



**Universidade do Estado do Rio de Janeiro**

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Cláudio Nobre Rapello

**Testes de sistemas de informações geográficas  
com lógica nebulosa**

Rio de Janeiro

2009

Claudio Nobre Rapello

**Testes de sistemas de informações geográficas  
com lógica nebulosa**

Dissertação submetida ao corpo docente da Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Computação. Área de concentração: Geomática.

Orientadores: Prof. Dr. Orlando Bernardo Filho

Prof.<sup>a</sup> Dr.<sup>a</sup> Vera Maria Benjamim Werneck

Rio de Janeiro

2009

CATALOGAÇÃO NA FONTE  
UERJ/REDE SIRIUS/CTCB

R216 Rapello, Cláudio Nobre

Testes de sistemas de informações geográficas com lógica nebulosa/  
Cláudio Nobre Rapello. – 2009.  
96 f.

Orientador : Orlando Bernardo Filho

Co-orientador: Vera Maria Benjamim Werneck

Dissertação (mestrado) – Universidade do Estado do Rio de Janeiro,  
Faculdade de Engenharia.

1. Engenharia de software – Teses. 2. Sistema de informações  
geográficas. I. Bernardo Filho, Orlando. II. Werneck, Vera Maria  
Benjamim. III. Universidade do Estado do Rio de Janeiro.  
Faculdade de Engenharia. IV. Título.

CDU 504.05

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial  
desta dissertação.

---

Assinatura

---

Data


Claudio Nobre Rapello

## Testes de Software com Lógica Nebulosa


Dissertação submetida ao corpo docente da Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Computação. Área de concentração: Geomática.

Aprovado em: 15/05/2009

Banca Examinadora:

  
Prof. Dr. Orlando Bernardo Filho (Orientador)

Faculdade de Engenharia da UERJ

  
Prof<sup>ª</sup>. Dr.<sup>a</sup> Vera Maria Benjamin Werneck (Co-orientadora)

Instituto de Matemática e Estatística da UERJ

  
Prof<sup>ª</sup>. Dr.<sup>a</sup> Neide dos Santos

Instituto de Matemática e Estatística da UERJ

  
Prof<sup>ª</sup>. Dr.<sup>a</sup> Fernanda Cláudia Alves Campos

Departamento da Ciência da Computação da UFJF

Rio de Janeiro  
2009

## **DEDICATÓRIA**

*Aos meus pais, esposa e filhas.*

## **AGRADECIMENTOS**

À UERJ.

Ao departamento de Engenharia de Sistemas e Computação, pelo padrão do curso.

Aos professores Orlando Bernardo Filho e Vera Maria Benjamin Werneck, por terem sido orientadores exemplares.

À professora Neide dos Santos, que sempre me apoiou e incentivou em minha jornada acadêmica.

Aos demais professores e amigos do curso de Geomática, que contribuíram de alguma forma para minha formação.

Ao SERPRO, nas pessoas de Claudio Santos e Jorginho James, que me liberaram para frequentar o curso e suas atividades.

Ao amigo Gustavo Guise, pelo apoio e idéias na área de testes de software.

## **RESUMO**

RAPELLO, Cláudio Nobre. Testes de sistemas de informações geográficas com lógica nebulosa, 2009. 96f. Dissertação (Mestrado em Engenharia da Computação)–Faculdade de Engenharia, Universidade Estadual do Rio de Janeiro, Rio de Janeiro, 2009.

Os testes são uma atividade crucial no desenvolvimento de sistemas, pois uma boa execução dos testes podem expor anomalias do software e estas podem ser corrigidas ainda no processo de desenvolvimento, reduzindo custos. Esta dissertação apresenta uma ferramenta de testes chamada SIT (Sistema de Testes) que auxiliará no teste de Sistemas de Informações Geográficas (SIG). Os SIG são caracterizados pelo uso de informações espaciais georreferenciadas, que podem gerar um grande número de casos de teste complexos. As técnicas tradicionais de teste são divididas em funcionais e estruturais. Neste trabalho, o SIT abordará os testes funcionais, focado em algumas técnicas clássicas como o particionamento de equivalência e análise do Valor Limite. O SIT também propõe o uso de Lógica Nebulosa como uma ferramenta que irá sugerir um conjunto mínimo de testes a executar nos SIG, ilustrando os benefícios da ferramenta.

Palavras-Chave: Engenharia de Software, Teste de Software, Lógica Nebulosa, Sistema de Informações Geográficas, SIG.

## **ABSTRACT**

Testing is a very important activity that is crucial in the development of information systems because most of the software costs could be avoided if better testing was performed. This dissertation describes a tool for a Testing System named SIT (System of Testing) to support the testing performance of Geographic Information Systems (GIS). GIS have the characteristics of spatial referenced persistent data that can generate a great and complex number of test cases. The traditional approaches for designing test cases can be divided into black box (functional) testing and white box testing. At first SIT will treat only the functionality testing method focused on the classic techniques of Equivalence Partitioning and Boundary Value Analysis (BVA). SIT also proposes the use of fuzzy logic as a tool to suggest the minimal test case to be used in those GIS to illustrate the benefits of the fuzzy logics module of this tool.

**Keywords:** Software Engineering, Software Testing, Fuzzy Logic, Geographic Information System, GIS.



## SUMÁRIO

	<b>INTRODUÇÃO</b>	1
1.1	<b>Motivação</b>	2
1.2	<b>Justificativa</b>	2
1.3	<b>Objetivo</b>	3
1.4	<b>Apresentação do trabalho</b>	3
1.5	<b>Trabalhos Relacionados</b>	4
2	<b>FUNDAMENTOS TEÓRICOS</b>	6
2.1	<b>Testes de Software</b>	6
2.1.1	<u>Projetos de casos de teste</u>	7
2.1.2	<u>Testes estruturais</u>	8
2.1.2.1	Notação de Grafo de Fluxo	8
2.1.2.2	Derivação de casos de teste	9
2.1.2.3	Matrizes e grafos	9
2.1.2.4	Teste de condição	10
2.1.2.5	Teste de fluxo de dados	10
2.1.2.6	Teste de ciclo	11
2.1.3	<u>Testes funcionais</u>	12
2.1.3.1	Métodos de teste baseados em grafos	12
2.1.3.2	Particionamento de equivalência	15
2.1.3.3	Análise do valor limite	15
2.1.3.4	Teste de comparação	16
2.1.3.5	Teste da matriz ortogonal	16
2.1.4	<u>Teste de ambiente, arquitetura e aplicações especializadas</u>	17
2.1.4.1	Teste de interface gráfica com o usuário (GUI)	17
2.1.4.2	Teste de arquitetura cliente-servidor	17
2.1.4.3	Teste de documentação e dispositivos de ajuda	18
2.1.5	<u>Estratégias de teste de software</u>	18
2.1.5.1	Uma abordagem estratégica do teste de software	18
2.1.6	<u>Aspectos estratégicos</u>	19
2.1.6.1	Teste de unidade	20
2.1.6.2	Teste de integração	20
2.1.6.3	Teste de regressão	21

2.1.6.4	Teste fumaça	22
2.1.6.5	Teste de validação	22
2.1.6.6	Teste de recuperação	22
2.1.6.7	Teste de segurança	23
2.1.6.8	Teste de estresse	23
2.2	<b>Arquitetura de Sistemas de Informação Geográfica</b>	23
2.2.1	<u>Estrutura geral de um SIG</u>	24
2.2.2	<u>Gerência de dados em um SIG</u>	25
2.2.2.1	Arquitetura Dual	26
2.2.2.2	Arquitetura Integrada para Gerência de Dados	27
2.2.3	<u>Uma visão geral da tecnologia de SIG</u>	30
2.2.3.1	Aplicativos SIG para desktop (GIS desktop)	31
2.2.3.2	A segunda geração: bancos de dados geográficos	31
2.2.3.3	A terceira geração: bibliotecas geográficas digitais	32
2.2.3.4	Bibliotecas de componentes	33
2.3	<b>Lógica Nebulosa</b>	33
2.3.1	<u>Conceitos de Conjunto nebuloso e função de pertinência</u>	34
2.3.2	<u>Operações com Conjuntos Nebulosos</u>	35
2.3.2.1	Operadores	35
2.3.2.2	Propriedades Algébricas	38
2.3.2.3	Variáveis Lingüísticas	39
2.3.2.4	Funções de implicação ou proposição difusa	40
2.3.2.5	Processo de Agregação	41
2.3.2.6	Defuzzificação	41
2.3.2.7	Inferência	44
2.3.3	<u>Sistema de Inferência Nebuloso (SIN)</u>	47
2.3.3.1	Fuzzificador	48
2.3.3.2	Base de Conhecimento	48
2.3.3.3	Unidade de inferência	49
2.3.3.4	Defuzzificador	49
3	<b>O SISTEMA DE TESTES (SIT)</b>	50
3.1	<b>Visão geral do Sistema de Testes</b>	50
3.2	<b>Sistemas de Inferência Nebulosos do SIT</b>	52

3.3	<b>Modelo conceitual do SIT</b>	56
3.4	<b>A implementação do Sistema de Testes (SIT)</b>	57
3.4.1	<u>Funcionamento do Sistema de Testes</u>	57
3.4.2	<u>A interface do SIT</u>	58
3.4.3	<u>Banco de dados</u>	60
3.4.4	A criação da DLL SITFuzzy	63
3.4.4.1	A classe clsFuncoesBD	64
3.4.4.2	A classe clsFuzzy	64
3.5	<b>Aplicação da teoria dos conjuntos nebulosos</b>	66
3.6	<b>Definição das Regras de Inferência</b>	67
3.7	<b>Definição dos tipos de entradas</b>	68
4	<b>ESTUDO DE CASO</b>	69
4.1	<b>SIGATDSC</b>	69
4.1.1	<u>Análise do resultados obtidos</u>	72
4.2	<b>SICH</b>	73
4.2.1	<u>Análise do resultados obtidos</u>	76
5	<b>CONCLUSÕES</b>	78
	<b>REFERÊNCIAS BIBLIOGRÁFICAS</b>	80
	<b>APÊNDICE A – Descrição dos Casos do Uso</b>	85
	<b>APÊNDICE B – Código fonte da DLL</b>	88
	<b>APÊNDICE C – Regras de Inferência</b>	97

## 1 INTRODUÇÃO

Teste de software é um elemento crítico na atividade da Engenharia de Software e na Garantia da Qualidade de Software. Dependendo da criticidade do software, um bom planejamento e uma boa execução do plano de testes podem economizar muito dinheiro, não sendo raro uma organização gastar entre 30% e 40% do esforço total do projeto em testes (BERTOLINO, 2007).

Apesar do uso dos melhores métodos de desenvolvimento e das melhores ferramentas de suporte e treinamento de pessoal, os erros permanecem nos produtos e são essencialmente humanos. Assim, as atividades de revisão e teste representam um papel fundamental para a obtenção de um produto de software com qualidade. Podem-se definir diferentes níveis de testes em relação às diversas fases do processo de desenvolvimento de software, focando classes de erros distintas para se prevenir ou detectar erros cometidos ao longo de todo processo de desenvolvimento (PRESSMAN, 2006).

Por outro lado, na atividade de teste, há uma grande quantidade de limitações, sendo os principais problemas: existência de um conjunto grande de dados de teste para executar todos os caminhos possíveis; retorno do programa com o resultado esperado, porém através de processamentos incorretos, não revelando a existência do erro. O erro pode ser revelado com a entrada de outro valor. Esse tipo de erro é chamado de correção coincidente.

Como o domínio de dados de entrada geralmente é muito grande ou infinito, a atividade de teste pode não mostrar com exatidão que o produto de software está correto, inviabilizando na maioria das vezes o teste de todas as possibilidades (teste exaustivo). Outro aspecto é a limitação de tempo e recursos destinados aos testes. Sendo assim, duas questões cruciais nesse contexto podem ser formuladas:

- Como seleccionar os dados para a realização dos testes?
- Como decidir se um produto foi suficientemente testado?

Para solucionar as questões apresentadas, critérios de teste são estabelecidos, que por sua vez estão incluídos nas técnicas distintas de teste. Deve-se, então, definir e adequar esses critérios e técnicas para a fase de teste a ser realizada. Diversas técnicas clássicas de teste podem ser utilizadas, tais como a técnica funcional (caixa-preta), estrutural (caixa-branca) e a técnica com base em erros.

Diante das dificuldades apontadas para a definição de quais valores das entradas de um sistema devem ser testados, muitos pesquisadores da área têm lançado mão de técnicas da Inteligência Computacional para analisar quais dados são mais relevantes para serem incluídos na massa de teste. Algoritmos Genéticos, por exemplo, é uma das técnicas já experimentadas (BUENO, 2008), todavia uma técnica também possível de ser aplicada seria a Lógica Nebulosa, visto que ela pode ser usada para inferir o quanto se deve testar de uma entrada com base em algumas das suas características mais importantes.

A técnica aqui desenvolvida neste trabalho aplicou Lógica Nebulosa na análise da massa de testes dos Sistemas de Informações Geográficas (SIG). Os SIG são mais abrangentes que os demais sistemas de informação, uma vez que possuem as mesmas características desses últimos, além de incluírem dados referenciados espacialmente, aumentando então a complexidade do teste de suas funcionalidades o que vem a tornar mais necessária a criação de forma semi-automática dos testes a serem feitos sobre eles.

O sistema de testes (SIT) foi proposto neste trabalho com o objetivo de sugerir um conjunto mínimo de valores a testar para cada entrada de um Sistema de Informações Geográficas (SIG). Esse conjunto mínimo é identificado através da aplicação das técnicas de testes funcionais associadas a um módulo composto por três sistemas de inferência nebulosos. As seguintes técnicas de teste funcionais foram utilizadas neste trabalho: particionamento de equivalência e análise do valor limite.

## **1.1 Motivação**

O presente trabalho tem como motivação a redução do espaço total de testes a serem executados em um sistema de informações geográficas, mantendo-se uma cobertura mínima que garanta a qualidade do software gerado.

## **1.2 Justificativa**

Devido à inviabilidade de cobrirmos todos os casos de teste possíveis de um sistema, foi desenvolvida uma ferramenta de testes de SIG que forneça e determine os testes a serem realizados de acordo com as características de cada entrada. Para isto foram utilizados métodos clássicos de teste associados à inteligência computacional (lógica nebulosa).

### **1.3 Objetivo**

O principal objetivo deste trabalho é descrever uma aplicação da teoria dos conjuntos nebulosos que indique um conjunto mínimo de testes a realizar para um Sistema de Informações Geográficas. Estes testes serão gerados a partir de informações cadastradas para cada entrada, diminuindo o universo de testes a realizar, uma vez que a realização de todos os testes é inviável. Em outras palavras, o principal objetivo do trabalho é comprovar a seguinte hipótese: “O uso de técnicas de teste clássicas em conjunto com a inteligência computacional (lógica nebulosa) na escolha de casos de teste de software reduzirá o espaço total de testes a serem executados”.

### **1.4 Apresentação do trabalho**

A dissertação está estruturada em 5 capítulos, resumidos a seguir.

No Capítulo 1 está a parte introdutória do trabalho, apresentando também sua motivação, justificativa e objetivos.

O Capítulo 2 proporciona uma visão geral dos conceitos necessários para o entendimento do escopo do trabalho desenvolvido, apresentando as técnicas de teste, os fundamentos dos sistemas de informações geográficas (SIG) e os conceitos de lógica nebulosa.

O Capítulo 3 apresenta o sistema de testes desenvolvido, proporcionando uma visão de suas principais características, módulos e interfaces com o usuário. Também são apresentadas as funções de pertinência dos módulos de inferência nebulosa e a aplicação da teoria dos conjuntos nebulosos.

No Capítulo 4 são descritos os testes realizados com a ferramenta desenvolvida, verificando sua eficiência. São também discutidos os resultados obtidos pelo SIT.

Finalmente, o Capítulo 5 apresenta as conclusões do trabalho desenvolvido e objetivos alcançados, citando suas principais contribuições e apontando trabalhos futuros.

## 1.5 Trabalhos Relacionados

DeMillo e Offutt (1991) propuseram a técnica baseada em restrições, com o objetivo de auxiliar na geração de dados de teste adequados aos testes de mutação. São usadas restrições algébricas projetadas para detectar um tipo particular de defeito no programa. O teste baseado em restrições usa os conceitos da análise de mutantes, detectando os defeitos descritos pelos operadores de mutação. As três condições são: alcançabilidade, necessidade e suficiência.

Bueno e Jino (1999) utilizaram algoritmos genéticos para manipular valores de entrada de um programa para que um caminho pretendido fosse executado, selecionando as melhores soluções. Cada novo caminho pretendido é inserido em uma base de soluções, para serem usados na composição da população inicial do algoritmo genético, o que permite que soluções passadas sejam reaproveitadas. Tal prática reduz o custo das buscas.

Pargas *et al.* (1999) utilizaram algoritmos genéticos para geração automática de dados para teste. A solução foi implementada em uma ferramenta chamada TGen, que apresentou resultados promissores para programas de maior porte.

Linkman et al. (2003) realizaram um estudo com o objetivo de avaliar a adequação do teste aleatório e funcional em relação ao critério de análise de mutantes. Foram gerados conjuntos de teste para o teste aleatório (TA), funcional sistemático (FS), particionamento de equivalência e análise do valor limite (PE) e para a análise de mutantes (AM). Todos os casos de teste PE apresentaram eficiência acima de 90%, enquanto que as outras técnicas apresentaram resultados bem variados. Apesar de estudos complementares serem necessários, observou-se que os critérios de teste funcionais apresentaram grande cobertura de teste.

Ferreira e Vergilio (2005) utilizaram algoritmos genéticos para alcançar a cobertura desejada de testes. Os operadores genéticos foram utilizados até que tal cobertura fosse alcançada ou um determinado número de gerações fosse alcançado. As técnicas de hibridização utilizadas foram as listas de tabu e elitismo.

Bertolino (2007), faz um apanhado geral (*roadmap*) do que já é realidade em termos de testes. O artigo aponta também as novas tecnologias emergentes na área e aponta os desafios para a geração de testes automáticos “inteligentes”. Tais testes terão como objetivo a maximização da cobertura, através do uso de padrões e inteligência computacional, o que leva a um menor custo.

Oliveira Neto *et al.* (2008) utilizaram um algoritmo a redução do número de casos de teste, baseado em um modelo comportamental conhecido como LTS (*Labelled Transition System*). Tal modelo permite um formalismo semântico de diversas notações de especificação, que podem ser facilmente obtidas de diversas ferramentas existentes como a SPACES (BARBOSA *et al.*, 2007) e a LTS-BT (CARTAXO *et al.*, 2008). Os grafos com os caminhos a serem percorridos para os diversos casos de teste são analisados. Os grafos com caminhos coincidentes são descartados, diminuindo o escopo de teste.

Bueno, Wong e Jino (2008) simularam o chamado algoritmo de repulsão, baseado em um sistema de partículas, para a geração automática de conjuntos de testes chamados DOTS. Estes conjuntos são gerados aleatoriamente e melhorados por meio de iterações. Os resultados de uma simulação são avaliados para indicar uma melhora na identificação de defeitos.



## 2 FUNDAMENTOS TEÓRICOS

Este capítulo proporciona uma visão geral de todos os conceitos necessários para o perfeito entendimento do escopo deste trabalho. Inicialmente, são apresentados os conceitos relativos ao teste de software, incluindo os tipos de teste estruturais e funcionais e suas principais técnicas. A seguir, são abordados os fundamentos dos sistemas de informação geográfica, sua evolução no tempo, implantação e aplicações. Finalmente, são discutidos os conceitos de lógica nebulosa, incluindo os principais operadores, funções algébricas e o funcionamento de um sistema de inferência nebuloso.

### 2.1 Testes de Software

O teste do software gerado é um elemento crítico na atividade Garantia de Qualidade. Este demonstra se o produto atende aos requisitos e demonstra as falhas ou defeitos do software (SOMMERVILLE, 2007).

Dependendo da criticidade do software, um bom planejamento e uma boa execução do plano de testes podem economizar muito dinheiro e até mesmo salvar vidas. Não é raro uma organização gastar entre 30 e 40% do esforço total do projeto em testes.

Segundo Deutsch (PRESSMAN, 2006):

O desenvolvimento de sistemas de software envolve uma série de atividades de produção em que as oportunidades para injetar falibilidade humana são enormes. Erros podem vir a ocorrer até o início do processo, onde os objetivos... podem ser errônea ou imperfeitamente especificados, bem como [em] estágios posteriores de projeto e desenvolvimento... Por causa da inabilidade humana de realizar e de se comunicar com perfeição, o desenvolvimento é acompanhado por uma atividade de garantia de qualidade.

Durante o processo de desenvolvimento do software, o engenheiro tenta construir um produto através de conceitos abstratos. Há a elicitacão dos requisitos, a criação de diagramas, etc., sendo possível a introdução de erros. Os testes expõem as anomalias do software e estas podem ser corrigidas ainda no processo de desenvolvimento.

Algumas premissas devem ser seguidas nos testes de software. Segundo Davis (PRESSMAN, 2006), "O Princípio de Pareto<sup>1</sup> se aplica a testes de software, ou seja, 80% dos problemas são relativos a 20% dos componentes. A questão é identificar estes componentes."

Os testes devem ser planejados muito antes do início de sua execução e executados durante todo o processo de desenvolvimento. O ideal é que logo que o modelo de requisitos esteja completo os casos de teste sejam projetados.

Pode-se começar os testes nos componentes individuais, progredindo para conjuntos de componentes, até que todo o sistema seja coberto.

Devido o número de permutações de caminhos a serem percorridos ser extremamente grande, mesmo em sistemas pequenos, é impossível cobrir todas as possibilidades na "força bruta". O que pode ser feito é cobrir a lógica do programa de acordo com os requisitos especificados.

Softwares devem ser projetados com alta testabilidade, ou seja, com facilidades para a execução de testes, tendo todos os fatores que afetam a saída do sistema visíveis ao testador, sempre que possível. Simplicidade e compreensibilidade aceleram os testes por razões óbvias.

### 2.1.1 Projetos de casos de teste

Qualquer produto que passa por engenharia pode ser testado de duas formas. Na primeira, pode-se testar todos os comandos, verificando-se se cada um deles está operacional, ou seja, uma abordagem estrutural ou caixa-branca. Na segunda, pode-se verificar se as saídas são as esperadas, de acordo com as entradas fornecidas. Esta segunda abordagem é conhecida como funcional ou caixa-preta.

Os testes funcionais são realizados na interface do software e seus resultados indicam que as funções do software estão operacionais.

Um teste estrutural testa os caminhos lógicos do software de forma rigorosa. A sua dificuldade está no número muito grande de caminhos possíveis, mesmo para um software de pequeno porte. Apesar desta dificuldade, um teste funcional não deve ser descartado. O ideal é que seja selecionado um número limitado de caminhos lógicos importantes para que os mesmos sejam exercitados.

---

<sup>1</sup> Princípio estabelecido por Vilfredo Pareto (1848-1923), também conhecido como princípio 80-20, que significa que 80% dos problemas está relacionado a 20% das causas. (WIKIPEDIA,2009 e PRESSMAN, 2006)

Os atributos, tanto dos testes estruturais quanto dos testes funcionais, podem ser combinados para que seja obtida uma abordagem válida da interface e que seja garantido seletivamente o funcionamento interno do software.

### 2.1.2 Testes estruturais

Os testes estruturais, também chamados de testes caixa-branca ou caixa-de-vidro, permitem o teste de todos os caminhos independentes e ciclos de um software. A idéia é que cada caminho seja percorrido pelo menos uma vez e que cada estrutura de dados seja validada.

Os testes estruturais tornam-se necessários para cobrir possibilidades que os testes funcionais não cobrem. Geralmente os erros lógicos não estão no fluxo principal do programa, pois ele é testado exaustivamente durante a construção do software, ou seja, os erros lógicos são inversamente proporcionais ao número de vezes que um caminho é executado. Além disso, pode-se ter erros de avaliação por parte do engenheiro de software, que pode acreditar que um fluxo do programa não é provável de ser executado, quando o mesmo é muito executado no dia-a-dia.

