionada sob o ponto de vista da validação e verificação do sistema quanto aos seus requisitos.

Durante a construção dos casos de teste, tanto a equipe de teste quanto os usuários e líderes de projetos contribuem com informações e situações pertinentes ao processo e que possibilitasse um teste mais efetivo. É incentivada inclusive, a experiência em casos de testes em versões anteriores para os possíveis testes de regressão.

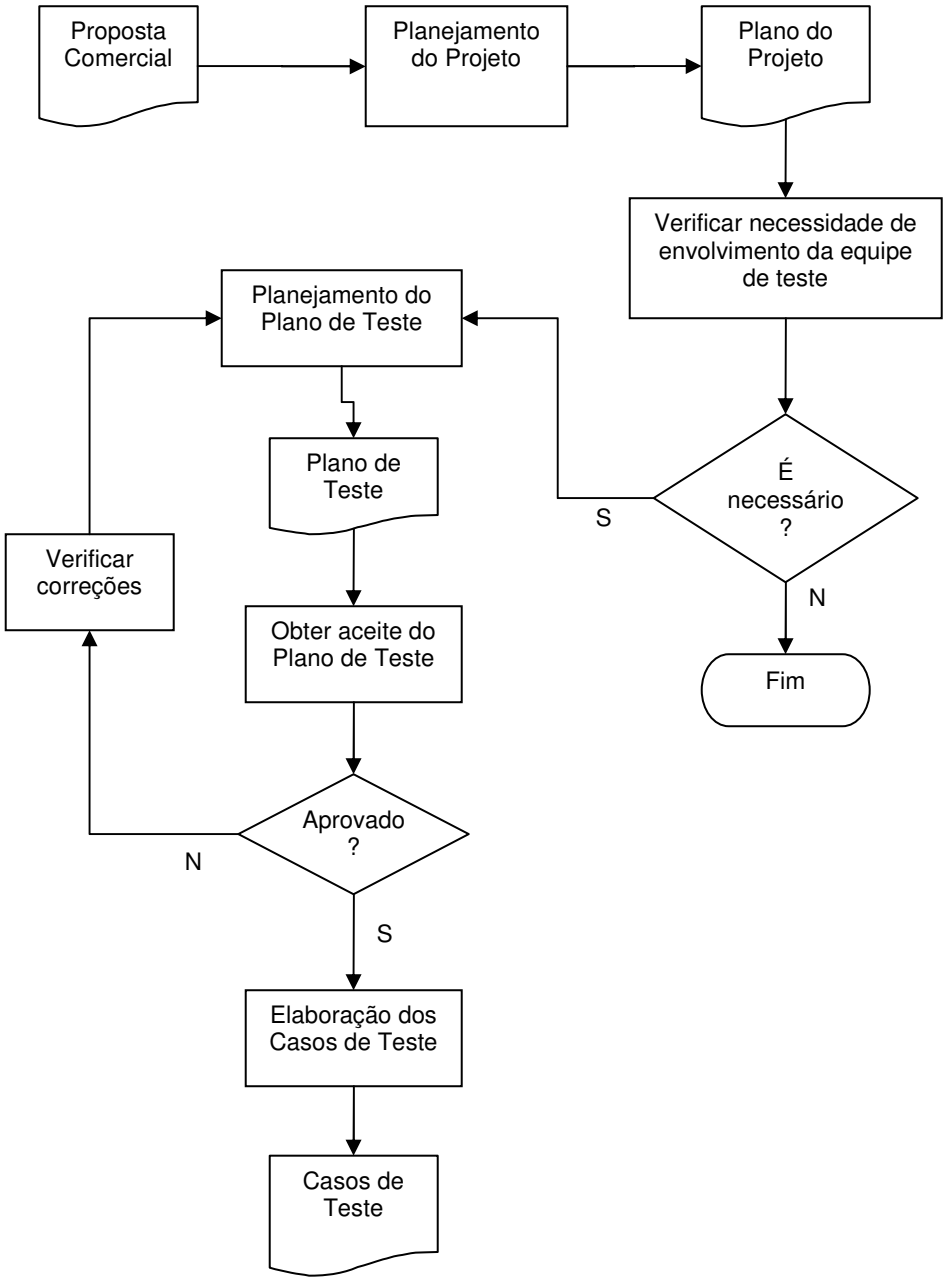

**Figura 8** - Fluxo do planejamento e especificação dos testes

O processo para criação dos casos de testes inicia-se pela identificação dos módulos do sistema a serem testados, de acordo com os requisitos. E dentro de cada módulo há a elaboração dos casos de testes. A definição dos módulos é feita de acordo com as características funcionais do sistema. Por exemplo, em uma aplicação pode existir o módulo de acesso (*login*), módulo de relatórios, módulo de cálculo de despesas, módulo de lançamento de despesas, módulo de consulta de clientes, etc.

