Teste de Software em Aplicações de Banco de Dados Relacional

Klausner Vieira Gonçalves

Trabalho Final de Mestrado Profissional

Instituto de Computação
Universidade Estadual de Campinas

Teste de Software em Aplicações de Banco de Dados Relacional

Klausner Vieira Gonçalves

Abril de 2003

Banca Examinadora:

- Profa. Dra. Eliane Martins (Orientadora)
  Instituto de Computação - UNICAMP
- Prof. Dr. Mario Jino
  Faculdade de Engenharia Elétrica e Computação - UNICAMP
- Prof. Dr. Célio Cardoso Guimarães
  Instituto de Computação - UNICAMP
- Prof. Dr. Geovane Cayres Magalhães (Suplente)
  Instituto de Computação - UNICAMP

Teste de Software em Aplicações de Banco de Dados Relacional

Este exemplar corresponde à redação final do Trabalho Final devidamente corrigida e defendida por Klausner Vieira Gonçalves e aprovada pela Banca Examinadora.

Campinas, 02 de Abril de 2003

Eliane Martins

Profª. Drª. Eliane Martins (Orientadora)

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 Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Gonçalves, Klausner Vieira

G586t Teste de software em aplicações de banco de dados relacional / Klausner Vieira Gonçalves -- Campinas, [S.P. :s.n.], 2003.

Orientadora: Eliane Martins

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

1. Software – Testes. 2. Engenharia de software. 3. Banco de dados relacional. I. Martins, Eliane. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 02 de abril de 2003, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Mario Jino
FEEC - UNICAMP

Prof. Dr. Célio C. Guimarães
IC - UNICAMP

Profª. Drª. Eliane Martins
IC - UNICAMP

Resumo

A evolução do uso de sistemas computacionais nas atividades humanas tem exigido cada vez mais um alto nível de qualidade dos *softwares*. Esta busca para garantir a qualidade do *software* torna o processo de desenvolvimento de *software* mais difícil e complexo. Na tentativa de reduzir os custos e aumentar a qualidade da atividade de teste, várias técnicas e critérios vêm sendo propostos, onde a diferença entre essas técnicas está na origem da informação que é utilizada para avaliar ou construir conjuntos de casos de teste, sendo que cada técnica possui uma variedade de critérios para esse fim.

Com o aumento expressivo de aplicações que utilizam Banco de Dados Relacional, torna-se necessária a adequação de técnicas e critérios de Teste de *Software* para os testes destas aplicações.

O presente trabalho visa apresentar técnicas de Teste de *Software* em Aplicações de Banco de Dados Relacional através da utilização de técnicas de Teste Estrutural (Caixa Branca) e Teste Funcional (Caixa Preta). A partir do estudo destas técnicas é definida uma estratégia de teste que cobre Teste Estrutural baseado em critérios de cobertura de Fluxo de Controle e Complexidade Ciclomática e Testes Funcionais baseados em Particionamento de Equivalência e Análise de Valor Limite. Para apoiar esta estratégia é desenvolvida também uma ferramenta de auxílio para execução dos testes (*STest for Delphi*).

Abstract

The evolution of the use of computational systems in the human activities beings has demanded each time plus one high level of quality of softwares. This search to guarantee the quality of software becomes the process of software development of more difficult and complex. In the attempt to reduce the costs and to increase the quality of the activity of test, several techniques and criteria come being proposed, where the difference between these techniques is in the origin of the information that is used to evaluate or to construct sets of test cases, being that each technique possesss a variety of criteria for this end.

With the expressive increase of applications that use Relational Database, one becomes necessary the adequacy of techniques and criteria of Test of Software for the tests of these applications.

The present work aims at to present techniques of Test of Software in Applications of Relational Database through the use of techniques of Structural Test (White Box) and Functional Test (Black Box). From the study of these techniques is defined a test strategy that has covered established Structural Test in criteria of covering of Control Flow and Cyclomatic Complexity and Funcional Tests based in Equivalence Partitioning and Boundary Value Analysis. To support this strategy, it is also developed, a tool of aid for execution of the tests (STest for Delphi).

Dedico aos meus pais, meus grandes amigos, companheiros e incentivadores, sem os quais este trabalho não teria sido realizado.

Agradecimentos

A professora Eliane Martins, por toda orientação, apoio e atenção fornecidos durante o desenvolvimento deste trabalho.

Aos colegas do Instituto de Computação.

A Deus, pela oportunidade de realização de mais este importante passo em minha vida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste capítulo são apresentados o contexto no qual este trabalho está inserido, os fatores que motivam a sua realização, os objetivos a serem atingidos durante o seu desenvolvimento, finalizando com a organização desta monografia.

1.1. Contexto

A crescente evolução do uso de sistemas computacionais nas atividades humanas tem exigido cada vez mais um alto nível de qualidade dos *softwares*, a fim de proporcionar aos usuários uma maior confiabilidade e segurança na sua utilização.

Esta busca para garantir a qualidade do *software* torna o processo de desenvolvimento de *software* mais difícil e complexo. Porém, mesmo com as atividades de garantia de qualidade conduzidas durante todo o processo de desenvolvimento de *software* (desde a análise de requisitos até a fase de manutenção do *software*), constata-se que muitos erros permanecem no *software* finalizado. De acordo com PRESSMAN (1995) e MYERS (1979), uma destas atividades é o teste de *software* que pode exigir aproximadamente 50% do esforço total do projeto. Desse modo, segundo SOUZA (1996) é necessário determinar quais os casos de testes que serão utilizados a fim de que a maioria dos erros existentes possam ser detectados e que o número de casos de teste utilizados não seja tão grande a ponto de ser impraticável.

Na tentativa de reduzir os custos e aumentar a qualidade da atividade de teste, várias técnicas e critérios vêm sendo propostos, onde a diferença entre essas técnicas está na origem da informação que

é utilizada para avaliar ou construir conjuntos de casos de teste, sendo que cada técnica possui uma variedade de critérios para esse fim. Na técnica de Teste Funcional ou Caixa Preta os requisitos de teste são obtidos a partir da especificação. Já na técnica de Teste Estrutural ou Caixa Branca derivam-se os requisitos a partir dos aspectos de implementação do *software*.

Com o aumento expressivo de aplicações que utilizam Banco de Dados Relacional, torna-se necessária a adequação de técnicas e critérios de teste de *software* para os testes destas aplicações.

Para que a atividade de teste caminhe rumo à obtenção do sucesso esperado e livre de erros que possam ser introduzidos com a intervenção humana, o desenvolvimento de ferramentas automatizadas torna-se fundamental.

1.2. Motivação

Segundo SPOTO (2000), apesar da existência de vários métodos e técnicas de teste de *software*, estes não cobrem todas as áreas de desenvolvimento de *software*. A necessidade da utilização de testes de *software* em aplicações de Banco de Dados Relacional torna motivante os esforços de pesquisa para o aprimoramento destas técnicas.

Para tanto, a idéia de utilizar técnicas de testes de *software* direcionadas para Aplicações de Banco de Dados Relacional torna-se viável devido ao grande crescimento de Sistemas de Informação que utilizam Banco de Dados Relacional.

Os esforços para adequação de técnicas e critérios de teste de *software* para os testes em Aplicações de Banco de Dados Relacional tornam desafiador o desenvolvimento desta pesquisa, como forma de apresentar as técnicas e critérios para teste destas aplicações, além de ferramentas automatizadas para auxílio na aplicação destes tipos de testes.

SPOTO (2000) apresenta uma metodologia de Teste Estrutural para Aplicações de Banco de Dados Relacional e uma adaptação das ferramentas POKE-TOOL, que dá apoio ao teste de unidade (CHAIM et al. (1997)), e VIEWGRAPH, que permite visualizar as etapas parciais do teste de unidade a partir das informações geradas pela POKE-TOOL (VILELA et al. (1997)). Esta ferramenta de apoio efetua Testes Estruturais em *software* com abordagem para Aplicações de Banco de Dados Relacional desenvolvidas com Linguagem C.

Entretanto, a estratégia que será proposta poderá ser aplicada em qualquer *software*, porém a ferramenta que será desenvolvida terá como base a utilização de Testes Estruturais em Aplicações de

Banco de Dados Relacional desenvolvidas com a Linguagem de Programação *Borland Delphi*, e além disso, a possibilidade de Testes Funcionais destas aplicações.

1.3. Objetivo

O objetivo deste trabalho é investigar critérios de teste existentes para Aplicações de Banco de Dados Relacional que utilizam *SQL* (*Structure Query Language*), definindo eventualmente uma estratégia que combine vários critérios. E, também, apresentar uma ferramenta para Teste de *Software* em Aplicações de Banco de Dados Relacional desenvolvida utilizando a Linguagem de Programação *Borland Delphi*, que apoie a estratégia adotada.

1.4. Organização do Trabalho

Neste capítulo foi apresentado o contexto no qual este trabalho se insere, as motivações para a sua realização e os objetivos a serem alcançados. O capítulo 2 traz em seu conteúdo a definição de Aplicações de Banco de Dados Relacional e uma breve exposição dos conceitos sobre Banco de Dados. No capítulo 3 são apresentados as fases da atividade de teste e os principais métodos e critérios de teste existentes. Já no capítulo 4 serão detalhados estes métodos e critérios de testes, baseando-se em Aplicações de Banco de Dados Relacional. O capítulo 5 apresenta a estratégia adotada para Teste de *Software* em Aplicação de Banco de Dados Relacional. No capítulo 6 é apresentada a ferramenta desenvolvida para os testes. O trabalho é finalizado com a conclusão, as referências bibliográficas e os apêndices.

Capítulo 2

Aplicação de Banco de Dados Relacional

Uma Aplicação de Banco de Dados Relacional é um conjunto de módulos de programas, onde cada um destes módulos pode ser escrito em uma linguagem de programação, onde conectada a um Sistema Gerenciador de Banco de Dados (SGBD) baseado no Modelo Relacional, utiliza a Linguagem de Consulta Estruturada (*SQL*) para manipulação do banco de dados.

2.1. Banco de Dados

De acordo com DATE (2000), um banco de dados é uma coleção de dados persistentes utilizada pelos sistemas de aplicação de uma determinada organização.

2.2. Sistema de Banco de Dados

Segundo SILBERSCHATZ et al. (1999), um Sistema de Banco de Dados é composto por:

- Usuários e sua Interação: Usuários Navegantes (usuários comuns que interagem com o sistema chamando um dos programas aplicativos já escritos), Programadores de Aplicações (profissionais que interagem com o sistema por meio de chamadas *DML*), Usuários Sofisticados (interagem com o sistema formulando suas solicitações por meio de linguagens de consultas) e Usuários Especialistas (escrevem aplicações especializadas, diferentes das tradicionais);

- Sistema Gerenciador de Banco de Dados (SGBD): é uma coleção de dados inter-relacionados e uma coleção de programas para acesso a estes dados, proporcionando assim, um ambiente conveniente e eficiente para armazenamento e recuperação de informações. É dividido em componentes de Processamento de Consultas (Compilador *DML*, Pré-Compilador para comandos *DML*, Interpretador *DDL*, Componentes para execução de consultas e programas de aplicações em código objeto) e Gerenciador de Memória (Gerenciador de Transações, Gerenciador de *Buffer* e Gerenciador de Arquivos);
- Armazenamento em Disco: estruturas de dados que são necessárias como parte da implementação física do sistema (Arquivo de Dados, Dicionário de Dados, Índices e Estatísticas de Dados).