Os testes estruturais partem de um caminho básico, que é então derivado para que o maior número de comandos do software seja testado. O ideal é que cada comando seja executado pelo menos uma vez.

#### 2.1.2.1 Notação de Grafo de Fluxo

Na abordagem estrutural pode-se utilizar um método auxiliar conhecido como *Grafo de Fluxo* (figura 1), que mostra o fluxo de controle lógico de um programa através de uma construção estruturada em nós e arestas. Pode-se então determinar a chamada *Complexidade Ciclomática*, que é uma métrica que fornece a complexidade lógica de um programa e ajuda na definição do número de caminhos lógicos independentes. A complexidade ciclomática pode ser calculada de três maneiras diferentes: pelo número de nós com decisão (nós predicados), pelo número de regiões e pelo número de arestas e grafos.

$$V(G) = \text{Arestas} - \text{Nós} + 2$$

$$V(G) = \text{Predicados} + 1$$

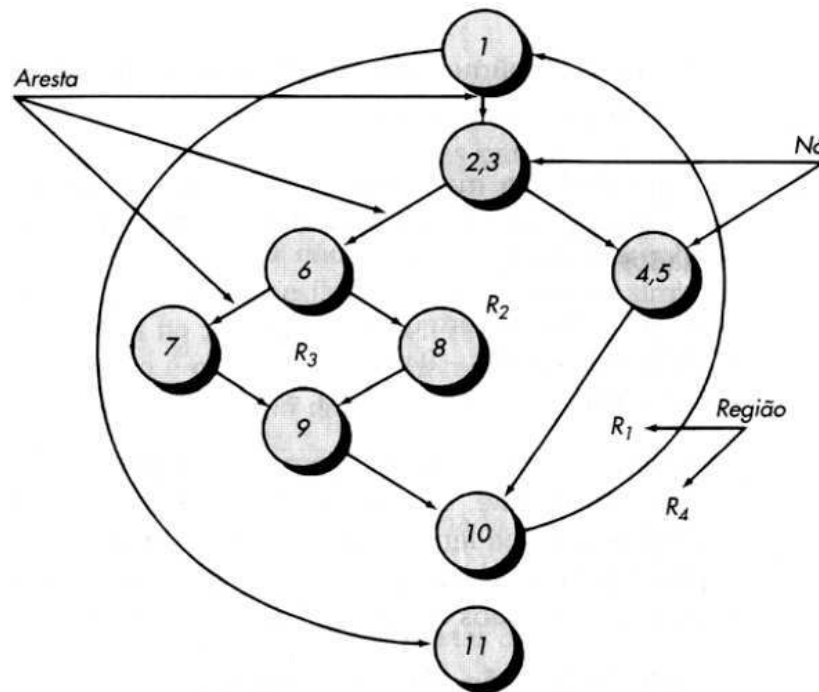


Figura 1 – Grafo de fluxo de um programa (Pressman, 2006)

#### 2.1.2.2 Derivação de casos de teste

O método do caminho básico pode ser aplicado a um projeto procedimental ou código-fonte. A partir da informação inicial pode-se elaborar grafo de fluxo e determinar a complexidade ciclomática. Então, pode-se identificar o total de caminhos independentes e preparar casos de teste que cubram todos os caminhos. Cada caso deve ser testado e comparado aos resultados esperados.

#### 2.1.2.3 Matrizes e grafos

O procedimento para originar grafos de fluxo e determinar o conjunto de caminhos básicos pode ser mecanizado. Para o desenvolvimento de uma ferramenta que auxilie o teste do caminho básico, pode ser interessante o uso de uma estrutura de dados chamada Matriz de Grafo.

Uma matriz de grafo (figura 2) é uma matriz quadrada, cujo tamanho é igual ao número de nós do grafo de fluxo. Podemos identificar os nós por números e as arestas por letras.

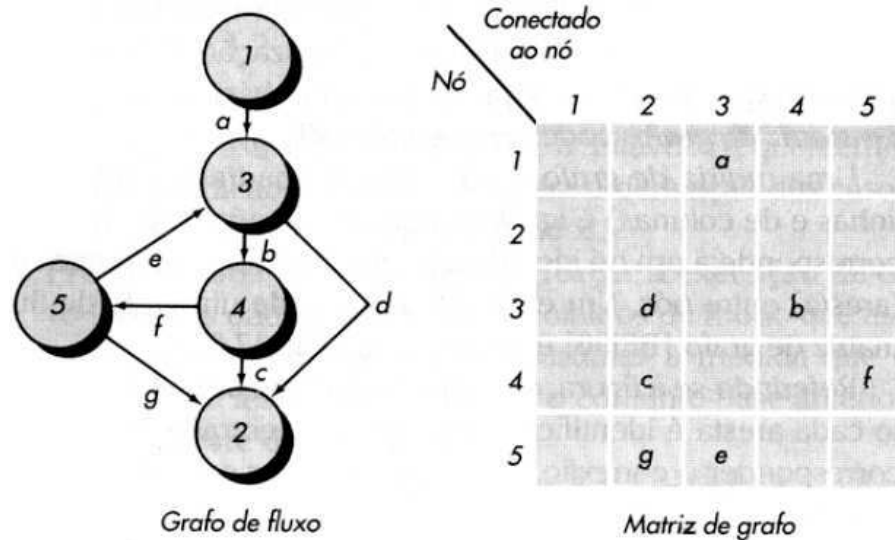


Figura 2 – Grafo de fluxo e Matriz de grafo (Pressman, 2006)

Adicionando-se pesos às ligações pode-se construir uma ferramenta de avaliação da estrutura e fluxo de controle de um software durante os testes. Os pesos podem ser atribuídos de acordo com propriedades relevantes para os testes a serem realizados.

Pode-se, por exemplo, atribuir um peso diferente a arestas com maior probabilidade de execução ou com maior tempo de processamento.

Beizer (PRESSMAN, 2006) fornece um rigoroso tratamento de outros algoritmos matemáticos, que podem ser aplicados a matrizes de grafo. Usando essas técnicas, a análise necessária para projetos de casos de teste pode ser parcialmente ou totalmente automatizada.

#### 2.1.2.4 Teste de condição

O teste de condição é um tipo de caso de teste que verifica as condições lógicas de um módulo de um software. Podemos ter condições simples, como uma condição booleana, ou condições compostas, que agrupam mais de uma condição simples. Este método foca o teste de cada condição do software, possibilitando testes simples e uma cobertura abrangente.

#### 2.1.2.5 Teste de fluxo de dados

O teste de fluxo de dados seleciona caminhos de teste de acordo com a localização das definições das variáveis de um programa e o uso das mesmas.

As estratégias de teste de fluxo de dados são úteis para selecionar caminhos de teste em um software que contém comandos aninhados e ciclos. O problema é que a seleção de caminhos de teste torna-se um pouco complexa.

#### 2.1.2.6 Teste de ciclo

O teste de ciclo é uma técnica estrutural focada exclusivamente na validade de construções de ciclos. Podemos classificar os ciclos em quatro tipos diferentes: ciclos simples, ciclos aninhados, ciclos concatenados (seqüenciais) e ciclos desestruturados.

Nos ciclos simples o seguinte conjunto de testes podem ser aplicados:

1. Pule todo o ciclo;
2. Apenas uma passagem pelo ciclo;
3. Duas passagens pelo ciclo;
4.  $m$  passagens pelo ciclo, onde  $n$  é o número máximo de passagens e  $m < n$ ;
5.  $n-1$ ,  $n$ ,  $n+1$  passagens pelo ciclo;

Se a abordagem de ciclos simples for estendida para ciclos aninhados, o número de testes cresce geometricamente. Beizer (PRESSMAN, 2006) ainda sugere uma outra abordagem:

1. Começar no ciclo mais interno e ajustar os outros ciclos para valores mínimos;
2. Fazer testes de ciclo simples para o ciclo mais interno, ajustando os outros ciclos para valores mínimos;
3. Passar para o próximo ciclo, mantendo-se os outros ciclos com valores mínimos;
4. Continuar nesta estratégia até que todos os ciclos sejam testados;

Os ciclos concatenados podem ser testados usando-se a técnica de ciclos simples, uma vez que cada ciclo é independente do outro.

O ciclos desestruturados devem ser reprojitados, sempre que possível, para que para ciclos simples ou compostos.

### 2.1.3 Testes funcionais

Um teste funcional tem como foco os requisitos funcionais de um sistema. Ele não é uma alternativa ao teste estruturais, mas uma estratégia complementar de testes.

O teste funcional tenta encontrar principalmente erros de interface, funções incorretas ou omitidas, erros de estruturas de dados, erros de acesso a bases de dados e erros de iniciação e término do software.

O teste funcional tende a ser aplicado nos últimos estágios do plano de testes, diferentemente do teste estrutural, que é aplicado durante todo desenvolvimento.

Na abordagem funcional os testes são projetados com base em algumas questões que devem ser respondidas:

- Como a validade funcional é testada?
- Como o comportamento e o desempenho do sistema são testados?
- Que classes de entrada vão constituir bons casos de teste?
- O sistema é particularmente sensível a certos valores de entrada?
- Como são isolados os limites de uma classe de dados?
- Que taxas e volumes de dados o sistema pode tolerar?
- Que efeito as combinações específicas de dados vão ter na operação do sistema?

#### 2.1.3.1 Métodos de teste baseados em grafos

O passo inicial do teste funcional é entender os objetos e as relações que os conectam, verificando a seguir se estas relações estão de acordo com o esperado. Pode-se criar um grafo como na abordagem estrutural, estabelecendo os testes que vão cobrir todos os objetos do grafo. Neste grafo criado, cada nó representa um objeto e cada aresta representa uma relação entre os objetos. Podemos atribuir pesos para os nós e pesos para as arestas.

No grafo da figura 3, cada nó é representado por um círculo. As ligações podem ser de três tipos:

- Direcionada: a relação se move em apenas uma direção. A representação é uma linha com uma seta;

- Bidirecional: a relação se aplica em ambas as direções. A representação é uma linha simples;
- Paralela: diversas relações diferentes são estabelecidas entre os nós do grafo. A representação é uma linha dupla;

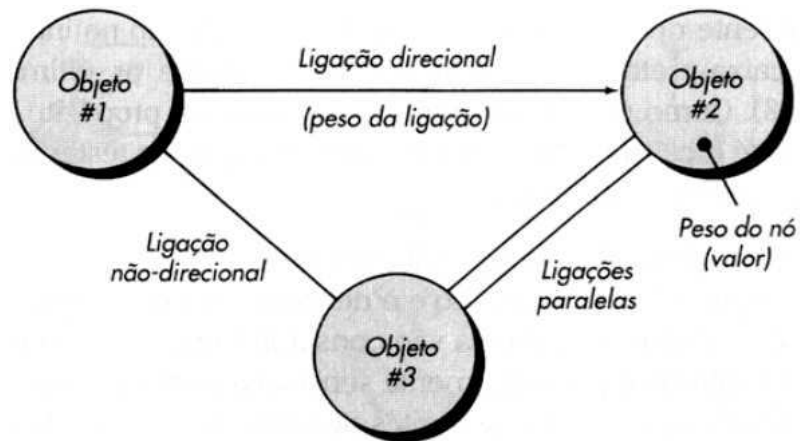


Figura 3 – Notação em grafo (PRESSMAN, 2006)

O modelo preenchido com informações dos objetos, pesos e atributos fica conforme apresentado na figura 4.

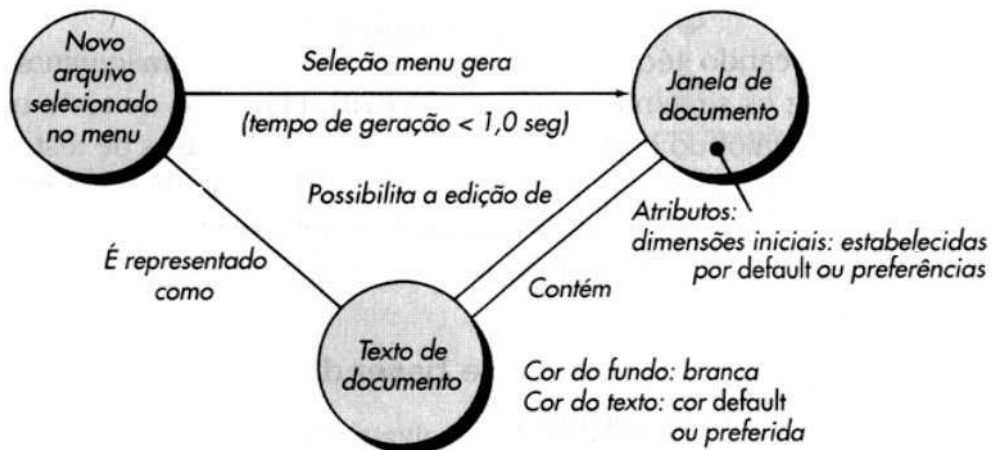


Figura 4 – Exemplo usando a notação em grafo (PRESSMAN, 2006)

Uma seleção de menu de *Novo Arquivo* gera uma *Janela de Documento*. O peso do nó de *Janela de Documento* fornece uma lista de atributos da janela. O peso da ligação indica que a janela deve ser gerada em menos de 1 segundo. As ligações paralelas indicam relações entre *Janela e Documento* e *Texto de Documento*.



Podemos gerar grafos bem mais complexos e então os casos de teste. Esses casos de teste têm o objetivo de encontrar erros em quaisquer das relações.

Beizer (PRESSMAN, 2006) descreve alguns métodos de teste comportamental que podem fazer uso de grafos:

- Modelagem do fluxo de transação: os nós representam passos em alguma transação e as ligações representam as conexões lógicas entre os passos. O diagrama de fluxo de dados pode ser usado para ajudar a criar grafos deste tipo;
- Modelagem de estado finito: os nós representam diferentes estados do software observáveis pelo usuário (uma tela, por exemplo) e as ligações representam as transições que ocorrem para ir de um estado para o outro. O diagrama de transição de estados pode ser usado para ajudar a criar grafos deste tipo;
- Modelagem do fluxo de dados: Os nós são objetos de dados e as ligações são as transformações que ocorrem para traduzir um objeto de dados em outro. Os nós são objetos de programa e as ligações são as conexões sequenciais entre esses objetos. Os pesos das ligações são usados para especificar os tempos de execução necessários, enquanto o software é executado.

O teste utilizando grafos tem início com a definição dos nós e seus respectivos pesos, utilizando o modelo de dados como ponto de partida. Deve-se ter o cuidado de verificar se há objetos não representados no modelo e definir nós de entrada e saída. Após a definição dos nós, devemos estabelecer as ligações e seus pesos.

Podemos identificar ciclos no grafo criado, pois o teste de ciclo pode ser aplicado na abordagem funcional. O grafo vai ajudar na identificação dos ciclos a serem testados.

A partir do grafo pode-se identificar e derivar casos de teste, verificando relações sequenciais que possuam a propriedade da transitividade e o impacto da propagação de modificações de um objeto nos demais.

A simetria (bidirecionalidade) de uma relação também deve ser avaliada, incluindo todas as exceções da relação. Finalmente, cada nó do grafo deve ter uma relação que volta a si próprio. Essas relações *reflexivas* também devem ser testadas.

O projeto de casos de teste tem dois objetivos básicos: o primeiro é conseguir a cobertura de nós, demonstrando que nenhum foi esquecido e que seus pesos estão corretos. O segundo é garantir a cobertura das ligações, verificando as relações transitivas e reflexivas.

### 2.1.3.2 Particionamento de equivalência

O particionamento de equivalência é um método de teste funcional que divide o domínio de entrada de um software em classes de dados, dos quais pode-se derivar casos de teste. Um bom caso de teste descobre sozinho um classe de erros que exigiria a execução de diversos casos de teste até que um erro geral fosse observado. O particionamento de equivalência descobre classes de erros, reduzindo bastante o número de caso de teste (PRESSMAN, 2006).

A avaliação das classes de equivalência é a base do projeto de casos de teste para particionamento de equivalência. Uma classe de equivalência representa um conjunto de estados válidos ou inválidos para condições de entrada. Tipicamente, valores de entrada são valores numéricos específicos, intervalos de valores, condições *booleanas* ou cadeias de caracteres (*strings*). Podemos definir classes de equivalência de acordo com as seguintes diretrizes:

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

### 2.1.3.3 Análise do valor limite

A experiência mostra que os erros tendem a acontecer nas fronteiras dos domínios, ou seja, nos valores limite. Por esta razão, a Análise de Valor Limite (*Boundary Value Analysis, BVA*) foi desenvolvida como técnica de teste. Esta técnica leva a seleção de casos de teste que exercitem os valores limite dos domínios (PRESSMAN, 2006).

A análise do valor limite é uma técnica que complementa o particionamento de equivalência, levando a seleção de casos de teste nos limites das classes. Em vez de focalizar somente nas condições de entrada, a análise do valor limite deriva casos de teste também nos domínios de saída.

As diretrizes para a análise do valor limite são bem semelhantes às do particionamento de equivalência.

1. Se uma condição de entrada especifica um intervalo limitado pelos valores  $a$  e  $b$ , casos de teste devem ser projetados com os valores  $a$  e  $b$  e imediatamente acima e imediatamente abaixo de  $a$  e  $b$ ;
2. Se uma condição de entrada especifica vários valores, casos de teste devem ser desenvolvidos para exercitar os números mínimo e máximo. Valores imediatamente acima e imediatamente abaixo do mínimo e do máximo também são testados;
3. Aplicar as diretrizes 1 e 2 às condições de saída.
4. Se as estruturas de dados internas do software têm limites prescritos, projetar um caso de teste para exercitar a estrutura de dados no seu limite;

#### 2.1.3.4 Teste de comparação

Quando a aplicação do software é crítica (segurança de aeronaves, por exemplo) é necessário que haja redundância, tanto de hardware quanto de software, para que a possibilidade de erro seja minimizada.

Uma técnica possível é desenvolver o mesmo programa por mais de uma equipe, usando a mesma especificação. Assim, as versões podem ser testadas utilizando-se os mesmos dados de entrada, verificando-se se as saídas são idênticas. Estas versões independentes formam a base da técnica de teste funcional conhecida como teste de comparação ou teste de emparelhamento.

O teste de emparelhamento não é a toda prova. Se a especificação estiver incorreta, todas as versões desenvolvidas provavelmente irão refletir os seus erros.

#### 2.1.3.5 Teste da matriz ortogonal

Quando a aplicação tem poucos parâmetros de entrada, e estes parâmetros têm domínios limitados pode-se usar a permutação de valores e testar exaustivamente o processamento dos domínios de entrada. No entanto, à medida que o número de valores de entrada cresce, o teste exaustivo torna-se inviável.

O teste da matriz ortogonal pode ser aplicado quando o número de valores de entrada cresce, mas o domínio continua relativamente pequeno, sendo útil para encontrar erros em regiões com falhas.

O teste da matriz ortogonal permite que sejam projetados testes que forneçam máxima cobertura com um número razoável de casos de teste.

#### 2.1.4 Teste de ambiente, arquitetura e aplicações especializadas

Apesar dos métodos já discutidos até agora serem aplicáveis em todos os ambientes, arquiteturas e aplicações, podemos ter diretrizes e abordagens especiais em alguns casos. As próximas seções serão abordadas tais diretrizes para teste de ambientes, arquiteturas e aplicações especializadas comumente encontradas por engenheiros de software.

##### 2.1.4.1 Teste de interface gráfica com o usuário (GUI)

A criação de interfaces gráficas com o usuário (GUI) tornou-se menos demorada e mais precisa devido a criação de componentes reutilizáveis. Porém sua complexidade aumentou, levando a mais dificuldade no projeto e execução de casos de teste.

Devido a muitas GUI terem comportamento e aparência semelhantes, pode-se ter um conjunto de casos de teste padrão, com a utilização de ferramentas automatizadas, devido ao número de permutações ser em uma quantidade considerável.

##### 2.1.4.2 Teste de arquitetura cliente-servidor

Arquiteturas cliente-servidor são um desafio para os testadores de software, devido a grande variedade de componentes associados. Podemos ter várias plataformas e *hardwares* diferentes, além de bases de dados centralizadas ou distribuídas, por exemplo. Isto significa um aumento de tempo e custo para a realização dos testes.

#### 2.1.4.3 Teste de documentação e dispositivos de ajuda

É importante estender os casos de teste para a documentação do software, pois erros na documentação podem ser tão devastadores para a aceitação do mesmo quanto erros no código-fonte.

O teste de documentação pode ser abordado em duas fases. Na primeira executa-se a revisão e a inspeção, examinando-se documentos quanto à clareza. Na segunda, chamada de *teste ao vivo*, confere-se a documentação em conjunto com o uso do programa.

Os testes de documentação podem fazer parte do conjunto de testes funcionais.

#### 2.1.5 Estratégias de teste de software

Uma estratégia de teste de software bem planejada resulta na construção bem-sucedida do produto final. Ela fornece um roteiro que conduz o processo de teste com o mínimo de esforço, tempo e custo.

O plano de testes deve conter tanto testes de baixo nível (trechos de código), quanto de alto nível (funções).

##### 2.1.5.1 Uma abordagem estratégica do teste de software

Para a condução dos testes é necessário que tenhamos uma estratégia minuciosamente planejada, com gabaritos e métodos bem definidos.

Algumas características genéricas de um bom gabarito de teste são:

- teste começa de "dentro para fora", ou seja, em nível de componente até a integração de todo o sistema;
- diferentes técnicas de teste são adequadas a diferentes momentos;
- teste é conduzido pelo desenvolvedor do software e por um grupo de testes independente;
- teste e a depuração são atividades diferentes, mas a depuração deve ser acomodada em qualquer estratégia de teste;

Uma estratégia de teste pode ser encarada de forma análoga ao ciclo de desenvolvimento espiral, percorrida de dentro para fora, aumentando sua complexidade a cada volta.

O teste de unidade começa no centro da espiral e concentra-se em cada unidade que compõe o software. O passo seguinte é o teste de integração, focado na arquitetura do software. Então é realizado o teste de validação, onde os requisitos são verificados frente ao software construído. Finalmente, é executado o teste de sistema, onde todos os elementos são testados conjuntamente como um todo.

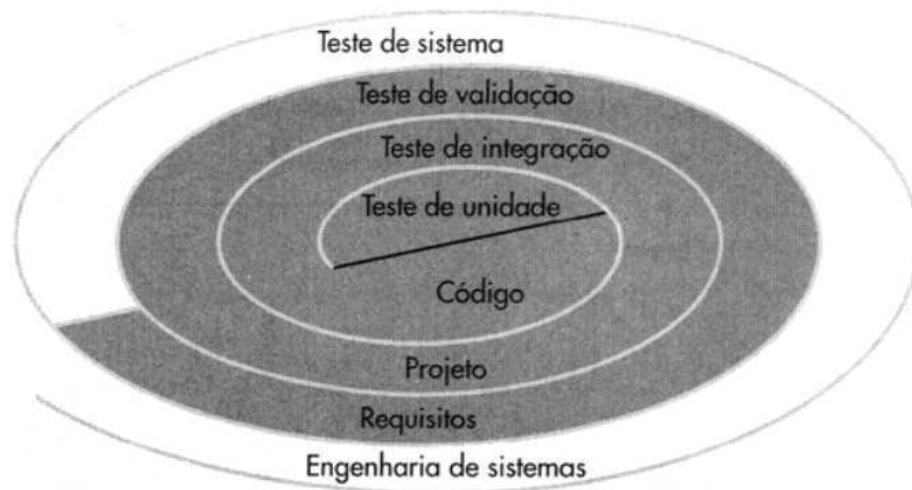


Figura 5 – Estratégia de teste (PRESSMAN, 2006)

O teste de unidade faz uso intenso das técnicas estruturais, exercitando caminhos da estrutura de controle dos módulos, garantindo uma cobertura completa para a máxima detecção de erros. Já as técnicas funcionais são utilizadas mais intensamente no teste de integração. Depois que o software foi integrado (construído), um conjunto de testes de alto nível pode ser conduzido.

#### 2.1.6 Aspectos estratégicos

Para uma estratégia de teste bem definida, alguns aspectos devem ser levados em consideração:

- Além do objetivo principal do teste, que é encontrar erros, devemos avaliar outras características como a usabilidade, portabilidade e manutenabilidade;
- Os objetivos do teste devem ser explícitos e enunciados em termos mensuráveis. Alguns exemplos são: custo de correção de um erro e cobertura do teste;
- Os cenários de interação para cada tipo de usuário

#### 2.1.6.1 Teste de unidade

O teste de unidade foca na menor unidade de projeto do software. Usando a descrição de projeto em nível de componente como guia, caminhos de controle importantes são testados para descobrir erros dentro dos limites de um módulo.

Podem-se iniciar os testes pela interface, partindo a seguir para as estruturas de dados. O próximo passo é testar as condições limite e os caminhos básicos de execução. Finalmente, todos os caminhos de manipulação de erro são testados.

O teste seletivo de caminhos de execução é uma tarefa essencial durante o teste de unidade. O teste de unidade é normalmente considerado como uma tarefa complementar à codificação. Pode-se construir módulos subordinados de teste, que podem ser chamados pelo módulo que está sendo testado, para ajudar na tarefa de teste.

O teste de unidade é simplificado quando um componente com alta coesão é projetado.

#### 2.1.6.2 Teste de integração

Após o teste de unidade, devemos juntar todos os módulos testados. O teste de integração é uma técnica sistemática para construir a estrutura do software enquanto, ao mesmo tempo, conduz testes para descobrir erros associados às interfaces. Podemos ter algumas estratégias de integração diferentes: *Big Bang* e *Incremental*.

Na estratégia *Big Bang*, todos os módulos são colocados juntos e testados de uma só vez.

Na estratégia *Incremental*, os módulos vão sendo adicionados e testados. Esta estratégia pode ser dividida em duas abordagens:

- Descendente (*top-down*) : os módulos são integrados movendo-se de forma descendente na hierarquia. O teste de regressão pode ser conduzido para garantir que os novos módulos não introduzam erros;
- Ascendente (*bottom-up*) : a construção e a integração começa nos módulos inferiores, movendo-se de forma ascendente na hierarquia;

A seleção da estratégia de integração depende das características do software. Em geral, as vantagens de um tipo de estratégia resulta em desvantagem na outra. Também pode ser adotada uma abordagem híbrida.

A principal desvantagem da Integração Descendente é a necessidade de pseudo-controladores, que pode ser compensada pela vantagem de se poder testar as principais funções do software precocemente.

A principal vantagem da Integração Ascendente é que os resultados do software só serão observados quando os últimos módulos forem integrados, compensada pelos casos de teste mais simples, ausência de pseudo-controladores e maior reuso dos módulos inferiores.

À medida que a integração prossegue, o testador deve identificar os módulos críticos. Eles podem ser identificados por algumas características básicas:

- Abordam vários requisitos de software;
- Tem alto escopo de controle, ou seja, situam-se em um ponto relativamente alto na estrutura do software;
- Complexo ou propenso a erros, com alta complexidade ciclomática;
- Tem requisitos de desempenho bem definidos;

#### 2.1.6.3 Teste de regressão

Ao adicionarmos módulos quando fazemos o teste de integração o software é modificado, pois são adicionados novos caminhos, fluxos de dados e funções, que podem interferir no que estava funcionando perfeitamente.

O teste de regressão é a reexecução de testes que já foram conduzidos para garantir que todos os módulos ainda funcionam como antes, ou seja, que as novas modificações não introduziram erros.

O teste de regressão pode ser conduzido manualmente, reexecutando-se um subconjunto de todos os casos de teste, ou pelo uso de ferramentas automatizadas.

O conjunto de testes a executar contém três classes de testes:

- Uma amostra representativa de testes, que vai exercitar todas as funções do software;
- Testes que focam as funções mais provavelmente afetadas pelas alterações;
- Testes dos componentes modificados;

Conforme o teste de integração prossegue, o total de testes de regressão pode crescer de forma significativa, sendo necessário que o plano de testes seja projetado para que sejam



incluídos apenas os testes de módulos afetados pela modificação e os módulos dos caminhos principais do software. Não é prático reexecutar cada função do software.

#### 2.1.6.4 Teste fumaça

O teste fumaça é comumente aplicado a softwares comerciais "de prateleira", como a família *Windows*. Este tipo de software geralmente tem prazos críticos de lançamento (PRESSMAN, 2006).

Nesta estratégia de teste, todo o software é testado quase que diariamente, conforme a integração de novos módulos vai sendo executada. Estes testes freqüentes têm por objetivo descobrir erros críticos que impeçam o funcionamento de todo o produto.

Quando é aplicado a projetos de software complexos e com prazo crítico, o risco da integração é minimizado, a qualidade do produto final é aperfeiçoada, a correção de erros se torna mais simples e o progresso da construção do software pode ser avaliado mais facilmente.

#### 2.1.6.5 Teste de validação

Após o teste de validação o sistema está com todos os seus módulos incorporados e funcionando sem erros, pois os mesmos devem ter sido detectados e corrigidos. Neste ponto pode ser feito o teste de validação, que verificará se os resultados produzidos pelo software são os esperados pelo cliente.

Expectativas razoáveis estão definidas na especificação de requisitos de software, que descreve os atributos visíveis ao usuário. Esta especificação tem critérios de validação que ajudam na condução dos testes de validação.

A conformidade do software é validada por meio de uma série de testes funcionais, que demonstram que a conformidade dos requisitos. São verificados todos os requisitos funcionais e não-funcionais.

#### 2.1.6.6 Teste de recuperação

Uma característica desejada em um sistema é que ele seja tolerante a falhas, pois uma falha de processamento não deve causar a parada de todo o sistema.

O teste de recuperação é um teste de sistema que força o software a falhar de diversos modos e verifica se a recuperação da falha é adequadamente realizada e dentro dos níveis aceitáveis.

#### 2.1.6.7 Teste de segurança

O teste de segurança verifica se os mecanismos de proteção de um sistema são eficientes quando há uma tentativa de invasão ou violação da integridade do sistema.

#### 2.1.6.8 Teste de estresse

O teste de estresse realiza processamentos em um sistema de um modo que demanda recursos em quantidade, frequência ou volume anormais, tentando verificar seu comportamento em situações extremas.

Uma variante do teste de estresse é o teste de sensibilidade, que tenta descobrir combinações de dados, dentro de classes de entrada válidas, que podem causar instabilidade ou processamento inadequado.

## 2.2 Arquitetura de Sistemas de Informação Geográfica

Aplica-se o termo Sistemas de Informação Geográfica (SIG) aos sistemas que fazem tratamento computacional de dados geográficos, recuperando informações tanto alfanumérica quanto espacial. Isso é possível quando a geometria e os atributos dos dados de um SIG estão georreferenciados, ou seja, localizados na superfície terrestre e representados em uma projeção cartográfica.

O requisito de armazenar a geometria dos objetos geográficos e de seus atributos representa uma *dualidade* básica para SIG. Para cada objeto geográfico, o SIG necessita armazenar seus atributos e as várias representações gráficas associadas. Devido à sua ampla gama de aplicações, que inclui temas como agricultura, floresta, cartografia, cadastro urbano e redes de concessionárias (água, energia e telefonia), há pelo menos três grandes maneiras de utilizar um SIG:

- como ferramenta para produção de mapas;
- como suporte para análise espacial de fenômenos;

- como um banco de dados geográficos, com funções de armazenamento e recuperação de informação espacial.

Essas três visões, apesar de parecerem conflitantes, são convergentes e refletem a importância do tratamento da informação geográfica. Para esclarecer ainda mais o assunto, apresentam-se a seguir algumas definições de SIG:

“Um conjunto manual ou computacional de procedimentos utilizados para armazenar e manipular dados georreferenciados” (ARONOFF, 1989);

“Conjunto poderoso de ferramentas para coletar, armazenar, recuperar, transformar e visualizar dados sobre o mundo real” (BURROUGH, 1986);

“Um sistema de suporte à decisão que integra dados referenciados espacialmente num ambiente de respostas a problemas” (COWEN, 1988);

A multiplicidade do uso das tecnologias de SIG está refletida em cada uma dessas visões, mostrando o aspecto interdisciplinar desta tecnologia. A partir desses conceitos, é possível indicar as principais características de SIG:

- Inserção e integração, em uma base de dados única, informações espaciais provenientes de dados cartográficos, dados censitários e cadastro urbano e rural, imagens de satélite, redes e modelos numéricos de terreno;
- Mecanismos de combinação de informações, através do uso de algoritmos de manipulação e análise, bem como para executar operações de consulta, recuperação, visualização e plotagem do conteúdo da base de dados georreferenciados.

### 2.2.1 Estrutura geral de um SIG

Os seguintes componentes estão presentes em um SIG:

- Interface com usuário;
- Entrada e integração de dados;
- Funções de consulta e análise espacial;
- Visualização e plotagem;

- Armazenamento e recuperação de dados (organizados sob a forma de um banco de dados geográficos).

O relacionamento entre esses componentes se dá de forma hierárquica. No nível mais alto (nível usuário), a interface homem-máquina define como o sistema é operado e controlado. O nível intermediário deve ter mecanismos para o processamento de dados espaciais (entrada, edição, análise, visualização e saída).

No nível mais baixo (nível interno), o armazenamento e recuperação de dados espaciais e seus atributos são realizados pelo sistema de gerência de bancos de dados geográficos.

As funções de processamento de um SIG fazem operações sobre dados em uma área de trabalho em memória principal. Mecanismos de seleção e consulta fazem a ligação entre os dados geográficos e funções de processamento do SIG, definindo restrições sobre o conjunto de dados. Alguns exemplos são:

- Recuperar os dados relativos à carta de Guajará-Mirim " (restrição por definição de região de interesse);
- Recuperar as cidades do Estado de São Paulo com população entre 100.000 e 500.000 habitantes (consulta por atributos não-espaciais);
- Mostrar os postos de saúde num raio de 5 km do hospital municipal de São José dos Campos (consulta com restrições espaciais).

A figura 6 mostra os principais componentes de um SIG e seus relacionamentos. A necessidade e os objetivos de cada sistema faz a implementação de forma distinta, porém todos os componentes devem estar presentes em um SIG.

### 2.2.2 Gerência de dados em um SIG

A forma de gerenciamento dos dados geográficos é a principal diferença entre os SIG. Existem três arquiteturas que usam os recursos do SGBD: dual, integrada baseada em sistemas gerenciadores de banco de dados (SGBD) relacionais e integrada baseada em extensões espaciais sobre SGBD objeto-relacionais.

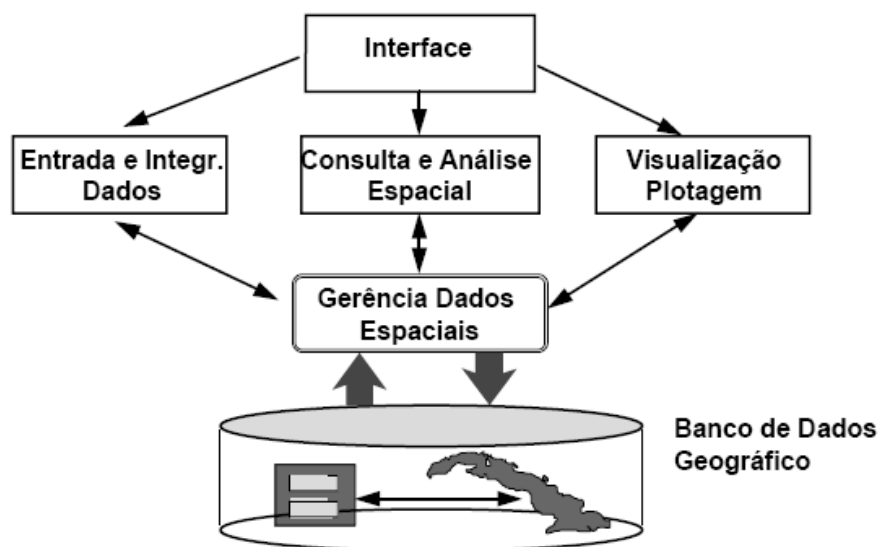


Figura 6 - Estrutura Geral de Sistemas de Informação Geográfica (DAVIS, 2001)

#### 2.2.2.1 Arquitetura Dual

A estratégia dual utiliza um SGBD relacional para o armazenamento dos atributos convencionais dos objetos geográficos na forma de tabelas e arquivos para guardar as representações geométricas desses objetos. No modelo relacional, os dados são armazenados em tabelas, onde as linhas correspondem aos dados e as colunas aos atributos.

Um SGBD relacional é usado para a entrada de atributos não espaciais, onde é imposto um identificador único ou rótulo para cada entidade gráfica inserida no sistema. Por meio desse rótulo é feita uma ligação lógica com os seus atributos não-espaciais, que estão nas tabelas do SGBD, conforme a figura 7.

A principal vantagem desta estratégia é poder utilizar os SGBD relacionais disponíveis no mercado. Porém, esta estrutura dificulta a otimização de consultas, gerência de transações e controle de integridade e concorrência, pois os objetos espaciais estão fora do controle do SGBD. Alguns exemplos de sistemas comerciais baseados na estratégia dual: *ARC/VIEW*, *MGE* e o *SPRING* (CAMARA *et al.*, 1996). As principais desvantagens desta arquitetura são:

- Dificuldades no controle e manipulação dos dados espaciais;
- Dificuldade em manter a integridade entre a componente espacial e a componente alfanumérica;

- Consultas mais lentas, pois são processadas separadamente. A parte convencional da consulta é processada pelo SGBD separado da parte espacial, que é processada pelo aplicativo utilizando os arquivos proprietários;
- Falta de interoperabilidade entre os dados. Cada sistema produz seu próprio arquivo proprietário sem seguir um formato padrão, o que dificulta a integração desses dados.

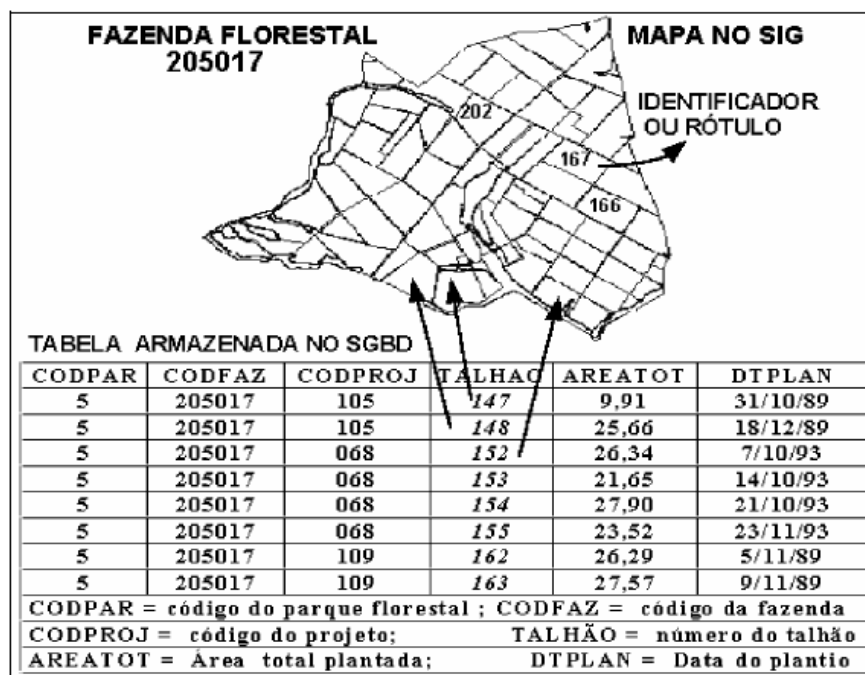


Figura 7 - Estratégia dual para bancos de dados geográficos. (DAVIS, 2001)

#### 2.2.2.2 Arquitetura Integrada para Gerência de Dados

O uso da arquitetura integrada consiste em armazenar tanto os dados alfanuméricos quanto os dados espaciais em um SGBD. O uso dos recursos de um SGBD é a principal vantagem, pois o SGBD controla as transações, a integridade dos dados e a concorrência.

A arquitetura integrada utiliza campos do tipo *BLOB* (*Binary Large Objects*), que são campos longos que armazenam a componente espacial do dado.

A arquitetura Integrada, mostrada a seguir na figura 8b, consiste em armazenar todo o dado espacial em um SGBD, tanto sua componente espacial como a parte alfanumérica. Sua principal vantagem é a utilização dos recursos de um SGBD para controle e manipulação de dados espaciais, como gerência de transações, controle de integridade e concorrência. Sendo assim, a manutenção de integridade entre a componente espacial e alfanumérica é feita pelo

SGBD. Há duas alternativas para a arquitetura integrada: (a) baseada em SGBD relacionais; (b) baseada em extensões espaciais sobre SGBD objeto-relacionais.



Figura 8a - Arquitetura Dual  
(DAVIS, 2001)



Figura 8b - Arquitetura Integrada  
(DAVIS, 2001)

Como já mencionado, a arquitetura integrada baseada em um SGBD relacional utiliza campos longos, chamados de BLOB, para armazenar a componente espacial do dado. Os BLOB armazenam tipicamente imagens, áudio ou outros objetos multimídia.

Suas principais desvantagens são:

- Impossibilidade de capturar os dados espaciais de forma semântica, pois os BLOB tratam o campo como uma cadeia de caracteres binária;
- SIG deve implementar os métodos de acesso espacial e otimizar as consultas: como os dados espaciais são tratados como uma cadeia binária de caracteres, não existem mecanismos que manipulem esses dados de forma satisfatória;
- A linguagem SQL não dispõe de mecanismos satisfatórios para o tratamento de BLOB.

O outro tipo de arquitetura integrada usa extensões espaciais desenvolvidas sobre SGBD objeto-relacionais. O armazenamento, acesso e análise dos dados espaciais de formato vetorial são realizados por funcionalidades e procedimentos dessas extensões. As principais desvantagens desta arquitetura são:

- Não há mecanismos de controle de integridade sobre os dados espaciais;
- Não há padronização das extensões da linguagem SQL;

A definição de novos tipos de dados e métodos ou operadores de manipulação desses tipos são funcionalidades oferecidas pelos SGBD objetos-relacionais (ou SGBD extensíveis). Este modelo de dados e linguagem de consulta pode ser estendido. Por essa razão, um SGBDOR é mais adequado para o tratamento de dados complexos, como os dados geográficos, do que um SGBDDR, pois esses recursos não são oferecidos por ele.

Um SGBDOR que possui uma extensão para tratar dados espaciais deve ter as seguintes características:

- Existência de tipos de dados espaciais (TDE), como ponto, linha e região, em seu modelo de dados e possibilidade de manipulação, assim como os tipos alfanuméricos básicos (inteiros, *strings*, etc.);
- Linguagem SQL estendida para suportar operações e consultas espaciais sobre TDE;
- Adaptar outras funções de níveis mais internos para manipular TDE eficientemente, tais como métodos de armazenamento e acesso (indexação espacial) e métodos de otimização de consultas (junção espacial).

Portanto, além dos TDE, as extensões espaciais fornecem operadores e funções que são utilizados, juntamente com a linguagem de consulta do SGBD, para consultar relações espaciais e executar operações sobre TDE. Além disso, fornecem métodos de acesso eficiente de TDE através de estruturas de indexação, como R-tree e QuadTree.

Algumas extensões disponíveis no mercado são: *Oracle Spatial* (RAVADA e SHARMA, 1999), *IBM DB2 Spatial Extender* (IBM, 2001), *Informix Spatial Datablade* (IBM, 2002), além da extensão *PostGIS* (RAMSEY, 2002) para o SGBD PostgreSQL, que é objeto-relacional, gratuito e de código fonte aberto.

Todas essas extensões baseiam-se nas especificações do *Open Geospatial Consortium* (OGC, 1996), que especifica padrões, seguidos por todas as extensões. Porém, essas extensões possuem variações relevantes entre os modelos de dados, semântica dos operadores espaciais e mecanismos de indexação.

O *OpenGIS* é um consórcio formado por organizações públicas e privadas envolvidas com SIGs. Seu principal objetivo é a criação e gerenciamento de uma arquitetura padrão para geoprocessamento. As principais características desta padronização são:



- Criação de um modelo universal de dados espaço-temporais e de processos, chamado modelo de dados *OpenGIS*;
- Uma especificação para cada uma das principais linguagens de consulta a banco de dados para implementar o modelo de dados *OpenGIS*;
- Uma especificação para cada um dos principais ambientes computacionais distribuídos para implementar o modelo de processo *OpenGIS*.

### 2.2.3 Uma visão geral da tecnologia de SIG

Hoje em dia, diferentes alternativas estão relacionadas ao conceito de “sistemas de informação geográfica”, havendo uma grande diversificação de oferta de sistemas deste tipo. Existem pelo menos quatro grandes tecnologias complementares:

- Os "GIS desktop", que oferecem interfaces amigáveis e um conjunto de funcionalidades crescente;
- Os "Gerenciadores de Dados Geográficos", que fazem o armazenamento dos dados espaciais em ambiente multiusuário;
- Os "Componentes GIS", que são ambientes de programação que fornecem insumos para que o usuário desenvolva aplicativos geográficos próprios;
- Os "Servidores Web de Dados Geográficos", utilizados para publicação e acesso a dados geográficos via Internet.

Os “GIS desktop” são sistemas que herdaram conceitos da Cartografia, oferecendo suporte a banco de dados limitado e cujo paradigma típico de trabalho é o mapa (chamado de “cobertura” ou de “plano de informação”). São sistemas desenvolvidos a partir da década de 80 e podem ser considerados a primeira geração de SIG. Essa geração também pode ser caracterizada como sistemas orientados a projeto (*project-oriented GIS*).

No início da década de 90 chegou ao mercado a segunda geração de SIG, caracterizada pelo uso de ambientes cliente-servidor, acoplado a SGBD relacionais e com pacotes adicionais para o processamento de imagens. Essa geração também pode ser vista como sistemas para suporte a instituições (*enterprise-oriented GIS*).

A terceira geração de SIG (bibliotecas geográficas digitais ou centros de dados geográficos), tem como características o uso de grandes bases de dados geográficos e acesso

de redes locais remotas, utilizando a interface *web*. O uso de tecnologias de bancos de dados distribuídos e federativos, possibilitou o compartilhamento de dados entre diversas instituições. Esses sistemas deverão seguir os requisitos de interoperabilidade, de maneira a permitir o acesso de informações espaciais por SIG distintos. Essa geração pode ser vista como orientada a troca de informações entre uma instituição e os demais componentes da sociedade (*society-oriented GIS*).

#### 2.2.3.1 Aplicativos SIG para desktop (GIS desktop)

Operações gráficas e de análise espacial sobre arquivos (*flat files*) caracterizam a primeira geração de SIG. A sua ligação com os bancos de dados é parcial ou inexistente, pois parte das informações descritivas se encontra no sistema de arquivos. Esse tipo de tecnologia é mais adequada à realização de projetos de análise espacial sobre regiões de pequeno e médio porte, pois esses sistemas enfatizam o aspecto de mapeamento. Os dados podem ser inseridos sem definição prévia do esquema conceitual, assemelhando-se a ambientes CAD que possuem a capacidade de representar projeções cartográficas e de associar atributos a objetos espaciais. Por essas características, tais ambientes não são adequados para a construção de grandes bases de dados espaciais.

Em um aplicativo SIG tradicional, os atributos descritivos são armazenados em tabelas e as geometrias em arquivos de formato proprietário, como os arquivos do tipo *shapefile* do *ArcView*. Esse tipo de SIG foi concebido originalmente para simples consultas e apresentação de dados, porém várias novas funcionalidades foram agregadas ao longo do tempo como combinação de dados vetoriais e matriciais, maior processamento de imagens, *scripts* de programação de tarefas, ferramentas de análise espacial e uso de bancos de dados geográficos como o *Oracle Spatial*.

#### 2.2.3.2 A segunda geração: bancos de dados geográficos

O aparecimento de gerenciadores de bancos de dados geográficos, que armazenam tanto a geometria quanto os atributos evidenciou as vantagens deste tipo de tecnologia: (a) evitar problemas de controle de integridade típicos do ambiente *desktop*, permitindo o acesso concorrente aos dados; (b) facilitar a integração com as bases corporativas já existentes, como sistemas legados, que já utilizam SGBD relacionais.

Um SGBD apresenta os dados numa visão independente dos sistemas aplicativos, além de garantir três requisitos importantes: eficiência (acesso e modificações de grandes volumes de dados); integridade (controle de acesso por múltiplos usuários); e persistência (manutenção de dados por longo tempo, independentemente dos aplicativos que acessem o dado). O uso de SGBD permite ainda realizar, com maior facilidade, a interligação de banco de dados já existente com o sistema de geoprocessamento. Estes gerenciadores possuem dois componentes, geralmente de fabricantes distintos: um SGBD com suporte a dados geográficos e uma "camada de acesso", que fornece um ambiente de armazenamento e recuperação, visível externamente ou integrado a um aplicativo SIG para desktop. As camadas de acesso interagem com estes servidores, para fornecer um ambiente de armazenamento e recuperação, através de uma interface de programação (API). Dentre os gerenciadores de bancos de dados com capacidades de armazenamento de dados geográficos, estão o *ORACLE*, *PostgreSQL* e *MySQL*. Dentre as API para acesso a esses bancos, estão o *ArcSDE* da ESRI e a *TerraLib* desenvolvida pelo INPE.

#### 2.2.3.3 A terceira geração: bibliotecas geográficas digitais

Uma biblioteca geográfica digital ou centro de dados geográfico é um banco de dados geográfico compartilhado por um conjunto de instituições. Além de armazenar dados geográficos, ela deve armazenar descrições relativas aos dados (metadados) e documentos multimídia associados. Além disso, esta biblioteca deve ser acessível remotamente. O cerne de uma biblioteca geográfica digital é um grande banco de dados geográficos.

O ambiente deve garantir o acesso concorrente a usuários, incluir uma linguagem de consulta, métodos de seleção e folheamento (*browsing*). A solução mais comum é o acoplamento de um servidor de dados geográficos, que possibilita a difusão de dados na internet. Além disso, adota-se também um servidor de imagens, que responda a pedidos remotos e envie imagens de tamanho fixo nos formatos GIF ou JPEG. Essa solução permite configurar o servidor para responder a diferentes tipos de consulta, sem requerer que todos os dados a ser transmitidos sejam pré-computados. Como exemplo, tem-se o *Internet Map Server*, da ESRI e o *MapServer* (produto de software livre). O protocolo WMS (*Web Map Server*) do consórcio *OpenGIS* encapsula essa funcionalidade.

#### 2.2.3.4 Bibliotecas de componentes

Uma tendência crescente é fornecer um ambiente de componentes, com tipos de dados geográficos básicos e métodos de acesso e apresentação. No caso dos produtos *MAPOBJECTS* da ESRI e *MAPX* da MAPINFO, a linguagem de programação utilizada é o *Visual Basic*.

Os objetos OLE (*Object Linking and Embedding*) e ODBC (*Open Database Connectivity*) do Windows fazem a comunicação com outras aplicações. A alternativa (utilizada por projetos de software livre) é oferecer bibliotecas em linguagens como C ou C++. Esse é o caso da biblioteca *TerraLib* (2003). Nesse caso, toda a funcionalidade do sistema está disponível, mas precisa ser remontada pelo programador.

### 2.3 Lógica Nebulosa

O modelo mais utilizado para o tratamento da informação vaga e imprecisa é a teoria dos conjuntos nebulosos. Através de graduações na pertinência de um elemento de dada classe, o modelo possibilita que um elemento pertença a uma classe com maior ou menor intensidade. Enquanto que na teoria dos conjuntos “clássica” a pertinência só pode assumir valores 0 ou 1, nos conjuntos nebulosos os valores de pertinência podem assumir quaisquer valores no intervalo dos números reais  $[0,1]$  (BITTENCOURT, 2001).

A grande vantagem desse modelo é sua capacidade de capturar, com um formalismo matemático, conceitos relativos como graus de satisfação, conforto, adequação, etc. (OLIVEIRA JR., 1999).

A teoria dos conjuntos nebulosos foi desenvolvida nos anos 60 por Lotfi A. Zadeh (ZADEH, 1965). Enquanto a tradicional lógica booleana trata o mundo real como tendo apenas duas classes (verdadeiro ou falso), a lógica nebulosa atribui às variáveis reais (temperatura, pressão, tensão, etc.) classes de conjuntos associados a termos lingüísticos (alto, baixo, médio, quase baixo). A grande vantagem dessa metodologia é a preservação do dado, ou seja, descreve de maneira satisfatória a riqueza das informações fornecidas.

Podemos tomar como exemplo um conjunto de valores de temperatura ambiente. Não há como definir claramente o que é uma temperatura quente, morna ou fria, ou seja, não há como definir os elementos de cada subconjunto. A teoria dos conjuntos nebulosos foi desenvolvida por Zadeh para tratar os aspectos vagos desse tipo de informação.

Quando é utilizada em sistemas baseados em conhecimento, que tem um contexto lógico, a teoria de conjuntos nebulosos é conhecida como lógica nebulosa, lógica difusa ou lógica *fuzzy* (SANDRI e CORREA, 1999) .

### 2.3.1 Conceitos de Conjunto nebuloso e função de pertinência

Segundo Oliveira Jr. (1999), para generalizar a idéia de conjuntos ordinários, foi criado o conceito de conjunto nebuloso (*fuzzy set*). Esses conjuntos ordinários podem ser denominados abruptos (*crisp sets*).

O conjunto nebuloso funciona como um predicado lógico, onde os valores estão no intervalo  $[0,1]$ . A seguir, são apresentados alguns conceitos e idéias básicas a respeito dos conjuntos nebulosos.

A função característica de um conjunto  $A \subseteq X$  assume o valor 1 em elementos de A e 0 em elementos  $X - A$ , onde:

$X$  = conjunto universo

$C[A]: X \rightarrow \{0, 1\}$ , é definida por:

$C[A](v) = 1$  para  $v \in A$  ; 0 para  $v \in A$

$X - A = \{v: v \in X \wedge v \in A\}$

Assim, observamos que:

1.  $C[A]$  só assume valores em  $\{0, 1\}$ ,
2. A transição da condição de pertinência para a de não-pertinência (ou vice-versa) é abrupta.

Se  $C[A]$  se estender a  $[0, 1]$ , pode-se obter um conjunto nebuloso, em que há elementos que podem ou não pertencer simultaneamente ao conjunto.

A forma que cada ponto de entrada é mapeado em um valor de pertinência no intervalo  $[0,1]$  é a função de pertinência.

Seja  $X$  um conjunto arbitrário qualquer (universo de discurso), e  $A \subseteq X$  um subconjunto qualquer, um conjunto nebuloso é um par  $(A, \mu_A(x))$ , sendo  $\mu_A(x): X \rightarrow [0, 1]$  uma função de pertinência, que é o grau em que os elementos do conjunto A pertencem ao conjunto nebuloso  $(A, \mu_A(x))$ .

Por exemplo, a figura 9 representa graficamente o conjunto nebuloso dos homens de meia idade.

O conjunto seria formado pelos seguintes pares:

$$V = \{0/5; 0/10; 0,2/15; 0,3/20; 0,4/25; 0,6/30; 0,8/35; 1/40; 0,8/45; 0,6/50; 0,4/55; 0,3/60; 0,2/65; 0/70; 0/75\}$$

Pode-se concluir que a função de pertinência caracteriza completamente um conjunto *fuzzy*.

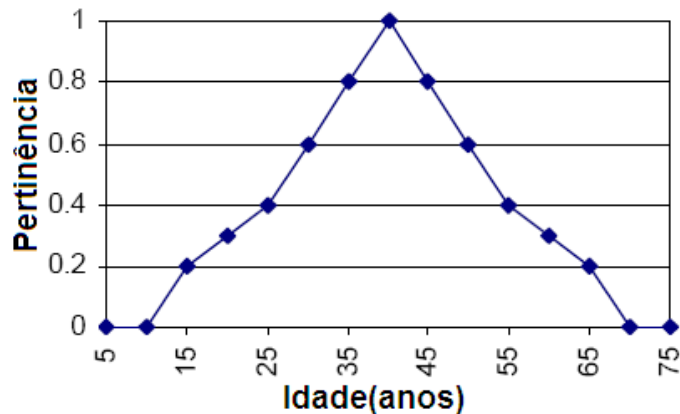


Figura 9 - Função de pertinência para “homens de meia-idade” (MARANHÃO, 2005)

### 2.3.2 Operações com Conjuntos Nebulosos

Nos conjuntos nebulosos, assim como nos conjuntos *crisp*, são necessárias algumas operações de interseção, união e negação, entre outras.

#### 2.3.2.1 Operadores

Para definir as operações de união e interseção, Zadeh (1965) fez uso das funções de máximo e mínimo, respectivamente. Por exemplo, o mínimo ou o máximo de dois elementos,  $a$  e  $b$  é definido pelas equações 1 e 2.

$$\begin{aligned} a \wedge b &= \min(a, b) = a \text{ se } a \leq b \\ &= b \text{ se } a > b \end{aligned} \tag{1}$$

$$\begin{aligned} a \vee b &= \max(a, b) = a \text{ se } a \geq b \\ &= b \text{ se } a < b \end{aligned} \tag{2}$$

Sejam A e B subconjuntos difusos de X. Sua **união** (equação 3) é um subconjunto difuso  $A \cup B$ , onde " $\vee$ " é utilizado para representar uma disjunção lógica (OLIVEIRA JR., 1999).

$$(A \cup B)(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x), \forall x \in X \quad (3)$$

Sejam A e B subconjuntos difusos de X. Sua **interseção** (equação 4) é um subconjunto difuso  $A \cap B$ , onde " $\wedge$ " representar uma conjunção lógica (OLIVEIRA JR., 1999).

$$(A \cap B)(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x), \forall x \in X \quad (4)$$

Seja A um subconjunto nebuloso de X. O **Complemento ou Negação** de A denotado por  $\text{neg}(A)$  é o conjunto nebuloso definido pela equação 5 (OLIVEIRA JR., 1999).

$$\text{Neg}(A) = X - A \text{ ou } (\text{neg}(A))(x) = 1 - \mu_A(x), \forall x \in X. \quad (5)$$

Sejam A e B subconjuntos nebulosos de X, o **produto algébrico** de A e B, representado por  $AB$  é definido pela equação 6 (FONSECA, 2003).

$$\mu_{AB}(x) = \mu_A(x) \mu_B(x), \forall x \in X \quad (6)$$

Sejam A e B subconjuntos nebulosos de X, a **soma algébrica** de A e B (equação 7), representada por  $A \oplus B$  é definida pela soma das suas respectivas funções de pertinência menos o seu produto algébrico.

$$\mu_{A \oplus B}(x) = \mu_A(x) + \mu_B(x) - \mu_{AB}(x), \forall x \in X \quad (7)$$

Um operador **gama** (equação 8) é definido a partir do produto algébrico e da soma algébrica.

$$\text{Operador Gama} = (\text{Soma Algébrica})^\gamma * (\text{Produto Algébrico})^{1-\gamma} \quad (8)$$

Os valores de  $\gamma$  variam no intervalo de 0 a 1, sendo que para  $\gamma=0$  o resultado se iguala ao produto algébrico *fuzzy* enquanto que para  $\gamma=1$  o resultado é o mesmo que o da soma algébrica *fuzzy* (BONHAM-CARTER, 1994).

Operações de união e interseção, são casos específicos de situações mais abrangentes de agregação de conjuntos nebulosos (FONSECA, 2003).

Para generalização dos conceitos de *união* e *interseção*, utilizam-se as t-conormas e t-normas, respectivamente (SOUZA, 2001).

As t-normas (**T**) generalizam o conceito da operação de interseção, e devem satisfazer os axiomas:

1. condições de contorno :  $x \mathbf{T} 0 = 0, \forall x \in [0,1]$   
 $x \mathbf{T} 1 = x, \forall x \in [0,1]$
2. Propriedade comutativa :  $x \mathbf{T} y = y \mathbf{T} x$
3. Propriedade associativa :  $x \mathbf{T} (y \mathbf{T} z) = (x \mathbf{T} y) \mathbf{T} z$
4. Condições monotônicas : para  $z \mathbf{T} w \leq x \mathbf{T} y$  se  $z \leq x$  e  $w \leq y$

Os operadores *mínimo* e *produto algébrico*, entre outros, são exemplos das t-normas.

As t-conormas (**⊥**) generalizam o conceito da operação de união, e devem satisfazer os axiomas:

1. condições de contorno :  $x \mathbf{\perp} 0 = x, \forall x \in [0,1]$   
 $x \mathbf{\perp} 1 = 1, \forall x \in [0,1]$
2. Propriedade comutativa :  $x \mathbf{\perp} y = y \mathbf{\perp} x$
3. Propriedade associativa :  $x \mathbf{\perp} (y \mathbf{\perp} z) = (x \mathbf{\perp} y) \mathbf{\perp} z$
4. Condições monotônicas : para  $z \mathbf{\perp} w \leq x \mathbf{\perp} y$  se  $z \leq x$  e  $w \leq y$

Os operadores *máximo* e *soma algébrica*, entre outros, são exemplos das t-conormas. Tanto os axiomas da t-norma como da t-conorma., são satisfeitas pelo operador *gama*.

Quando os conjuntos são *crisp*, pode-se notar que as t-normas e t-conormas são reduzidas aos operadores clássicos de união e interseção (SANDRI e CORREA,1999).



### 2.3.2.2 Propriedades Algébricas

Através da utilização das definições de união, interseção e complemento, é possível verificar que diversas propriedades algébricas dos conjuntos ordinários também se aplicam a conjuntos nebulosos (TANSCHKEIT, 1999):

Involução	$(A')' = A$
Idempotência	$A \cap A = A$ e $A \cup A = A$
Comutatividade	$A \cap B = B \cap A$ e $A \cup B = B \cup A$
Associatividade	$(A \cap B) \cap C = A \cap (B \cap C)$ e $(A \cup B) \cup C = A \cup (B \cup C)$
Distributividade	$(A \cap B) \cup C = (A \cap B) \cup (A \cap C)$ e $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
Absorção	$A \cap (A \cup B) = A$ e $A \cup (A \cap B) = A$
Lei Transitiva	se $A \subset B$ e $B \subset C \Rightarrow A \subset C$
Leis de De Morgan	$(A \cap B)' = A' \cup B'$ e $(A \cup B)' = A' \cap B'$

Como o conjunto vazio e o universo são definidos como 0 e 1, respectivamente, as seguintes propriedades válidas:

$$A \cap \emptyset = \emptyset \text{ e } A \cap X = A \quad (9)$$

$$A \cup \emptyset = A \text{ e } A \cup X = X, \quad (10)$$

As propriedades de conjuntos clássicos  $A \cap A' = \emptyset$  e  $A \cup A' = X$  **não** se verificam para conjuntos fuzzy :

$$\mu_{A \cap A'}(x) = \mu_A(x) \wedge (1 - \mu_A(x)) \neq 0 \Rightarrow A \cap A' \neq \emptyset \quad (11)$$

$$\mu_{A \cup A'}(x) = \mu_A(x) \vee (1 - \mu_A(x)) \neq 1 \Rightarrow A \cup A' \neq X \quad (12)$$

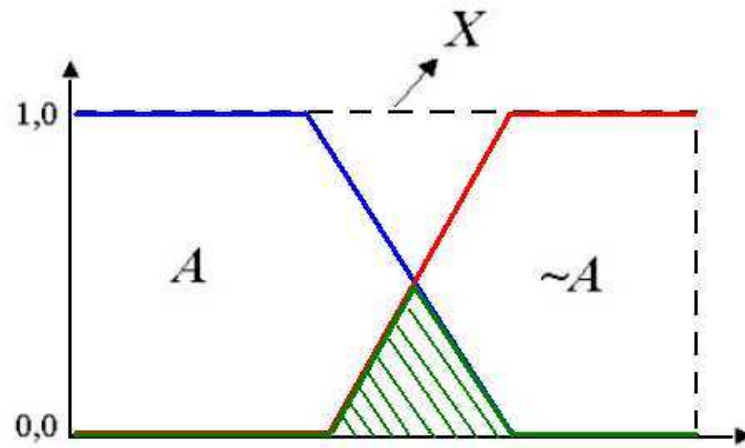


Figura 10 - Interseção de conjuntos nebulosos (MARANHÃO, 2005)

Pela figura 10, temos:

Caso clássico:  $A \cap \sim A = \emptyset$

Caso *fuzzy*:  $A \cap \sim A \neq \emptyset$

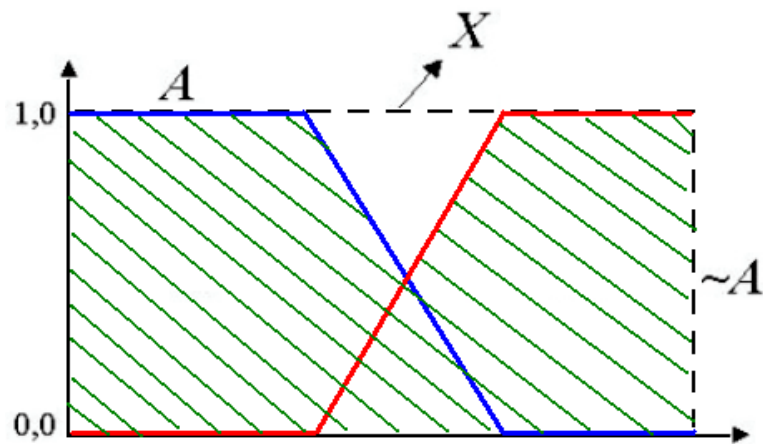


Figura 11 - União de conjuntos nebulosos (MARANHÃO, 2005)

Pela figura 11, temos:

Caso clássico:  $A \cup \sim A = X$

Caso *fuzzy*:  $A \cup \sim A \neq X$

### 2.3.2.3 Variáveis Linguísticas

Segundo Tanscheit (1999), uma variável linguística é aquela cujos valores são rótulos (*labels*) de conjuntos nebulosos. Por exemplo, poderíamos ter um processo cujos valores de temperatura assumidos fossem *pequena*, *média*, *alta*, *etc.*, com esses valores sendo descritos

por meio de conjuntos nebulosos. De forma geral, as variáveis podem conter sentenças ou valores em uma linguagem especificada.

Nesse caso, a variável é chamada lingüística. Por exemplo, os valores da variável nebulosa temperatura poderiam ser expressos por *alta*, *não alta*, *muito alta*, *bastante alta*, *não muito alta*, *alta mas não muito alta*. Os valores do conjunto nebuloso são sentenças formadas a partir do rótulo *alta*, da negação *não*, dos conectivos *e* e *mas* e dos modificadores *muito* e *bastante*.

A caracterização sistemática de fenômenos complexos ou mal definidos é a principal função das variáveis lingüísticas. Essencialmente, a utilização de descrições lingüísticas empregadas por seres humanos, e não de variáveis quantificadas, permite que sistemas muito complexos, para ser analisados por meios matemáticos convencionais sejam tratados. Quem descreve e quantifica as variáveis lingüísticas é um especialista do sistema.

#### 2.3.2.4 Funções de implicação ou proposição difusa

Uma declaração de implicação nebulosa ou condicional nebulosa descreve uma relação entre variáveis lingüísticas. Consideremos dois conjuntos nebulosos  $A$  e  $B$ , que contêm valores lingüísticos dos universos  $X$  e  $Y$ , respectivamente. Podemos formar, matematicamente, uma declaração nebulosa da forma *SE A ENTÃO B* como:

$$R : \text{SE } A \text{ ENTÃO } B = A \rightarrow B = A \times B \quad (13)$$

A relação nebulosa  $A \times B$  denota, nesse caso, a implicação  $A \rightarrow B$  no produto cartesiano dos dois universos  $X \times Y = \{(x, y) \mid x \in X, y \in Y\}$ . (TANSCHKEIT, 1999)

Chamamos de base de regras a combinação de várias regras. Especialistas definem as funções de pertinência e as regras.

Proposição difusa é outro termo que denota função de implicação. Segundo Oliveira Jr. (1999) a proposição difusa expressa relações entre variáveis lingüísticas e conjuntos difusos, podendo apresentar composições por meio de conectivos e transformadores. O processo de avaliação de proposições consiste em aferir o “nível de verdade” ou pertinência apresentado em relação a uma dada situação.

Segundo Tanscheit (1999), os operadores de implicação mais utilizadas são *mínimo* e *produto algébrico*. Para sentenças com um antecedente apenas (do tipo SE  $A$  ENTÃO  $B$ ), tem-se:

$$\text{mínimo} : \mu_B(x, y) = \min(\mu_A(x), \mu_B(y)) = \mu_A(x) \wedge \mu_B(y) \quad (14)$$

$$\text{produto} : \mu_B(x, y) = \mu_A(x) \times \mu_B(y) \quad (15)$$

#### 2.3.2.5 Processo de Agregação

Agregação é o processo de combinação de conjuntos nebulosos que representam cada regra de saída em um único conjunto de saída. A entrada do processo de agregação é a lista das funções de saída retornadas pelo processo de implicação de cada regra.

O processo de agregação é comutativo, ou seja, a ordem de execução de cada regra não muda o resultado final. (MATLAB, 2000).

Os métodos mais comumente utilizados são *máximo* e *soma algébrica*, sendo implementados em programas de computador.

#### 2.3.2.6 Defuzzificação

O conjunto nebuloso resultante do processo de agregação é a entrada do processo de *fuzzificação*, sendo a saída um número. A questão é que os conjuntos nebulosos não são entendidos pelo mundo “físico”, sendo necessário gerar números que expressem ou resumam da melhor forma possível a informação contida nesses resultados. (OLIVEIRA JR.,1999)

O procedimento consiste em identificar o domínio das variáveis de saída em um universo de discurso correspondente. Com a ação de controle nebulosa inferida evolui-se uma ação de controle não-nebulosa. (SANDRI e CORREA,1999)

A *defuzzificação* pode ser feita por três métodos distintos: valor máximo, média de valores máximos ou cálculo do centróide. O método do máximo calcula como valor *defuzzificado*, o ponto do universo de discurso em que a função de pertinência é máxima. Esse método pode provocar certa confusão nos casos em que a função de pertinência possui mais de um valor máximo. Então foi proposta a utilização da média dos máximos para achar o valor a ser *defuzzificado*. Porém, numa situação como a mostrada pela figura 12, a média dos

máximos produziria um valor onde a função de pertinência é zero, o que não faria muito sentido (ZADEH, 1965).

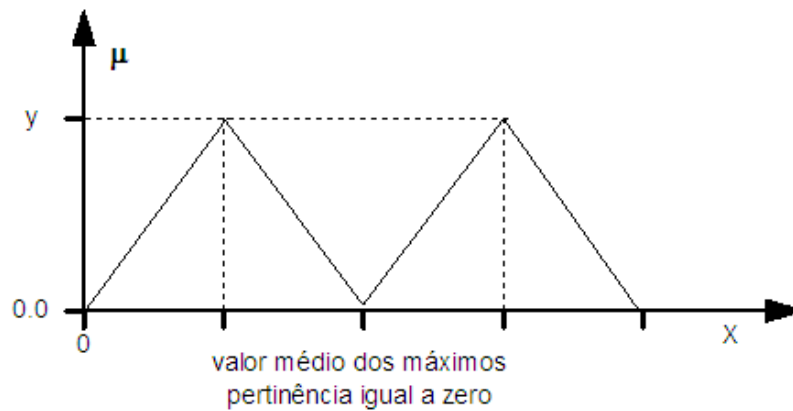


Figura 12 – Conjunto em que a defuzzificação produziria um valor igual a zero  
(BERNARDO FILHO, 1999)

Diante das limitações dos métodos de máximo e média dos máximos, foi proposto o método do centróide, onde a saída determinística se baseia no cálculo do centro de gravidade do conjunto de suporte da saída. Esse conjunto é gerado pelo módulo de inferência, como é mostrado na figura 13.



Figura 13 – Valor final calculado pelo método do centróide (BERNARDO FILHO, 1999)

Matematicamente, o centróide de uma função de pertinência é calculado pelas integrais das funções do termo, como proposto na equação 2.16.

Na equação 16,  $\bar{x}$  é o valor defuzzificado de um dado parâmetro que corresponde ao centróide (centro de massa) do conjunto nebuloso com função de pertinência  $\mu_A(x)$  e suporte  $S$ .

$$\bar{x} = \frac{\int_S x \mu_{\tilde{A}}(x) dx}{\int_S \mu_{\tilde{A}}(x) dx} \quad (16)$$

Na implementação as *funções de pertinência* são segmentos de reta. As integrais representadas na equação 16 são os somatórios das integrais da equação de cada segmento. Então, será preciso calcular as integrais do numerador e denominador da equação 16.

A integral de um segmento de reta do numerador é dada pela equação 17.

$$\text{Centróide} = m(b)^3/3 + q(b)^2/2 - m(a)^3/3 - q(a)^2/2 \quad (17)$$

A integral de um segmento de reta do denominador é dada pela equação 18.

$$\text{Área} = m(b)^2/2 + qb - m(a)^2/2 - qa \quad (18)$$

Nas duas fórmulas  $m$  é o *coeficiente angular* do segmento de reta,  $q$  é o *coeficiente linear*,  $a$  é o *intervalo esquerdo* e  $b$  é o *intervalo direito* no eixo  $x$  do segmento de reta conforme a figura 14.

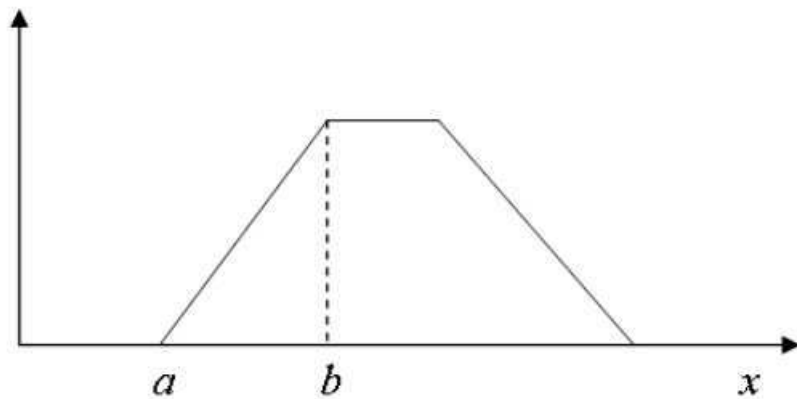


Figura 14 – Função de pertinência (BERNARDO FILHO, 1999)

Para calcular o centróide do trapézio da figura 14, basta calcular a soma das integrais de cada reta (numerador e denominador) e dividir pelo denominador.

Para gerar a saída escalar, deve-se calcular o somatório dos centróides multiplicado pela área do termo de saída ativados, dividindo-se pela soma das áreas de cada termo. A equação 19 mostra o cálculo.

$$\bar{X} = \frac{\sum (\text{Centróide} * \text{Área})}{\sum \text{Área}} \quad (19)$$

### 2.3.2.7 Inferência

Um sistema que trabalha com lógica nebulosa é composto por várias regras de inferência, que manipulam termos de variáveis lingüísticas. Por exemplo:

- SE Largura ( $\epsilon$ ) É estreita E Trânsito ( $\epsilon$ ) É intenso ENTÃO Velocidade ( $\epsilon$ ) é muito\_lenta;
- SE Largura ( $\epsilon$ ) É estreita E Trânsito ( $\epsilon$ ) É livre ENTÃO Velocidade ( $\epsilon$ ) é rápida;
- SE Largura ( $\epsilon$ ) É estreita E Trânsito ( $\epsilon$ ) É moderado ENTÃO Velocidade ( $\epsilon$ ) é lenta;

As regras de inferência ajudam a modelar um raciocínio nebuloso, estabelecendo a largura com o limite de velocidade de uma estrada  $\epsilon$ . As variáveis lingüísticas empregadas nesse exemplo são:

- $T(\text{velocidade}) = \{\text{muito\_lenta}, \text{lenta}, \text{rápida}, \text{muito\_rápida}\}$
- $T(\text{largura}) = \{\text{ampla}, \text{média}, \text{estreita}\}$
- $T(\text{trânsito}) = \{\text{livre}, \text{moderado}, \text{intenso}\}$

As funções de pertinência de cada um dos termos das variáveis lingüísticas devem ser definidas de acordo com o universo de discurso. Por exemplo, a variável *velocidade* teria como universo de discurso um intervalo de valores reais entre 10 e 120 Km/h. A variável *largura* poderia ter um intervalo entre 5 e 40m.

Os sistemas construídos com lógica clássica possuem um conjunto de implicações ou premissas (axiomas), que descrevem o conhecimento sobre um determinado assunto. Ao se questionar um esse conjunto de axiomas básicos, pode-se deduzir uma resposta através da aplicação de regras de inferência. Uma das regras de inferência da lógica clássica é a *modus ponens*, que é demonstrada pela equação 20 (BERNARDO FILHO, 1999).

$$\frac{A \supset B}{A} B \quad (20)$$

Na regra *modus ponens*, tem-se a implicação SE  $A$  ENTÃO  $B$  (axioma), ou seja, caso  $A$  (pergunta) então conclui-se  $B$ . Isso é empregado na lógica clássica, mas na lógica nebulosa é necessário estender a definição dessa regra de inferência que passa, nesse caso, a ser chamada de *modus ponens generalizada*, pois os dados da lógica nebulosa são aproximações de uma realidade. Portanto, a regra *modus ponens generalizada* ficaria da seguinte forma (BERNARDO FILHO, 1999):

$$\frac{A \supset B}{A'} B' \quad (21)$$

A equação 21 mostra que a pergunta  $A'$  não é exatamente igual ao antecedente  $A$  do axioma ou regra, na verdade, a pergunta é “parecida” com o antecedente, o que leva então a concluir, que a resposta  $B'$  também será “parecida” com o conseqüente  $B$  da implicação. Um exemplo, empregando termos lingüísticos seria o apresentado na figura 15.

PREMISSA → O tomate está muito vermelho
IMPLICAÇÃO → Tomate vermelho é maduro
CONCLUSÃO → O tomate está muito maduro

Figura 15 - Exemplo de axiomas com termos lingüísticos. (Bernardo Filho, 1999)

No caso de várias regras com dois antecedentes, existem dois conjuntos nebulosos de entrada para comparar com cada um dos antecedentes das regras. No caso especial em que tais entradas fossem iguais aos termos das variáveis lingüísticas *largura* e *trânsito*, de alguma



regra, então a resposta corresponderia ao conseqüente dessa mesma regra. Entretanto, no caso geral as entradas não são iguais aos antecedentes de nenhuma regra e sim a uma boa aproximação dos antecedentes de alguma regra e a uma má aproximação dos antecedentes das demais regras.

A aplicação da inferência *modus ponens generalizada* ao caso de várias regras é feita por intermédio de um cálculo que tem início com a determinação do grau de aderência  $\alpha$  que as entradas (pergunta) possuem com cada uma das regras. Essa aderência atenua ou não (dependendo do seu valor) a influência do conseqüente de cada regra com o resultado que, por sua vez, é uma combinação de todos os conseqüentes de todas as regras. Tal cálculo, apresentado detalhadamente em pseudo-código, segue os seguintes passos:

#### INICIO DA CLASSE INFERÊNCIA

somatorioCentroide = 0

somatorioArea = 0

centroideRegra = 0

areaRegra = 0

SELECIONAR as Regras QUANDO Sistema = escolhido pelo usuário

PARA cada Regra FAZER

centroideRegra = 0, areaRegra = 0;

Obter o 1º Termo da Regra

mi1 = Avaliar o mi do 1º Termo

Obter o 2º Termo da Regra

mi2 = Avaliar o mi do 2º Termo

Obter menorMi entre 1 e 2

SE menorMi > 0 ENTÃO

Obter termoConsequente Atenuado da Regra

CalcularCentróideRegra(termoConsequente, centroideRegra, areaRegra)

somatorioCentroide = somatorioCentroide + (centroideRegra \* areaRegra);

somatorioArea = somatorioArea + areaRegra;

FIM SE

LOOP

ValorFinal = somatorioCentroide / somatorioArea;

FIM INFERÊNCIA

### 2.3.3 Sistema de Inferência Nebuloso (SIN)

Também chamado de *Controle Nebuloso*, é hoje uma importante aplicação da teoria dos conjuntos nebulosos. É a técnica que conquistou espaço como área de estudo em diversas instituições de ensino, pesquisa e desenvolvimento no mundo.

Segundo Sandri e Correa (1999), são encontrados na literatura alguns controladores clássicos, como o modelo Mamdani, Larsen, além dos modelos de interpolação Takagi-Sugeno e Tsukamoto. A principal diferença entre os modelos é a forma de representação dos termos na premissa, quanto à representação das ações de controle e quanto aos operadores utilizados para implementação do controlador.

O modelo mais difundido é o Mamdani, representado em vários programas computacionais de SIF como o Fuzzy Logic Toolbox do MATLAB (MATLAB, 2000). O modelo Mamdani foi proposto por Ebrahim Mamdani (SOUZA, 2001) baseado no estudo publicado por Lotfi Zadeh (1975). Foi o primeiro modelo de SIN a ser construído a partir da Teoria dos Conjuntos Nebulosos. Este foi o método escolhido para ser aplicado nos SIN da dissertação.

Nos modelos clássicos, dado um conjunto de valores de variáveis de estado, um conjunto nebuloso é obtido como valor da variável de controle. Este conjunto nebuloso representa uma ordenação no conjunto de ações de controle aceitáveis naquele momento. Então, uma ação de controle global é finalmente selecionada dentre as aceitáveis.

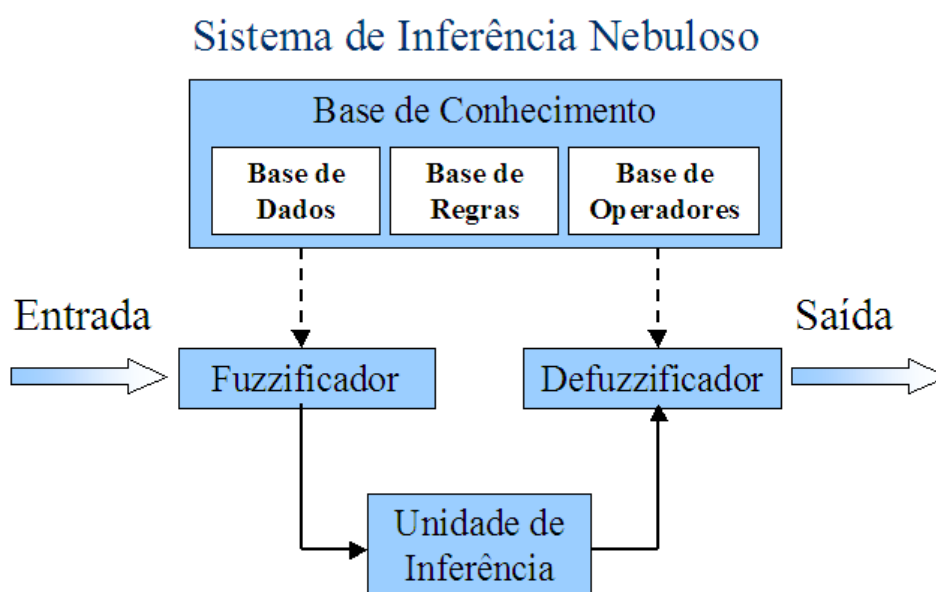


Figura 16 - Representação modular de um SIN (SOUZA,2001)

A figura 16 representa a estrutura básica de um SIN. Há muitas variações propostas na literatura de acordo com o objetivo do projeto, mas este modelo geral é suficiente para a identificação dos módulos que o compõem, fornecendo uma idéia do fluxo da informação. A seguir, uma breve descrição de cada módulo que compõe um SIN.

#### 2.3.3.1 Fuzzificador

Identifica as variáveis de entrada, que são as variáveis que caracterizam o estado do sistema (variáveis de estado), e as normaliza em um universo de discurso padronizado. Esses valores são *fuzzificados*, transformando as entradas em graus de pertinência de conjuntos nebulosos, tornando-se instâncias das variáveis lingüísticas.

#### 2.3.3.2 Base de Conhecimento

Consiste de uma base de dados , base de regras e descrição dos operadores, de maneira a caracterizar a estratégia de controle e as suas metas.

Na *base de dados* ficam armazenadas as definições sobre discretização e normalização dos universos de discurso, e as definições das funções de pertinência dos termos nebulosos.

A *base de regras* é a forma como a representação do conhecimento fica armazenada em um Sistema de Inferência Nebuloso (SIN).

A *descrição dos operadores* define que tipo de operações é utilizado para os operadores de interseção e união entre conjuntos nebulosos. As regras e os dados de entrada, são processados pela unidade de inferência, onde são inferidas as ações de controle de acordo com o estado do sistema, aplicando o operador de implicação.

É importante que existam tantas regras quantas forem necessárias em um controlador nebuloso, para que seja possível mapear totalmente as combinações dos termos das variáveis. Isso significa que a base deve ser completa, garantindo que exista sempre ao menos uma regra a ser disparada para qualquer entrada.

Também é essencial a consistência, onde procura-se evitar a possibilidade de contradições e a interação entre as regras, gerenciada pela função de implicação.

#### 2.3.3.3 Unidade de inferência

A partir de um conjunto de regras nebulosas do tipo SE-ENTÃO (IF-THEN) é usado o procedimento de inferência para a agregação de conclusões. Isso resulta no conjunto nebuloso de saída. Pode-se afirmar que as regras traduzem o conhecimento enquanto a inferência traduz o modo de raciocinar (SOUZA, 2001).

#### 2.3.3.4 Defuzzificador

Após o processo de agregação do conjunto de regras nebuloso, uma função de pertinência é gerada, estabelecendo uma decisão de caráter nebuloso. Nesse caso, é necessário efetuar uma interpretação de modo a traduzi-la para um valor determinístico. Utiliza-se o método de determinação do centro de gravidade (COG) ou centro de área (COA), cujo o valor da abcissa é o valor de saída do SIN.

### 3 O SISTEMA DE TESTES (SIT)

Neste capítulo é apresentado o Sistema de Testes desenvolvido. Na primeira parte, é apresentada uma visão geral dos módulos que compõem o sistema, com suas principais características e entradas.

A seguir são apresentadas as funções de pertinência do módulo de lógica nebulosa e a tabela FAM (*Fuzzy Associative Memory*) com as saídas do módulo.

As seções posteriores apresentam a interface dos sistema com o usuário, as tabelas do banco de dados e as classes que compõem o módulo de inferência nebuloso.

Na parte final deste capítulo é mostrada a aplicação da teoria dos conjuntos nebulosos no SIT, com a definição das regras de inferência e os tipos de entradas cadastrados.

#### 3.1 Visão geral do Sistema de Testes

O sistema de testes (SIT) fornece o valor mínimo a testar para cada entrada de um Sistema de Informações Geográficas (SIG). Esse conjunto mínimo foi associado à aplicação das técnicas funcionais de particionamento de equivalência, análise do valor limite, grafo de causa-efeito e por três sistemas de inferência nebulosos. A razão de ser para a obtenção de tais valores por inferência baseia-se na constatação de que testar todos os valores possíveis de uma entrada para um SIG ou para um software qualquer pode ser muito extenso ou até mesmo inviável, uma vez que algumas entradas podem ter valores infinitos o que levaria a uma óbvia explosão combinatória (DELAMARO, 2007).

O particionamento de equivalência utiliza classes de equivalência para testar domínios de entrada de um programa, de onde são derivados os casos de teste. O objetivo principal dessa técnica é diminuir os casos de teste, selecionando-se apenas um caso de teste de cada classe, partindo-se do princípio que todos os elementos da classe se comportam de maneira equivalente. Segundo Myers (PRESSMAN, 2006 e DELAMARO, 2007), essa equivalência é aproximada, pois é impossível caracterizar exatamente a equivalência.

A análise do valor limite é uma técnica que complementa o particionamento de equivalência, utilizando-se os casos de teste nos limites de cada classe de equivalência. Segundo PRESSMAN (2006) e DELAMARO (2007), esse critério é bastante relevante, pois os erros costumam ocorrer com mais frequência nos limites dos domínios de entrada.

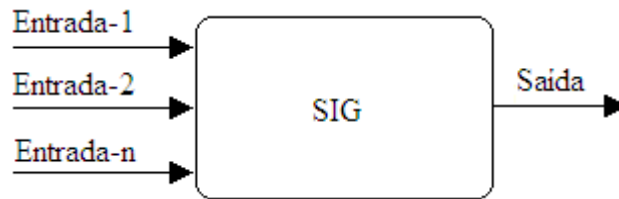


Figura 17 - Teste funcional de um SIG.

Tomando como base um SIG que possui várias entradas  $E1, E2, E3, \dots, En$ , a saída  $S$  do SIG trata-se de uma função dessas entradas, ou seja,  $S = f(E1, E2, E3, \dots, En)$ . Para testar o SIG, deve-se observar se para certos valores de  $E1, E2, E3, \dots, En$ ,  $S$  apresenta também o valor que se espera (ver figura 17).

Inicialmente, pode-se pensar que as entradas possuem dois tipos básicos, a saber:

- (1) Entrada com valores ilimitados em um intervalo ilimitado;
- (2) Entrada com valores ilimitados (ou não) dentro de um intervalo fechado.

Um exemplo de entrada do tipo (1) seria o caso das *strings* que representam nomes de objetos. Para esse exemplo, pode-se observar que todos os valores válidos de *string* são equivalentes, o que leva a concluir que testar um único valor válido já seria suficiente e possível. A preocupação seria apenas pelos valores inválidos de *strings*, como, por exemplo, *strings* nulas, com caracteres especiais, iniciando por números, longas demais, etc. Nessa situação, seria necessário testar tais valores anômalos no sentido de observar a reação do sistema. Por outro lado, caso a entrada do sistema fosse dotada de validação, não seria necessário testar os valores anômalos.

No caso de entradas do tipo (2), se o valor da entrada estiver dentro do intervalo fechado, conclui-se que se trata de um valor válido, porém resta saber quantos valores de tal intervalo seria necessário testar.

Nesse ponto, já é possível apresentar a idéia inicial do Sistema de Testes (SIT) que foi desenvolvido, o qual empregou a lógica nebulosa para auxiliar na inferência de valores das entradas de um SIG a ser testado. A figura 18 apresenta a concepção do SIT.

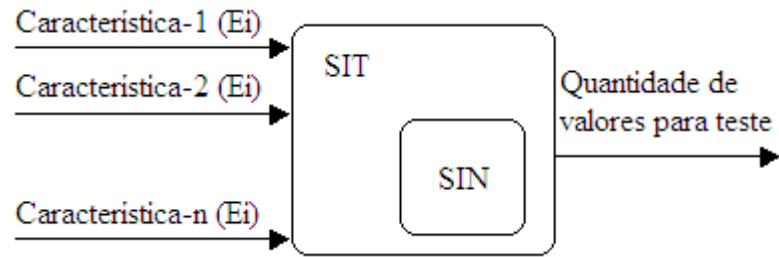


Figura 18 - Sistema de Teste (SIT) proposto.

O SIT recebe como entrada as diversas características de uma entrada  $E_i$  do SIG sob análise. O SIT internamente é composto de um módulo com três sistemas de inferência nebulosos (SIN) de acordo com a quantidade de características previstas para a entrada  $E_i$  do SIG. A figura 19 ilustra as entradas e saídas dos três SIN que foram utilizados.

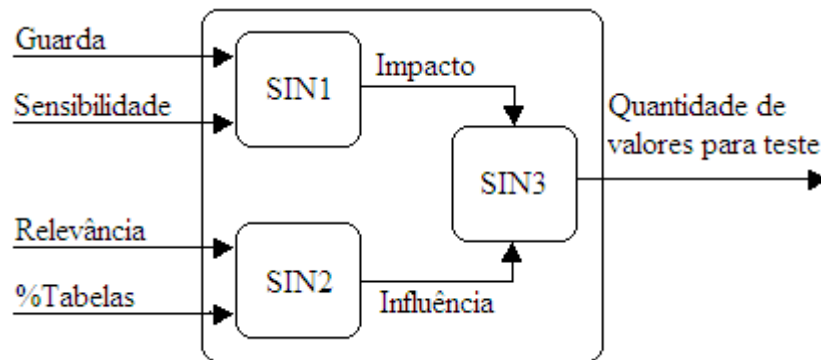


Figura 19 - A composição do módulo de inferência nebulosa (SIN)

### 3.2 Sistemas de Inferência Nebulosos do SIT

O SIT tem como entrada os seguintes dados: a condição de guarda, o grau de sensibilidade, grau de relevância e quantidade de tabelas acessadas.

Para o caso de entradas com validação, pode-se definir uma relação de guarda  $G$ , ou seja, uma entrada  $E_i$ , após passar por um processo de validação caracterizado por diversas condições lógicas  $G$ , assumiria a forma  $\hat{E}_i$  detentora de valores válidos apenas, expurgando quaisquer valores inválidos.

$$E_i \xrightarrow{G} \hat{E}_i \quad (22)$$

A **Condição de Guarda** é definida como um valor percentual de 0 a 100 e tem pertinência entre 0 a 1 para termos nebulosos. Os termos lingüísticos são: *fraca* (FR), *média* (M) ou *forte* (FT). A figura 20 mostra as funções de pertinência para a condição de guarda.

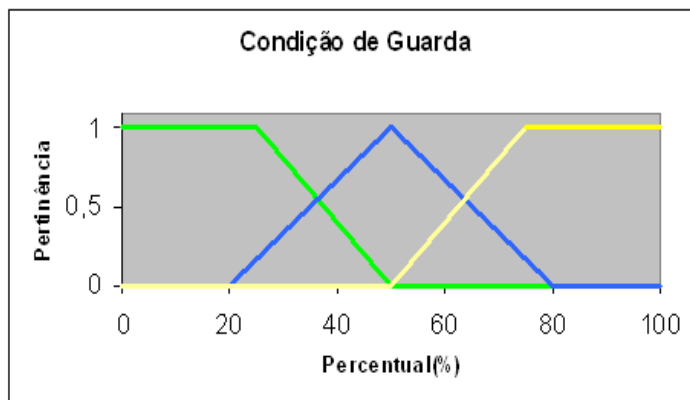


Figura 20 - Funções de pertinência para a condição de guarda

O **Grau de Sensibilidade** denota o quanto a variação de valores de uma entrada causa distúrbios na saída do SIG. Pode haver uma grande variação nos valores e no tempo de processamento da saída caso uma entrada com grau de sensibilidade alto exista no SIG o que vem a indicar uma necessidade de mais testes com essa entrada. Os termos lingüísticos para o grau de sensibilidade são: *muito baixo* (MB), *baixo* (B), *médio* (M), *alto* (A) ou *muito alto* (MA). A figura 21 mostra as funções de pertinência para o grau de sensibilidade.

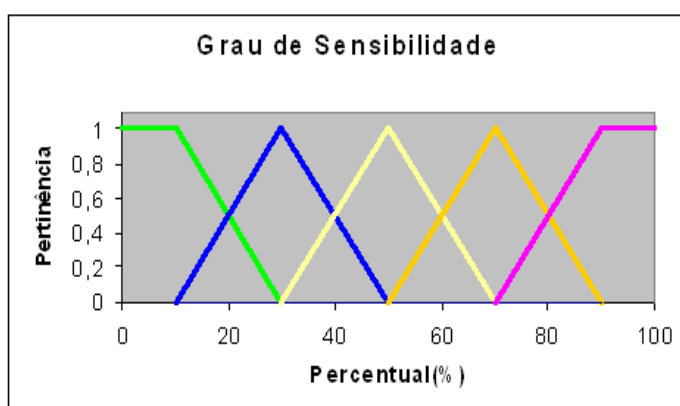


Figura 21 - Funções de pertinência para o grau de sensibilidade

O **Grau de Relevância** define o quanto uma entrada é importante para a saída produzida, levando em conta os principais serviços oferecidos pelo SIG. Podem-se ter entradas muito relevantes ou irrelevantes para a saída esperada. Os termos lingüísticos são:



*irrelevante (I), opcional (OP), desejável (D) ou obrigatória (OB).* A figura 22 mostra as funções de pertinência para o grau de relevância.

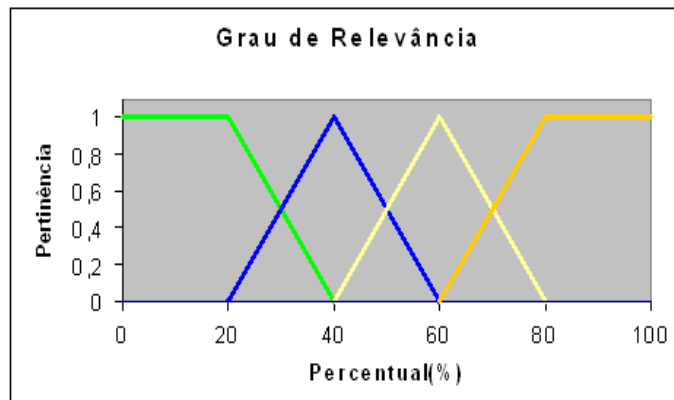


Figura 22 - Funções de pertinência para o grau de relevância

O **Percentual de Tabelas Acessadas** foi definido pelo intervalo fechado de 0 a 100% em relação ao total de tabelas do sistema acessadas pela entrada. Os termos lingüísticos são:  *muito baixa (MB), baixa (B), média (M), alta (A) ou muito alta (MA).* A figura 23 mostra as funções de pertinência para o percentual de tabelas acessadas.

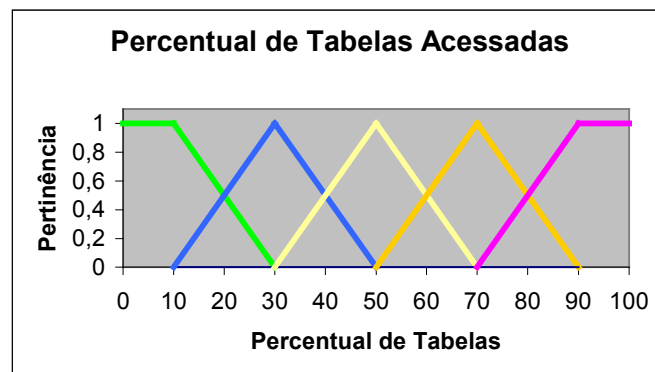


Figura 23 - Funções de pertinência para o percentual de tabelas acessadas

A partir das entradas, foram montadas as tabelas FAM, relacionando os termos lingüísticos. A saída desse primeiro SIN será chamada de **Impacto da Entrada**. Os termos lingüísticos serão:  *muito pouca (MP), pouca (P), média (M), alta (A) ou muito alta (A).* A tabela 1 mostra os relacionamentos entre guarda e sensibilidade.

A saída do segundo SIN será chamada de **Influência da Entrada**. Os termos lingüísticos serão:  *muito pouca (MP), pouca (P), média (M), alta (A) ou muito alta (A).* A tabela 2 mostra os relacionamentos entre relevância e quantidade de tabelas.

A saída do terceiro SIN é chamada de **Valores a Testar**. Essa saída é gerada para cada entrada  $E_i$  e é dada como um valor entre 0 e 100% em relação ao universo de valores de entrada. Os termos linguísticos serão:  *muito pouco* (MP), *pouco* (P), *médio* (M), *alto* (A) ou *muito alto* (A).

Tabela 1 - Guarda X Sensibilidade

		Guarda		
		FR	M	FT
Sensibilidade	MB	P	MP	MP
	B	M	P	MP
	M	M	M	P
	A	A	M	M
	MA	MA	A	M

Tabela 2 – Relevância X Percentual de Tabelas

		Relevância			
		I	OP	D	OB
Perc. Tabelas	MB	MP	MP	P	M
	B	MP	P	M	M
	M	P	M	M	A
	A	M	M	A	MA
	MA	M	A	MA	MA

Os gráficos de saída (impacto de entrada, influência da entrada e quantidade de valores para teste) são iguais, podendo ainda sofrer ajustes para um melhor resultado futuro. A figura 24 apresenta as funções de pertinência dos termos linguísticos das variáveis impacto de entrada, influência da entrada e quantidade de valores para teste, enquanto a tabela 3 evidencia as regras nebulosas do SIN3 o qual fornece a saída final da inferência do SIT sobre quantos valores de uma entrada  $E_i$  devem ser testados.

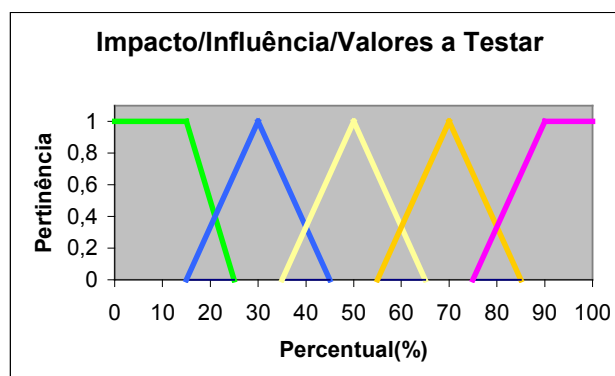


Figura 24 - Funções de pertinência para impacto e influência da entrada e valores a testar

Tabela 3 - Impacto X Influência

		Impacto				
		MP	P	M	A	MA
Influência	MP	MP	P	P	M	M
	P	P	P	M	M	A
	M	P	M	M	A	A
	A	M	M	A	A	MA
	MA	M	A	A	MA	MA

Para exemplificar, após a execução do SIN3 relativa a uma entrada  $E_i$ , sendo composta por um intervalo de valores entre 1 e 40 e, no caso do valor inferido na saída do SIT for 10%, será suficiente testar somente 4 valores dos 40 possíveis. Dois desses quatro valores serão obrigatoriamente os extremos do intervalo considerado para a variação de  $E_i$ .

### 3.3 Modelo conceitual do SIT

Para uma melhor visualização das funcionalidades do SIT, optou-se por utilizar o modelo de casos de uso da UML. O diagrama de casos de uso do sistema de testes é mostrado na figura 25. A descrição dos casos de uso estão no Apêndice A.

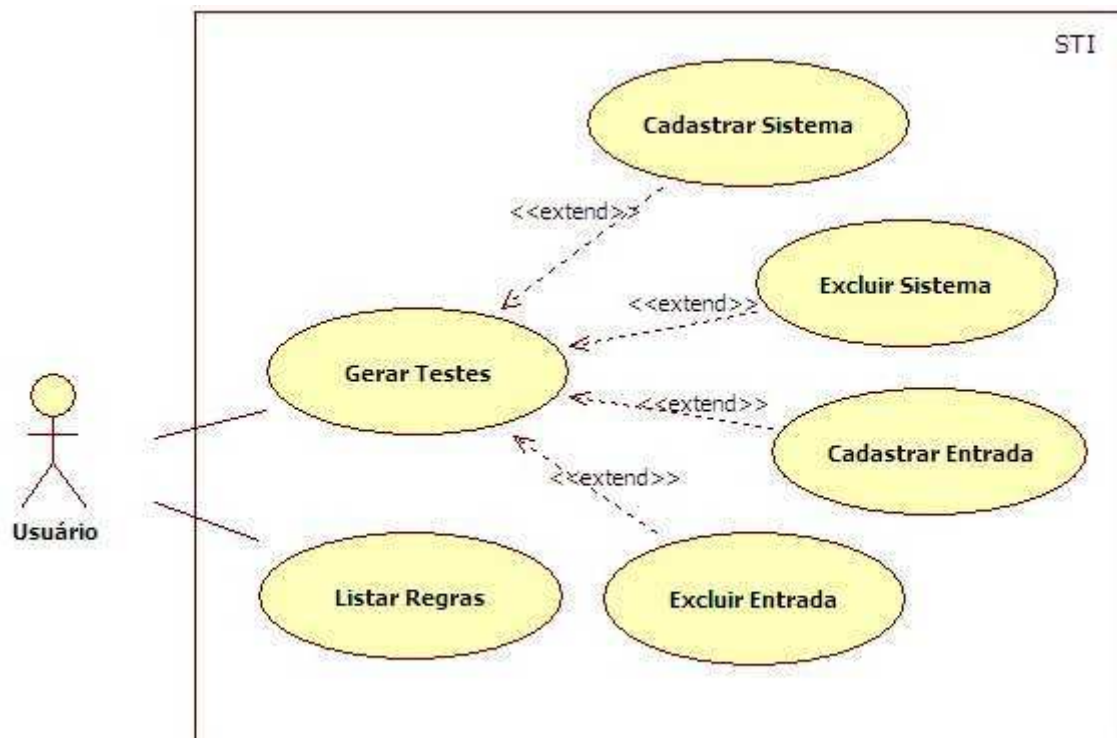


Figura 25 – O diagrama de casos de uso do SIT

### 3.4 A implementação do Sistema de Testes (SIT)

O Sistema de Testes (SIT) foi desenvolvido na forma de uma DLL (*Dynamic Link Library*) codificada em *Visual Basic 6.0*, que contém as classes que manipulam o banco de dados *Access* e fazem as inferências. Essa DLL será instanciada na forma de um objeto *ActiveX* por páginas ASP (*Active Server Pages*).

O processamento é executado em máquinas que possuam o IIS (*Internet Information Server* da Microsoft) ou outro servidor de páginas ASP, que gera uma página de resposta HTML (*Hypertext Markup Language*). O cliente recebe páginas HTML como resposta das páginas ASP executadas no servidor. Essas páginas HTML podem ser lidas em qualquer navegador (*browser*) existente no mercado.

A linguagem HTML é uma linguagem de formatação e exibição, enquanto o ASP oferece estruturas lógicas e de decisão necessárias para a exibição dos testes gerados, pois os mesmos têm uma resposta diferente para cada tipo de entrada cadastrada.

A linguagem ASP é capaz de manipular os dados, mas não é tão eficiente para o processamento de cálculos e criação do motor de inferência do sistema. Essas funcionalidades são fornecidas pelas classes que tratam a parte de lógica nebulosa e funções de conexão e consulta ao banco de dados. As páginas ASP foram usadas como interface entre o usuário e as funcionalidades presentes na DLL escrita em *Visual Basic*. O ASP também foi usado para possibilitar o cadastro de informações por parte do usuário.

A linguagem *JavaScript* foi utilizada em conjunto com o ASP, pois enquanto o ASP é executado no servidor, o *JavaScript* é executado no cliente. Isso possibilita que algumas críticas possam ser validadas antes da página ser enviada para processamento no servidor.

#### 3.4.1 Funcionamento do Sistema de Testes

O esquema básico do SIT é mostrado na figura 26, onde é fornecida uma visão geral do fluxo principal da geração dos testes. Tudo começa pela seleção do sistema que será testado, que exibirá as entradas cadastradas previamente. Selecionam-se então as entradas desejadas e geram-se os testes. Todas as páginas acessam o banco de dados pelos métodos de uma classe da SITFuzzy.dll.

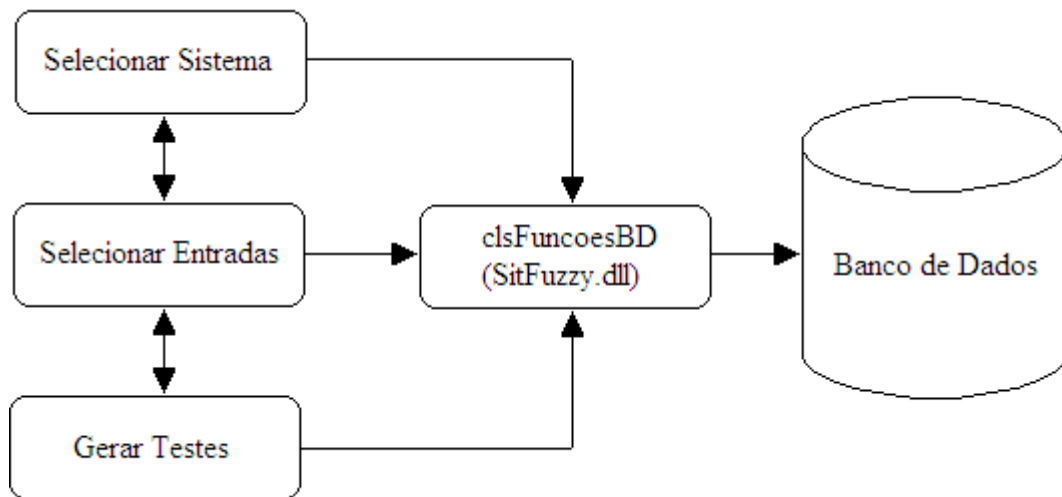


Figura 26 – Esquema básico do Sistema de Testes

### 3.4.2 A interface do SIT

A figura 27 apresenta a página inicial do sistema de testes, com links para as funcionalidades dos sistema.

A página SelecionarSistema.asp (figura 28) é a página onde o sistema que terá os testes gerados é selecionado. Há também a opção de exclusão e inclusão de um novo sistema.

A página SelecionarEntrada.asp (figura 29) é a página onde as entradas do sistema em uso são selecionadas. Há também a opção de exclusão e inclusão de novas entradas.

A página GerarTestes.asp (figura 30) é a página onde os testes realmente gerados para as entradas do sistema selecionado. O classe de lógica nebulosa é usada nessa página para realizar as inferências, além da classe que acessa o banco de dados.

A página VisualizarRegras.asp (figura 31) é a página onde são visualizadas todas as regras de todos os sistemas de inferência nebulosos do SIT.

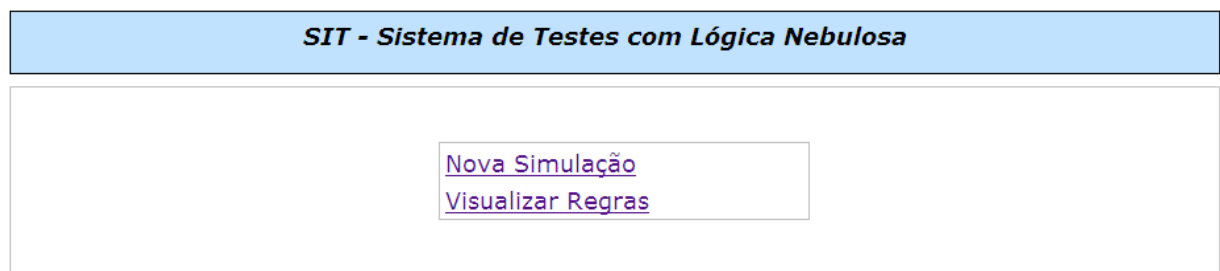


Figura 27 – A tela da página MenuSIT.asp

<b>SIT - Sistema de Testes com Lógica Nebulosa</b>	
<b>Selecionar Sistema</b>	
<b>SISTEMA</b>	<b>DESCRIÇÃO</b>
1	SICH
2	SIGATDSC
<input type="button" value="Exclui Sistema"/> <input type="button" value="Novo Sistema"/> <input type="button" value="Voltar"/>	

Figura 28 – A tela da página SelecionarSistema.asp

<b>SIT - Sistema de Testes com Lógica Nebulosa</b>										
<b>SICH</b>										
<b>Selecionar Entradas</b>										
	NR	TIPO	DESCRIÇÃO	PARÂMETRO1	PARÂMETRO2	PARÂMETRO3	GUARDA	SENSIBILIDADE	RELEVÂNCIA	%TABELAS
<input type="checkbox"/>	1	VALORES - CONJUNTO NAO EQUIVALENTE	FAIXA ETARIA	PESSOAS  HOMENS  MULHERES  P0-14  P15-29  P30-59  P60			100	10	60	20
<input type="checkbox"/>	2	VALORES - CONJUNTO NAO EQUIVALENTE	MOSTRA LINHAS	TUDO  30  60			100	0	0	0
<input type="checkbox"/>	3	VALORES - CONJUNTO NAO EQUIVALENTE	PESSOAS	0  1  2  3  4  5			100	10	60	20
<input type="checkbox"/>	4	VALORES - CONJUNTO NAO EQUIVALENTE	POSTOS DE SAUDE	0  1  2  3  4  5			100	10	60	20
<input type="button" value="Exclui Entrada"/> <input type="button" value="Nova Entrada"/> <input type="button" value="Ok"/> <input type="button" value="Voltar"/>										

Figura 29 – A tela da página SelecionarEntradas.asp

<b>SIT - Sistema de Testes com Lógica Nebulosa</b>							
<b>SICH</b>							
<b>Testes Calculados</b>							
NR	DESCRIÇÃO	PARÂMETRO1	PARÂMETRO2	PARÂMETRO3	PARÂMETROS NEBULOSOS	VALORES A TESTAR (TÉCNICAS CLÁSSICAS)	VALORES A TESTAR (LÓGICA NEBULOSA)
1	FAIXA ETARIA	PESSOAS  HOMENS  MULHERES  P0-14  P15-29  P30-59  P60			100 10 60 20	100,00% (7 de 7)	29,99% (3 de 7)
2	MOSTRA LINHAS	TUDO  30  60			100 0 0 0	100,00% (3 de 3)	10,20% (1 de 3)
3	PESSOAS	0  1  2  3  4  5			100 10 60 20	100,00% (6 de 6)	29,99% (2 de 6)
4	POSTOS DE SAUDE	0  1  2  3  4  5			100 10 60 20	100,00% (6 de 6)	29,99% (2 de 6)
<input type="button" value="Voltar"/>							

Figura 30 – A tela da página GerarTestes.asp

### SIT - Sistema de Testes com Lógica Nebulosa

#### Visualizar Regras

REGRA	SIN		ANTECEDENTE 1				ANTECEDENTE 2			CONSEQUENTE			
1	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Muito baixa	ENTÃO	Impacto	É	Pouco
2	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Baixa	ENTÃO	Impacto	É	Medio
3	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Media	ENTÃO	Impacto	É	Medio
4	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Alta	ENTÃO	Impacto	É	Alto
5	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Muito alta	ENTÃO	Impacto	É	Muito alto
6	1	SE	Guarda	É	Media	E	Sensibilidade	É	Muito baixa	ENTÃO	Impacto	É	Muito pouco
7	1	SE	Guarda	É	Media	E	Sensibilidade	É	Baixa	ENTÃO	Impacto	É	Pouco
8	1	SE	Guarda	É	Media	E	Sensibilidade	É	Media	ENTÃO	Impacto	É	Medio
9	1	SE	Guarda	É	Media	E	Sensibilidade	É	Alta	ENTÃO	Impacto	É	Medio
10	1	SE	Guarda	É	Media	E	Sensibilidade	É	Muito alta	ENTÃO	Impacto	É	Alto

Figura 31 – A tela da página VisualizarRegras.asp

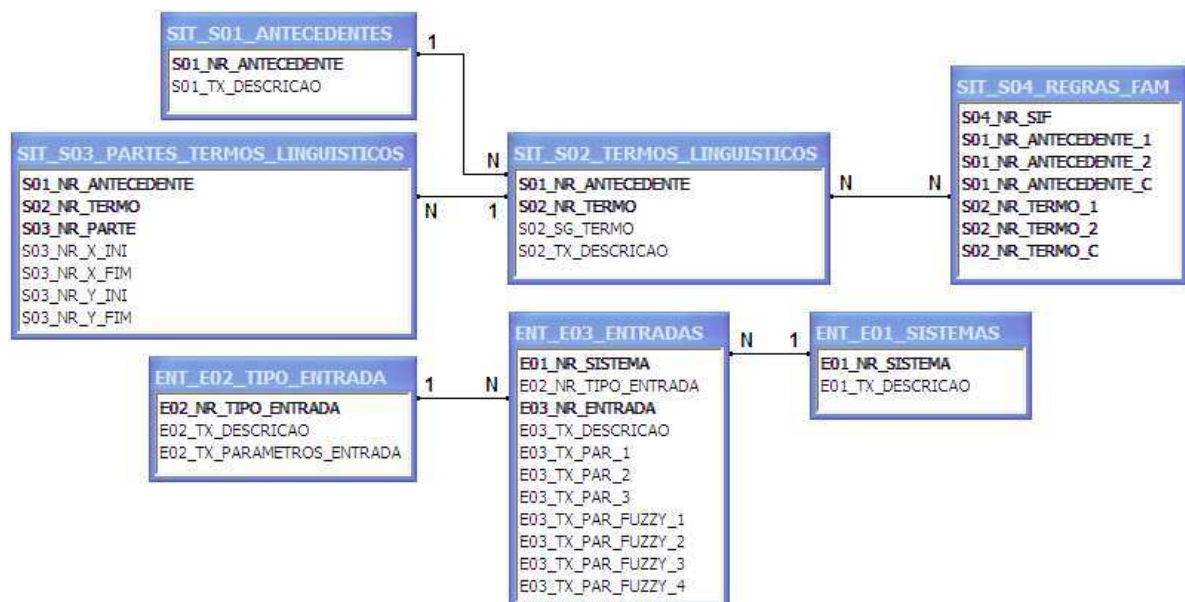


Figura 32 - O modelo relacional das tabelas do SIT (chaves das tabelas em negrito).

#### 3.4.3 Banco de dados

O banco de dados escolhido para armazenamento das informações do SIT foi o *Microsoft Access*, pois o mesmo é de fácil utilização e o SIT não ter grandes exigências de desempenho de consultas. Todas as consultas do SIT foram escritas usando a linguagem SQL padrão, o que possibilita a migração para um banco de dados mais robusto como *Microsoft SQL Server*, *Oracle*, *Sybase*, etc.

Foram criados dois conjuntos independentes de tabelas. No primeiro, foram modeladas todas as tabelas utilizadas pelo módulo de lógica nebulosa: funções de pertinência, termos

lingüísticos de cada antecedente e conseqüente e regras da tabela FAM. Todas as tabelas deste grupo têm o prefixo “SIT”.

Na segunda parte, foram modeladas todas as tabelas relativas ao cadastro e manipulação das entradas do SIG. Todas as tabelas deste grupo têm o prefixo “ENT”. A figura 32 mostra o modelo relacional das tabelas do SIT.

A tabela SIT\_S01\_ANTECEDENTES armazena os nomes dos antecedentes e conseqüentes dos SIN utilizados no SIT. Os campos são mostrados na tabela 4.

Tabela 4 – Os campos da tabela SIT\_S01\_ANTECEDENTES

Campo	Descrição	Tipo
<b>S01_NR_ANTECEDENTE</b>	Número do antecedente/conseqüente	Inteiro longo
<b>S01_TX_DESCRICA0</b>	Nome do antecedente/conseqüente	Texto(50)

A tabela SIT\_S02\_TERMOS\_LINGUISTICOS armazena os termos lingüísticos de cada antecedente e conseqüente, com suas respectivas descrições. Os campos são mostrados na tabela 5.

Tabela 5 – Os campos da tabela SIT\_S02\_TERMOS\_LINGUISTICOS

Campo	Descrição	Tipo
<b>S01_NR_ANTECEDENTE</b>	Número do antecedente/conseqüente	Inteiro longo
<b>S02_NR_TERM0</b>	Número do termo lingüístico	Inteiro longo
<b>S02_SG_TERM0</b>	Sigla do termo lingüístico	Texto(3)
<b>S02_TX_DESCRICA0</b>	Nome do termo lingüístico	Texto(50)

A tabela SIT\_S03\_PARTES\_TERMOS\_LINGUISTICOS armazena as partes dos termos lingüísticos de cada antecedente e conseqüente, com suas respectivas descrições. Os campos são mostrados na tabela 6.

Tabela 6 – Os campos da tabela SIT\_S03\_PARTES\_TERMOS\_LINGUISTICOS

Campo	Descrição	Tipo
<b>S01_NR_ANTECEDENTE</b>	Número do antecedente/conseqüente	Inteiro longo
<b>S02_NR_TERM0</b>	Número do termo lingüístico	Inteiro longo
<b>S02_NR_PARTE</b>	Parte do termo lingüístico	Inteiro longo



S03_NR_X_INI	Valor inicial do intervalo (eixo X)	Inteiro longo
S03_NR_X_FIM	Valor final do intervalo (eixo X)	Inteiro longo
S03_NR_X_INI	Valor inicial da pertinência (eixo Y)	Inteiro longo
S03_NR_X_FIM	Valor final da pertinência (eixo Y)	Inteiro longo

A tabela SIT\_S04\_REGRAS\_FAM armazena as regras FAM para cada sistema de inferência nebuloso ou *fuzzy* (SIN ou SIF). Os campos são mostrados na tabela 7.

Tabela 7 – Os campos da tabela SIT\_S04\_REGRAS\_FAM

Campo	Descrição	Tipo
<b>S04_NR_SIF</b>	Número do SIN (SIF)	Inteiro longo
<b>S01_NR_ANTECEDENTE_1</b>	Número do primeiro antecedente	Inteiro longo
<b>S01_NR_ANTECEDENTE_2</b>	Número do segundo antecedente	Inteiro longo
<b>S01_NR_ANTECEDENTE_C</b>	Número do conseqüente	Inteiro longo
<b>S02_NR_TERMOS_1</b>	Termo lingüístico do primeiro antecedente	Inteiro longo
<b>S02_NR_TERMOS_2</b>	Termo lingüístico do segundo antecedente	Inteiro longo
<b>S02_NR_TERMOS_C</b>	Termo lingüístico do conseqüente	Inteiro longo

A tabela ENT\_E01\_SISTEMAS armazena os sistemas que terão os testes gerados pelo SIT. Os campos são mostrados na tabela 8.

Tabela 8 – Os campos da tabela ENT\_E01\_SISTEMAS

Campo	Descrição	Tipo
<b>E01_NR_SISTEMA</b>	Número do sistema	Inteiro longo
E01_TX_DESCRICAO	Nome do sistema	Texto(50)

A tabela ENT\_E02\_TIPO\_ENTRADA armazena os tipos das entradas dos sistemas que terão os testes gerados pelo SIT. Os campos são mostrados na tabela 9.

Tabela 9 – Os campos da tabela ENT\_E02\_TIPO\_ENTRADA

Campo	Descrição	Tipo
<b>E02_NR_TIPO_ENTRADA</b>	Número do tipo de entrada	Inteiro longo

E02_TX_DESCRICAO	Nome do tipo de entrada	Texto(50)
E02_TX_PARAMETROS_ENTRADA	Parâmetros necessários para cada tipo de entrada	Texto(50)

A tabela ENT\_E03\_ENTRADAS armazena as entradas dos sistemas que terão os testes gerados pelo SIT. Os campos são mostrados na tabela 10.

Tabela 10 – Os campos da tabela ENT\_E03\_ENTRADAS

Campo	Descrição	Tipo
<b>E01_NR_SISTEMA</b>	Número do sistema	Inteiro longo
E02_NR_TIPO_ENTRADA	Número do tipo de entrada	Inteiro longo
<b>E03_NR_ENTRADA</b>	Número da entrada	Inteiro longo
E03_TX_DESCRICAO	Nome da entrada	Texto(50)
E03_TX_PAR_1	Primeiro parâmetro do tipo da entrada	Texto(50)
E03_TX_PAR_2	Segundo parâmetro do tipo da entrada	Texto(50)
E03_TX_PAR_3	Terceiro parâmetro do tipo da entrada	Texto(50)
E03_TX_PAR_FUZZY_1	Primeiro parâmetro <i>fuzzy</i> (guarda)	Texto(50)
E03_TX_PAR_FUZZY_2	Segundo parâmetro <i>fuzzy</i> (sensibilidade)	Texto(50)
E03_TX_PAR_FUZZY_3	Terceiro parâmetro <i>fuzzy</i> (relevância)	Texto(50)
E03_TX_PAR_FUZZY_4	Quarto parâmetro <i>fuzzy</i> (qtd. tabelas)	Texto(50)

#### 3.4.4 A criação da DLL SITFuzzy

O SIT é baseado no ambiente *web*, ou seja, em páginas processadas por um servidor. Para que essas páginas possam usar m programa escrito em *Visual Basic* é necessário que o código seja compilado na forma de uma DLL. Essa DLL é registrada no sistema operacional *Windows*, ficando disponível para ser instanciada por qualquer programa.

Um aspecto importante que não pode ser esquecido é que as DLL não possuem interface com o usuário, ou seja, podem somente receber parâmetros e retornar respostas. A DLL foi criada utilizando-se duas classes independentes: a classe *clsFuncoesBD*, que faz toda a comunicação com o banco de dados e a classe *clsFuzzy*, que trata a parte de lógica nebulosa. As classes contam com métodos públicos, que são visíveis fora da classe, e

privados, que não têm visibilidade fora da classe. O código fonte da dll está listado no Apêndice B.

#### 3.4.4.1 A classe clsFuncoesBD

A classe clsFuncoesBD é a responsável por toda a comunicação com o banco de dados, tanto por parte da clsFuzzy quanto pelas páginas ASP. A seguir, são descritos os seus métodos e propriedades.

O método público *conectaBanco* faz a conexão com o banco de dados padrão. Retorna *TRUE* se houve sucesso e *FALSE* em caso de erro, além de preencher a mensagem de retorno com a descrição do erro. Este método não recebe parâmetros.

O método público *desconectaBanco* faz a conexão com o banco de dados padrão. Retorna *TRUE* se houve sucesso e *FALSE* em caso de erro. Este método não recebe parâmetros.

O método público *sqlConsulta* faz a consulta ao banco recebida pela variável *sql* do tipo *string*, preenchendo um objeto do tipo *recordset* em caso de sucesso da consulta. Retorna *TRUE* se houve sucesso e *FALSE* em caso de erro, além de preencher a mensagem de retorno com a descrição do erro.

O método público *sqlExecuta* faz a execução do comando recebido pela variável *sql* do tipo *string*. Retorna *TRUE* se houve sucesso e *FALSE* em caso de erro, além de preencher a mensagem de retorno com a descrição do erro.

O método público *getRS* retorna o objeto do tipo *recordset* que armazenou o resultado de uma consulta realizada pelo método *sqlConsulta*.

O método público *getMensagem* retorna o conteúdo da variável do tipo *string* que armazenou a mensagem de erro que será retornada para a página ou classe que instanciou algum método da classe .

#### 3.4.4.2 A classe clsFuzzy

A classe clsFuzzy é responsável pelas inferências realizadas pelo motor de lógica nebulosa. A seguir, são descritos os seus métodos e propriedades.

O método Público *Inicia* é o método que chama todos os outros para realizar as inferências. Recebe os valores dos antecedentes (guarda, sensibilidade, relevância e quantidade de tabelas) do tipo *double*. Retorna um valor do tipo *double* obtido após todas as

inferências. Usa a classe *clsFuncoesBD* e chama os seguintes métodos privados *CalculaValorSaidaSIF*.

O método privado *calculaValorSaidaSIF* recebe como parâmetros o número do sistema de inferência nebuloso (SIF), do tipo *integer* e os valores dos dois antecedentes que compõem o sistema de inferência nebuloso ou *fuzzy* (SIF), do tipo *double* e retorna um valor do tipo *double* obtido na inferência do SIF em questão. Este método chama os métodos *carregaPartesTermosLinguisticos*, *carregaRegrasFAM*, *obterMi*, *obterConsequente* e *calcularCentroideRegra*.

O método privado *calcularCentroideRegra* recebe como parâmetros as variáveis que armazenam os valores do centróide e a área para o consequente selecionado, ambos do tipo *double*. O vetor com as partes do termo selecionado é percorrido e os valores do centróide e a área são acumulados. Este método não retorna parâmetros.

O método privado *obterConsequente* recebe como parâmetros o nome do termo lingüístico e o termo atenuante das funções ativadas. Sua função principal é preencher o vetor de termos selecionados com os termos lingüísticos ativados, percorrendo o vetor com todos os valores. Estes termos selecionados são então atenuados pelo valor de *dMiSelecioneado*, calculando os novos coeficientes das partes se necessário. Este método chama o método *calculaCoeficientesReta*.

O método privado *obterMi* que recebe como parâmetros o valor da entrada (*double*), os vetores de antecedentes e o termo lingüístico (*string*). Calcula o maior valor de cada parte do termo lingüístico ( $\mu$ ) caso ela seja ativada. Retorna o maior  $\mu$  encontrado (*double*).

O método privado *carregaPartesTermosLinguisticos* é responsável por carregar as partes dos termos lingüísticos no vetor de antecedentes. Recebe como parâmetros de entrada o número do SIF (*integer*), o tipo do antecedente (*string*) e o vetor de antecedentes a ser preenchido. O tipo de antecedente pode ser “1” para o primeiro antecedente do SIF, “2” para o segundo antecedente e “C” para o consequente. Retorna *TRUE* se houve sucesso e *FALSE* em caso de erro. Chama o método *calculaCoeficientesReta*.

O método privado *carregaRegrasFAM* é responsável por carregar as regras da tabela FAM no vetor de regras. Recebe o número do SIF como parâmetro (*integer*). Retorna *TRUE* se houve sucesso e *FALSE* em caso de erro.

O método privado *calculaCoeficientesReta* é responsável por calcular os coeficientes das retas que compõem um determinado termo lingüístico. Recebe como parâmetros o vetor com as partes dos termos lingüísticos dos antecedentes e o índice do item a calcular (*integer*), atualizando a informação no vetor. Este método não retorna parâmetros.

### 3.5 Aplicação da teoria dos conjuntos nebulosos

A implementação da classe que realiza as inferências no SIT começou com a escolha das variáveis lingüísticas de cada um dos três SIN e com a definição de seus respectivos termos. Foram também definidas 60 regras de associação dos termos para o preenchimento da tabela FAM.

Todos os termos de todas as variáveis lingüísticas das funções de pertinência foram definidos como segmentos de reta, definindo funções lineares com perfis triangulares. Isso facilita a manutenção das tabelas que armazenam as partes dos termos lingüísticos.

Para se realizar as inferências é necessário calcular os coeficientes angulares e lineares para cada função, através de seus valores iniciais e finais. A equação da reta pode ser definida através da seguinte forma:

$$Y = mX + q \quad (23)$$

Onde,  $m$  é o coeficiente angular e  $q$  é o coeficiente linear. A figura 33 mostra um segmento de reta com seus valores iniciais e finais em relação aos eixos X e Y.

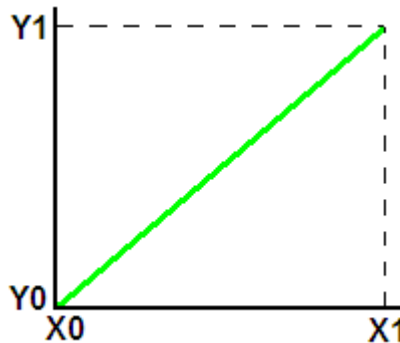


Figura 33 – Um segmento de reta com seus valores iniciais e finais.

Dessa forma, para um determinado segmento, pode-se obter tais valores através de um simples cálculo:

$$m = (Y1 - Y2) / (X1 - X2) \quad (24)$$

$$q = Y1 - (m * X1) \quad (25)$$

Para exemplificar o processo, tomemos o gráfico da função de pertinência para o termo lingüístico *fraca* da função de pertinência *Condição de Guarda*, mostrado na figura 34.

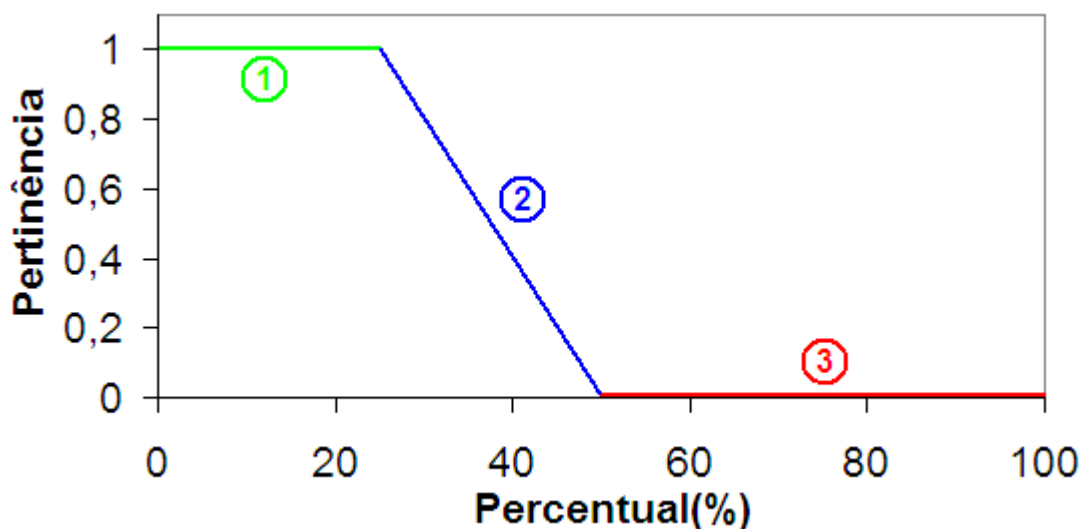


Figura 34 – Partes do termo linguístico *fraca* para a *Condição de Guarda*.

Conforme pode ser observado, o termo linguístico foi dividido em três retas, que foram armazenadas na tabela *SIT\_S03\_PARTES\_TERMOS\_LINGUISTICOS*. A partir das informações dos valores dos intervalos inicial e final de cada reta são calculados os coeficientes angular e linear. A tabela 11 mostra como as partes são armazenadas.

Tabela 11 – Partes armazenadas na tabela *SIT\_S03\_PARTES\_TERMOS\_LINGUISTICOS*

Parte	X <sub>0</sub>	X <sub>1</sub>	Y <sub>0</sub>	Y <sub>1</sub>
1	0	25	1	1
2	25	50	1	0
3	50	100	0	0

### 3.6 Definição das Regras de Inferência

As regras de inferência são de fundamental importância para a obtenção de uma resposta satisfatória e com a melhor precisão possível. O total de regras do sistema pode ser obtido somando-se o total de regras de cada um dos três SIN que compõem o SIT.

O total de regras de cada SIN é obtido através da multiplicação da quantidade de termos de cada uma das variáveis linguísticas. Após a geração das regras, elas foram incluídas na tabela *SIT\_S04\_REGRAS\_FAM*.

Todas as regras estão listadas no Apêndice C. A título de exemplo, observemos a figura 35, com as regras relativas ao termo lingüístico *fraca* da função de pertinência *Condição de Guarda*:

REGRA	SIN		ANTECEDENTE 1				ANTECEDENTE 2				CONSEQÜENTE		
1	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Muito baixa	ENTÃO	Impacto	É	Pouco
2	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Baixa	ENTÃO	Impacto	É	Medio
3	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Media	ENTÃO	Impacto	É	Medio
4	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Alta	ENTÃO	Impacto	É	Alto
5	1	SE	Guarda	É	Fraca	E	Sensibilidade	É	Muito alta	ENTÃO	Impacto	É	Muito alto

Figura 35 – Regras relativas ao termo lingüístico *fraca* da *Condição de Guarda*

### 3.7 Definição dos tipos de entradas

As entradas cadastradas no SIT para a geração dos testes devem ser dos tipos cadastrados na tabela de tipos de *ENT\_E02\_TIPO\_ENTRADA*. O sistema permite, com pequenas alterações no código, que novos tipos de entradas sejam adicionados. Os tipos cadastrados e seus parâmetros de entrada são listados na tabela 12.

Tabela 12 – Tipos de entradas cadastrados.

Tipo da entrada	Parâmetros
Texto	Tamanho do texto, valor válido, valor inválido
Booleana	Valor verdadeiro, valor falso
Conjunto de valores não-equivalentes	Lista de valores
Conjunto de valores equivalentes	Lista de valores
Intervalo de valores não-equivalentes	Valor inicial, valor final, passo (variação)
Intervalo de valores equivalentes	Valor inicial, valor final, passo (variação)

## 4 ESTUDO DE CASO

Este capítulo descreve os testes realizados em alguns SIG com a ferramenta desenvolvida, para verificar sua eficiência na indicação do percentual de testes a realizar. Em seguida, foram feitas análises dos resultados obtidos pelo SIT.

### 4.1 SIGATDSC

O primeiro teste da ferramenta SIT foi na avaliação dos testes que deveriam ter sido feitos no trabalho de Salmaso (2007) no qual tinha sido desenvolvido um sistema de informações geográficas, chamado de Sistema de Informação Geográfica para Apoio à Tomada de decisão no Setor de Cobrança (SIGATDSC), para consultar as informações dos inadimplentes de um sistema de cobrança. As informações sobre os inadimplentes eram do estado do Rio de Janeiro. O SIG desenvolvido ofereceu consultas sobre o grau de risco da cobrança em um determinado bairro após adotar uma estratégia de cobrança, avaliar a distribuição geográfica dos inadimplentes por estado e cidades e o índice de pagamento. O SIGATDSC empregou um sistema de inferência nebuloso para auxiliar na gestão do acordo dentro do processo de cobrança. A tela da consulta principal do SIGATDSC é mostrada na figura 36.



**Perfil da Consulta**

**Empresa:** .....Selecione..... ▼

**Tipo:** .....Selecione..... ▼

**Consultar por:** .....Selecione..... ▼

**Modo de Exibição:** .....Selecione..... ▼

**Gerar mapa**

Figura 36 – A tela da consulta principal do SIGATDSC (SALMASO, 2007)



Tal sistema fornece como saída o grau de risco da aplicação de uma estratégia de cobrança, levando em conta o resultado do contato com o inadimplente (variável lingüística *Acertividade*, e o efetivo recebimento do débito (variável lingüística *Efetividade*). As outras entradas da funcionalidade são *Empresa*, *Tipo de consulta*, *Consultar por* e *Modo de exibição*.

As variáveis lingüísticas *Acertividade* e *Efetividade* têm como universo de discurso uma faixa de valores em percentagem de 0% a 100%. Para testar o seu sistema de inferência nebuloso desenvolvido, seria necessário entrar com valores na faixa de 0% a 100% para ambas as variáveis e observar se o grau de risco calculado (valor também dentro da faixa de 0% a 100%) estaria de acordo com as regras de inferência nebulosas implementadas. A figura 37 mostra a tabela FAM para as regras do SIGATDSC.

		Efetividade				
Acertividade	GRAU DE RISCO	MB	B	M	A	MA
	MB	MA	MA	A	A	M
	B	MA	A	A	M	M
	M	A	M	M	M	B
	A	M	M	B	B	MB
	MA	M	M	B	MB	MB

Legenda: MA-Muito Alto, A – Alto, M- Médio, B – Baixo e MB – Muito Baixo

Figura 37 - Tabela FAM do SIGATDSC (SALMASO, 2007)

As tabelas do SIGATDSC são as seguintes: TB\_USUARIO, TB\_INADIMPLENTE, TB\_CONTRATO, TB\_EMPRESA, TB\_TITULO, TB\_SITUACAO, TB\_ESTADO, TB\_ACORDO, TB\_PAGAMENTO, TB\_TIPOPAG, TB\_ESPECIE, TB\_POSTO, TB\_CIDADE, TB\_BAIRRO, TB\_TPACIONAMENTO, ANTECEDENTE, REGRA, SEGMENTO, TERMO e VARIABEL\_LINGUISTICA.

As entradas do SIGATDSC foram divididas em dois grupos com características semelhantes. No primeiro grupo ficaram as entradas *Empresa*, *Tipo de consulta*, *Consultar por* e *Modo de exibição* e no segundo grupo as entradas *Acertividade* e *Efetividade*.

As entradas lingüísticas *Acertividade* e *Efetividade* são referentes ao mesmo universo de discurso, além de serem fornecidas como entrada da mesma forma no sistema. Ambas têm com universo de discurso uma faixa de valores entre 0% e 100%.

Será necessário agora definir as características das entradas *Acertividade* e *Efetividade* para que seja possível observar a saída do Sistema de Inferência Nebuloso do SIT, o qual informará quantos valores de *Ei* deverão ser testados. Começando com a característica **Guarda**, os valores de *Ei* são calculados a partir de dados cadastrados diretamente no banco de dados do SIGATDSC pelo usuário com o uso de uma ferramenta de acesso a SGBD externa. Sendo assim, não há qualquer proteção sobre os valores inseridos, o que nos leva a concluir que a sua guarda é *fraca* com um valor de 0%. Ainda no diz respeito ao impacto de *Ei*, a **Sensibilidade** é média (50%), pois cada entrada *Ei* tem igual impacto na produção da saída para a funcionalidade.

As entradas *Acertividade* e *Efetividade* são obrigatórias para a produção da saída para a funcionalidade. Sendo assim sua **Relevância** é *obrigatória* e pode ser arbitrado em 100%.

As tabelas TB\_INADIMPLENTE, TB\_EMPRESA, TB\_CIDADE e TB\_BAIRRO são usadas no cálculo dos índices de Acertividade e Efetividade. Então, o **Percentual de Tabelas** foi calculado em 20%.

No segundo grupo de entradas foi analisado de forma semelhante. A **Guarda** é *forte* (100%), pois os dados são selecionados em um objeto do tipo *listBox*, o que impossibilita valores inválidos. A **Sensibilidade** é muito baixa, pois este grupo de entradas não impacta na produção da saída. Tais entradas são irrelevantes para a produção da saída, tendo **Relevância** arbitrada em 0%. A entrada Empresa usa 5 tabelas das 20 do sistema, tendo **seu Percentual de Tabelas** calculado em 25%, enquanto as demais entradas do grupo usam somente 4 tabelas, com percentual calculado em 20%.

Os valores dos parâmetros nebulosos para as entradas do SIGATDSC são mostrados na figura 38.

DESCRIÇÃO	GUARDA	SENSIBILIDADE	RELEVÂNCIA	%TABELAS
EMPRESA	100	0	0	25
TIPO CONSULTA	100	0	0	20
CONSULTAR POR	100	0	0	20
MODO EXIBICAO	100	0	0	20
ACERTIVIDADE	0	50	100	20
EFETIVIDADE	0	50	100	20

Figura 38 – Os valores dos parâmetros nebulosos para as entradas do SIGATDSC

Entrando com esses valores das características de Ei no Sistema de Inferência Nebuloso do SIT, temos o cálculo dos percentuais a testar pelas técnicas clássicas e pela lógica nebulosa. A figura 39 mostra os resultados após os cálculos. Ao se colocar o ponteiro do *mouse* sobre qualquer percentual é exibida uma caixa de mensagem com os valores sugeridos para teste.

DESCRIÇÃO	PARÂMETROS NEBULOSOS	VALORES A TESTAR (TÉCNICAS CLÁSSICAS)	VALORES A TESTAR (LÓGICA NEBULOSA)
EMPRESA	100 0 0 25	100,00% (1 de 1)	10,20% (1 de 1)
TIPO CONSULTA	100 0 0 20	100,00% (3 de 3)	10,20% (1 de 3)
CONSULTAR POR	100 0 0 20	100,00% (3 de 3)	10,20% (1 de 3)
MODO EXIBICAO	100 0 0 20	100,00% (4 de 4)	10,20% (1 de 4)
ACERTIVIDADE	0 50 100 20	<1,00% (1 de 101)	50,00% (51 de 101)
EFETIVIDADE	0 50 100 20	<1,00% (1 de 101)	50,00% (51 de 101)

Figura 39 – Os resultados do processamento das entradas pelo SIT.

#### 4.1.1 Análise do resultados obtidos

Ao analisar os resultados para cada entrada do SIGATDSC, pode-se observar que o módulo de inferência nebulosa do SIT recomenda que sejam testados 50% valores para a entrada referente aos índices de *Acertividade* e *Efetividade* do SIGATDSC, que são as entradas mais importantes na produção da saída. A princípio, poder-se-ia imaginar que tais valores são muitos. Porém, ao levar-se em conta a importância da análise considerada para o SIGATDSC e que os valores de cada entrada *EX* afetam fortemente o comportamento do seu sistema de inferência para calcular o grau de risco da estratégia de cobrança adotada, pode-se, dessa forma, concluir que o SIT não exagerou em sugerir tal percentual de valores a testar. Somente ao testar-se este percentual de valores ter-se-ia a certeza de que o SIGATDSC está fazendo deduções corretas para as suas estratégias de cobrança.

Analogamente, as demais entradas *EX* (*Empresa*, *Tipo de consulta*, *Consultar por* e *Modo de exibição*) apresentaram obviamente um percentual de teste bem mais baixo, relativo às características informadas, indicando que a ferramenta apresenta resultados coerentes.

O uso do módulo de lógica nebulosa sugeriu uma diminuição do percentual de testes a realizar de 100% para cerca de 10%. Tal ganho seria muito mais representativo se cada entrada tivesse um conjunto de valores maior.

## 4.2 SICH

O segundo SIG utilizado na avaliação da ferramenta de testes foi o Sistema de Cálculo do Índice de Carência Hospitalar (SICH), desenvolvido por Rapello (2007), com o objetivo de fornecer um índice de carência hospitalar para municípios do estado do Rio de Janeiro. O índice foi calculado a partir de informações de um conjunto de tabelas disponibilizadas pelo Instituto Brasileiro de Geografia e Estatística (IBGE).

A tela inicial do SICH contém três links para as suas funcionalidades: *Nova Simulação*, *Carrega Simulação* e *Exclui Simulação*. O uso do SIT se concentrou na funcionalidade *Nova Simulação*, pois somente a mesma tem entrada de dados. A tela é mostrada na figura 40.

As tabelas do SICH são as seguintes: MUNICIPIOS, PESSOAS, SAUDE, SINTESE e SIMULACAO.

O ICH é calculado para cada município a partir de uma fórmula que utiliza pesos para cada parâmetro que o compõe. O cálculo do ICH é dado pela equação 26.

$$ICH = \sum_{i=1}^5 W_i * P_i \quad (26)$$

Na equação 4.1,  $P_i$  é o parâmetro e  $W_i$  é o peso no cálculo do ICH. Os parâmetros, lidos a partir das tabelas do IBGE, são os seguintes: *Número de pessoas na faixa etária escolhida (P1)*, *Número de postos de saúde (P2)*, *Número de leitos hospitalares (P3)*, *Saldo do município (P4)* e *Saldo do município per capita (P5)*. Cada parâmetro  $P_i$  e peso  $W_i$  é uma entrada  $E_i$  do sistema.

Os pesos ( $W_i$ ) são determinados na seção de pesos da tela, podendo ser selecionados valores em uma *listBox* com universo de discurso o intervalo de 0 a 5. A tela também possui um filtro onde pode-se escolher a faixa etária de interesse em *listBox*. O universo de discurso é: *todas as pessoas, somente homens, somente mulheres, pessoas de 0 a 14 anos, pessoas de 15 a 29 anos, pessoas de 30 a 59 anos e pessoas a partir de 60 anos*. Ainda no filtro, pode-se escolher a quantidade de linhas a exibir na tabela: *30, 60 ou Todas*.

Na tela há ainda uma seção onde pode-se fazer uma simulação para um município escolhido (o Rio de Janeiro tem 91 municípios), onde são adicionados novos leitos para

verificar uma nova posição do município na tabela, com universo de discurso o intervalo de 0 a 999 (0 ou vazio desabilita o campo).

**SICH - Sistema de Cálculo de Índice de Carência Hospitalar**

**Filtro**

Faixa Etária: Pessoas

Mostra Linhas: 30

**Pesos**

Pessoas: 1

Postos de Saúde: 1

Leitos Hospitalares: 1

Saldo Município: 1

Saldo per Capita: 1

**Simulação**

Município: Angra dos Reis

Leitos:  [\[Inclui Simulação\]](#)

[Atualizar](#) [Voltar](#)

Posição	Município	Nome	ICH	Pessoas	Postos	Leitos	Saldo	SaldoPC	Latitude	Longitude
1	330455	Rio de Janeiro	3,369	5551538	0,0047	0,4725	3939139	0,7096	-22,90278	-43,2075
2	330490	São Gonçalo	1,9622	833379	0,0516	0,0456	847324	1,0167	-22,82694	-43,05389
3	330390	Petrópolis	1,9414	269669	0,0915	0,0435	863373	3,2016	-22,505	-43,17861
4	330350	Nova Iguaçu	1,9083	826188	0,007	0,0269	1119602	1,3551	-22,75917	-43,45111
5	330170	Duque de Caxias	1,8983	715089	0,0188	0,0277	895065	1,2517	-22,78556	-43,31167
6	330330	Niterói	1,8849	450364	0,007	0,0547	887469	1,9706	-22,88333	-43,10361
7	330190	Itaboraí	1,8645	161209	0,054	0,0216	817284	5,0697	-22,74444	-42,85944
8	330100	Campos dos Goytacazes	1,8465	389547	0	0,0272	976873	2,5077	-21,75417	-41,32444
9	330510	São João de Meriti	1,8335	434323	0	0,018	785209	1,8079	-22,80389	-43,37222
10	330200	Itaguaí	1,8295	70126	0,0587	0,0045	517673	7,382	-22,85222	-43,77528
11	330630	Volta Redonda	1,8192	232287	0	0,0123	1090916	4,6964	-22,52306	-44,10417
12	330045	Belford Roxo	1,8172	399319	0	0,0052	795900	1,9931	-22,76417	-43,39944
13	330220	Itaperuna	1,8147	82650	0,0399	0,0118	428684	5,1867	-21,205	-41,88778
14	330320	Nilópolis	1,8119	155272	0,0188	0,0074	794144	5,1145	-22,8075	-43,41389
15	330340	Nova Friburgo	1,805	169246	0	0,0172	807707	4,7724	-22,28194	-42,53111
16	330360	Paracambi	1,8033	39441	0,0164	0,036	254592	6,455	-22,61083	-43,70917
17	330070	Cabo Frio	1,7994	101401	0,0282	0,005	471088	4,6458	-22,87944	-42,01861
18	330250	Magé	1,7985	183113	0,0023	0,0073	799435	4,3658	-22,65278	-43,04056
19	330040	Barra Mansa	1,7981	166745	0	0,0105	812681	4,8738	-22,54417	-44,17139
20	330270	Maricá	1,7913	60286	0,0329	0,0015	351276	5,8268	-22,91944	-42,81861
21	330370	Paraíba do Sul	1,7886	33737	0,0376	0,001	249967	7,4093	-22,16194	-43,29278
22	330050	Bom Jardim	1,7859	21805	0,0376	0,0011	213960	9,8124	-22,15194	-42,41944
23	330600	Três Rios	1,7833	66223	0,0164	0,006	438710	6,6247	-22,11667	-43,20917
24	330580	Teresópolis	1,7818	125122	0,0023	0,0068	569936	4,555	-22,41222	-42,96556
25	330080	Cachoeiras de Macacu	1,7816	43482	0,0258	0,0016	331510	7,6241	-22,4625	-42,65306
26	330480	São Fidélis	1,7787	36534	0,0235	0,0031	304731	8,341	-21,64611	-41,74694
27	330110	Cantagalo	1,7766	18858	0,0282	0,0008	238706	12,6581	-21,98111	-42,36806
28	330380	Parati	1,7752	27127	0,0258	0,0016	222155	8,1894	-23,21778	-44,71306
29	330030	Barra do Piraí	1,7747	85391	0	0,0117	443452	5,1932	-22,47	-43,82556
30	330240	Macaré	1,774	112971	0	0,0046	525599	4,6525	-22,37083	-41,78694

Figura 40 – A tela da funcionalidade *Nova Simulação* (RAPELLO, 2007)

A entrada *faixa etária (P1)* tem a **Guarda** definida como *forte*, pois os valores são selecionadas em uma *listBox*, o que impede valores inválidos. Uma variação nos valores dessa entrada causa um impacto no valor do ICH de cerca de 2%. Sendo assim, a **Sensibilidade** é  *muito baixa*. A **Relevância** é *desejável*, pois o parâmetro é relevante para a produção da saída, mas pode ser desabilitado pela seleção do valor 0 (zero) na entrada *pessoas*. A entrada usa apenas a tabela PESSOAS, tendo o **Percentual de Tabelas** calculado em 20% (*muito baixo/baixo*).

A entrada *mostrar linhas* tem a **Guarda** definida como *forte* por ter seus valores selecionados em uma *listBox*. Sua **Sensibilidade** é *muito baixa*, pois uma variação não causa impacto na posição do município na lista de municípios. A **Relevância** para a produção da saída é *irrelevante*. A entrada não usa nenhuma tabela do sistema, tendo o **Percentual de Tabelas** definido em *muito baixo* (0%).

Na seção *Pesos*, todas as entradas (*Ei*) têm comportamento semelhante e têm os parâmetros nebulosos praticamente iguais. A **Guarda** foi definida como *forte*, uma vez que o usuário está impedido de inserir valores inválidos. A **Sensibilidade** foi definida como *muito baixa/baixa*, pois uma variação no seu valor causa um impacto de cerca de 25% no valor do

ICH. Cada entrada tem **Relevância** *desejável*, pois a seleção do valor 0 (zero) desabilita o peso no cálculo do ICH (na verdade, pelo menos uma entrada deve ser informada). Finalmente, a entrada *pessoas* usa 1 tabela (PESSOAS), as entradas *postos de saúde e leitos hospitalares* usam 1 tabela (SAUDE) e as entradas *saldo do município e saldo per capita* usam 1 tabela (SINTESE), tendo o **Percentual de Tabelas** calculado em 20% ( *muito baixo/baixo*).

Na seção *Simulação*, a entrada *município* tem **Guarda** *forte*, pois cada município pode ser selecionado em uma *listBox*. A **Sensibilidade** é *muito baixa*, pois não há impacto no cálculo do ICH. A **Relevância** é *irrelevante* e o **Percentual de Tabelas** é *baixo/médio* (40%), pois a entrada usa as tabelas MUNICIPIO e SINTESE.

A última entrada da funcionalidade é *novos leitos*, que tem **Guarda** definida como *média*, pois nem todas as validações necessárias estão presentes (foi possível colar um valor alfanumérico em um campo numérico). A **Sensibilidade** é *muito baixa*, pois uma variação de valores nesta entrada mas o município variar muito pouco na lista de municípios, impactando em cerca de 2% no cálculo do ICH. A **Relevância** é *opcional*, pois a entrada só é usada quando se quer fazer simulações. Finalmente, a entrada usa a tabela SINTESE, tendo o **Percentual de Tabelas** calculado em 20% ( *muito baixo/baixo*). Os valores cadastrados no SIT são mostrados na figura 41.

DESCRIÇÃO	GUARDA	SENSIBILIDADE	RELEVÂNCIA	%TABELAS
FAIXA ETARIA	100	2	60	20
MOSTRA LINHAS	100	0	0	0
PESSOAS	100	25	60	20
POSTOS DE SAUDE	100	2	60	20
LEITOS HOSPITALARES	100	25	60	20
SALDO MUNICIPIO	100	25	60	20
SALDO PER CAPITA	100	25	60	20
MUNICIPIO	100	0	0	40
LEITOS	50	2	40	20

Figura 41 – As entradas para a funcionalidade *Nova Simulação* do SICH

Entrando com esses valores das características de *Ei* no Sistema de Inferência Nebuloso do SIT, temos o cálculo dos percentuais a testar pelas técnicas clássicas e pela lógica nebulosa. A figura 42 mostra os resultados após os cálculos. Ao se colocar o ponteiro do *mouse* sobre qualquer percentual é exibida uma caixa de mensagem com os valores sugeridos para teste.

DESCRIÇÃO	PARÂMETROS NEBULOSOS	VALORES A TESTAR (TÉCNICAS CLÁSSICAS)	VALORES A TESTAR (LÓGICA NEBULOSA)
FAIXA ETARIA	100 2 60 20	100,00% (7 de 7)	29,99% (3 de 7)
MOSTRA LINHAS	100 0 0 0	100,00% (3 de 3)	10,20% (1 de 3)
PESSOAS	100 25 60 20	100,00% (6 de 6)	29,99% (2 de 6)
POSTOS DE SAUDE	100 25 60 20	100,00% (6 de 6)	29,99% (2 de 6)
LEITOS HOSPITALARES	100 25 60 20	100,00% (6 de 6)	29,99% (2 de 6)
SALDO MUNICIPIO	100 25 60 20	100,00% (6 de 6)	29,99% (2 de 6)
SALDO PER CAPITA	100 25 60 20	100,00% (6 de 6)	29,99% (2 de 6)
MUNICIPIO	100 0 0 40	1,09% (1 de 91)	14,69% (14 de 91)
LEITOS	50 2 40 20	<1,00% (1 de 1000)	14,69% (147 de 1000)

Figura 42 – A tela com o resultado dos cálculos para cada entrada do SICH.

#### 4.2.1 Análise do resultados obtidos

Ao analisar os resultados para cada entrada *Ei* do SICH, pode-se observar que o módulo de inferência nebulosa do SIT recomenda cerca de 30% de testes para as entradas *Faixa etária*, *Pessoas*, *Postos de saúde*, *Leitos hospitalares*, *Saldo município* e *Saldo per capita*, que são as entradas mais importantes para a produção da saída. Isso se deve ao fato de tais entradas terem parâmetros nebulosos semelhantes, diferindo apenas nos valores da Sensibilidade. Aparentemente, esta variação na sensibilidade não tem impacto suficiente no cálculo do percentual a testar. Os valores sugeridos pelo SIT indicam que nestes casos houve um grande ganho no total de testes a realizar no SICH, que passaram de 100% para cerca 30%. Tal ganho seria muito mais representativo se cada entrada tivesse um conjunto de valores maior.

As entradas *Mostrar linhas*, *Município* e *Leitos*, apresentaram um percentual a testar bem mais baixo que as entradas mais relevantes, o que já era esperado, indicando que a ferramenta apresenta resultados coerentes. Nestes casos, apesar das técnicas clássicas de teste apontarem um percentual menor a testar, intuitivamente o desenvolvedor ou o testador do

sistema testa um pouco mais. Por exemplo, a entrada Município tem 91 valores. As técnicas clássicas sugerem que se teste apenas 1 valor, o que pode ser insuficiente para que um erro seja identificado. Já o módulo de lógica nebulosa sugere que se teste 14 valores, o que amplia significativamente a cobertura dos testes e as chances de se encontrar um erro.



## 5 CONCLUSÕES

Na área de testes existem diversos estudos de geração de casos de teste associados a algoritmos genéticos, tentando-se obter um conjunto mínimo de testes a realizar para um software (Pargas *et al.*, 1999, Bueno e Jino, 1999 e Ferreira e Vergilio, 2005) . Os algoritmos genéticos são usados para refinar um conjunto de casos de teste inicial, geralmente escolhido de forma aleatória, até chegar a um resultado pré definido satisfatório. Também existem estudos que empregam técnicas de análise de mutantes ou usam semântica para a redução do escopo dos testes a realizar (DeMillo e Offutt, 1991 e Linkman *et al.*, 2003). Entretanto não foi encontrado nenhum trabalho que associa-se geração de testes com lógica nebulosa.

A tentativa de se indicar o total de testes a realizar em cada entrada do sistema pode não ser a ideal, uma vez que vários fatores subjetivos estão envolvidos. Apesar dessa subjetividade, o trabalho serve como um ponto de partida para um melhor planejamento da atividade de testes.

O uso da lógica nebulosa na atividade de testes de SIG contribuiu para reduzir o espaço total de testes a serem executados para alguns tipos de entrada, comparando-se com o total de testes a realizar sugeridos pelas técnicas clássicas de teste. Porém o aumento do total de testes para alguns tipos de entrada deve ser encarado como uma forma de melhorar a cobertura dos testes e, conseqüentemente, sua confiabilidade, uma vez que em alguns casos as técnicas clássicas sugerem que se teste apenas um valor.

Espera-se que esse trabalho contribua para o desenvolvimento de novos estudos sobre o assunto, aprimorando a ferramenta e fomentando discussões.

A proposta da ferramenta, inicialmente desenvolvida para testes de SIG, pode ser aplicada a sistemas de informação em geral, com alguns pequenos ajustes que se façam necessários.

Pode-se destacar alguns trabalhos futuros que podem ser definidos como melhoria e continuação deste trabalho, tais como, (i) adicionar novos tipos de entradas aos tipos já existentes; (ii) adicionar relatórios com os testes a executar para cada entrada; (iii) adicionar a possibilidade de cadastros de entradas com referências cruzadas; (iv) contribuir para a disseminação do conhecimento adquirido e os dados obtidos para que outros pesquisadores possam aproveitar as informações; e (v) divulgar o trabalho em congressos, simpósios e seminários.

O desenvolvimento das pesquisas para elaboração dessa dissertação permitiram a apresentação do RAPELLO, BERNARDO e WERNECK, (2009) no XIV Simpósio Brasileiro de Sensoriamento Remoto.

## REFERÊNCIAS BIBLIOGRÁFICAS

ARONOFF, S. **Geographical Information Systems: A Management Perspective**. Ottawa, WDI Publications, 1989.

BARBOSA, D. L.; LIMA, H. S.; MACHADO, P. D. L.; FIGUEIREDO, J. C. A.; JUCÁ, M. A.; ANDRADE, W. L. **Automating functional testing of components from uml specifications**. Int. Journal of Software Eng. and Knowledge Engineering, 17:339–358, 2007.

BEIZER, B. **Software Testing Techniques**. 2ª ed., Van Nostrand-Reinhold, 1990.

BERNARDO FILHO, O. **Verificação de Protocolos de Comunicação com Lógica Nebulosa**. Rio de Janeiro, 1999, COPPE/UFRJ, D.Sc., Engenharia Elétrica, 1999.

BERNARDO FILHO, O.; Pedroza, A.C.P.; Leão, J.L.S. **Uma ferramenta para verificação de sistemas distribuídos com lógica nebulosa: Implementação e experiências**. Artigo do Departamento de Engenharia de Sistemas e Computação/FEN-UERJ, Rio de Janeiro, 2001.

BERTOLINO, A. **Software testing research: achievements, challenges, dreams**. In: Future of Software Engineering at ICSE 2007, Minneapolis, USA, 2007.

BITTENCOURT, G. **Inteligência Artificial: ferramentas e teorias**. 2ª ed., Ed. UFSC, 2001.

BONHAM-CARTER, G.F. **Geographic Information Systems for Geocientists: Modelling with GIS**, Kindlington: Ed. Pergamon, 1994.

BUENO, P.M.S.; JINO, M. **Geração Automática de Dados e Tratamento de não Executabilidade no Teste Estrutural de Software**. In: XIII Simpósio Brasileiro de Engenharia de software, 1999, Florianopolis - S.C.. Anais do XIII Simpósio Brasileiro de Engenharia de software, 1999. v. 01.

BUENO, P.M.S.; WONG, W.E.; JINO M. **Automatic test data generation using particle systems**. In: Proceedings of the 2008 ACM symposium on applied computingm, 809-814 New York, NY, USA, 2008.

BURROUGH, P.A. **Principles of Geographic Information Systems for Land Resources Assessment**. Oxford, Oxford University Press, 1986.

CÂMARA, G.; FREITAS, U.M.; SOUZA, R.C.M.; GARRIDO, J. **SPRING: Integrating Remote Sensing and GIS by Object-Oriented Data Modelling**. Computers and Graphics, vol. 15, n.6, July 1996.

CARTAXO, E. G.; ANDRADE, W. L.; OLIVEIRA NETO, F. G.; MACHADO, P. D. L. **LTSBT: a tool to generate and select functional test cases for embedded systems**. In: SAC '08: Proceedings of the 2008 ACM symposium on Applied computing, volume 2, pages 1540–1544, New York, NY, USA, ACM, 2008.

CASANOVA, M.; CÂMARA, G.; DAVIS, C.; VINHAS, L.; QUEIROZ, G.R. **Bancos de Dados Geográficos**. <http://www.dpi.inpe.br/livros/bdados/index.html> (acessado 28 Junho. 2007)

COWEN, D.J. **GIS versus CAD versus DBMS: what are the differences**. Photogrammetric Engineering and Remote Sensing, 54:1551-4, 1988.

COX, E. **The fuzzy systems handbook: a practitioner's guide to building, using, and maintaining fuzzy systems**. 1994.

CRESPO, A. **Aplicação da Norma IEEE 829 como Mecanismo de Gerência do Processo de Teste de Produtos de Software**. CenPRA, Campinas, 2003.

DAVIS, A. *et al.* **Identifying and Measuring Quality in a Software Requirements Specification**. Proc. First In: Software Metrics Symposium, IEEE, Baltimore, MD, 1993.

DAVIS, C.; CÂMARA, G. **Arquitetura de Sistemas de Informação Geográfica. Introdução à Ciência da Geoinformação**. cap.3, São José dos Campos, 2001, Disponível em: <<http://www.dpi.inpe.br/gilberto/livro/introd/>>. Acesso em: 30.out.2008.

DEMILLO, R.A.; OFFUTT, A.J. **Constraint-based automatic test data generation**. IEEE Transactions on Software Engineering, pp. 900-910, 1991.

DEUTSCH, M. **Verification and Validation**. In: Software Engineering (R. Jensen e Tonies eds.), Prentice-Hall, 1979, pp. 329-408.

DELAMARO, M.E.; MALDONADO, J.C.; JINO, M. **Introdução ao teste de software**. Ed. Elsevier, Rio de Janeiro, 2007.

FERREIRA, L.P.; VERGILIO, S.R. **Tdsgen: An environment based on hybrid genetic algorithms for generation of test data**. In: 17th International Conference on Software Engineering and Knowledge Engineering, volume 3103/2004. P. 1.431-1.432, Springer, 2005.

FONSECA, O.L.H. **Aplicação de métodos de análise espacial e da teoria dos conjuntos nebulosos em estudos sobre Pobreza**. 146p. Dissertação (Mestrado em Engenharia de computação - Área de concentração Geomática). Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2003.

IBM Corporation **DB2 Spatial Extender User's Guide and Reference** [online]. Disponível em <http://www-3.ibm.com>, 2001.

IBM Corporation **Working with the Geodetic and Spatial DataBlade Modules** [online]. Disponível em [http://www-3.ibm.com/software/data/informix/pubs/manuals/geo\\_spatial.pdf](http://www-3.ibm.com/software/data/informix/pubs/manuals/geo_spatial.pdf), 2002.

LINKMAN, S.; VICENZI, A.M.R.; MALDONADO, J. **An evaluation of systematic functional testing using mutation testing**. In: 7th International Conference on Empirical Assessment in Software Engineering – EASE, Keele, UK, abr. 2003, The IEEE.

MARANHÃO, M.R.A. **Modelo de seleção de áreas para atualização do Mapeamento Sistemático baseado em Lógica Nebulosa**. Rio de Janeiro, 2005. 102p. Dissertação (Mestrado em Engenharia de computação - Área de concentração Geomática). Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2005.

**Matlab Fuzzy Logic Toolbox-Tutorial.** USA. The Mathworks Inc. pp.2-19.

OGC **The OpenGIS™ Guide - Introduction to Interoperable Geoprocessing.** Massachusetts, USA: Open GIS Consortium , 1996.

OLIVEIRA JR., H.A. **Lógica Difusa : Aspectos Práticos e Aplicações.** Rio de Janeiro : Ed. Interciência, 1999. 192p.

OLIVEIRA NETO, F G; CARTAXO, E. G.; MACHADO, P. D. L.; FELIPE O. J. **Reducing the Size of Test Cases Based on Similarities.** In: 2nd Brazilian Workshop on Systematic and Automated Software Testing, 2008, Campinas. 2nd Brazilian Workshop on Systematic and Automated Software Testing, 2008.

PARGAS, R. P.; HARROLD, M. J.; PECK, R. R. **Test-data generation using genetic algorithms.** Software Testing, Verification and Reliability, Vol. 9, No. 4. (1999), pp. 263-282, 1999.

PRESSMAN, R.S. **Engenharia de Software.** 6ª ed., Rio de Janeiro, McGraw-Hill Interamericana, 2006.

RAPELLO, C.N. **Sistema de Cálculo do Índice de Carência Hospitalar – SICH.** