FATEC

Faculdade de Tecnologia

Reginaldo José de Souza

RA: 012230-0

Qualidade e Teste de Software

FATEC – AM
Americana – SP
1º Semestre de 2004

Reginaldo José de Souza

RA: 012230-0

Qualidade e Teste de Software

Trabalho de graduação apresentado à Faculdade de Tecnologia de Americana, como parte dos requisitos para obtenção do título de Tecnólogo em Processamento de dados

Orientador: Antonio Alfredo Lacerda

FATEC – AM
Americana – SP
1º Semestre de 2004

Dedico este trabalho com muito carinho a minha família: meu pai José, minha mãe Geni, minhas irmãs Regiane, Talita e Thaís e a todos os meus amigos.

Vocês sempre serão muito importante na minha vida.

Agradecimentos

Agradeço prir eiramente a Deus pela força dada nos momentos de dificuldade na elaboração do trabalho, a todos meus amigos, em especial ao meu amigo Ed ıardo, que de uma forma ou de outra ajudaram-me dando força, incentiv ando para que o trabalho fosse concluído, agradeço especialmente ao professor Antonio A. Lacerda por orientar-me, esclarecendo minhas dúvidas nos momentos de dificuldade e a professora Doralice por compor a banca examinadora.

"Se queremos progredir, não devemos repetir a história, mas fazer uma história nova"

(Gandhi)

Sumário

Lista de Figuras viii

Lista de tabelas ix

Resumo x

Abstract xi

Introdução 12

1. Qualidade de software 13
1.1 O que é qualidade de software ? 13
1.2 Atributos de qualidade do software 14
1.3 Atributos de qualidade de produtos de software 16
2. Teste de software 19
2.1 Plano de teste 20
2.1.1 Principais itens do plano de teste 20
2.2 Casos de teste 21
2.3 Tipos de teste 24
2.3.1 Teste de unidade 24
2.3.2 Teste de integração 25
2.3.3 Teste de validação 25
2.3.4 Teste de sistema 26
2.3.4.1 Teste de recuperação 26
2.3.4.2 Teste de segurança 27
2.3.4.3 Teste de estresse 27
2.3.4.4 Teste de desempenho 27
2.3.5 Teste de Caixa-Preta (Black-Box Test) 28
2.3.5.1 Técnica de valor limite 29
2.3.5.2 Classes de equivalência 29
2.3.5.3 Tabelas de decisão 30
2.3.5.4 Grafos de causa e efeito 31
2.3.6 Teste de Caixa-Branca (White-Box Test) 32
2.3.6.1 Teste de caminho básico 32
2.3.6.2 Teste de estrutura de controle 33
2.3.7 Principais tipos de testes 33
3. Testes de software orientados a objetos 38
3.1 Tipos de teste 38
3.1.1 Teste de estado 39
3.1.2 Teste Incremental 39
3.1.3 Teste de Integração 39
3.1.4 Teste de caso de uso 42
4. Estudo de caso 44
4.1. Plano de teste 44
4.2. Casos de teste 47
4.3. Estratégias de teste 48
4.3.1. Cadastro de Clientes 48

4.3.2. Cadastro de fitas 51
4.3.3 Cadastro de locações 52
4.4 Problemas encontrados no software 54
4.5. Registro dos erros 58
Conclusão 59
Referências Bibliográficas 60
Sites utilizados na pesquisa 61

Lista de Figuras

Fig.1: Conj. de caract. a serem verif. num software–Norma ISO/IEC 9126/NBR 13596..... 17

Fig.2: Atividades de teste..... 19

Fig.3: Visão Macro do teste caixa-preta..... 28

Fig.4: Particionamento de equivalência 30

Fig.5: Estratégia Bottow-up..... 40

Fig.6: Estratégia Top-Down..... 41

Fig.7: Casos de teste – Cadastro de clientes..... 47

Fig.8: Cadastro de clientes..... 48

Fig.9: Cadastro de fitas..... 51

Fig.10: Cadastro de locações..... 52

Fig.11: Erro no cadastro de clientes..... 54

Fig.12: Trecho de código do cadastro de clientes..... 54

Fig.13: Cadastro de fitas incompleto..... 55

Fig.14: Trecho de código do cadastro de fitas..... 55

Fig.15: Locação de filmes.....56

Fig.16: Trecho do código de locação de filmes..... 56

Lista de Tabelas

Tabela 1: Exemplo de caso de teste 22

Tabela 2: Tabela de decisão com entradas limitadas.....31

Tabela 3: Principais tipos de teste.....37

Tabela 4: Plano básico de testes.....46

Tabela 5: Estratégias de teste – cadastro de clientes.....50

Tabela 6: Estratégias de teste – cadastro de fitas.....51

Tabela 7: Estratégias de teste – cadastro de locações.....53

Resumo

Souza, Reginaldo José (2004) - Qualidade e teste de software – Americana-SP
Dissertação (graduação) – FATEC Americana

O objetivo deste trabalho é abordar sobre uma área da engenharia de software que é responsável pela qualidade final do software: *testes de software*. Os testes de software não irão acabar com os bugs dos sistemas, mas com certeza, se bem feitos reduzirão e muito os problemas existentes.

Primeiramente foi feito um estudo sobre qualidade de software, abordando seus principais conceitos. Depois deste conteúdo inicial, foi descrito sobre teste de software, na qual fui elucidando sobre os passos iniciais da atividade de teste de software até os tipos e estratégias mais utilizadas no momento. Na etapa seguinte descrevi sobre os tipos e estratégias de testes voltadas para a tecnologia orientada a objetos, visto que é a mais utilizada atualmente no mundo do desenvolvimento de sistemas. Posteriormente, foi elaborado um estudo de caso em que foi aplicado os principais conceitos descritos no desenvolvimento do trabalho. Concluindo o trabalho, uma opinião pessoal de quem está trabalhando nesta área sobre a importância deste trabalho tão importante e que prima tanto pela qualidade do sistema fornecido ao cliente.

Abstract

Souza, Reginaldo José (2004) – Quality and Software Test – Americana-SP

Dissertation (graduation) – FATEC Americana

The objective of this project is to explain about an area of software engineering that is responsible for the final software quality: *software tests*. The software tests are not out there to completely bring system bugs to an end; notwithstanding , when well performed these tests will greatly reduce existents problems.

First, a study was conducted concerning quality of software covering its main concepts. After this initial stage, software testing is explained and covered from its initial steps up until the types strategies most commonly used at the present time. In the following stage, I describe the tests' type and strategies used towards objective-oriented technologies, since the latter is the mostly used nowadays in the system development field. In addition, a study case was elaborated in which the main concepts described in the project's development were applied. Concluding the work, one finds a personal opinion from a professional in the field, who cares greatly on the system delivered to the client, regarding the importance of such essential task.

Introdução

O mercado do software está atualmente cada vez mais competitivo. Pensando nisso, as empresas desenvolvedoras de sistemas estão cada vez mais procurando investir na qualidade do seu produto. Consumidores mais exigentes, um mundo cada vez mais 'recheado' de informações, tudo isso faz com que se invista cada vez mais na qualidade do software produzido.

Antigamente, o tempo dedicado aos testes de sistema era o mais escasso no processo de desenvolvimento, mas isso está mudando em virtude dessa demanda maior por um produto melhor.

Como tudo na engenharia de software existe métodos e ferramentas, nesta fase também não é diferente. No decorrer no trabalho será exposto as principais técnicas e métodos usados atualmente e que com certeza ajudam e muito na qualidade final do sistema.

Teste de software não limita-se somente ao que está escrito neste trabalho. Além destas técnicas, existem também outros segmentos para se testar. Sendo assim nunca é demais continuar pesquisando sobre o assunto, pois sempre haverá novos métodos para aprendermos.

1. Qualidade de software

Atualmente todos exigem qualidade nos produtos que adquirem. No mercado do software também não é diferente. As empresas estão cada vez mais exigente no quesito *perfeição* por parte dos sistemas que utilizam. Podemos dizer que a qualidade de um software é um diferencial de competitividade entre um produto e outro disponível no mercado.

> Segundo Deming [1] *"o homem sempre ensinou sobre qualidade no Japão – todos querem qualidade, porém é certo que cada ser humano tem uma definição diferente de qualidade".*
> Deming [1] ainda afirmou que *"A qualidade somente pode ser definida em termos de quem avalia".*

Com base nas citações acima, podemos concluir que o conceito de qualidade torna-se um tanto relativo, pois apesar de todos exigirem qualidade, um produto pode ser de qualidade para uma pessoa e no entanto para outra não. Tudo depende se os objetivos foram alcançados, se houve retorno do investimento , se o prazo de entrega foi cumprido, entre outros fatores. No mundo do software esse conceito um tanto *relativo* também é aplicado, pois desde que as especificações [2] foram satisfeitas, pode-se dizer que a empresa tem um sistema de qualidade.

1.1 O que é qualidade de software ?

Antes de entrar propriamente no conceito de qualidade de software, darei uma explicação sobre o que são requerimentos.

Requerimentos são condições ou capacidades que um sistema deve ter, podendo derivar das necessidades do usuário, do que está estabelecido em contrato, padrões, especificações, necessidade de funcionamento do sistema ou até mesmo de documentos informais.

No mundo computacional, qualidade de software é:

[1] William Edwards Deming, Professor / consultor de renome internacional na área de qualidade

[2] Especificação aqui pode ser entendida como o *fim ou a meta* do sistema

1 - Ir de encontro aos requerimentos, tornando o conceito de qualidade algo binário: qualidade existe ou não existe. Para a equipe envolvida na produção de software, qualidade é o encontro com os requerimentos e especificações.

2 - Ir de encontro às necessidades do usuário. Neste ponto o software faz o que o usuário precisa, ou seja, ele está pronto para ser usado.

Em termos de visão popular, qualidade de software é a ausência de bugs[3]. Isso é o conceito básico de que o software está em conformidade com as necessidades básicas do cliente (requisitos), pois caso o software possua muitas falhas, estará fugindo das especificações.

1.2 Atributos de qualidade do software

O desenvolvimento de software com qualidade depende, na grande maioria das vezes, da clareza de suas especificações no momento em que ele está sendo projetado. Mas o que deve ser entendido como especificações neste ciclo de vida do software ? *Especificações* nesta etapa são as descrições de como o sistema é, e o que ele deve fazer. O essencial é construir especificações com alto nível de qualidade.

Sendo assim, podemos definir os seguintes atributos de qualidade do software:

a) Manutenibilidade

Consiste na facilidade com que uma aplicação é modificada. Suas especificações sãos fáceis de serem alteradas e trabalhadas na medida em que se for detalhando o sistema.

b) Reutilizabilidade

Uma aplicação deve servir de apoio para o desenvolvimento de outro produto semelhante. Sua reutilizabilidade deve ser total ou parcial.

c) Avaliabilidade

O sistema desenvolvido deve permitir que seja feito um controle de sua qualidade através da avaliação de como é apresentado (forma) e do que é apresentado (conteúdo) .

[3] Problema ou falha no software.

d) Implementabilidade

Após toda a especificação do sistema, é necessário avaliar a viabilidade de sua implementação. Não há porque continuar com o desenvolvimento de um sistema que não seja viável. Para verificar sobre a viabilidade de um sistema, deve-se levar em conta os seguintes pontos:

- Viabilidade econômica;
- Viabilidade técnica;
- Viabilidade financeira;
- Viabilidade de mão-de-obra;
- Viabilidade de recursos de suporte;
- Viabilidade de cronograma (tempo);
- Viabilidade social.

e) Comunicabilidade

Para que o usuário possa avaliar se o software é o que realmente ele deseja, é preciso que ele leia e entenda o que está escrito. Neste quesito entra a comunicabilidade como item de qualidade de software.

A comunicabilidade de um sistema é obtida através de cinco pontos considerados como fundamentais:

- Uso apropriado da linguagem de especificação;
- Mínimo volume de texto e máximo volume de informação;
- Terminologia igual do início ao fim das especificações;
- Mesmo grau de detalhe em todos os níveis de abstração;

- Modularidade, para que o usuário tenha perfeito conhecimento exatamente no ponto em que procura.

f) Manipulabilidade

Este item refere-se ao controle de revisões de uma especificação. Todas as pessoas envolvidas no processo de desenvolvimento, baseia-se sempre na última revisão das especificações e caso seja necessário, consultar especificações anteriores.

A manipulabilidade possui três sub-características:

- *Fidelidade:* é o grau entre a especificação e a percepção que o especificador tem do problema;
- *Consistência*: não pode haver contradições entre o que foi especificado;
- *Não ambigüidade*: deve haver somente uma interpretação possível.

1.3 Atributos de qualidade de produtos de software

Medir qualidade de um produto muitas vezes é uma tarefa fácil. Pode-se julgar seu tamanho, sua cor, espessura, etc. Porém nem sempre é fácil medir a qualidade do software. O usuário, por mais que tenha em mente o que precisa, nem sempre tem condições para fazer uma avaliação metódica e concisa.

A ISO (Organização Internacional de Padrões) publicou uma norma na qual padronizou a qualidade do produto de software. Trata-se da ISO/IEC 9126 publicada em 1991. Sua tradução foi publicada no Brasil em agosto de 1996 e chama-se NBR 13596. Esta norma traz um conjunto de características que devem ser verificadas em um software para que ele seja considerado um produto de qualidade e está dividida em seis grupos, conforme figura:

Figura 1. Conjunto de características a serem verificadas num software – Norma ISO/IEC 9126 / NBR 13596

a) Funcionalidade

Trata-se do conjunto de funções que satisfazem explicita ou implicitamente as necessidades do usuário;

b) Confiabilidade

O software deve ser capaz de manter um nível de desempenho, sob condições estabelecidas, por um certo período de tempo.

c) Utilizabilidade

É o esforço necessário que o usuário faz para utilizar o software e também o julgamento individual desse uso por um conjunto de usuários.

d) Eficiência

É o nível de desempenho do software e a quantidade de recursos disponibilizados, sob condições estabelecidas.

e) Manutenibilidade

Diz respeito aos esforços despendidos para efetuar modificações no software.

f) Portabilidade

Refere-se à capacidade que o software tem de ser transferido de um ambiente para o outro.

2. Teste de software

No processo de desenvolvimento, a etapa de testes é a que garante a qualidade do software. Através dos testes realizados, é possível reduzir a níveis altamente consideráveis o número de bugs de um sistema. Sendo assim, este é o último passo do desenvolvimento do software.

Podemos afirmar que:

- Teste é uma das áreas da engenharia de software;
- Teste é o processo de executar um programa com o propósito de descobrir erros;
- Um bom caso de teste é aquele com alta probabilidade de revelar um erro ainda não descoberto;
- Um teste bem sucedido é aquele que revela um erro ainda não descoberto, através de esforço e tempo mínimo.

A atividade de teste envolve basicamente:

- *Entradas:* especificação dos requisitos[4] e do projeto do software[5];
- *Processo:* utilização dos planos de procedimentos de testes e ferramentas de auxílio;
- *Saídas:* resultados obtidos na fase anterior.

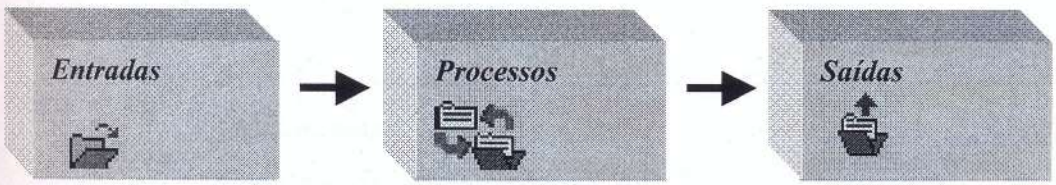

Figura 2. Atividades de teste

[4] Necessidades básicas do cliente
[5] Análise, especificações, design, desenvolvimento e outras documentações associadas.

2.1 Plano de teste

> *" Devemos planejar o teste como um general organiza sua batalha, e devemos testar a aplicação como um ninja, com precisão. Afinal, mesmo que você não veja os "bugs", eles estarão lá na aplicação, esperando por você, aguardando o momento certo de aparecer."*[6]

A partir da especificação do software a ser testado, ou seja, como o software é ou o que faz, gera-se um roteiro de execução manual, automática ou mista. Este roteiro, na qual é chamado de *plano de teste,* visa testar as características funcionais do software em questão. O plano de teste tem por função descrever como e quais são os testes que serão aplicados sobre o sistema. É basicamente um documento que agrega todos os testes relacionados e determina a seqüência e o enfoque da aplicação.

2.1.1 Principais itens do plano de teste

- *Título:* Consiste na identificação do projeto e do plano de testes, incluindo versão e data;
- *Identificação do software:* nome e versão do software;
- *Visão geral:* apresenta uma visão geral do software a ser testado e suas funcionalidades;
- *Pré*-requisitos: conhecimentos prévios que o responsável pelo teste deve ter, bem como os documentos relacionados que devem ser lidos;
- *Ambiente:* Hardware e software necessários para execução dos testes;
- *Requerimentos de testes:* itens (casos de uso, requerimentos funcionais e não funcionais) que devem ser identificados e testados. O Objetivo é identificar o que será testado no software;

[6] Frase colocada no SQA User Mailing List pelo palestrante como elemento motivacional para equipes de testes nos EUA.

- *Estratégias de teste:* técnicas e critérios que serão utilizados nos testes. O Objetivo é identificar como o software será testado;
- *Tipos de testes:* lista os tipos de testes que serão aplicados, como por exemplo, teste de sistema, performance, instalação, etc;
- *Eventos:* lista as atividades de testes com a informação de seu início/fim;
- *Relatórios de resultados:* documento na qual é registrado os resultados obtidos com os procedimentos realizados;

2.2 Casos de teste

Um caso de teste é composto por um conjunto de dados de entrada e por um conjunto de resultados esperados, a fim de determinar se uma característica da aplicação está sendo executada corretamente ou não. A questão da elaboração dos casos de testes é um tanto complexa, pois para um mesmo caso podem existir inúmeras entradas gerando inúmeras saídas.

Vejamos no exemplo abaixo a complexidade da questão:

Especificação do sistema: Um programa recebe pela linha de comando três valores inteiros. Os três valores devem ser interpretados como sendo os comprimentos dos lados de um triângulo. O programa deve imprimir uma mensagem dizendo se o triângulo é eqüilátero, isósceles ou escaleno.

Pergunta: Qual o grau de confiança nos testes desenvolvidos ? Será que o programa foi completamente testado ?

Vejamos a tabela abaixo:

Valor de entrada	Resultado obtido	Justificativa
2,2,2	Triângulo Eqüilátero	Entradas válidas

3,3,4 / 3,4,3 / 4,3,3	Triângulo isósceles	Entradas válidas
2,4,3	Triângulo escaleno	Entradas válidas
3,3,0	Mensagem de erro	Triângulo com lado nulo
-1,3,3	Mensagem de erro	Triângulo c/ lado negativo
1,2,3 / 2,1,3 / -3,1,2	Mensagem de erro	Soma de 2 lados = 3° lado
1,2,4 / 30,12,15 / 10,60,8	Mensagem de erro	Soma de 2 lados < 3 ° lado
0,0,0	Mensagem de erro	Lados nulos
3.3, 3.3, 4.5	Mensagem de erro	Lados não inteiros
4,B,8 / A,B,5 / +$@	Mensagem de erro	Entradas não numéricas
2,4,... / ...,2,4 / 2,...,4	Mensagem de erro	Omissão de um dos valores

Tabela 1. Exemplo de caso de teste

Analisando a tabela, podemos responder a questão acima: não é possível confiar 100% nos testes desenvolvidos, já que inúmeras são as possibilidades de valores de entradas. Por conseqüência, nenhum programa é completamente testado.

Devemos procurar explorar ao máximos os casos de testes possíveis, pois onde menos esperamos, o bug está presente.

Sendo assim, podemos tirar como conclusão na tabela acima que nenhum sistema está livre de bugs, já que nem sempre conseguimos explorar todas as possibilidades de testes.

2.3 Tipos de teste

Para que os testes sejam realizados de maneira correta, é necessário antes que seja estabelecido os tipos de testes que serão aplicados. Os tipos de testes propiciam a elaboração de um plano de teste mais conciso, possibilitando assim extrair o máximo de proveito das atividades. Cabe salientar que nenhum tipo de teste deve ser aplicado isoladamente e sim de maneira que uma complemente a outra.

Existe inúmeros tipos de testes[7] que podem ser aplicados e nesta parte do trabalho, estarei explanando sobre os principais. Vários dos tipos que serão descritos abaixo tem fundamentos nos testes caixa-preta e caixa-branca, que serão descritos mais ao final do capítulo.

2.3.1 Teste de unidade

No software cujo o principio utilizado foi da programação estruturada, ele é aplicado sobre a menor unidade: o módulo ou função. Para os sistemas desenvolvidos na tecnologia orientada a objetos, ele é realizado a nível de componente ou de classe [8].

Um teste de unidade deve verificar os seguintes pontos:

- A interface com o módulo. As informações de entradas e saídas devem ser consistentes entre si;
- A estrutura de dados local. Os dados contidos nas variáveis devem manter sua integridade em todas as etapas de execução do código;
- As condições de limite. Os limites usados para marcar ou restringir o processamento devem garantir a execução correta da unidade;
- Os caminhos básicos. Todas as instruções de uma unidade devem ser executadas pelo menos uma vez;
- Os caminhos de tratamento de erros. Similar aos caminhos básicos, os de tratamento de erros devem ser executados pelo menos uma vez. Através de

[7] Ao final deste capítulo estarei colocando uma tabela com os principais tipos de testes.
[8] Testes orientados a objetos serão discutidos no decorrer do trabalho.

sua execução será verificado que os valores não verdadeiros foram devidamente tratados.

2.3.2 Teste de integração

Este tipo de teste tem por finalidade garantir que os componentes (ou unidades) que foram combinados entre si estão funcionando corretamente.

A integração pode ser de duas formas:

- *não incremental:* As unidades são combinadas e o programa é testado como um todo e;
- *Incremental:* As unidades são gradativamente testadas e integradas.

Destas duas formas, a mais eficiente é a *incremental.* Isso porque é mais fácil encontrar um erro quando o sistema é testado em partes menores.

2.3.3 Teste de validação

Este teste é realizado após o teste de integração, pois depende do sistema funcionando para ser realizado. Podemos tomar como definição deste tipo de teste a seguinte: o sistema passou no teste de validação quando o cliente recebe o software que esperou ou que ele esteja conforme as especificações. As validações do software são feitas através de *testes caixa-preta*[9], pois visam demonstrar a conformidade com os requisitos.

Na maioria das vezes, torna-se praticamente impossível saber como o cliente irá usar o software, devido as fatores como: mal interpretação do uso, combinações não usuais de dados, entre outros fatores. Neste tipo de teste, o usuário também passa a ser um "integrante" da equipe de testes. São realizados dois tipos de teste com a participação do cliente: teste alfa e teste beta.

No caso do *teste alfa,* o cliente testa o produto e o desenvolvedor o acompanha com a finalidade de registrar os erros e problemas na utilização. Este teste apresenta como problema a presença incômoda do desenvolvedor junto ao cliente. O cliente tem dificuldades em agir com naturalidade no seu trabalho.

[9] Técnica que será comentada no decorrer do trabalho

No *teste beta* um ou mais clientes executam os testes no seu próprio ambiente de trabalho e o desenvolvedor não acompanha o processo. O cliente fica responsável por registrar todos os erros e encaminhá-los ao desenvolvedor para correções.

2.3.4 Teste de sistema

Neste tipo de teste, o software tem que funcionar como um todo. O objetivo principal do teste de sistema é verificar se o sistema funciona corretamente em conjunto com outros softwares e elementos de hardware, pois são com eles que o software irá "trabalhar" após ser entregue para o cliente.

O teste de sistema é divido em:

- Teste de recuperação;
- Teste de segurança;
- Teste de estresse e;
- Teste de desempenho.

2.3.4.1 Teste de recuperação

Um sistema nunca está livre de falhas e sendo assim o mesmo deve tolerar falhas. No teste de recuperação, o sistema é "forçado" a falhar de diversas maneiras a fim de verificar como ele se comporta. A recuperação destas falhas pode ser *automática* (realizadas pelo próprio sistema) ou *não automática* (é necessário a intervenção humana). Caso seja necessário uma intervenção humana, é avaliado se o tempo gasto está dentro dos limites aceitáveis.

O teste de recuperação usa casos de testes para verificar como o sistema se comporta diante de uma falha no disco, interface, memória, etc. O sistema deve possui um registro de log[10] com a finalidade de registrar as atividades antes da falha acontecer.

[10] Registro das operações de processamento do computador.

2.3.4.2 Teste de segurança

Visa testar a segurança da aplicação nas mais diversas formas. Tem por objetivo verificar se todos os mecanismos de proteção estão realmente funcionando e garantir que o sistema funcione adequadamente mediante tentativas de acessos ilegais. É importante que estes testes não sejam feitos pelos desenvolvedores, pois estes tendem a testar somente os mecanismos de proteção implementados, visto que são de seu conhecimento.

2.3.4.3 Teste de estresse

Também é necessário testar a aplicações nas situações inusitadas. Neste caso de teste, o sistema é carregado com um grande volume de informações com a intenção de interromper o seu bom funcionamento. Então, é monitorado a sua perda de desempenho e suscetibilidades a falhas.

O testador deve utilizar o sistema com quantidade e freqüência em volume anormais. Basicamente, pode ser usado as seguintes técnicas:

- Procura em excesso dos dados em disco;
- Grande aumento de índices;
- Abertura de muitas janelas;
- Arquivos em formato não compatível, etc.

2.3.4.4 Teste de desempenho

Um sistema pode até satisfazer as condições exigidas, porém se o desempenho for baixo, torna-se inaceitável perante o cliente. Neste tipo de teste, é realizado um monitoramento e registro de desempenho durante as cargas de estresse regulares, baixas e altas.

O teste de desempenho preocupa-se basicamente com o tempo de resposta e capacidade, especialmente do banco de dados. Ocorre que nem sempre a equipe de teste possui um grande volume de informações a fim de realizar este teste. Nesse caso, pode-se "simular" um grande número de informações reduzindo, por exemplo, a capacidade do

hardware. Entretanto, deve-se ter em mente que isso é apenas um subterfúgio para suprir essa dificuldade e o resultado pode não ser o esperado.

2.3.5 Teste de Caixa-Preta (Black-Box Test)

O tipo de teste caixa-preta tem por objetivo verificar se o conjunto de valores de entradas produziu o conjunto de valores de saídas esperado. Os casos de testes gerados nesta técnica baseia-se exclusivamente na especificação dos requisitos do sistema. Podemos definir que este tipo de teste tem como objetivo macro garantir que todos os requerimentos ou comportamentos da aplicação estejam funcionando adequadamente.

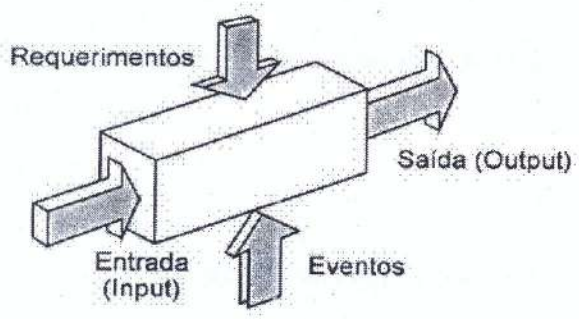

Figura 3. Visão Macro do teste caixa-preta

Os testes caixa-preta visam descobrir erros nas seguintes categorias:

- Funções incorretas ou ausentes;
- Interfaces;
- Estrutura de dados ou acesso a banco de dados externos;
- Desempenho;
- Inicialização e término.

Existe uma série de técnicas nas quais são usadas para gerar casos de testes caixa-preta. Estarei explanando um pouco sobre algumas das técnicas conhecidas porém sem adentrar em estudos mais profundos de cada técnica visto que não é o foco do trabalho.

2.3.5.1 Técnica de valor limite

A técnica de valor limite cerca os limites dos valores de entradas. Parte-se do princípio que os erros costumam ocorrer próximos aos valores limites das variáveis de entrada. A idéia desta técnica é utilizar valores de entradas no seu valor máximo, logo abaixo do máximo, um valor nominal, logo acima do mínimo e um valor mínimo para elaborar os casos de teste. Nomeamos estes valores de: *MIN, MIN+,NOM, MAX-, MAX.*

O uso do valor limite na elaboração dos casos de testes depende muito do que se deseja testar. Quando estamos lidando com quantidades físicas ela tem funcionalidade. Num teste de temperatura, por exemplo, precisamos saber o que acontecerá se uma caldeira atingir seu valor máximo de temperatura. Agora se estamos testando um programa que controla número de telefone, para que iremos elaborar casos de testes com os números 000, 001, 998,999 ?

2.3.5.2 Classes de equivalência

Classes de equivalência representam um conjunto de estados válidos ou inválidos para determinadas condições de entradas. O domínio dos dados de entradas são divididos de forma a evitar redundância nos casos de testes, ocorrendo então *o particionamento de equivalência.* Baseando-se na teoria dos conjuntos, um conjunto de partição é válido somente se a união de todos os elementos particionados representarem o domínio das entradas, ou seja, ao dividirmos um conjunto de valores (*A*) em sub-conjuntos (A_1, A_2, A_3, etc), estes sub-conjuntos somente serão considerados partição de *A* se e somente se a união de A_1, A_2, A_3 for igual a *A*.

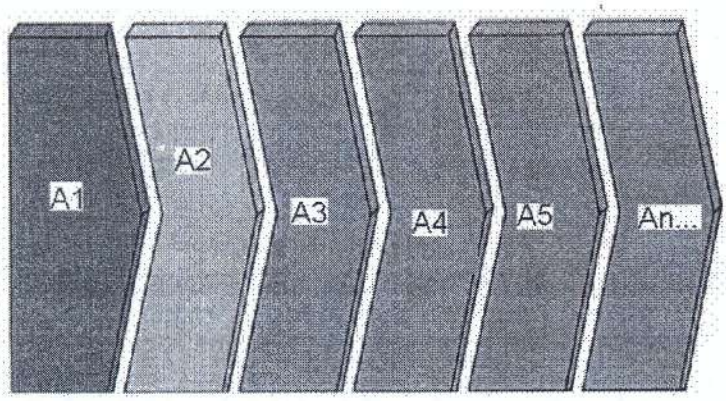

Figura 4. Particionamento de equivalência

O conjunto de partição é importante para a realização de testes porque traz a idéia de completude (todos os elementos estão em um sub-conjunto) e não há redundância (nenhum elemento está em mais de um sub-conjunto).

2.3.5.3 Tabelas de decisão

As tabelas de decisões descrevem situações nas quais as possíveis combinações de resultados dependem das várias combinações das condições de entradas. As tabelas com entradas binárias (V/F, 0,1) são chamadas *tabelas de decisão com entradas limitadas*. Os principais elementos de uma tabela de decisão são:

- *Condições:* Variáveis que serão levadas em conta para as tomadas de decisões;
- *Entradas:* Condições que estão valendo para uma dada variável;
- *Ações:* Conjunto de operações que serão desenvolvidas condicionadas pelas respostas das condições;
- *Regras:* Conjunto de situações que são verificadas em resposta às condições.

Para a determinação do número de casos de testes possíveis com a tabela usamos a seguinte fórmula: 2^n, onde n = número de condições.

Condição 1	V	V	F	**F**
Condição 2	V	F	V	**F**
Ação 1		X		**X**
Ação 2	X	X		**X**
Ação 3				**X**

Tabela 2. Tabela de decisão com entradas limitadas

2.3.5.4 Grafos de causa e efeito

As técnicas apresentadas até então (técnica de valor limite, classes de equivalência, tabelas de decisão) possuem como ponto fraco a dificuldade de explorar combinações de entradas, pois estas costumam ser enormes. Para auxiliar na seleção dos casos de testes possuímos a técnica dos grafos de causa e efeito. O *grafo de causa e efeito* é uma técnica de projetos de caso de teste que oferece uma representação das condições lógicas e ações correspondentes. Uma *causa* é uma condição de entrada ou uma classe de equivalência sobre variáveis de entrada. Um *efeito* é uma condição de saída ou uma transformação no estado do sistema. Não estarei adentrando no assunto da elaboração do grafo, pois não é este o objetivo do trabalho, porém os passos para a elaboração são os seguintes:

- Dividir a especificação em unidades ou módulos;
- Desenvolver um grafo de causa e efeito;
- Converter o grafo em uma tabela de decisão;
- Converter as regras da tabela de decisão em casos de teste.

2.3.6 Teste de Caixa-Branca (White-Box Test)

Os testes funcionais (caixa-preta), como foi dito acima, são projetados para verificar se o software satisfaz os requisitos descritos na especificação. Neste tipo de teste não é necessário conhecer o código-fonte do programa. Nenhum tipo de teste cobre 100% das possibilidades de casos de entrada, porém pode haver casos em que dificilmente um teste caixa-preta detectaria um bug. Por exemplo: dado um determinado sistema de cadastro de funcionário. Os testes funcionais poderiam executar todas as possibilidades de entradas e saídas, garantindo que o programa está livre de falhas. Porém vamos supor que a cada execução da rotina o sistema atualize a base de dados, aumentando em 10% o salário de uma determinada categoria. Isso dificilmente um seria visto, pois a especificação do sistema não diria que não deverá ser alterado o salário dessa ou daquela categoria.

Os testes de caixa-branca (teste estrutural) garantem que todas as linhas de código e condições foram executas e estejam corretas. Os casos de testes deste tipo tem por objetivo:

- Garantir que todos os caminhos independentes dentro de um módulo foram executados pelo menos uma vez;
- Exercitar todas as condições lógicas verdadeiras ou falsas;
- Executar todos os laços em suas fronteiras e dentro de seus limites;
- Exercitar as estruturas de dados internas a fim de garantir sua validade.

O teste caixa-branca são agrupados em duas categorias:

2.3.6.1 Teste de caminho básico

Através do teste de caminho, definimos a relação entre um caso de teste e a parte do programa que será exercitada por ele. Um caso de teste sempre executa o programa por completo. Na técnica do caminho básico, cada caso de teste corresponde a um caminho diferente no grafo do programa[11].

[11] Grafo onde os nodos do programa representam comandos executáveis. Este tema não será abordado no trabalho por fugir do propósito da pesquisa.

2.3.6.2 Teste de estrutura de controle

- *Teste de condição*: Este método tem por objetivo testar as condições lógicas do programa.
- *Teste de fluxo de dados*: Este método testa caminhos do programa que contenham instruções de laços e if aninhados.
- *Teste de laços (loops):* Concentra-se os testes na validade de construções de laços, pois estes são o fundamento para a maioria dos algoritmos implementados no sistema.

2.3.7 Principais tipos de testes

Procurei no decorrer do capítulo explanar sobre os principais tipos de testes, porém existem outros que não foram colocados. Como um 'complemento' e para fins de conhecimento, segue abaixo uma tabela [2] com os principais tipos de testes. Nesta tabela , por se tratar de um 'guia', também constará alguns do tipos já explicados acima.

Tipos de teste	Descrição
Teste de Unidade	Teste em nível de componente ou classe. É o teste cujo objetivo é um "pedaço do código".
Teste de Integração	Garante que um ou mais componentes combinados (ou unidades) funcionam corretamente.
Teste de Sistema	A aplicação tem que funcionar como um todo. Neste momento a aplicação tem que "fazer aquilo que diz que faz".

Teste Operacional	Garante que a aplicação pode "rodar"muito tempo sem falhar.
Teste Negativo-Positivo	Garante que a aplicação vai funcionar no "caminho feliz"de sua execução e vai funcionar no seu fluxo de exceção.
Teste de Regressão	Um dos mais importantes testes. "Para irmos para o futuro, temos que voltar ao passado, sempre". Toda vez que formos inserir uma característica nova na aplicação, devemos testar t-o-d-a a aplicação. Afinal, podemos, ao "consertar algo, quebrar outro".
Teste de Caixa-preta ou Black-Box Test	Testar todas as entradas e saídas desejadas. Não se está preocupado com o "código".
Teste de Caixa-branca ou White-Box Test	O objetivo é testar o código. Às vezes existem partes do código que nunca foram testadas.
Teste Beta	O Objetivo é testar a aplicação em produção.
Teste de Verificação de Versão	Toda vez que se libera uma nova versão da aplicação, existem condições mínimas que validam se a versão liberada está OK. Este teste é usado durante o processo de construção da aplicação. Pode querer testar às vezes apenas uma parte da aplicação.
Teste Funcional	Testar se as funcionalidades presentes na documentação funcionam como especificado. Incluem-se as regras de negócio.

Teste de Interface	Verifica se navegabilidade e os objetos de tela funcionam corretamente, em conformidade com os padrões vigentes (em nível de interface).
Teste de Performance	Verifica se o tempo de resposta é o desejado para "momento"de utilização da aplicação e suas respectivas telas envolvidas.
Teste de Carga	Verifica se a aplicação suporta a quantidade de usuários simultâneos requeridos.
Teste de Aceitação do Usuário ou UAT (user acceptance Test).	A meta é clara. É um teste exploratório voltado para validar aquilo que o usuário deseja, tendo um objetivo claro: dar o aceite ou não.
Teste de Estresse	Testar a aplicação em situações inesperadas.
Teste de Volume	Testar a quantidade de dados envolvidos (pode ser pouca, normal, grande ou além de grande).
Teste de Configuração	Testa se a aplicação funciona corretamente em diferentes ambientes de hardware e software. Uma aplicação pode funcionar bem numa máquina Pentium com 128MB de RAM e 3 Giga de HD, porém pode funcionar melhor ainda num Pentium com 64MB de RAM e 10 Giga de HD. Parece ser o contrário, mas cada aplicação é um caso. Esteja sempre preparado quando se trata de ambientes de testes. Pode mascarar outros testes.
Teste de Instalação	Verificar se a instalação da aplicação (hardware e software) foi OK.

	OK.
Teste de Documentação	A documentação existe? Mostra que o software faz efetivamente? Falta algo na documentação?
Teste de Integridade	O Objetivo é testar a integridade dos dados armazenados.
Teste de Segurança	Testar a segurança da aplicação nas mais diversas formas.
Teste de Aplicações Mainframe	Teste de aplicações mainframe que requer um formalismo e um forte planejamento de testes.
Teste de Aplicações Client	Neste momento se está preocupado mais com a funcionalidade, com a interface e a performance do que com outras coisas.
Teste de aplicações Server	Neste momento se está preocupado mais com desempenho dos processos que com a integridade dos dados.
Teste de Aplicações Network	A análise estatística do desempenho das aplicações é enfoque pouco utilizado, daí o fato de certas aplicações serem testadas em produção.
Teste de Aplicações Web	Na verdade desfez-se um mito, em que não somente a interface é fundamental, mas o desempenho e a adequação das necessidades aliados a uma alta flexibilidade das ferramentas envolvidas são fator-chave de sucesso. Um planejamento estratégico dos testes passou a ser fundamental.
Teste de Monitoração	São, na realidade, testes funcionais, que visam verificar o status e disponibilidade de diversas funcionalidades e da

	status e disponibilidade de diversas funcionalidades e da aplicação em si. Isto incluir verificar o SLA e o SLM da aplicação.
Teste de Ameaça ou Thread	Semelhante ao de segurança, mas em escala mais em nível de falhas.
Monkey Test	Testa o aplicativo de forma aleatória e inesperada. O teste do macaco é antes de tudo "sem planejamento".
Teste de Módulo	Teste de um módulo, porém em nível menor. Semelhante ao de unidade, porém é mais abrangente que este.

Tabela 3. Principais tipos de testes

3. Testes de software orientados a objetos

O teste em software desenvolvido com base na programação orientada a objetos apresenta as seguintes facilidades em relação aos testes de software desenvolvido na programação estruturada:

- As interfaces de classes[12] e métodos[13] estão bem definidas e explicitas;
- Há um número reduzidos de parâmetros e isso implica em menos casos de testes;
- A herança[14] sugere uma maneira de utilizar os mesmos casos de teste.

Porém nem tudo é facilidade. As desvantagens também existem nestes tipos de testes:

- O encapsulamento[15] pode causar dificuldades em avaliar se a classe está correta ou não;
- Os vários métodos de uma classe dificultam os testes;
- O polimorfismo[16] expande as possíveis ações entre os objetos.

3.1 Tipos de teste

No capítulo 2 foi descrito sobre técnicas estruturais e funcionais voltadas para o paradigma da programação estruturada. Neste capítulo, estarei abordando sobre a aplicação das principais técnicas voltadas para a programação orientada a objetos. Os tipos de teste funcional (caixa-preta) e estrutural (caixa-branca) também são utilizados no paradigma da programação orientada a objetos, já que visam testar entradas/saídas de dados e código respectivamente. Muitas técnicas são comuns aos dois paradigmas de programação (caixa-preta, caixa-branca, teste de validação, teste de sistema, etc) então não estarei explicando sobre essas técnicas novamente devido aos detalhes já terem sidos mencionados no capítulo anterior.

[12] Modelo para criação de vários objetos (elemento independente).
[13] Instrução ou função relacionada a uma classe.
[14] Mecanismo na qual uma classe é criada com base em outra já existente.
[15] Forma de esconder atributos e variáveis de um determinado objeto.
[16] Diferentes comportamentos que um método pode assumir ao ser requisitado.

3.1.1 Teste de estado

O comportamento de uma determinada classe do sistema, é determinado por um conjunto de valores encapsulados num determinado momento, por uma seqüência de mensagens[17] ou até mesmo por ambos os casos. O objetivo do teste de estado é testar o sistema sem utilizar todas as combinações possíveis. Um estado é definido como sendo uma partição (subconjunto) de todas as combinações dos valores de atributos[18] da classe. Nos casos de testes, esses atributos são valores que podem ser combinados e tem alguma propriedade de interesse comum.

3.1.2 Teste Incremental

O propósito da programação orientada a objetos é a criação de classes que poderão ser reutilizadas e que sejam confiáveis. Sendo assim é preciso que estas classes sejam testadas. O Objetivo deste tipo de teste é verificar as classes de maneira isolada, de forma incremental e através da herança, reutilizar os casos de teste sempre que possível. É utilizada uma abordagem *top-down* (de cima para baixo) onde as superclasses (classe principal) são testadas inicialmente partindo-se então para as sub-classes onde os casos de testes são reaproveitados na medida do possível. Existe entretanto casos em que os novos atributos da sub-classe requer novos casos de teste devido ao seu novo contexto. Este teste é chamado de incremental porque utiliza os resultados de teste de uma hierarquia superior para reduzir os esforços necessários nos níveis subseqüentes.

3.1.3 Teste de Integração

No capítulo 2 foi dada uma explicação sobre os testes de integração. Neste capítulo estarei entrando mais na aplicação nos sistemas orientados a objetos.

Após ter sido realizados os testes de unidade[19], chega o momento de realizar o teste de integração do sistema. A integração dos módulos deve ser feita de maneira incremental (unidades gradativamente testadas e integradas). Neste tipo de teste devem ser construídos módulos de apóio, chamados *drivers e stubs* . *Driver* é um módulo *que chama* outro que está sendo testado e possui em seu corpo apenas inicializações das variáveis globais, chamadas de rotinas e inicializações dos parâmetros necessários. Já um *stub* é um

[17] Mecanismo que permite a comunicação entre os objetos.
[18] Valores que são armazenados numa classe como por exemplo: código, nome, etc.
[19] Discutido no capitulo 2 do trabalho

módulo *que é chamado* pelo módulo que está sendo efetivamente testado e contém em seu corpo apenas a atribuição de valores que deverão ser retornados, caso seja necessário. A utilização de um driver ou de um stub depende da estratégia a ser aplicada nos testes. Quando é utilizada a estratégia *top-down* (teste de sistema ao teste de unidade), a integração inicia-se com o módulo inicial do sistema (módulo do topo) partindo-se então para os outros módulos do sistema. Os valores iniciais para o teste do módulo inicial são fornecidos pelos *stubs,* pois os módulos que possuem funções de entradas ou saídas estão localizados na base do diagrama dos módulos. Quando todos os módulos e respectivos *stubs* forem testados, cada *stub* é substituído pelo módulo correspondente de maneira que o teste de integração termina quando todo o sistema estiver integrado e as interfaces entres os módulos também estiverem testadas.

Também é utilizada a estratégia *bottom-up* (teste de unidade ao teste de sistema), os testes são iniciados pelos módulos de base do sistema. Para este tipo de estratégia, é necessária a construção de *drivers* para que os outros módulos do sistema sejam chamados. Neste tipo de teste não é construído inicialmente um 'esqueleto' do sistema e o programa passa realmente a existir quando o último módulo é integrado. Este é tipo como um dos grandes problemas desta técnica. Uma vantagem desta técnica é que as funções de entradas e saídas são integradas no início dos testes e sendo assim não é necessário a construção de tantos *drivers, o* que não acontece com o número de *stubs* da estratégia *top-down.*

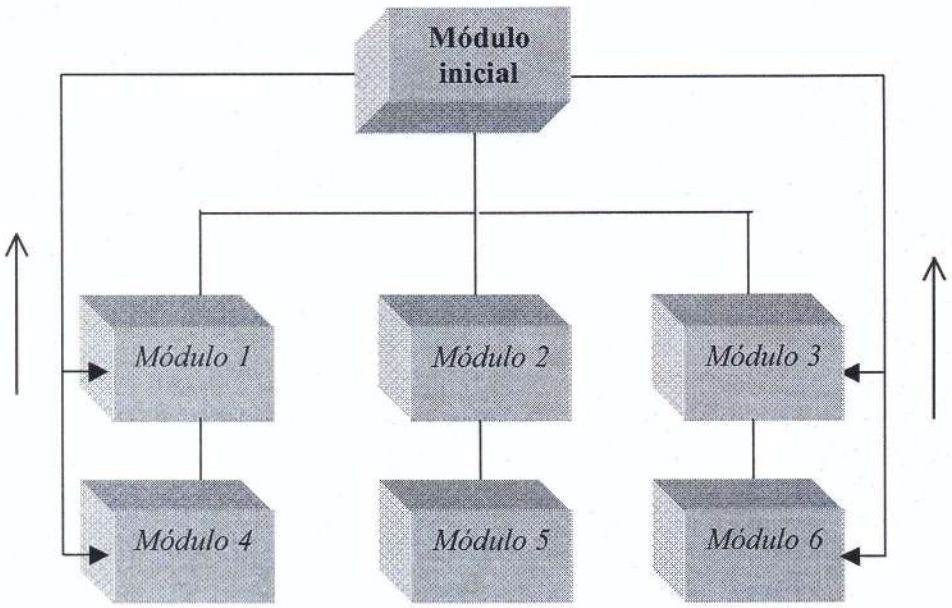

Figura 5. Estratégia bottom-up

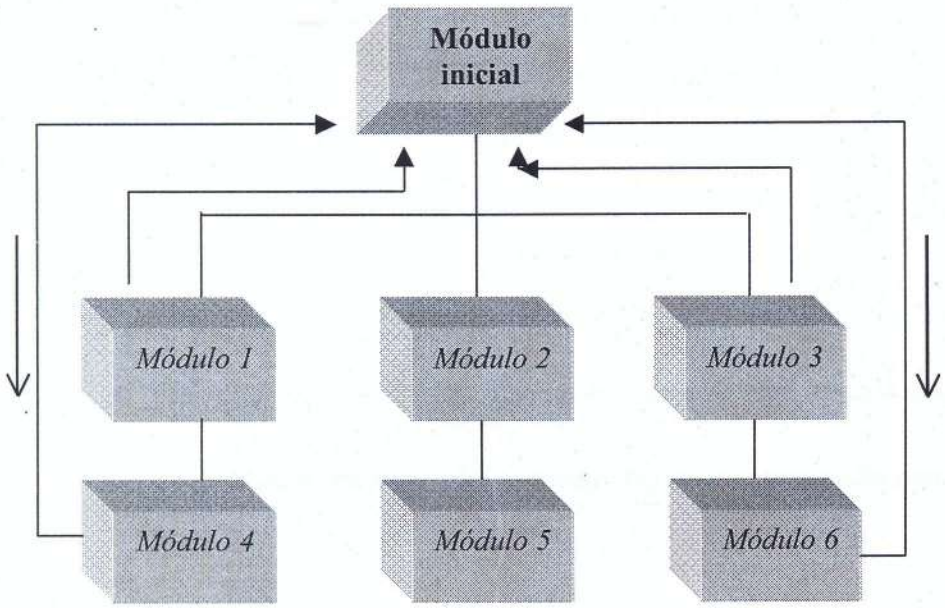

Figura 6. Estratégia top-down

3.1.4 Teste de caso de uso

Casos de uso são maneiras específicas de utilização do sistema na qual é executado parte de sua funcionalidade. Cada *caso de uso* é realizada através de uma seqüência de ações realizadas por alguém que interage com o sistema. O conjunto desses casos especifica todas as formas de utilização do sistema. Ao utilizar *casos de uso,* devem ser atingidos três objetivos:

- Definir o sistema, através da linguagem do usuário, com base em três princípios: visão do usuário final, do testador e do desenvolvedor;
- Casos de usos podem ser refinados em casos de teste de aceitação;
- Fornecimento de base para a construção da documentação para o usuário final.

Exemplo de um caso de uso: Locar uma fita de vídeo

- Funcionário da locadora efetua uma locação em quantidade menor que o número de fitas disponível na prateleira;
- Funcionário efetua uma locação em quantidade superior ao disponível na prateleira;
- Funcionário efetua uma locação em quantidade igual ao número de fitas disponível na prateleira;
- Funcionário tenta efetuar uma locação de fita não existente na locadora.

Para definir *casos de uso,* devem ser seguidas as seguintes diretrizes:

- Não devem ser escritas pelos usuários, já que eles tendem a elaborar casos complexos na qual descrevem telas e outras situações que dificultam a testabilidade;
- Ser baseados nas entrevistas com os usuários;
- Visar problemas que devem ser resolvidos pelos usuários;

- Atingir os objetivos dos usuários;
- Verificar outros sistemas que interagem com o sistema;
- Estar atento as notificações que devem ser feitas aos usuários quando ocorrerem alguns eventos;
- Observar as tarefas feitas com freqüência pelos usuários;

4. Estudo de caso

Para que se cumpra o objetivo deste trabalho, que é de mostrar se foi produzido um software de qualidade, irei elaborar um estudo de caso sobre um sistema de locadora de filmes. Não estarei demonstrando o teste completo do sistema e sim algumas partes para que o leitor saiba o que se tem para fazer e como fazer.

Testar um software é algo muito mais complexo do que o apresentado neste estudo de caso, pois envolve ambiente, pessoas e mão-de-obra. Além de que existe inúmeros testes que realmente somente são possíveis de fazer em presença de alguns recursos materiais, algo impossível de mostrar no papel.

4.1. Plano de teste

Segue abaixo uma tabela representando um plano básico de teste. O mesmo está totalmente voltado para testes funcionais, na qual não se tem por objetivo estar analisando o código-fonte do programa.

Título	***Finalidade***: sistema de locadora que tem por objetivo controlar o processo de locação de filmes. ***Data de desenvolvimento***: 14/06/2004 ***Versão do sistema***: 1.00
Identif. do software	Discover, versão 1.00
Visão geral	Sistema possui como principal finalidade o controle rápido e seguro da locação de filmes. Através dele é possível manter o cadastro dos clientes, filmes, atores, locações, entre outros. Através dele, o cliente poderá lançar as locações, emitir recibos, consultar as locações efetuadas em qualquer época, bem como saber qual funcionário realizou. O sistema possui relatórios que o cliente poderá estar usufruindo para sua maior comodidade. Um item de bastante utilidade no sistema é a consulta dinâmica. Através dela, o cliente poderá buscar qual tabela e determinar

	condições para a consulta.
Pré-requisitos	O responsável pela execução dos testes deverá ter conhecimento pleno das rotinas de uma locadora de filmes, bem como estar atento aos documentos de projeto do sistema.
Ambiente	Para execução dos testes deverá ser usando um microcomputador com processador Pentium III 750MHZ 128MB de memória RAM.
Requerimentos de testes	Serão testados no sistema as seguintes rotinas: **Cadastros**: inserir, excluir, salvar, editar, cancelar, atualizar, bem como a navegação através de botões. **Consultas**: verificar a integridade relacional das tabelas e respectivos campos, critérios e organização. **Relatórios:** verificar a consistência das informações, bem como a impressão em vídeo, impressora matricial, laser e jato de tinta. **Login:** verificar se as condições para o cadastro, alteração de senhas, permissões, estão corretas para o nível de usuário. **Funcionalidades gerais:** verificar se todas as demais funcionalidades estão devidamente corretas.
Estratégias de testes	Verificar todas as entradas e saídas válidas e inválidas do sistema.
Tipos de teste	Testes funcionais
Eventos	Inicio das atividades de teste: 15/06/2004 Previsão de término das atividades de teste: 18/06/2004

Relatórios de resultados	Vide modelo item 4.4

Tabela 4. Plano básico de testes

4.2. Casos de teste

O sistema na qual estou tomando por base este estudo de caso é de locadora de filmes, na qual desenvolvi como trabalho final na disciplina de LTP IV. Como já mencionei na sub-seção 2.2, um caso de teste é composto por um conjunto de dados de entradas e por um conjunto de resultados esperados. Este sistema não requer combinações de dados de entradas, como no exemplo do sistema do triângulo[20], e sim que eles sejam válidos ou satisfaçam algumas condições (que estarei falando na próxima sub-seção). Sendo assim não estarei dando tanta ênfase neste tópico e sim um breve exemplo de caso de teste na inserção de um cliente.

Figura 7. Casos de teste – Cadastros de clientes

No processo de inserção de clientes, deverão ser observados os seguintes aspectos:

- O campo código do cliente deverá ser auto-enumerável. Caso o último cliente tenha sido excluído, o próximo cadastro deverá então assumir este número;
- Os campos nome, sobrenome, endereço, e-mail poderão aceitar letras e números, como por exemplo: João da Silva, js45@ig.com.br, etc.;
- O campo cpf somente deverá aceitar números. Exemplo: 139.658.699-99;

[20] Mencionado na sub-seção 2

- O campo Rg deverá ser alfanumérico. Exemplo: 25.792.238-AB

4.3. Estratégias de teste

Com base nas especificações dos requisitos do sistema, podemos elaborar procedimentos de testes. Através da elaboração desta rotina de trabalho, podemos estabelecer o que iremos testar no sistema (requerimentos de testes), determinando condições, ações, resultados esperados e métodos que serão acionados. Nas sub-seções abaixo, estarei apresentando alguns dos módulos do sistema e uma tabela com os procedimentos adotados. As estratégias que poderiam ser adotadas são inúmeras. Sendo assim, estarei colocando algumas possibilidades dos módulos, a título de exemplo, para que o leitor possa vir a ter um conhecimento prático.

4.3.1. Cadastro de Clientes

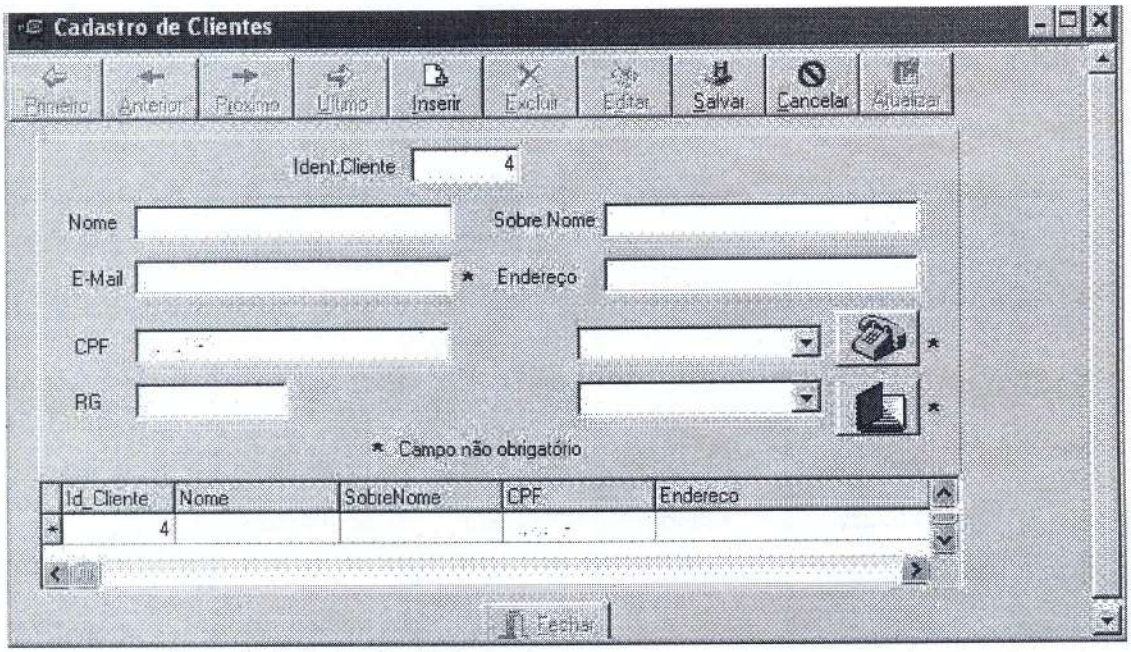

figura 8. Cadastro de clientes

Procedimentos	*Pré-condição*	*Condição de entrada*	*Ação a ser executada*	*Resultado esperado*

Inserir cliente	-	Informação de todos os dados válidos	Clicar no botão Salvar	O cliente deverá ser gravado no arquivo
Inserir cliente	-	Deixar em branco campos não obrigatórios	Clicar no botão Salvar	O cliente deverá ser gravado no arquivo
Inserir cliente	-	Informar CPF inválido	Ir para próximo campo	"CPF inválido"
Inserir cliente	-	Deixar em branco campo(s) obrigatório	Clicar no botão Salvar	"Preenchimento incompleto"
Excluir cliente	cliente já cadastrado	Localizar cliente	Clicar no botão excluir	Cliente excluído da base de dados
Excluir cliente	Nenhum cliente cadastrado	-	Clicar no botão excluir	"Não há registros para excluir"
Editar cliente	Cliente já cadastrado	-	Clicar no botão editar	Habilitar todos os campos para edição

				edição

Tabela 5. Estratégias de teste – Cadastro de clientes

4.3.2. Cadastro de fitas

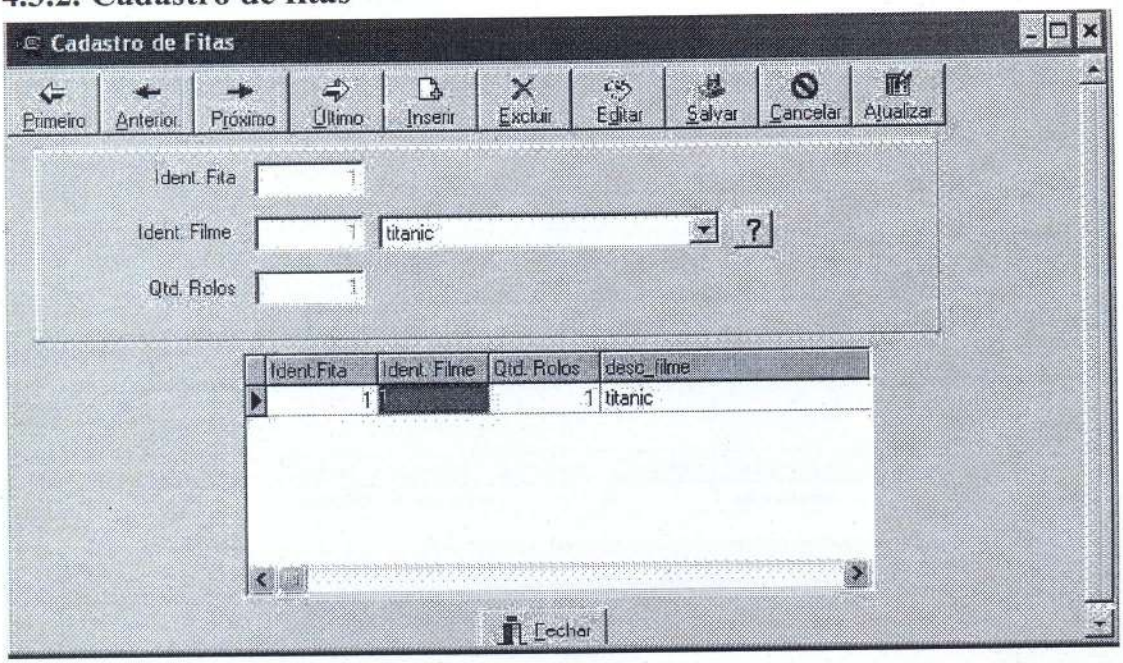

Figura 9. Cadastro de Fitas

Procedimentos	*Pré-condição*	*Condição de entrada*	*Ação a ser executada*	*Resultado esperado*
Inserir uma fita	Filme já cadastrado	Informar qtd de rolos	Clicar no botão salvar	Fita deverá ser inclusa na base de dados
Inserir uma fita	Filme sem cadastro	Informar qtd de rolos	Clicar no botão salvar	"Filme não cadastrado"
Inserir uma fita	Filme já cadastrado	Informar qtd de rolos negativa	Clicar no botão salvar	"número de rolos inválidos"

Tabela 6. Estratégias de teste - Cadastro de Fitas

4.3.3 Cadastro de locações

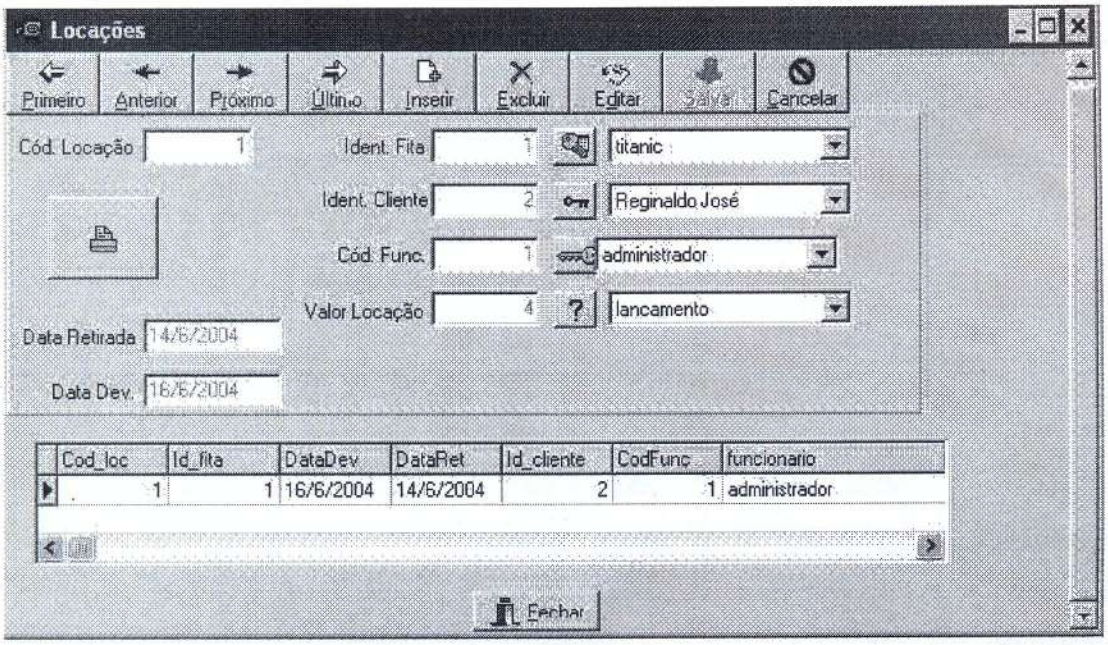

Figura 10. Cadastro de locações

Procedimentos	*Pré-condição*	*Condição de entrada*	*Ação a ser executada*	*Resultado esperado*
Efetuar locação	Fita, cliente, funcionário e valor locação já cadastrados	Cód. Da fita, cód. Do cliente, cód. Do func., valor da locação, datas retirada/dev.	Clicar no botão salvar	Locação inclusa na base de dados
Efetuar Locação	Fita sem cadastro	Cód. Da fita, cód. Do cliente, cód. Do func., valor da locação, datas	Clicar no botão Salvar	"Fita não cadastrada"

		retirada/dev.		
Efetuar Locação	Campo data retirada em branco	Cód. Da fita, cód. Do cliente, cód. Do func., valor da locação, data devolução	Clicar no botão Salvar	"Preenchimento incompleto"
Efetuar locação	Campo data retirada maior que a data devolução	Cód. Da fita, cód. Do cliente, cód. Do func., valor da locação, datas retirada/dev.	Clicar no botão salvar	"Data da retirada maior que data da devolução"

Tabela 7. Estratégias de testes - Cadastro de locações

4.4 Problemas encontrados no software

Na sub-seção acima descrevi algumas estratégias de teste utilizada no sistema. Nesta parte do estudo de caso, irei mostrar alguns bugs encontrados no software, onde estarei utilizando figuras e explicações para elucidar os erros ocorridos.

Situação 1:

Após o término do cadastro de clientes, o programador permitiu que o botão "Salvar" ficasse acessível ao usuário. Isso é uma falha no sistema, pois caso o usuário clique novamente no botão, é acionado o método salvar e a tabela não se encontra mais no modo de inserção ou edição.

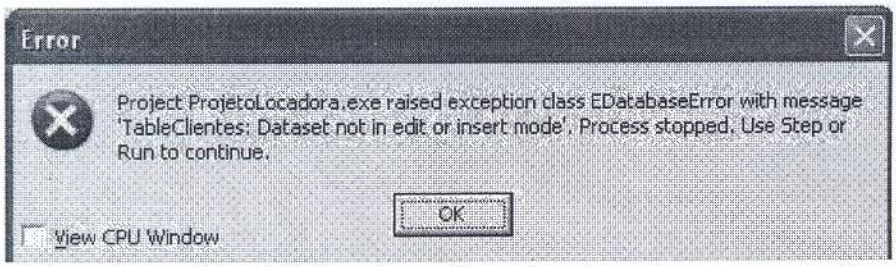

figura 11. Erro no cadastro de clientes

Uma linha de código extremamente simples poderia ter resolvido esse bug do sistema. Bastaria nublar o botão de salvar após a tabela ter entrado em modo de gravação. Vejamos a figura abaixo:

```
      DBEDitNome.SetFocus;
   End
else
        Dm.Tableclientes.post;
        tbsalvar.enabled := false;
end;
```

figura 12. Trecho de código do cadastro de clientes

Situação 2:

O usuário conseguiu cadastrar uma fita sem o filme estar cadastrado. Isso realmente é um bug do sistema, pois as tabelas possuem relacionamento entre si. Nunca é possível se ter uma fita caso não exista um filme dentro dela!!! Podemos observar pela figura abaixo a inclusão da fita sem a informação do respectivo filme.

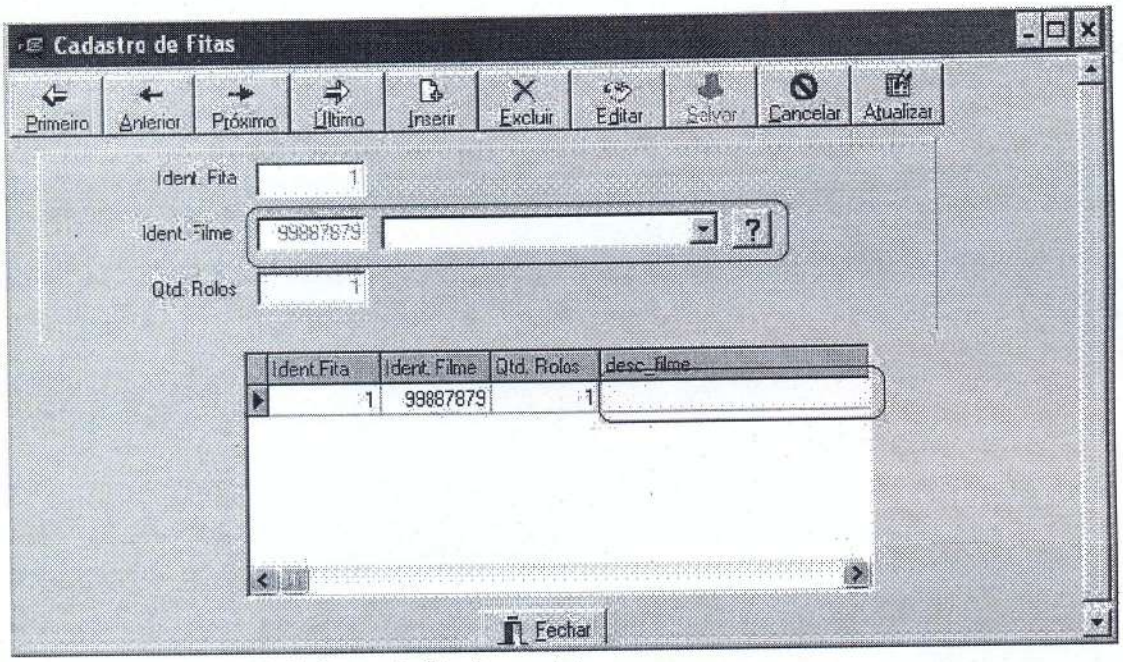

figura 13. Cadastro de fitas incompleto

A solução para este caso também é muito simples. Bastaria incluir uma linha de pesquisa na tabela de filmes junto ao método salvar e caso o filme não estive cadastrado, uma mensagem seria exibida. Veja na figura abaixo o trecho de código, já implementado, e que realiza esta função:

```
if not DM.TableFilme.Locate('id_Filme',DBEditFilme.Text,
[LoCaseInsensitive]) then
    begin
       ShowMessage( 'Filme não cadastrado ! ');
       DBEditFilme.enabled := true;
       DBEditFilme.Setfocus;
    end
```

figura 14. Trecho de código do cadastro de fitas

Situação 3:

Num processo de locação, a data de devolução da fita tem que ser maior ou (no máximo) igual a data de retirada. Uma simples inversão no nome do componente que recebeu estas informações originou uma validação contrária a que realmente deveria ter sido feita. O sistema passou a permitir que a data de retirada fosse maior que a data de devolução e que ainda numa situação absolutamente normal (conforme mostra a figura abaixo), apresentou uma mensagem que deveria ser da situação contrária. Ocorreu neste caso 'duplo' bug. Vejamos a figura abaixo:

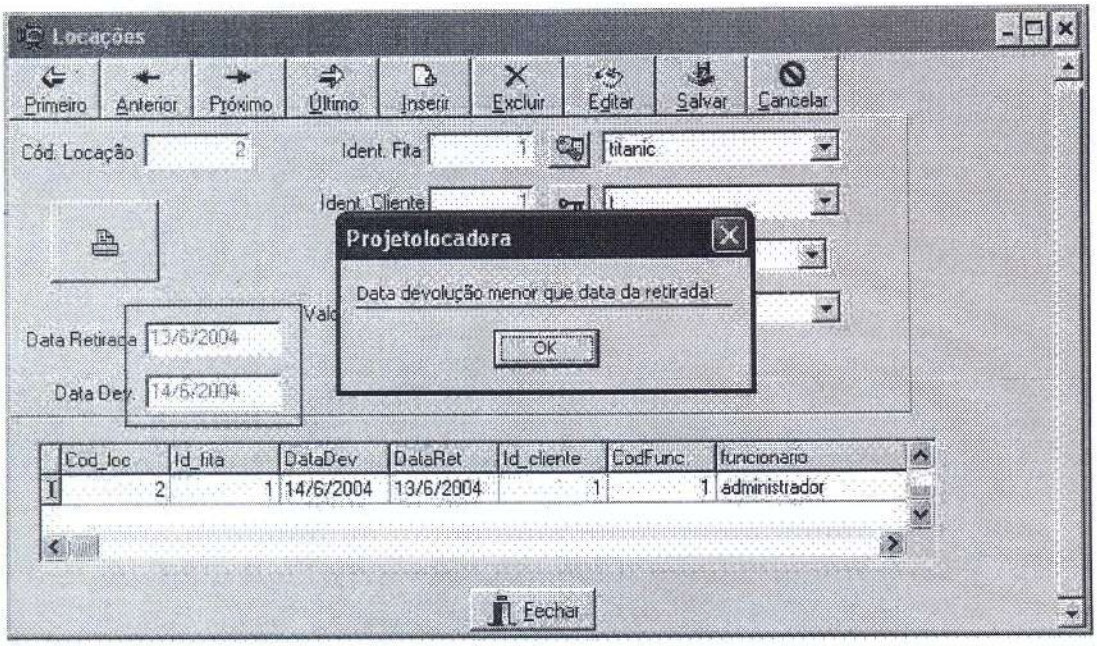

Figura 15. Locação de filmes

Vejamos agora a parte do código que originou esse erro no sistema:

```
    end

else If StrToDate(DBERet.Text) < StrToDate(DBEDev.Text) then
begin
    ShowMEssage('Data devolução menor que data da retirada!');
    dbedev.enabled := true;
    dbedev.setFocus;
 end
else Dm.TableLocacao.post;
tbsalvar.enabled := false;
```

figura 16. Trecho do código da locação de filmes.

Se o trecho **'else if'** do código tivesse sido escrito: 'else if strtodate(dbedev.text) < strtodate(dberet.text)then', com certeza não teria havido o bug no sistema, pois da maneira correta ela compara se a data da devolução é menor que a data da retirada e caso for verdade, mostra a mensagem e não permite a gravação da locação.

4.5. Registro dos erros

Com a finalidade de registrar os erros e enviá-los ao programador, as empresas de software dispõe de várias maneiras possíveis. Normalmente isso é feito através de algum gerenciador de processos que é responsável por controlar o trabalho da equipe de desenvolvimento. Segue abaixo um modelo que via de regra não foge do padrão, seja ele automatizado ou até manual.

RJS DESENVOLVIMENTO DE SISTEMAS S/C LTDA
Av. Nossa Senhora de Fátima, 1500 – Jd. N.S.Fátima
Americana-SP
Sistema: Discover
Cliente: Discover Locadora ME
Responsável pelos testes: Reginaldo José de Souza
Número de Horas do teste: 15H
Responsável pelo desenvolvimento: Fulano de Tal
Número de horas de desenvolvimento: 30H
Histórico de desenvolvimento:
Desenvolver uma rotina de locação de filmes.....
Histórico dos Erros:
1) Quando a data da devolução é maior que a data da retirada,
o sistema está apresentando uma mensagem que 'data da
devolução menor que a data da retirada'
2) Quando a data da retirada é maior que a data da devolução,
o sistema não está acusando. Neste caso não deverá permitir
gravar e apresentar a mensagem citada acima.

Conclusão

Testar (e bem !!!) o sistema antes de liberá-lo para o cliente é hoje em dia uma tarefa das mais primordiais que existem. Simplesmente colocar no mercado de forma rápida para suprir uma necessidade emergencial não é mais a solução. Custa muito mais caro para a empresa liberar uma release de um software do que investir o que realmente é necessário em equipe e tempo para se testar. Não é somente custo de mão-de-obra e tempo necessário para 'consertar' o software que deve ser levado em conta quando se fala de prejuízo. Deve-se ter em mente a credibilidade que a empresa que desenvolveu o sistema está perdendo junto ao mercado consumidor.

Considerando esta avidez por qualidade, conclui-se então que testar software é hoje uma obrigatoriedade no mundo computacional, pois assim proporciona produtos de melhor qualidade.

Visto também que dispomos de ferramentas e métodos, podemos analisar está muito mais fácil trabalhar e conseguir fazer um bom trabalho.

Como consideração final e alguma experiência na área, posso afirmar: "Vale a pena investirmos no trabalho de teste, pois não a nada mais compensador que um cliente satisfeito com o produto que adquiriu e que você 'testador' é o responsável final por isso."

Referências bibliográficas

[1] INTHURN, Cândida (2001), Qualidade e Teste de Software. Visual Books

[2] MOLINARI, Leornado (2003), Testes de software – Produzindo sistemas melhores e mais confiáveis. Editora Érica

[3] REZENDE, Alcides (1997), Engenharia de Software Empresarial. Brasport

Sites utilizados na pesquisa

http://www.inf.pucrs.br/~flavio/teste

Acessado em 15/03/2004

http://www.psphome.hpg.ig.com.br

Acessado em 02/04/2004

http://www.pr.gov.br/batebyte/edicoes/2003/bb136/testes.shtml

Acessado em 15/04/2004

http://sapp.telepac.pt/Imaf/ai/tabelasdecisao.htm

Acessado em 18/04/2004