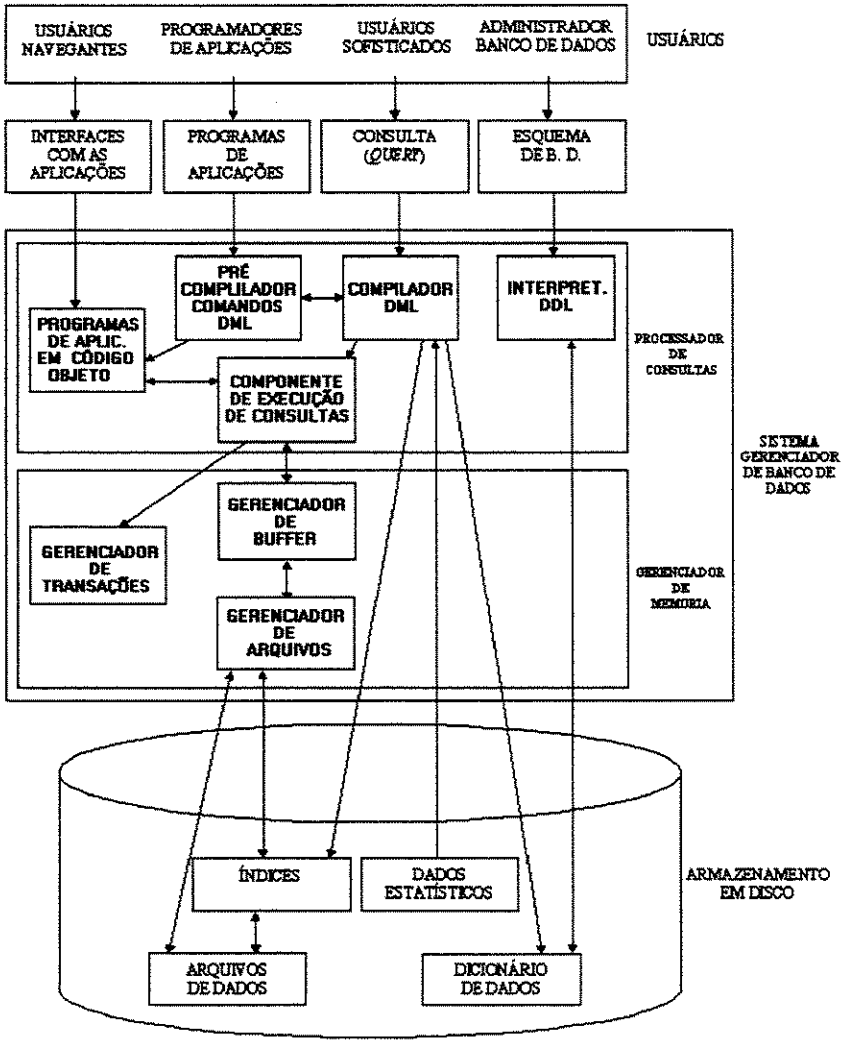


Figura 1: Sistema de Banco de Dados (SILBERSCHATZ et al. (1999))

2.3. Modelo Entidade-Relacionamento

Modelos semânticos ou conceituais se prestam à etapa de projeto conceitual, pois nesta etapa a base de dados é concebida de uma forma bastante preliminar pelos futuros usuários (ainda não está associada a nenhum SGBD ou plataforma). Cada construção tem uma semântica bem definida, pois possuem notações diagramáticas de fácil entendimento. O Modelo Entidade-Relacionamento é considerado um modelo semântico.

Segundo SILBERSCHATZ et al. (1999), o Modelo Entidade-Relacionamento tem por base a percepção de que o mundo real é formado por um conjunto de objetos chamado entidades e pelos conjuntos dos relacionamentos entre estes objetos. A extensão deste modelo permitiu a inclusão de conceitos, tais como classes, herança e regras, entre outros.

Suas construções básicas são entidades (retângulos), relacionamentos (losangos) e atributos (ovais), que oferecem uma interpretação melhor que o modelo relacional.

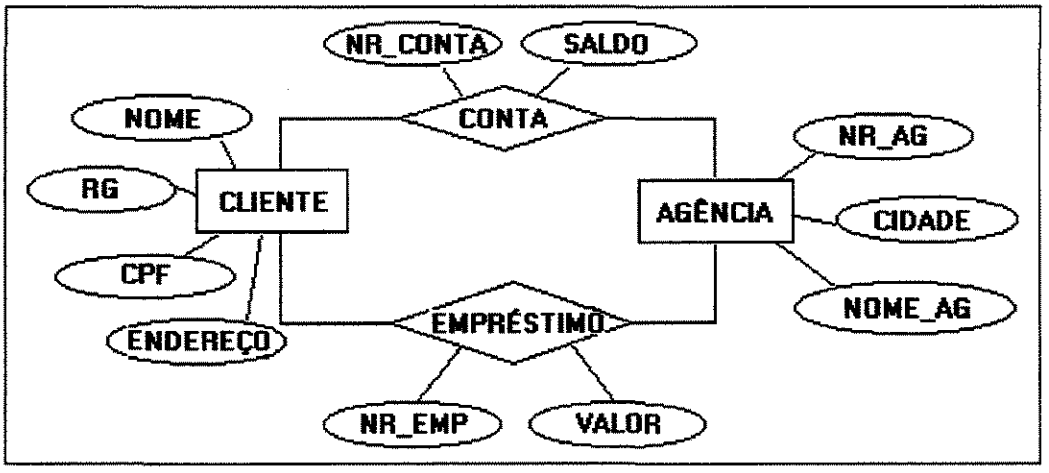


Figura 2: Exemplo de Diagrama Entidade-Relacionamento

2.4. Modelo Relacional

O Modelo Relacional está formalmente baseado em Teoria dos Conjuntos e Álgebra Relacional. A organização dos dados ocorre na forma de relações que são tabelas de tuplas (linhas), constituídas por valores especificados de acordo com um conjunto de atributos.As operações existentes neste modelo formam uma álgebra, que produzem relações (seleção, projeção, *join*, união, etc.).

O modelo tornou possível que vários aspectos da área de Banco de Dados se desenvolvessem (otimização de consultas, independência de dados, transações, redundância, fragmentação, etc.)

Porém, existem algumas restrições quanto ao modelo relacional:

- A natureza bi-dimensional indica que todos os elementos em uma base relacional são tabelas;
- Como tudo é tabela, os ponteiros para outras tabelas não são distinguíveis como ponteiros (se perdem na relação).

Segundo DATE (2000), uma razão pela qual os sistemas de banco de dados relacionais se tornaram tão dominantes (tanto acadêmico quanto comercial) é que eles admitem a interpretação precedente de dados e banco de dados de forma direta, onde:

- Os dados são representados por meio de linhas em tabelas, e essas linhas podem ser interpretadas diretamente como proposições verdadeiras;
- São fornecidos operadores para operações sobre linhas em tabelas, e esses operadores admitem de forma direta o processo de dedução de proposições verdadeiras adicionais a partir das proposições dadas.

O exemplo a seguir apresenta um esquema de banco de dados relacional, onde os atributos comuns geram as relações entre tabelas.

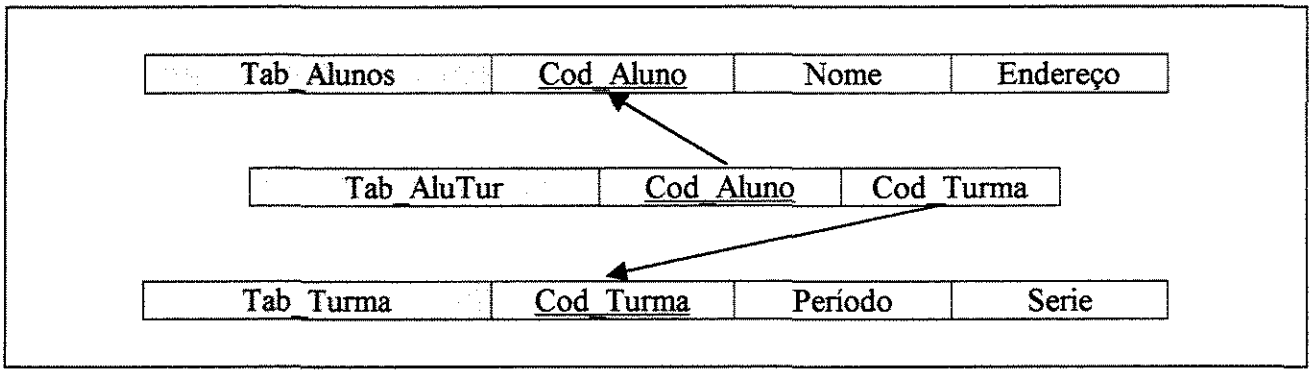


Figura 3: Exemplo de Modelagem Relacional

2.5. Linguagem de Consulta Estruturada (*SQL*)

Desenvolvida no laboratório de pesquisa da *IBM* em *San Jose* no começo dos anos 70, a Linguagem *SQL* usa uma combinação de construtores em álgebra e cálculo relacional (modelo relacional), permitindo ao usuário o armazenamento e recuperação de dados do Banco de Dados.

De acordo com SILBERSCHATZ et al. (1999), *SQL* além de possuir recursos de consulta ao banco de dados, também apresenta outros recursos como definição da estrutura de dados para modificação de dados no banco de dados e para especificações de restrições de segurança.

A linguagem *SQL* tem diversas partes apresentar a seguir.

2.5.1. Linguagem de Definição de Dados (*DDL*)

Os comandos *DDL* para são utilizados para definir os esquemas de relações e índices.

a) CREATE TABLE: permite criar uma tabela, indicando os nomes e tipos de dados das colunas da tabela, além da chave primária.

Exemplo:
```
CREATE TABLE Alunos
        ( Codigo Numeric (6),
          Nome Char(40),
          PRIMARY KEY (Codigo) )
```

b) ALTER TABLE: altera a estrutura de uma tabela criada.

Exemplo: ALTER TABLE Alunos MODIFY Nome (CHAR(50) NOT NULL)

c) DROP TABLE: permite excluir uma tabela.

Exemplo: DROP TABLE Alunos

d) CREATE INDEX: cria índices para as tabelas.

Exemplo:
```
CREATE DESC INDEX IndNomeAlunos
        ON Alunos (Nome)
```

e) DROP INDEX: remove um índice criado.

Exemplo: DROP INDEX IndNomeAlunos

2.5.2. Linguagem de Manipulação de Dados (*DML*)

Os comandos DML são utilizados para operações de manipulação de dados.

a) SELECT: seleciona os dados de uma ou mais tabelas.
   Exemplo: SELECT * FROM Alunos WHERE Codigo = "101102"

b) INSERT: insere um registro em uma tabela.
   Exemplo: INSERT INTO Alunos (Codigo, Nome) VALUES ("103221", "CARLA")

c) UPDATE: atualiza um registro ou mais registros de uma tabela.
   Exemplo: UPDATE Alunos SET Nome = "CARLA SILVA" WHERE Codigo = "103221"

d) DELETE: exclui um registro ou mais registro de uma tabela.
   Exemplo: DELETE FROM Alunos WHERE Codigo = "103221"

2.5.3. *SQL* Embutida (*Embedded SQL*)

A SQL fornece uma poderosa linguagem de consulta declarativa que permite escrever consultas de forma mais simples do que em linguagens de programação de uso geral (linguagens hospedeiras). Entretanto o acesso a um banco de dados é necessário para as linguagens de programação de uso geral (*PL/I*, *C*, *COBOL* e *FORTRAN*), pois nem todas as consultas podem ser geradas apenas com *SQL*. Para isso foi criado então, a *SQL* Embutida (*Embedded DML*) que incorpora comandos *SQL* em linguagens hospedeiras.

Exemplo: EXEC SQL
SELECT Nome FROM Alunos WHERE CODIGO = "102565"
END EXEC

2.5.4. Visões

Definição de Visões: comandos para definição de visões.

Para definir uma visão, precisamos dar um nome a ela e escrever a consulta que criará a visão.

Exemplo: CREATE VIEW Doc_Alunos AS

SELECT Nome, RG, CPF FROM Alunos

2.5.5. Transações

Segundo SILBERSCHATZ et al. (1999), uma Transação é uma unidade de execução de programa que acessa e, possivelmente, atualiza vários itens. Um Transação, geralmente, é o resultado da execução de um programa de usuário escrito em uma linguagem de manipulação de dados de alto nível ou em uma linguagem de programação (*SQL*, *COBOL*, *C* ou *Pascal*) e é delimitada por declarações: *Begin Transaction* e *End Transaction*.

Exemplo: BEGIN TRANSACTION;

UPDATE Alunos SET Nome = "FLAVIA" WHERE Codigo = "123212"

IF ocorreu algum erro THEN ROLLBACK;

END IF;

COMMIT;

RETURN;

END TRANSACTION;

Capítulo 3

A Atividade de Teste

O processo de desenvolvimento de *software* engloba várias atividades, onde apesar dos métodos, técnicas e ferramentas empregados, erros ainda podem ocorrer. Com isso, a atividade de teste é fundamental para a identificação e eliminação de erros que persistem, representando a última revisão da especificação, projeto e codificação (PRESSMAN (1995)).

Segundo MYERS (1979) os principais princípios para a atividade de teste são:

- A atividade de teste é o processo de executar um programa com a intenção de descobrir um erro;
- Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto;
- Um teste bem sucedido é aquele que revela um erro ainda não descoberto.

De acordo com PRESSMAN (1995), a atividade de teste pode ser conduzida em quatro fases:

- Teste de Unidade: concentra-se no esforço de verificação da menor unidade de projeto de *software* chamado de unidade ou módulo.
- Teste de Integração: é uma técnica sistemática para a construção da estrutura de programa, realizando-se, ao mesmo tempo, testes para descobrir erros associados a *interfaces*. O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto.

- Teste de Validação: ao término da atividade de teste de integração, o *software* está completamente montado como um pacote, erros de *interface* foram descobertos e corrigidos e uma série final de testes de *software* conhecida como os testes de validação podem ser iniciados. A validação, portanto, é definida como bem-sucedida quando o *software* funciona de uma maneira razoavelmente esperada pelo cliente.
- Teste de Sistema: é uma série de diferentes testes, cujo principal propósito é pôr completamente à prova o sistema baseado em computador, onde todo o trabalho deve verificar se todos os elementos do sistema foram adequadamente integrados e realizam as funções atribuídas.

Um dos pontos críticos da atividade de teste é a escolha dos casos de teste a serem aplicados. O programa, para ser demonstrado correto, deveria ser exercitado com todos os valores possíveis do domínio de entrada. Entretanto, sabe-se que o teste exaustivo é impraticável devido a restrições de tempo e custo para realizá-lo, sendo necessário determinar um conjunto de casos de teste que seja eficaz em encontrar erros cuja cardinalidade seja reduzida (MYERS (1979)). Com isso, várias técnicas e critérios de teste têm sido elaborados visando fornecer uma maneira sistemática e rigorosa para selecionar um subconjunto do domínio de entrada e assim ser eficaz para revelar os erros existentes.

O sucesso da atividade de teste conduzirá à detecção de erros no *software* possibilitando, assim, a correção dos mesmos e, conseqüentemente, proporcionando uma indicação de maior confiabilidade do *software* (qualidade do *software*).

3.1. Principais Métodos de Teste de *Software*

Na tentativa de reduzir os custos associados ao teste, é fundamental a aplicação de técnicas que indiquem como testar o *software* e de critérios que respondam quando parar os testes, de forma que essa atividade possa ser conduzida de modo planejado e sistemático (DEMILLO (1980)).

Segundo PRESSMAN (1995), nenhuma das técnicas de teste é completa, no sentido de que nenhumas delas é, em geral, suficiente para garantir a qualidade da atividade de teste. Essas diferentes técnicas são complementares e devem ser aplicadas em conjunto para assegurar um teste de boa qualidade.

Segundo BEIZER (1995), três estratégias podem ser utilizadas para se realizar teste de *software*:

- Teste Estrutural: Baseia-se na estrutura do objeto testado. Requer, portanto, acesso completo ao código-fonte do programa para ser realizado. É também conhecido por teste de caixa branca (*White-Box Test*);
- Teste Funcional: Checa se um programa está de acordo com a sua especificação, independentemente do código-fonte. Teste funcional é também conhecido como teste de caixa preta (*Black-Box Test*) ou teste comportamental;
- Teste Híbrido: Combina as duas estratégias de teste anteriores.

3.1.1. Teste Estrutural

A técnica de Teste Estrututal, também conhecida como Teste de Caixa Branca, usa a estrutura de controle do projeto procedimental para derivar casos de teste, portanto, são técnicas de teste baseia-se na implementação do *software*.

De acordo com PETERS & PEDRYCZ (2001), no Teste Estrutural o interesse está no desenvolvimento de casos de teste baseados na estrutura (lógica interna) do código em teste. Existem diversas classes de teste que dependem de quão completo e consumidor de tempo o processo de teste deva ser.

Este teste se baseia no exame detalhado das rotinas procedimentais do programa, onde os caminhos lógicos são testados e fornecem casos de teste que põem à prova conjuntos específicos de condições e/ou laços.

De acordo com PRESSMAN (1995), o Teste de Caixa Branca permite derivar os casos de teste que:

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

Como o Teste Estrutural é baseado na análise do código-fonte, as técnicas evidentemente dependem da linguagem de programação utilizada.

Os critérios de Teste Estrutural mais conhecidos são os baseados no Fluxo de Controle e no Fluxo de Dados apresentados a seguir.

3.1.1.1. Critérios baseados no Fluxo de Controle

Critérios baseados em Fluxo de Controle utilizam apenas características de controle de execução do programa, como comandos ou desvios, para determinar quais estruturas estão sendo utilizadas e quais podem ser eliminadas.

Para facilitar a análise do Fluxo de Controle, uma ferramenta para geração de Grafos de Fluxo de Controle pode ser utilizada. Cada programa tem uma seqüência finita de comandos da linguagem de programação. Estes blocos de comandos são representados por nós arredondados no Grafo de Fluxo de Controle. De acordo com HUANG (1975), um bloco de comando representa uma seqüência de um ou mais comandos tendo a característica de que, sempre que o comando inicial for executado, todos os demais comandos do bloco também serão.Segundo PRESSMAN (1995), a notação utilizada para representar Grafos de Fluxo de Controle é dada na Figura 4. Nesta notação é definido que cada círculo, denominado nó, representa uma ou mais instruções do programa e cada arco (seta), denominada ramo, representa um fluxo de controle.

Os critérios de cobertura de Fluxo de Controle mais conhecidos são:

- Todos-Nós: Todos os nós (ou comandos) devem ser executados pelo menos uma vez, onde dessa forma, a execução do programa deve abranger cada vértice do Grafo de Fluxo de Controle pelo menos uma vez;
- Todos-Arcos: Todos os arcos (ou decisões) devem ser executados pelo menos uma vez, ou seja, exige que a execução do programa passe pelo menos uma vez em cada aresta do Grafo de Fluxo de Controle;
- Todos-Caminhos: Todos os caminhos possíveis em um Grafo de Fluxo de Controle são executados pelo menos uma vez, requerendo assim, que todos os caminhos possíveis do programa sejam executados (Impraticável, pois o número de caminhos é muito grande).Segundo FRANKL (1987), um caminho é executável se existe um conjunto de

valores que possa ser atribuído às variáveis de entrada do programa e que causa a execução desse caminho.

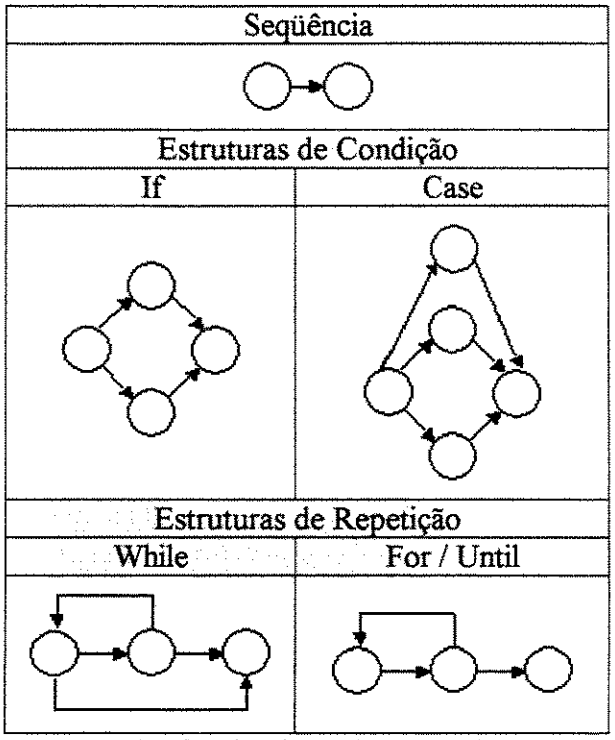
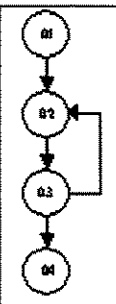


Figura 4: Notação para Grafos de Fluxo de Controle (PRESSMAN (1995))

```
(01) procedure T_CalcFatorial.BotaoCalcular(Sender: TObject);
(01) var vConta, vFat, vCalc: Integer;
(01) begin
(01)    vFat := StrToInt(txtFat.Text);
(01)    vCalc := 0;

(02)    for vConta := 1 to vFat - 1 do
(03)        vCalc := vCalc + (vFat * vConta);

(04)    showmessage('O fatorial de ' + IntToStr(vFat) + ' é ' + IntToStr(vCalc));
(04) end;
```



Todos-Nós: {1,2,3,4}	Todos-Caminhos: {(1,2,3,4) (1,2,3,2,3,4) . . . (1,2,3,2,3,...,2,3,4)}
Todos-Arcos: (1,2,3,2,3,4)	

Figura 5: Critérios de Cobertura de Fluxo de Controle

Na Figura 5 é apresentado um exemplo de Critérios de Cobertura de Grafo de Fluxo de Controle, no qual se podem verificar os critérios: Todos-Nós, onde cada nó (círculo) deve ser executado pelo menos uma vez ({1,2,3,4}); Todos-Arcos, que requer a cada execução de cada arco pelo menos uma vez (1,2,3,2,3,4); e Todos-Caminhos, onde cada caminho deve ser executado pelo

menos uma vez (1,2,3,2,3,4). Neste último, se pode confirmar a citação anterior de impraticável, pois existe um número de caminhos muito grande.

3.1.1.2. Critérios baseados na Complexidade Ciclomática

A Complexidade Ciclomática é uma métrica de *software* que utiliza informações sobre a complexidade do programa para determinar os requisitos de teste, permitindo assim, a medida quantitativa da complexidade lógica de um programa.

THOMAS MCCABE (1976), propôs essa métrica baseada numa representação do Fluxo de Controle de um programa, onde é definido o número de caminhos independentes em um programa, que nos oferece um limite máximo para o número de testes que devem ser realizados para garantir que cada instrução seja executada pelo menos uma vez. Na prática é uma contagem do número de condições de teste em um programa.

Segundo PRESSMAN (1995), a Complexidade Ciclomática tem suas bases na teoria dos grafos e pode ser calculada de três formas (ver Figura 6):

a) Número de regiões do grafo de fluxo;
b) Número de ramos (R) e número de nós (N) do grafo de fluxo: V(G)=R–N+2;
c) Número de nós predicados (P) do grafo de fluxo: V(G)=P+1;

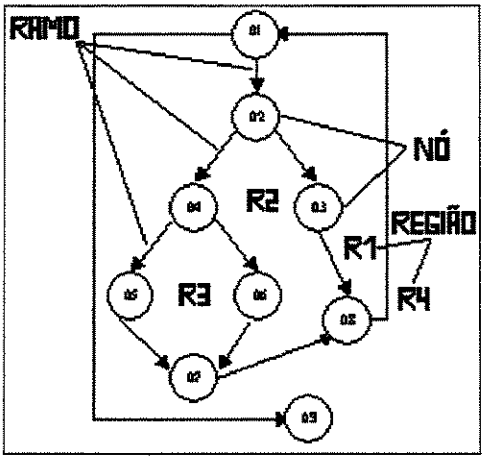


Figura 6: Grafo de Fluxo indicando RAMO, NÓ e REGIÃO (PRESSMAN (1995))

Esta métrica pode ser aplicada na técnica de Teste de Caminhos Básicos proposta por THOMAS MCCABE (1976) que pode ser dividida em 4 fases:

1 - A partir do Código-Fonte do Programa a ser testado criar Grafo de Fluxo de Controle

```
(01)  procedure T_Acesso.Logar(Sender: TObject);
(01)  begin

(02)     if vEscola.Text <> 'G' then
(03)        showmessage('Acesso ao BD da Graduação ... ')
(04)     else
(04)        showmessage('Acesso ao BD da Pós-Graduação ... ');

(05)     Menu.show;
(06)  end;
```

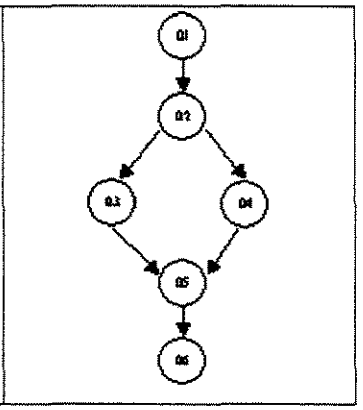


Figura 7: Grafo de Fluxo de Controle

2 - Calcular a Complexidade Ciclomática do Grafo de Fluxo de Controle resultante

Número de Ramos (R) = **6**
Número de Nós (N) = **6**
V(G) = R – N + 2
V(G) = 6 – 6 + 2
V(G) = **2 (Complexidade Ciclomática)**

3 - Determinar um Conjunto Básico de Caminhos Linearmente Independentes

Caminho 1: 01-02-03-05-06
Caminho 2: 01-02-04-05-06

4 - Preparar os Casos de Teste que forcem a execução de cada Caminho no Conjunto Básico

Casos de Teste do Caminho 1: vEscola = entrada válida, onde vEscola = "G" Resultados Esperados: - Mensagem "Acesso ao BD da Graduação" - Abrir o Formulário de Menu
Casos de Teste do Caminho 2: vEscola = entrada válida, onde vEscola <> "G" Resultados Esperados: - Mensagem "Acesso ao BD da Pós-Graduação" - Abrir o Formulário de Menu

Tabela 1: Casos de Teste do Teste de Caminhos Básicos

3.1.1.3. Critérios baseados no Fluxo de Dados

De acordo com PRESSMAN (1995), o método de Teste de Fluxo de Dados seleciona caminhos de teste de um programa de acordo com as localizações das Definições e Usos de variáveis no programa. A Definição de uma variável refere-se a uma instrução em que um valor é atribuído a uma variável. Já o Uso ocorre quando os valores das variáveis são utilizados.

RAPPS e WEYUKER (1982) definem uso de uma variável como sendo:

- c-Uso (uso computacional): quando a variável é usada na avaliação de uma expressão ou em um comando de saída (o uso está associado ao Nó);
- p-Uso (uso predicativo): quando a variável ocorre em um predicado e, portanto, afeta o fluxo de controle do programa (este uso está associado ao Arco).

Segundo NTAFOS (1984), o uso de uma variável ocorre sempre que existir recuperação do valor em posição de memória associada à variável.

A associação entre a Definição e Uso da variável é conhecida como associação DU e, para que ela se caracterize, o caminho de uma Definição de variável em relação ao seu Uso não pode conter nenhuma redefinição da variável.

A notação utilizada para representar o Grafo de Fluxo de Dados é mesma a utilizada para representar Grafo de Fluxo de Controle mostrado na Figura 8.

```
(01)  procedure T_Pesq_Lista.BotaoProcurar(Sender: TObject);
(01)  var vContador, vAchou: Integer;
(01)      vTabela: Array [1..10] of String;
(01)  begin
(01)     vContador := 1;
(01)     vAchou := 0;

(02)     while (vAchou = 0) and (vContador <= 10) do
(02)     begin
(03)        if vTabela [vContador] = txtPesq.Text then
(04)           vAchou := vAchou + 1;
(05)        vContador := vContador + 1;
(05)     end;

(06)     showmessage('O Item procurado foi encontrado '
(06)              + IntToStr(vAchou) + ' vezes ');
(06)  end;
```

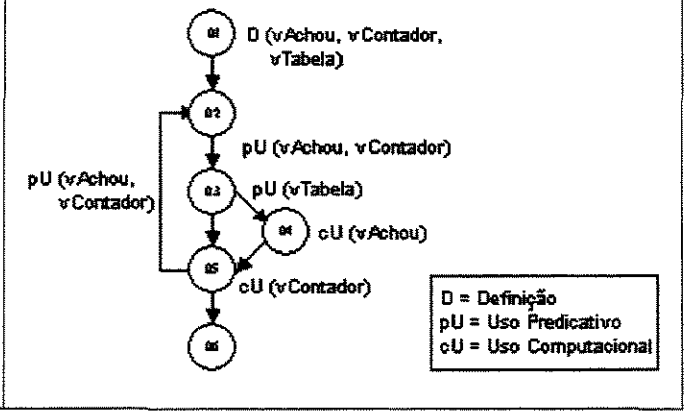


Figura 8: Grafo de Fluxo de Dados

3.1.2. Teste Funcional

Os métodos de Teste Funcional, também conhecidos como Testes de Caixa Preta, concentram-se nos requisitos funcionais do *software*, permitindo somente visualizar o lado externo do *software*, ou seja, os dados de entrada e as respostas produzidas como saída. Portanto, estas técnicas de teste são baseadas na especificação do *software*.

Os Testes Funcionais são realizados nas *interfaces* do *software*, tendo como propósitos, além de descobrir erros, demonstrar que as funções do *software* são operacionais, que a entrada é adequadamente aceita, a saída é corretamente produzida e que a integridade das informações externas é mantida.

Segundo CHANG e RICHARDSON (1999), o Teste Funcional é freqüentemente mais fácil de compreender e realizar do que as técnicas de teste tradicionais baseadas em código, porque o Teste Funcional baseia-se em uma descrição *Black-Box* da funcionalidade do *software*, enquanto que as técnicas de Teste Estrutural exigem um detalhado conhecimento da implementação.

Segundo PRESSMAN (1995), o teste de caixa preta não é uma alternativa para as técnicas de Caixa Branca; ao contrário, é uma abordagem complementar que procura revelar uma classe de erros diferente do teste estrutural:

- Funções Incorretas ou Ausentes;
- Erros de *Interface*;
- Erros nas Estruturas de Dados ou no Acesso a Bancos de Dados Externos;
- Erros de Desempenho;
- Erros de Inicialização e Término.

Atualmete, na literatura, são encontradas diversas técnicas de Teste Funcional, porém nesta seção serão apresentadas algumas destas técnicas, tais como: Particionamento de Equivalência e Análise de Valor Limite.

3.1.2.1. Particionamento de Equivalência

De acordo com PAULA FILHO (2001), o Particionamento de Equivalência é um método que divide o domínio de entrada em categorias de dados. Cada categoria revela uma classe de erros, permitindo que casos de teste na mesma categoria sejam eliminados sem que se prejudique a cobertura dos testes.

Segundo PRESSMAN (1995), as classes de equivalência podem ser definidas de acordo com as seguintes diretrizes:

- Se uma condição de entrada especificar um intervalo, uma classe de equivalência válida e duas classes de equivalência inválidas são definidas;
- Se uma condição de entrada exigir um valor específico, uma classe de equivalência válida e duas classes de equivalência inválidas são definidas;
- Se uma condição de entrada especificar um membro de um conjunto, uma classe de equivalência válida e uma classe de equivalência inválida são definidas;
- Se uma condição de entrada for booleana, uma classe de equivalência válida e uma classe de equivalência inválida são definidas.

Aplicando-se as diretrizes para a derivação de classes de equivalência, os casos de teste para cada item de dados do domínio de entrada poderiam ser desenvolvidos e executados.

O exemplo de Teste de Particionamento de Equivalência mostrado nas Tabelas 2 e 3 é baseado no cadastro de consultas de um consultório médico. Uma vez definidas as Classes de Equivalência, os Casos de Testes são criados.

Classes de Equivalência (V - Classe Válida; I - Classe Inválida)		
Nr_Paciente	Data	Hora
6 dígitos (V) < 6 dígitos (I) > 6 dígitos (I) não número (I)	10 dígitos (V) >= 10/01/2000 (V) < 10/01/2000 (I) vazio (I)	5 dígitos (V) >= 08:00 e < 12:00 (V) >= 14:00 e < 18:00 (V) >= 12:00 e < 14:00 (I) <= 08:00 (I) >= 18:00 (I) vazio (I)

Tabela 2: Exemplo de Classes de Equivalência

Casos de Testes				
S	Nr_Paciente	Data	Hora	Justificativa
1	000526	10/12/2001	08:30	Válido. Cobre o Nr_Paciente, Data e Hora Válidos.
2	00012	15/02/2002	10:00	Válido. Cobre somente uma Classe Inválida (< 6 dígitos).
3	0088901	25/11/1985	06:45	Inválido. Cobre três Classes Inválidas.
4	025565	17/04/2002	20:30	Válido. Cobre somente uma Classe Inválida (>= 18:00).
5	000025	03/02/1990	16:30	Válido. Cobre somente uma Classe Inválida (< 10/01/2000)
6	X01156	20/07/2000	15:00	Válido. Cobre somente uma Classe Inválida (não número).
7	002569	15/01/2003	13:00	Válido. Cobre somente uma Classe Inválida (>= 12:00 e < 14:00).
8	125699		15:00	Válido. Cobre somente uma Classe Inválida (vazio).
9	PA6689	29/08/1995	03:00	Inválido. Cobre três Classes Inválidas.
10	005569	12/04/2001	17:00	Válido. Cobre o Nr_Paciente, Data e Hora Válidos.

Tabela 3: Casos de Testes de Particionamento de Equivalência

3.1.2.2. Análise de Valor Limite

De acordo com PAULA FILHO (2001), a Análise de Valor Limite é uma técnica para seleção de casos de teste que exercitam os limites. O emprego dessa técnica deve ser complementar ao emprego da técnica de Particionamento de Equivalência. Portanto, ao invés de selecionar um elemento aleatório de cada classe de equivalência, selecionam-se os casos de teste nas extremidades de cada classe.

A aplicação desta técnica permite identificar possíveis erros como:

- Uma unidade abaixo do mínimo;
- Uma unidade acima do máximo;
- Arquivo Vazio;
- Arquivo maior ou igual à capacidade máxima de armazenamento;

- Cálculo que pode levar a "*overflow*" / "*underflow*"
- Erro no primeiro / último registro.

De acordo com PRESSMAN (1995), as diretrizes para a Análise de Valor Limite são semelhantes àquelas apresentadas para o Particionamento de Equivalência:

- Se uma condição de entrada especificar um intervalo delimitado pelos valores a e b, os casos de teste devem ser projetados com valores a e b, logo acima e logo abaixo de a e b, respectivamente;
- Se uma condição de entrada especificar uma série de valores, os casos de teste que ponham à prova números máximos e mínimos devem ser desenvolvidos. Valores logo abaixo e logo acima do mínimo e do máximo também devem ser testados;
- Aplique as diretrizes 1 e 2 às condições de saída. Por exemplo, suponhamos que uma tabela de temperatura versus pressão seja exigida como saída de um programa de análise de engenharia. Os casos de teste devem ser projetados para criar um relatório de saída que produza um número máximo (e mínimo) permissível de entradas na tabela;
- Se as estruturas internas de dados do programa tiverem prescrito fronteiras (por exemplo, um *array* tem um limite definido de 100 entradas), certifique-se de projetar um caso de teste para exercitar a estrutura de dados em sua fronteira.

Considere, por exemplo, um programa com 2 variáveis, x1 e x2, com limites claramente determinados:

$a <= x1 <= b$

$c <= x2 <= d$

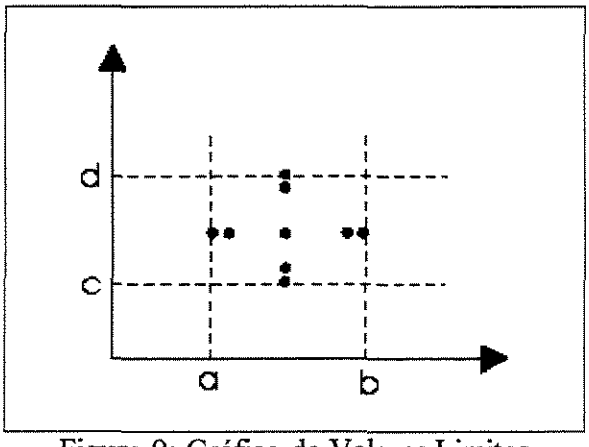


Figura 9: Gráfico de Valores Limites

Como a técnica de Análise de Valor Limite foca os limites dos espaços de entrada de maneira a determinar casos de teste, a idéia básica desta técnica é usar os valores de entrada no seu máximo (MAX), logo abaixo do máximo (MAX-), um valor nominal (NOM), logo acima do mínimo (MIN+) e o valor mínimo (MIN) para gerar casos de teste (Figura 9).

Capítulo 4

Critérios de Teste para Aplicação de Banco de Dados Relacional

Neste capítulo serão apresentadas abordagens encontradas na literatura para Testes de *Software* em Aplicações de Banco de Dados Relacional.

4.1. Teste Estrutural

Para abordagem de Aplicações de Banco de Dados Relacional, os métodos de Teste Estrutural foram estendidos e adaptados. Estas adaptações permitem uma melhor interpretação dos Grafos de Fluxo de Controle, Grafos de Fluxo de Dados e da Complexidade Ciclomática. Permitem, ainda, a definição de novos critérios de cobertura para estes métodos.

4.1.1. Critérios baseados no Fluxo de Controle

Na abordagem de testes para Aplicações de Banco de Dados Relacional, SPOTO (2000) estende a representação do Grafo de Fluxo de Controle, em que os nós arredondados representarão blocos de comandos da linguagem de programação e os nós retangulares representarão comandos *SQL*.

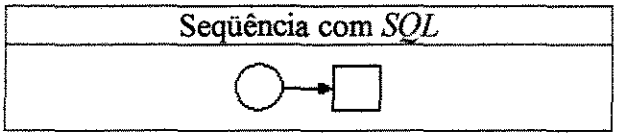


Figura 10: Notação de Grafo de Fluxo de Controle para Aplicação de Banco de Dados Relacional

Portanto, a notação utilizada para representar o Grafo de Fluxo de Controle de Aplicações de Banco de Dados Relacional é aquela mostrada no capítulo 3, na Figura 4, porém com a extensão nós retangulares para indicação de comandos SQL.

Os critérios de cobertura de Fluxo de Controle de Aplicações de Banco de Dados Relacional também são: Todos-Nós, Todos-Arcos e Todos-Caminhos.

O exemplo mostrado na Figura 11 apresenta os Critérios de Cobertura de Fluxo de Controle de Aplicação de Banco de Dados Relacional que tem objetivo de exportar dados dos alunos de uma faculdade.

```
(01)  procedure T_Exp_Alunos.BotaoExportar(Sender: TObject);
(01)  begin
(02)     Q1.Close;
(02)     Q1.SQL.Clear;
(02)     Q1.SQL.Add('Select * From AL_TURMAS, ALUNOS '
(02)              + 'Where AL_TURMAS.CGM = ALUNOS.CGM And '
(02)              + 'AL_TURMAS.Turma = ''' + CT.Text + ''' Order By Nome');
(02)     Q1.Open;

(03)     Q2.Close;
(03)     Q2.SQL.Clear;
(03)     Q2.SQL.Add('Select * From TEMP');
(03)     Q2.Open;

(04)     while not (Q1.EOF) do
(04)     begin
(05)       Q2.Insert;
(05)       Q2.FieldByName('CGM').Value     := Q1.FieldByName('CGM').AsString;
(05)       Q2.FieldByName('Nome').Value    := Q1.FieldByName('Nome').AsString;
(05)       Q2.FieldByName('Turma').Value   := Q1.FieldByName('Turma').AsString;

(06)       if (Q1.FieldByName('Situacao').AsString = 'CANCELADO') then
(07)          Q2.FieldByName('Situacao').Value := 'Aluno Cancelado na Turma'
(08)       else
(08)          Q2.FieldByName('Situacao').Value := 'Aluno Normal';

(09)       Q2.Post;

(10)       Q1.Next;
(11)     end;
(11)  end;
```

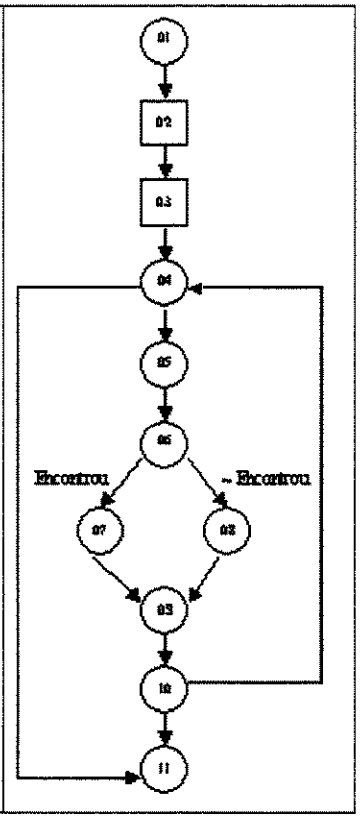


Figura 11: Grafo de Fluxo de Controle de Aplicação de Banco de Dados Relacional

4.1.2. Critérios baseados no Fluxo de Dados

De acordo com SPOTO (2000), em uma Aplicação de Banco de Dados Relacional, as variáveis utilizadas podem ser classificadas em três tipos:

- Variáveis de Programa (P): variáveis definidas e usadas apenas na Linguagem Hospedeira;
- Variáveis Hosts (H): também conhecidas como Variáveis de Ligação, são definidas e usadas em qualquer parte do Módulo de Programa (dentro e fora da *SQL*) e são fundamentais para estabelecer a comunicação entre as duas linguagens;
- Variáveis de Tabela (T): definidas e usadas apenas nos comandos executáveis da *SQL*.

Os comandos *DML* (*SELECT*, *INSERT*, *UPDATE* e *DELETE*) caracterizam a ocorrência de definição da Variável de Tabela.

Além das definições de Uso de uma variável vistas no capítulo 3 (c-Uso e p-Uso), SPOTO (2000) define, para a abordagem de Aplicações Banco de Dados Relacional, a referência s-Uso (uso na *SQL*): quando a variável afetar o resultado de uma consulta ou de uma computação em um comando executável da *SQL* (Uso associado ao nó da *SQL*).

```
(01)  procedure T_Pesq_Lista.BotaoProcurar(Sender: TObject);
(01)  var vConta: Integer;
(01)  begin
(01)      vConta := 0;

(02)      Q1.Close;
(02)      Q1.SQL.Clear;
(02)      Q1.SQL.Add('Select * From ALUNOS '
                   + 'Where CIDADE = ''RIO CLARO'' '
                   + 'Order By NOME');
(02)      Q1.Open;

(03)      while not Q1.EOF do
(03)      begin
(04)          if Q1.FieldByName('Curso').AsString = txtCurso.Text then
(04)             vConta := vConta + 1;
(05)          Q1.Next;
(05)      end;

(06)      showmessage('A qtde. de Alunos é ' + IntToStr(vConta));
(06)  end;
```

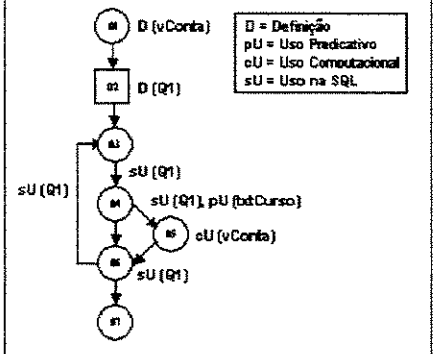


Figura 12: Grafo de Fluxo de Dados de uma Aplicação de Banco de Dados Relacional

4.2. Teste Funcional

BINDER (2000) propõe uma estratégia de teste para Associação de Classes, que estabelece uma abordagem sistemática para transformar parâmetros de multiplicidade de associação em casos de teste.

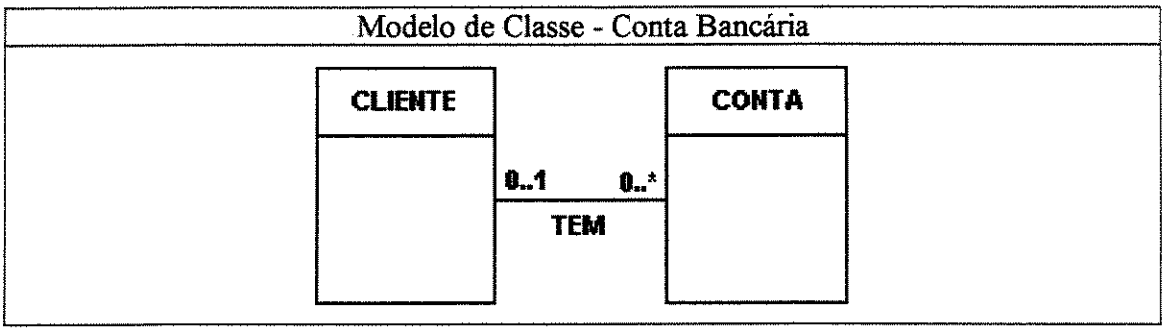


Figura 13: Modelo de Classe

Condições Limite de Multiplicidade para Conta Bancária			
Associação		Condições Limite	
Cliente tem Conta	0..*	Cliente: n(Conta) >= 0	Cliente: n(Conta) <= N
Conta é de Cliente	0..1	Conta: n(Cliente) >= 0	Conta: n(Cliente) <= 1

Tabela 4: Condições Limite de Multiplicidade

Matriz de Domínio para Conta Bancária									
Variável / Condição / Tipo			Casos de Teste						
			1	2	3	4	5	6	7	8
Nº de Contas	Cliente: n(Conta) >= 0	On	0							
		Off		-1						
	Cliente: n(Conta) <= N	On			N					
		Off				N+1				
	Típico	In					2	10	52	96
Nº de Clientes	Conta: n(Cliente) >= 0	On					0			
		Off						-1		
	Conta: n(Cliente) <= 1	On							1	
		Off								2
	Típico	In	1	1	1	1				
Resultado Esperado			✓	x	✓	x	✓	x	✓	x

Tabela 5: Matriz de Domínio

Esta estratégia de teste possibilita a partir do Modelo de Classe fazer Análise do Limite de Multiplicidade deste modelo. Na Figura 13 é apresentado um Modelo de Classe de Conta Bancária, composta por 2 classes (Cliente e Conta) associadas. A partir desta associação, são definidas as Condições Limite (Tabela 4). Para a análise é criada uma Matriz de Domínio para Conta Bancária (Tabela 5) com a seguinte estrutura:

- Variável: são as Classes do Modelo de Classes;
- Condição: são as definições da associação no Modelo de Classes;
- Tipo (On / Off): entrada possível / entrada que extrapola a associação;
- Típico: valores (entradas) aleatórios para testes;
- Casos de Teste: possíveis entradas;
- Resultados Esperados: Verificação dos casos de teste das associações do Modelo de Classe.

Devido a várias características comuns entre Modelo de Classes e Modelo Entidade-Relacionamento, esta estratégia pode ser aproveitada para Testes Funcionais em Aplicações de Banco de Dados Relacional, através da análise de cardinalidade entre entidades e relacionamentos.

4.3. Considerações Finais

Para a aplicação das técnicas de Teste Estrutural podem ser utilizados critérios baseados em Fluxo de Controle / Complexidade Ciclomática, a partir da aplicação do Teste de Caminhos Básicos, que permite definir uma maneira de determinar um conjunto básico de caminhos linearmente independentes, de modo que os executando, garante-se a execução de todos os comandos ao menos uma vez. Já os critérios baseados em Fluxo de Dados estabelecem requisitos de teste que seguem o modelo de dados usado dentro do programa. Deve-se, ainda, levar em consideração, a extensão proposta por SPOTO et al. (2000), para Aplicações de Banco de Dados Relacional.

No entanto, a escolha por técnicas de Teste Funcional permite por a prova a especificação do sistema. Para isso, podem ser utilizadas as técnicas de Particionamento de Equivalência e Análise de Valor Limite. A primeira técnica citada possibilita a redução do tamanho do domínio de entrada, pois se concentra em criar dados de teste baseados unicamente na especificação, e é indicado principalmente para especificação de aplicações, onde as variáveis de entrada podem ser detectadas mais facilmente. Complementando esta técnica, a Análise de Valor Limite coloca sua atenção em uma fonte propícia a erros, os limites de uma classe ou partição de equivalência. Portanto, estas técnicas se aplicadas em conjunto permitem um melhor aproveitamento para a análise dos Casos de Testes que devem ser gerados. Além disso, a estratégia proposta por BINDER (2000), permite uma adaptação para a especificação do Modelo Entidade-Relacionamento, possibilitando assim, testes nas especificações de Aplicações de Banco de Dados Relacional.

Capítulo 5

A Estratégia adotada e a Ferramenta desenvolvida para Teste de Software em Aplicação de Banco de Dados Relacional

Qualquer *software* desenvolvido pode ser testado de duas formas: verificando-se a função específica que deve ser executada pelo *software* (Teste Funcional) ou conhecendo-se o funcionamento interno de um *software* (Teste Estrutural).

Em relação ao Teste Funcional, o Teste Estrutural ajuda a detectar comportamentos que foram implementados, mas não especificados. Por outro lado, em relação ao Teste Estrutural, o Teste Funcional detecta comportamentos que foram especificados, mas não implementados e que o teste estrutural jamais testaria. Portanto, estes se complementam, e devem ser utilizados para garantir a qualidade do *software*.

5.1. A Estratégia Adotada

A estratégia de testes adotada foi baseada na idéia de Testes Híbridos definida por BEIZER (1995), onde são utilizadas as técnicas de Teste Estrutural e Funcional combinadas. Quanto a Atividade de Teste, esta foi definida baseando na fase de Testes de Unidade, onde serão utilizadas as técnicas de Teste Estrutural a partir de critérios baseado no Fluxo de Controle e critérios baseados na

Complexidade Ciclomática. Já a estratégia de Teste Funcional é definida através da utilização de Particionamento de Equivalência e Análise de Valor Limite a partir das variáveis de entrada.

Os passos para execução desta estratégia de teste são:

Teste Estrutural	1 - Inserção do Código-Fonte do Programa a ser testado 2 - Criação do Grafo de Fluxo de Controle 3 - Cálculo para determinar a Complexidade Ciclomática 4 - Identificação dos Caminhos Básicos (possíveis) 5 - Aplicação dos casos de teste referentes aos caminhos básicos
Teste Funcional	6 - Identificação do domínio de entrada para os testes Funcionais 7 - Definição das Classes de Equivalência 8 - Identificação dos Valores Limites 9 - Aplicação dos casos de teste 10 - Identificação dos possíveis erros do software

Tabela 6: Estratégia de Teste adotada

5.2. Ferramenta Desenvolvida para Suporte a Estratégia

A necessidade de se desenvolver uma ferramenta de testes surgiu da ausência de ferramentas com foco para Aplicações de Banco de Dados Relacional desenvolvidas com a Linguagem de Programação *Borland Delphi*.

Criou-se, então, um planejamento para desenvolvimento de uma Ferramenta para suporte a estratégia de teste apresentada anteriormente (ver Apêndice A e B). Esta ferramenta foi desenvolvida e batizada de *STest* for *Delphi* - Versão 1.0.

Para exemplificar a utilização da ferramenta *STest for Delphi* será utilizada uma rotina de gravação de notas dos alunos nas disciplinas da faculdade.

O primeiro passo é a execução do Teste Estrutural, através da inserção do código-fonte do programa a ser testado (Figura 14). Em seguida, aciona-se o botão "Processar" e se obtém como resultado o cálculo da Complexidade Ciclomática, o conjunto de Caminhos Básicos e o Grafo de Fluxo de Controle (Figura 15). Dando continuidade aos testes, o Teste Funcional pode ser executado e o

domínio de entrada pode ser definido automaticamente através do acionamento do botão "Identificar Domínio".

Após a identificação do Domínio de Entrada, complementam-se as Classes de Equivalência (Classe Válidas e Inválidas) para possível geração de Casos de Teste (Figura 16). Para poder definir o número de Casos de Teste a ser gerado, aciona-se botão "Qtd. Casos Teste" e digita-se a quantidade. Por fim, pressiona-se o botão "Processar" para obter os resultados (Casos de Teste - Particionamento de Equivalência e Análise de Valor Limite (Figura 17)).

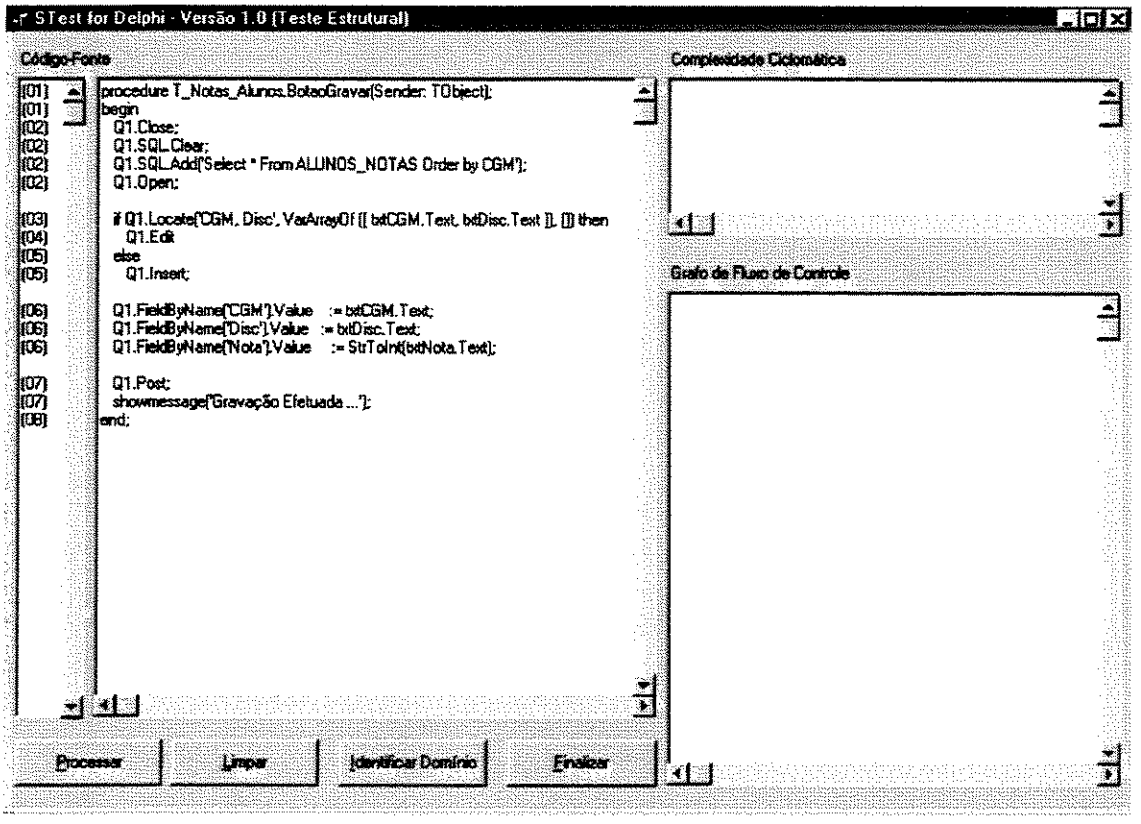


Figura 14: Inserção do Código-Fonte do Programa para a execução do Teste Estrutural

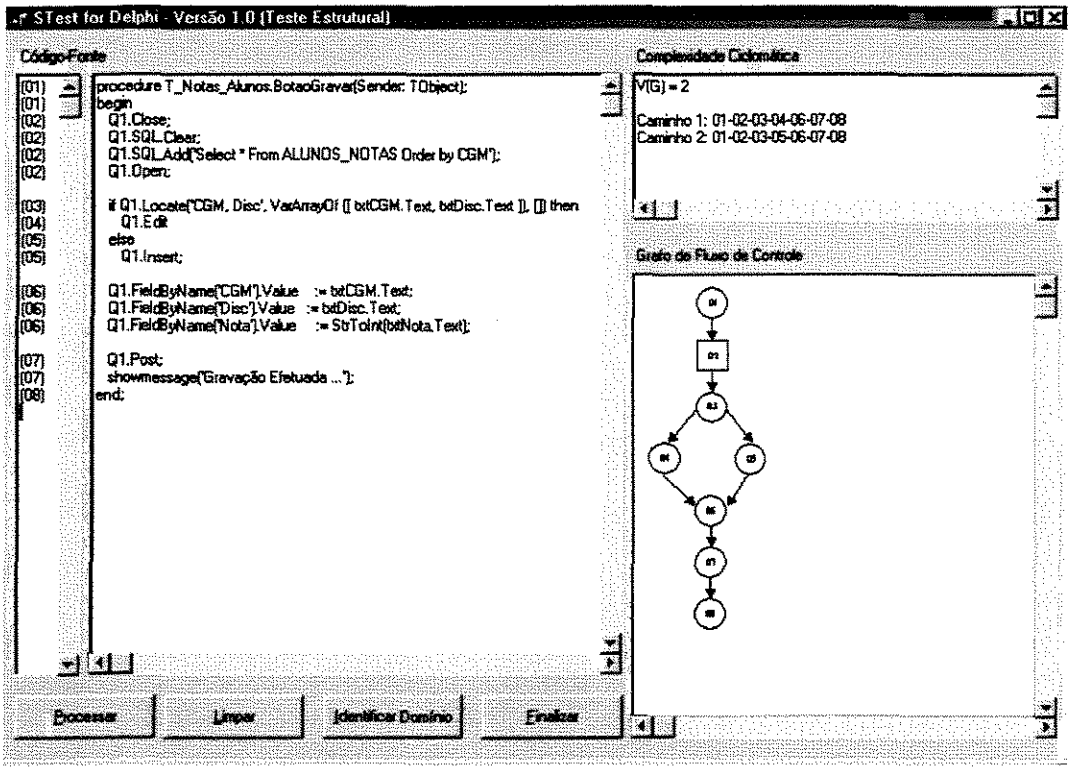


Figura 15: Complexidade Ciclomática, conjunto de Caminhos Básicos e Grafo de Fluxo de Controle

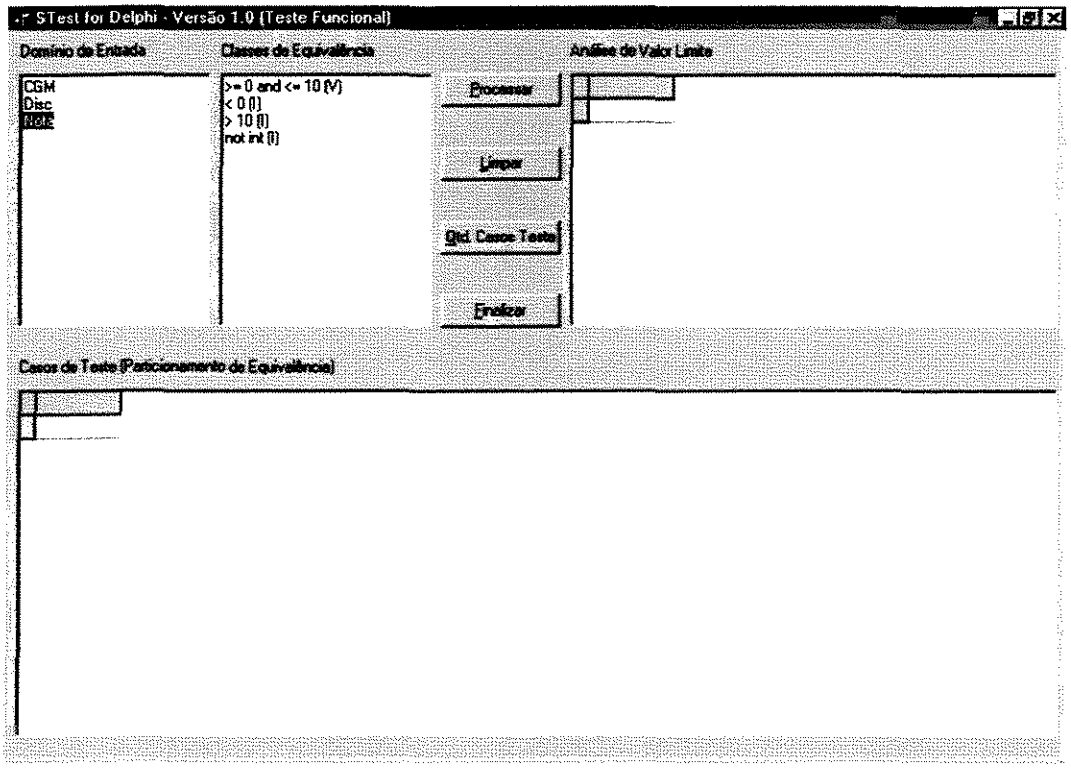


Figura 16: Domínio de Entrada e Classes de Equivalência

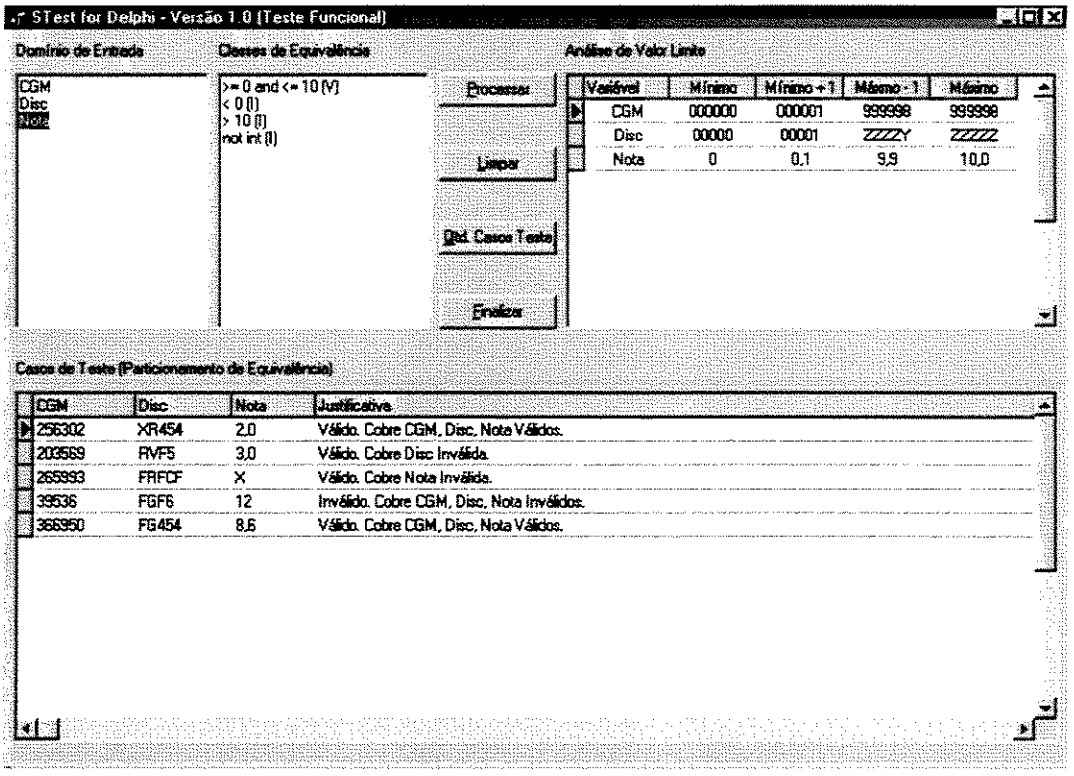


Figura 17: Casos de Teste - Particionamento de Equivalência e Análise de Valor Limite

Capítulo 6

Conclusão

Neste trabalho foram abordadas as técnicas de Teste de *Software* estendidas para Aplicações de Banco de Dados Relacional que foram necessárias, devido ao tipo de sistema no qual seriam aplicados os testes.

A definição da estratégia adotada e implementada integra conceitos encontrados na literatura e adaptados para esta abordagem. As Técnicas de Teste Estrutural escolhidas foram baseados em Fluxo de Controle e Complexidade Ciclomática através de Testes de Caminhos Básicos, e pode-se verificar a partir de alguns testes que estas técnicas cobriam as necessidades de testes do sistema. Já para Testes Funcionais foram adotadas as técnicas de Particionamento de Equivalência e Análise de Valor Limite para geração dos Casos de Teste baseados na especificação. Para tanto, as técnicas cobriram as necessidades de teste do sistema, pois eram analisados o domínio de entrada e os valores limites. Não foi utilizada a técnica proposta por BINDER (2000) de Modelo de Classe, pois se verificou que a técnica de Análise de Valor poderia cobri-la, e também a aplicação destas 3 técnicas poderia tornar a execução desta estratégia de teste muita exaustiva e desgastante.

A ferramenta desenvolvida (*STest for Delphi*) permitiu a execução da estratégia de teste com mais agilidade, segurança e confiabilidade. A Integração de Teste Estrutural e Funcional, que foi estudada e implementada na ferramenta, permitiu mais agilidade para a estratégia de teste adotada, pois a partir do Teste Estrutural era possível gerar automaticamente o domínio de entrada para Teste Funcional, o qual proporcionava apenas a conferência na especificação da aplicação.

Como conclusão, pode-se verificar que esta estratégia de teste que foi adotada permitiu detectar a presença de código-fonte inútil e algumas falhas de programação. Além disso, permitiu detectar erros

na especificação do sistema, em função da não atualização destas especificações mediante a manutenção do sistema.

Para pesquisas futuras, gostaria de propor novas estratégias de Testes de *Software* para Aplicações de Banco de Dados Relacional baseadas em critérios de Fluxo de Dados, pois elas complementam a estratégia de critérios de Fluxo de Controle, permitindo assim, a execução de teste nas variáveis do programa. Além disso, atualizar a ferramenta de testes para aplicação destas novas estratégias.

Bibliografia

BEIZER, Boris. "**Black-Box Testing: Techniques for Functional Testing of Software and Systems**". New York: John Wiley & Sons, 1995.

BINDER, Robert. "**Testing Object Oriented Systems: Models, Patterns and Tools**". Addison-Wesley, 2000.

CHAIM, M. L.; MALDONADO, J. C.; JINO, M. "**Ferramentas para Teste Estrutural de Software baseado em Análise de Fluxo de Dados: O Caso POKE-TOOL"**. Águas de Lindóia: Workshop do Projeto de Validação e Teste de Sistemas de Operação, 01/1997.

CHANG, J.; RICHARDSON, D. J. "**Structural Specification-Based Testing: Automated Support and Experimental Evaluation**". France: Proceedings of the 7th European Software Engineering Conference (ESEC/FSE'99), 09/1999.

DATE, C. J. "**Introdução a sistemas de Banco de Dados"**. Rio de Janeiro: Campus, 2000.

DEMILLO, R. A. "**Mutation Analysis as a Tool for Software Quality Assurance"**. Chicago: Proceedings of COMPSAC80, 10/1980.

FRANKL, F. G. "**The Use of Data Flow Information fo the Selection and Evaluation of Software Test Data**". New York: PhD Thesis of New York University, 10/1987.

HATLEY, D. J.; PIRBHAI, I. A. "**Strategies for Real-Time System Specification**". Dorset House, 1987.

HUANG, J. C. "**An Approach to Program Testing**". Computing Surveys, N° 3, Vol. 7, 1975.

MCCABE, T. "**A Software Complexity Measure**", IEEE TSE, N° 4, Vol. 2, 12/1976.

MYERS, Glen. "**The Art of Software Testing**". New York: John Wiley & Sons, 1979.

NTAFOS, S. C. "**On Required Element Testing**", IEEE TSE, Vol. 10, 11/1984.

PAULA FILHO, W . P. "**Engenharia de Software: Fundamentos, Métodos e Padrões**". Rio de Janeiro: Editora LTC, 2001.

PERRY, D. E.; WOLF, A. L. "**Foundations for the study of Software Architecture**". ACM SIGSOFT Software Engineering Notes, 1992.

PETERS, James F.; PEDRYCZ, Witold. "**Engenharia de Software: Teoria e Prática**". Rio de Janeiro: Campus, 2001.

PRESSMAN, Roger S. "**Engenharia de Software**". São Paulo: Makron Books, 1995.

RAPPS, S.; WEYUKER, E. J. "**Data Flow Analysis Techniques for Test Data Selection**", Tokio: Internacional Conference on Software Engineering, 09/1982.

RAPPS, S.; WEYUKER, E. J. "**Selecting Software Test Data Using Data Flow Information**", IEEE Transactions on Software Engineering - Vol 11, 04/1985.

SILBERSCHATZ, Abraham; KORTH, Henry; SUDARSHAN, S. "**Sistema de Banco de Dados**". São Paulo: Makron Books, 1999.

SOUZA, S. R. S., "**Avaliação do Custo e Eficácia do Critério Análise de Mutantes na Atividade de Teste de Software**", Dissertação de Mestrado - ICMC/USP, São Carlos: 06/1996.

SPOTO, Edmundo Sérgio; JINO, Mario; MALDONADO, José Carlos. "**Teste Estrutural de Software: Uma Abordagem para Aplicações de Banco de Dados Relacional**". João Pessoa: XIV Simpósio Brasileiro de Engenharia de Software, 10/2000.

VILELA, P.; MALDONADO, J. C.; JINO, M. "**Program Graph Visualization**". Software-Practice and Experience, 11/1997.

Apêndice A

Documento de Especificação de Requisitos do *Software* STest for Delphi - Versão 1.0

Este documento de Especificação de Requisitos de *Software* desenvolvido foi baseado no modelo de Especificação de Requisitos de *Software* de PAULA FILHO (2001).

A.1. Objetivos deste documento

O *software STest for Delphi* - Versão 1.0 visa a oferecer apoio informatizado ao processo de teste de software em aplicações (inclusive com Banco de Dados Relacional) desenvolvidas utilizando a Linguagem de Programação *Borland Delphi*. O Público-Alvo são Engenheiros de *Software*.

A.2. Escopo do *Software*

O *software STest for Delphi* - Versão 1.0 permite a geração de testes somente em aplicações desenvolvidas em *Borland Delphi*. A estratégia de Teste Estrutural é baseada nos critérios no Fluxo de Controle e Complexidade Ciclomática. Já a estratégia de Teste Funcional é baseada em Particionamento de Equivalência e Análise de Valor Limite.

Os Testes Estruturais são executados a partir da inserção do código-fonte da rotina a ser testada, enquanto que os Testes Funcionais são criados a partir da definição das classes de Equivalência.

A.3. Definições e Siglas

Item	Termo	Definição
1	Código-Fonte	Código de Especificação do Programa
2	Complexidade Ciclomática	Técnica de Teste Estrutural
3	Grafo de Fluxo de Controle	Técnica de Teste Estrutural
4	Particionamento de Equivalência	Técnica de Teste Funcional
5	Análise de Valor Limite	Técnica de Teste Funcional
7	Instruções *SQL*	Instruções da Linguagem de Consulta Estruturada (Banco de Dados)

Tabela 7: Definições e Siglas

A.4. Descrição Geral do Software

O *software STest for Delphi* - Versão 1.0 possui as qualificações a seguir.

A.4.1. Diagrama de Contexto

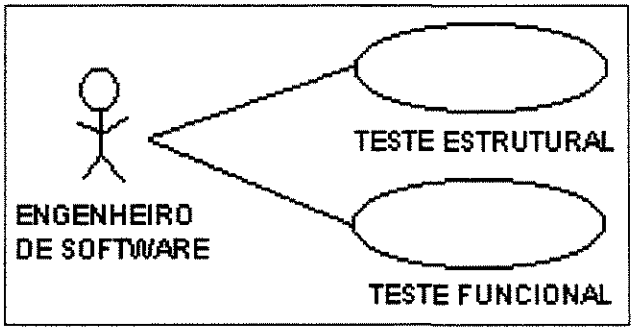


Figura 18: Diagrama de Contexto

A.4.2. Interfaces de Usuário

Item	Tela	Ator	Descrição
1	Abertura	Engenheiro de *Software*	Inicialização do *Software*
2	Menu	Engenheiro de *Software*	Seleciona o Tipo de Teste
3	Teste Estrutural	Engenheiro de *Software*	Gera o Teste Estrutural
4	Teste Funcional	Engenheiro de *Software*	Gera o Teste Funcional

Tabela 8: Interfaces do Usuário

A.4.3. Interfaces de Comunicação

O *software* pode ser utilizado por várias pessoas da equipe de desenvolvimento de *software*, portanto o mesmo deve ser multi-usuário e a rede de comunicação será baseada na plataforma *Windows*, através de seus serviços e protocolos de comunicação.

A.4.4. Restrições de Memória

O *software* deve ocupar no máximo 10 *MBytes* no Disco Rígido.

A.4.5. Modos de Operação

O modo de operação será interativo através da inserção dos requisitos e geração dos resultados esperados.

A.4.6. Funções do *Software*

Item	Caso de Uso	Descrição
1	Abertura	Inicialização do *Software*
2	Menu	Seleciona o Tipo de Teste (Estrutural ou Funcional)
3	Teste Estrutural	Inserção do Código-Fonte e cálculo da Complexidade Ciclomática, Caminhos Possíveis e geração do Grafo de Fluxo de Controle.
4	Teste Funcional	Inserção das Classes de Equivalência, geração dos casos de teste, baseados em Particionamento de Equivalência e Análise de Valor Limite.

Tabela 9: Funções do *Software*

A.4.7. Características dos Usuários

Os Engenheiros de *Software* responsáveis pelo Testes serão os usuários deste *software*.

A.5. Requisitos Específicos

A.5.1. Requisitos para Interfaces Gráficos de Usuário

O *software* será desenvolvido com *Interface* Gráfica baseada na plataforma *Windows*. Além da Tela de Abertura e Menu, o *software* possui uma tela para Teste Estrutural e uma para Teste Funcional.

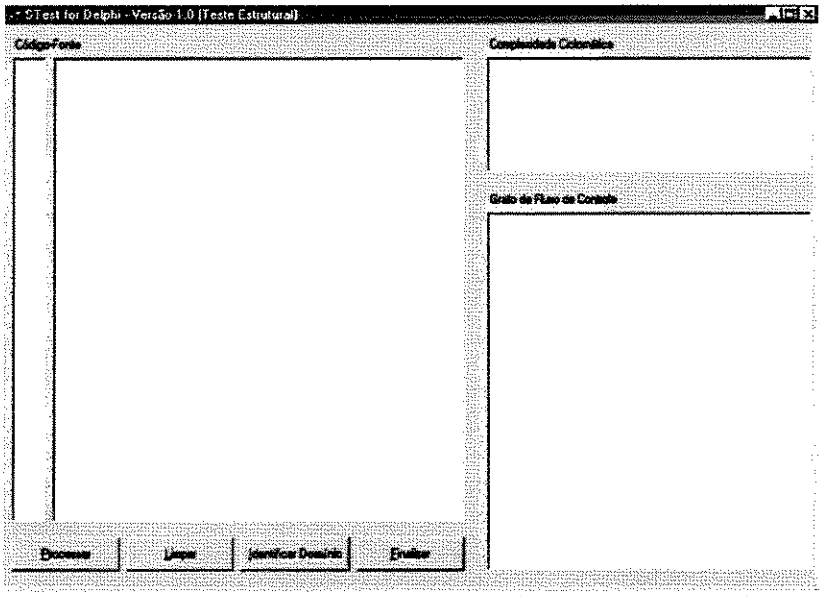


Figura 19: Protótipo de Interface de Teste Estrutural

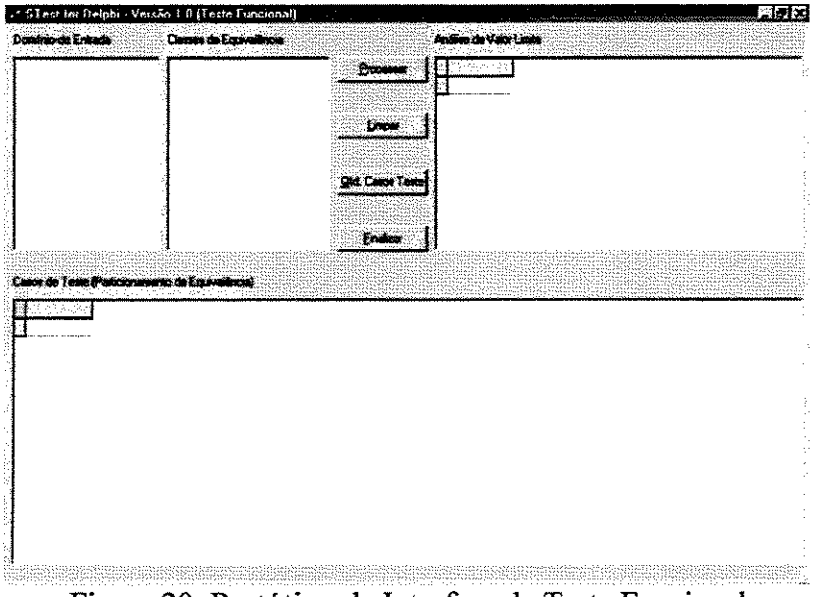


Figura 20: Protótipo de Interface de Teste Funcional

A.5.2. Requisitos Funcionais

A.5.2.1. Diagramas de Casos de Uso

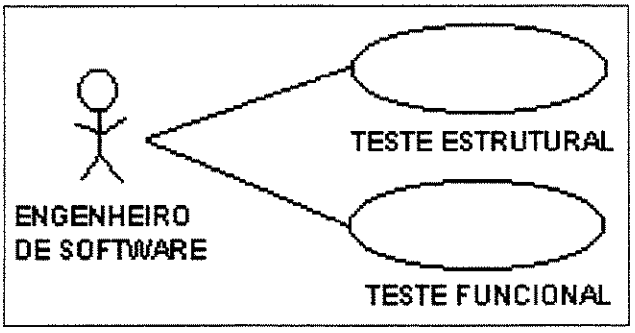


Figura 21: Diagrama de Casos de Uso

A.5.2.2. Fluxos de Casos de Uso

Entrada:

O *software* deve permite a inserção do código-fonte da rotina a ser testada para execução das técnicas de Teste Estrutural para critérios baseados no Fluxo de Controle e Complexidade Ciclomática.

O *software* também deve permitir a inserção das Classes de Equivalência para execução das técnicas de Teste Funcional baseadas em Particionamento de Equivalência e Análise de Valor Limite.

Saída:

Para os Testes Estruturais, o *software* deve calcular a Complexidade Ciclomática, definir os caminhos possíveis e gerar o Grafo de Fluxo de Controle.

Para os Testes Funcionais, o *software* deve gerar os casos de teste a partir da definição das Classes de Equivalência, e também apresentar os valores limites.

A.5.3. Requisitos Não-Funcionais

A.5.3.1. Usabilidade

- O sistema deve ter uma *interface* amigável para o usuário baseada na plataforma *Windows*.
- O sistema deve ter fácil acesso as suas funções através de menus.

A.5.3.2. Confiabilidade

- O sistema deve ter capacidade para recuperar os dados perdidos da última operação que realizou em caso de falha.
- O sistema deve ter a capacidade de manter um nível de desempenho no caso de violação de *interface*.

A.5.3.3. Eficiência

- O sistema deve responder a solicitações *on-line* em menos de 10 segundos.
- O sistema deve permitir o acesso de multi-usuários.

A.5.3.4. Manutenibilidade

- O sistema deve estar validado sempre que for atualizado (teste das rotinas).
- O sistema, sempre que alterado, deve manter a sua integridade (operacionalidade).

A.5.3.5. Portabilidade

- O sistema deve ser executado em computadores *PC Pentium 100 mHz* ou superior, com sistema operacional *Windows* 95 ou acima.
- O sistema deve ter capacidade de migração para outro ambiente estabelecido.

Apêndice B

Arquitetura do *Software* STest for Delphi - Versão 1.0

PERRY e WOLF (1992) apresentaram um modelo para descrição das arquiteturas de *software*, o qual é composto por 3 elementos básicos:

- Um Elemento de Processamento é uma estrutura de *software* que transforma suas entradas em saídas necessárias.
- Um Elemento de Dados consiste nas informações necessárias para o processamento, ou em informações a serem processadas por um elemento de processamento.
- Os Elementos de Conexão são a ponte que une as diferentes partes de uma arquitetura.

Baseado neste modelo, a Arquitetura do *Software STest for Delphi* - Versão 1.0 é composta por:

- Elementos de Processamento: Subsistema de Cálculo da Complexidade Ciclomática e Análise de Fluxo de Controle e Subsistema de Análise de Valor Limite e Particionamento de Equivalência.
- Elemento de Dados: Código-Fonte e Classes de Equivalência.
- Elementos de Conexão: Tela de Teste Estrutural e Teste Funcional.

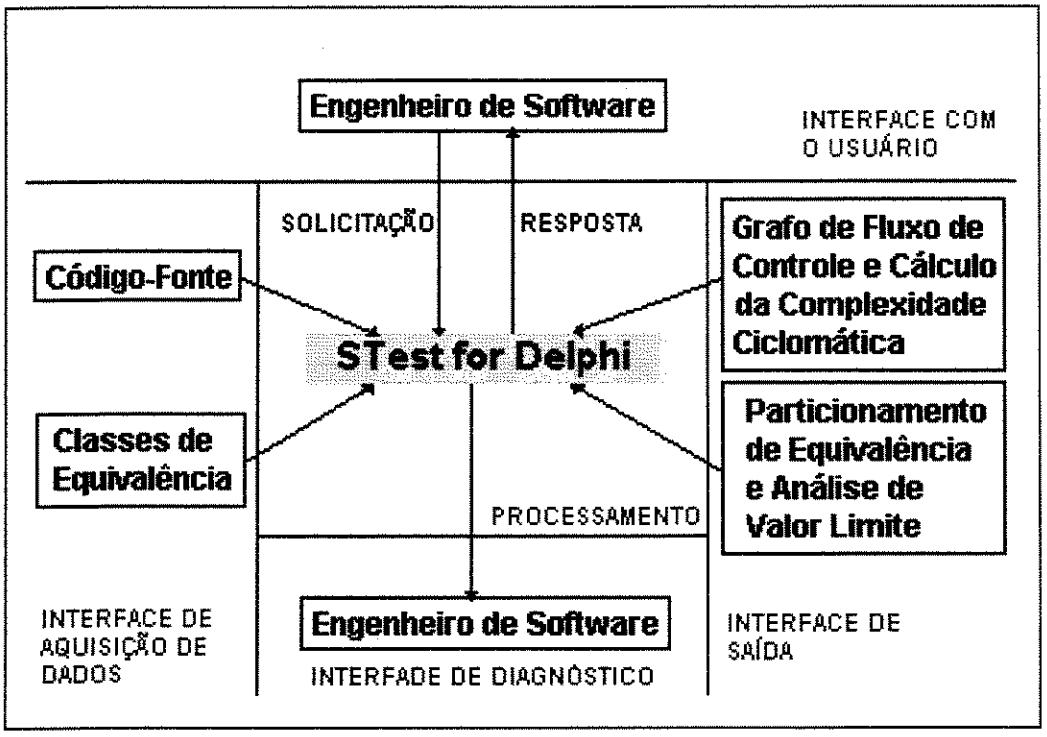


Figura 22: Arquitetura do *Software STest for Delphi* baseada na notação de HATLEY e PIRBHAI (1987)