Através dos critérios particionamento de equivalência e análise de valor limite, os testes são elaborados considerando situações de sucesso utilizando valores de entrada válidos dentre as especificações, e também para situações de erros, ou seja, valores fora da utilização correta do sistema, como entradas inválidas. Desta forma, possibilitaria verificar quanto ao comportamento do sistema e se as mensagens possuíam informações suficientes para entendimento do problema ocorrido e qual ação a ser tomada.

No sentido de melhorar a qualidade dos produtos entregues pelos desenvolvedores, a empresa adotou a estratégia de enviar juntamente com as especificações do projeto o documento CTe contendo os casos de testes funcionais elaborados. Na entrega do software pelo desenvolvedor, todos os casos de testes deveriam obrigatoriamente estar devidamente preenchidos com status de cada item testado.

Entretanto, durante a fase de codificação do software pelo desenvolvedor, novos casos de testes poderão ser identificados e registrados no CTe. Apenas ressaltando que, estes novos itens não poderão ser exigidos dos desenvolvedores.

Como aspecto negativo nesta estratégia tem-se o aumento de horas apontado pelos desenvolvedores no desenvolvimento do projeto, pois passaram a executar e documentar os testes agora obrigatórios.

#### *4.3.5. Execução e Entrega*

A execução dos testes seguia o plano estabelecido no PTe com datas, horários e local conforme descritos. O local de realização dos testes era de uso exclusivo, ficando inclusive, separada do local de ambiente normal de trabalho. Tudo isto, para garantir uma maior produtividade sem interrupções desnecessárias.

A execução dos testes contava com a participação de usuário, líder de projetos, equipe de teste e desenvolvedor. A participação da equipe de teste limitava-se a preparação dos testes como reserva de salas, preparação dos equipamentos e softwares, preparação do banco de dados com as informações e permissões de acesso necessárias, acompanhamento e a documentação dos testes. A execução dos testes propriamente dita, é de responsabilidade dos usuários ou dos líderes do projeto.

À medida que os testes são trabalhados, informações da execução são registradas no CTe. Nas situações de sucesso da execução de cada caso de teste era registrada quem o fez e uma evidência do sucesso como, por exemplo, o *print* da tela com a mensagem de sucesso. Para as situações de erros na execução de algum caso de teste é sinalizado no CTe como não ok, e uma ocorrência é registrada no RIn com o número do caso de teste, o status, a descrição detalhada do erro e um *print* de tela com informações importante, quando possível.

Ao final do dia, o RIn é repassado ao desenvolvedor para análise dos erros com uma previsão de data para correção. Com o erro corrigido, o desenvolvedor devolve o documento RIn informando que o caso de teste poderá ser refeito. Refeito o teste com sucesso, tanto o CTe e RIn são atualizados com status de sucesso.

Os status de um RI são:

- Em aberto - aguardando correção do desenvolvedor
- Pronto para re-teste – já corrigido pelo desenvolvedor
- Fechado – teste executado com sucesso
- Re-teste falho – teste executado com erro e aguardando correção
- Fechado com restrições – quando não é possível refazer o teste, fecha-se a ocorrência

Ao final da execução de todos os casos de testes ou ao término do tempo definido no PTe, o passo seguinte é a consolidação das informações dos documentos CTe, RIn e PTe no documento RRT. A partir do RRT é feita uma apresentação a todos envolvidos com os resultados obtidos, conforme figura 9.

A entrega do documento RRT tem como objetivo colocar um marco final no projeto, que a partir daquele momento resta apenas fazer a implantação em produção, tarefa do líder de projeto. A equipe de teste a partir deste momento está disponível para outras atividades.

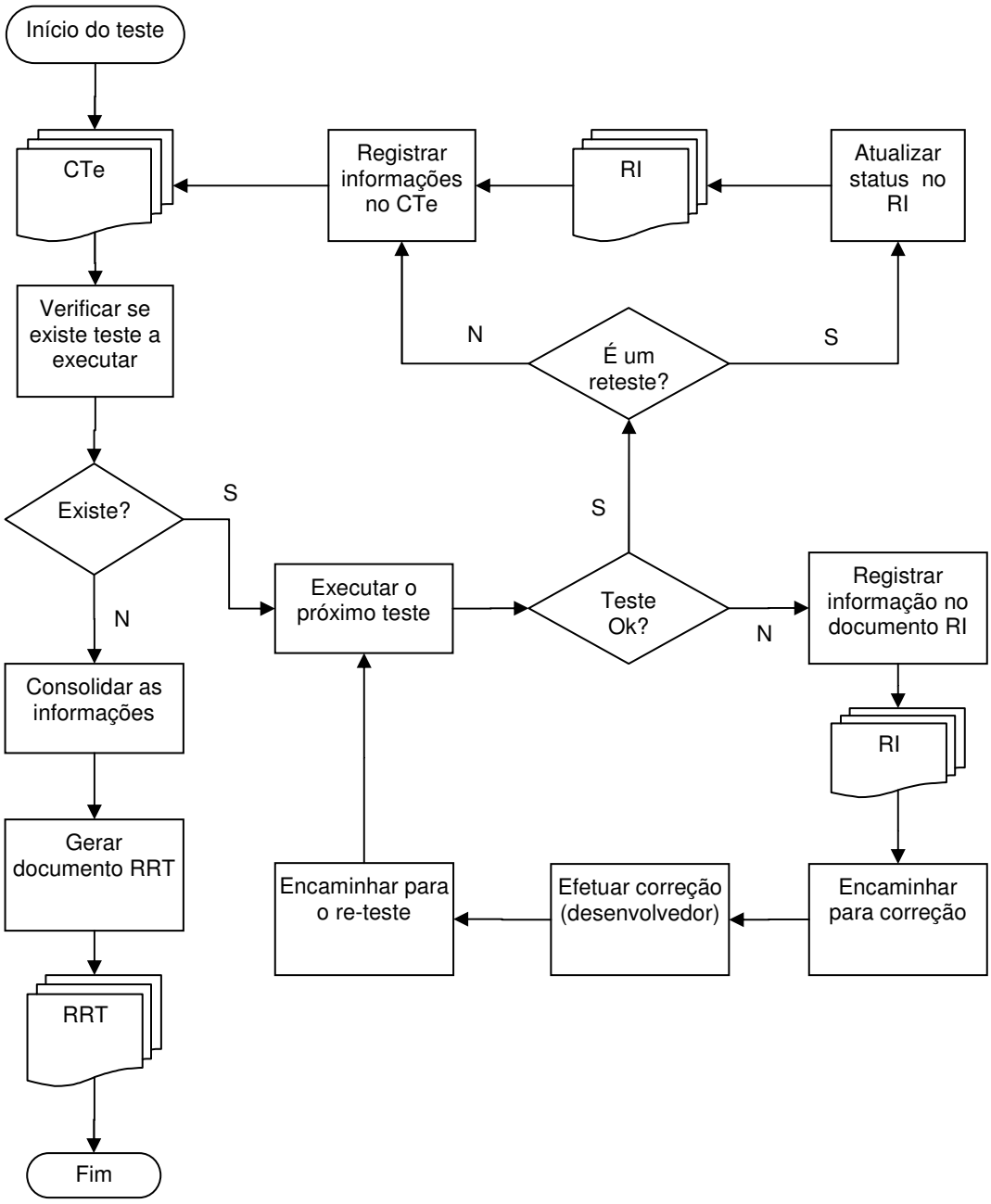


**Figura 9** - Fluxo da execução dos testes

## 4.4. Resultados

Os resultados a serem apresentados dizem respeito a projetos implantados seguindo a metodologia apresentada anteriormente com acompanhamento da equipe de teste reportando e documentado os eventos ocorridos durante os testes. Os dados de projetos implantados sem a utilização da metodologia de testes, também serão apresentados.

Os projetos, conforme listados na Tabela 3, contaram com a participação da equipe de teste a partir da fase de levantamento de dados e análise de requisitos. A elaboração do plano e dos casos de testes foram feitos juntamente com os líderes de projetos à medida que o projeto fosse avançando.

Os dados pesquisados foram divididos em três fases: levantamento, fase de teste e produção. A fase de *levantamento* correspondente ao período da análise de requisitos, levantamento de dados, elaboração da proposta técnica e desenvolvimento.

A *fase de teste* corresponde ao período da execução dos testes propriamente dito, ou seja, logo após a codificação e entrega do produto pelo desenvolvedor.

As informações pertencentes à fase de *produção* foram identificadas após a implantação do projeto em produção, ou seja, os erros nesta fase causaram algum impacto negativo na empresa como, por exemplo, uma parada de sistema ou disponibilização incorreta de informações.

Tabela 3. Projetos realizados com a utilização da metodologia

| | Levantamento | | | | Teste | | | Produção | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Nome do Projeto** | **Requisitos (#)** | **Levantamento (H)** | **Elaboração do teste (H)** | **Casos de testes criados (#)** | **Execução dos Testes (H)** | **Erros encontrados (#)** | **Erros de requisitos (#)** | **Erros encontrados (#)** | **Horas gastas corrigindo erros** | **Erros de requisitos (#)** |
| **Prj 1** | 21 | 170 | 48 | 85 | 110 | 33 | 2 | 3 | 8 | 1 |
| **Prj 2** | 4 | 14 | 16 | 20 | 22 | 8 | 0 | 0 | 0 | 0 |
| **Prj 3** | 6 | 30 | 22 | 31 | 40 | 13 | 0 | 1 | 3 | 0 |

Os projetos listados na Tabela 4 referem-se a projetos desenvolvidos antes da implantação da metodologia de testes e que tinham um número elevado de erros em produção.

Todas as fases do projeto como *levantamento*, *teste* e *produção* eram de responsabilidade do líder de projeto. Sendo que a preparação dos testes e a criação de casos de testes não faziam parte do escopo do projeto.

A fase de teste representa os valores de testes feitos entre o líder de projeto e o desenvolvedor. O objetivo aqui era apenas verificar se o desenvolvedor implementou as funcionalidades requeridas conforme a proposta técnica. E no caso afirmativo, o líder de projeto assinava a carta de aceite de modo a desenvolvedor receber pelo serviço.

Como conseqüência pela falta de preparação, a realização dos testes junto ao desenvolvedor decorria de maneira desordenada com pouca documentação do trabalho e sem uma eficiente base de dados para teste.

Tabela 4. Projetos realizados sem a utilização da metodologia

| | Levantamento | | | | Teste | | | Produção | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Requisitos # | Horas de Levantamento | Horas na elaboração do teste | Número de casos de testes criados | Horas na execução dos Testes | Erros encontrados | Erros de requisitos | Erros encontrados | Horas gastas corrigindo erros | Erros de requisitos |
| **Prj 1** | 15 | 135 | 0 | 0 | 40 | 16 | 2 | 7 | 20 | 3 |
| **Prj 2** | 8 | 105 | 5 | 4 | 24 | 19 | 0 | 6 | 24 | 2 |

Erros identificados e encaminhados para correção do desenvolvedor podiam ser “esquecidos”, consequentemente não corrigidos.

Validações mais detalhadas das principais funcionalidades do software corriqueiramente não eram realizadas e sendo somente identificadas, pelos usuários, no ambiente de produção.

Através das tabelas apresentadas, pode-se afirmar que houve um aumento no número de casos de testes criados durante a fase de levantamento. Assim como um maior número de horas dedicadas a execução dos testes. Em conseqüência, houve a redução do número de erros de requisitos encontrados após a implantação em produção.

# 5.CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresenta a utilização de uma metodologia de teste aplicada desde o início de desenvolvimento de software no propósito da melhoria da qualidade. Através dessa metodologia foi possível destacar o aumento na qualidade dos objetos produzidos, menor número de erros implantados em produção, identificação de erros na definição de requisitos, logo na fase inicial do projeto que refletiram na satisfação e confiança dos clientes internos.

Quanto ao aspecto dos desenvolvedores pode-se ressaltar o aumento da qualidade dos produtos entregues com um número bem menor de erros e, um maior comprometimento em garantir o tempo de entrega.

Quanto ao nível de dificuldade na implantação da metodologia, pode-se dizer que, apesar das resistências iniciais em quebrar paradigmas e romper com zonas de conforto de cada um, o método possibilitou notar melhorias significativas no processo de desenvolvimento, de tal forma que todos compraram a idéia e apoiaram a utilização obrigatória da metodologia para os próximos projetos.

A utilização de uma metodologia de teste possibilitou dar um importante passo para a melhoria do software, a propósito, foi o primeiro passo. Contudo, a elaboração de casos de testes mostrou-se um processo frágil, pois dependia da experiência, conhecimento sistêmico e até mesmo da imaginação dos envolvidos para a elaboração de testes eficazes.

Como trabalhos futuros, sugere-se o uso de outras metodologias de teste que se baseiem no uso de ferramentas para criação e execução dos casos de testes, como a AGEDIS, pois podem melhorar ainda mais a qualidade do software. A utilização de técnicas caixa-branca para módulos críticos e que exigem a execução de um maior número de caminhos independentes pode ser uma boa alternativa, assim como a utilização de teste de regressão após as correções em aplicações ou após a manutenção das mesmas.

# 6.REFERÊNCIAS

[AD,1997] Apfelbaum, L.; Doyle, J. **Model-based testing,** Proceedings of the Software Quality Week Conference, maio 1997.

[AGEDIS] AGEDIS Consortium, Automated Generation and Execution of Test Suites for DIstributed Component-based Software, http://www.agedis.de

[AMARO,1999] Amaro, A. **Encarando o Teste de Sistemas Como um Projeto**, Developer´s Magazine, p. 36-37, Ano IV, número 37, julho 1999.

[BEIZER,1990] Beizer, B., **Software Testing Techniques**, 2nd ed., Van Nostrand-Reinhold, 1990.

[BEIZER,1995] Beizer, B., **Black-Box Testing**, Wiley, 1995.

[BLACK,2005] Black, R. **The Cost of Software Quality**. Disponível em: <http://www.stickyminds.com>, consultado em 20/08/2005

[BULLOCK,2005] Bullock, J. **Calculating the Value of Testing**. Disponível em: <http://www.stickyminds.com>, consultado em 29/11/2005.

[CenPRA,2001a] **Centro de Pesquisas Renato Archer – CenPRA**, Divisão de Melhoria de Processos de Software - DMPS, RT – Guia para Elaboração de Documentos de Teste de Software, Relatório Técnico, 2001.

[CenPRA,2001b] **Centro de Pesquisas Renato Archer – CenPRA**, Divisão de Melhoria de Processos de Software - DMPS, RT – Processos para Elaboração de Documentos de Teste de Software, Relatório Técnico, 2001.

[CSBSAJ,2004] Crespo, A. N.; Silva, O. J.; Borges, C. A.; Salviano, C. F., Argolo, M. T, Jino, M.. **Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo**. III Simpósio Brasileiro de Qualidade de Software (SBQS), Brasília, 2004.

[FR,2002] Filho, T. R. M.; Rios, E. **Projeto e Engenharia de Software: Teste de Software**, 1ª ed. Rio de Janeiro: Alta Books, 2002, cap. 2-3.

[IEEE,1990] The Institute of Electrical and Electronics Engineers. **Standard 610 (1990)**, reprinted in IEEE Standards Collection: Software Engineering 1994 Edition.

[IEEE,1998] The Institute of Electrical and Electronics Engineers. **IEEE Std 829: Standard for Software Test Documentation. New York**: IEEE Computer Society, September, 1998.

[JO,1995] Jin, Z.; Offutt, J. **Integrating Testing with the Software Development Process**. Technical Report ISSE-TR-95-112, George Mason university, Agosto 1995.

[JOHN,2001] John, D. McGregor and David A. Sykes. **A Pratical Guide to Testing Object-Oriented Software**. Object Technology Series. Addison-Wesley, 2001.

[JONES,1981] Jones, T. C., **Programming Productivity: Issues for the 80s,** IEEE Computer Society Press, 1981**.**

[KANNER,2003] Kanner, C. **What Is a Good Test Case?** Software Testing Analysis & Review Conference (STAR) East*, Orlando, FL, May 12-16, 2003*.

[McCABE,1976] McCabe, T., "**A Software Complexity Measure,"** IEEE Trans. Software Engineering, vol. SE-2, December 1976, pp. 308-320.

[MLQ,1999] Martins, G.; Luz C.; Queiroz L. C. **Mainframe versus Cliente/Servidor: a Eterna Polêmica**, Developer´s Magazine, p. 36-37, Ano III, número 35, julho 1999.

[MYERS,1979] Myers, G. **The Art of Software Testing**. New York: Wiley, 1979.

[OLAN,2003] Olan, M. Unit Testing: Test Early, Test Often, In Journal of Computing Sciences in Colleges, page 319. The Consortium for Computing in Small Colleges, 2003.

[PRESSMAN,2002] Pressman, R. S. **Engenharia de Software**. 5ª ed. Rio de Janeiro: McGraw-Hill, 2002, cap. 17-18.

[SOMMERVILLE,1999] Sommerville, I. **Software Engineering**. 5ª ed. Addison-Wesley, 1999.

[ZALLAR,2001] Zallar, K. **Are You Ready for Test Automation Game?**. STQE – Software Testing and Quality Engineering Magazine, vol.3, Issue 6, Nov/Dec 2001.

# APÊNDICE A. MODELOS

Este apêndice apresenta exemplo dos documentos Plano de Teste, Casos de Teste, Relatório de Incidente e Relatório Resumo de Teste. Alguns campos foram preenchidos com informações hipotéticas apenas para ajudar na compreensão.

## A.1 Plano de Teste

| **Plano de Teste** | |
|---|---|
| ***Projeto:*** | |
| **Escopo** | |
| | |
| | |
| | |
| **Funcionalidades ou Módulos** | |
| | |
| | |
| | |
| **Riscos e Contingências** | |
| | |
| | |
| | |
| | |
| **Equipamentos / Softwares** | |
| | |
| | |
| | |
| **Pessoas Envolvidas e Responsabilidade** | |
| | |
| | |
| | |
| **Cronograma** | |
| **Data de Início e Fim do Projeto:**<br>**Data de Início e Fim do Teste:**<br>**Local dos Testes:** | |
| **Observações** | |
| *Documentos que serão criados;* | |

## A.2 Casos de Teste

| Casos de Teste | | | | | | |
|---|---|---|---|---|---|---|
| ***Projeto:*** | | | | | | |
| | | | | | | |
| **# CT** | **Módulo** | **Descrição do Teste** | **Roteiro de Teste** | **Resultado Esperado** | **Resultado Obtido Desenvolvedor** | **Resultado Obtido no Teste** |
| *1* | *Relatório* | *Gerar informações dos clientes com saldo negativo* | *1)Escolher relatório de clientes com saldo negativo;<br>2)Informar o período 10/11 à 15/11.<br>3) Escolher a opção gerar relatório.* | *Gerar relatório com as informações do nome do relatório, data de processamento, período informado, nome dos clientes e o valor devedor.* | | |
| *2* | *Relatório* | *Gerar informações do clientes cancelados* | *1) Escolher a opção de relatório de clientes cancelados;<br>2) Informar o período de cancelamento;<br>3) Escolher a opção gerar relatório.* | *Gerar relatório com as informações completas do cabeçalho nome dos clientes e datas de cancelamento.* | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

## A.3 Relatório de Incidente

| Relatório de Incidentes | | | | | |
|---|---|---|---|---|---|
| **Nome do Projeto:** | | | | | |
| | | | | | |
| **Status** | **# CT** | **Prioridade** | **Responsável pela correção** | **Descrição do Erro** | **Data e Nome de quem Corrigiu** |
| *Pronto para re-teste* | *1* | *Alta* | *Beltrano* | *O relatório não imprime os clientes com saldo negativo.* | *11/11/05 - Beltrano Correções efetuadas.* |
| *Em Aberto* | *2* | *Baixa* | *Fulano* | *Cabeçalho não contém nome do relatório.* | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## A.4 Relatório Resumo de Teste

| **Resumo do Teste** | |
|---|---|
| ***Nome do Projeto*** | |
| | |
| **Início dos Testes:** | **Fim dos Testes:** |
| | |
| **Descrição detalhada do teste** | |
| *Descrever todas as informações pertinentes ao teste, assim como, o status final do teste.* | |
| **Números do Teste** | |
| TC´s criados antes do início dos testes | 100 |
| TC´s criados durante os testes | 6 |
| TC´s executados | 106 |
| TC´s com sucesso | 95 |
| TC´s com erros | 11 |
| TC´s enviados para correção | 100 |
| **Percentuais** | |
| TC executados: | 100,00% |
| TC executados com Sucesso | 89,62% |
| TC executados com incidência de Erro | 10,38% |
| TC corrigidos pelo desenvolvedor | 94,34% |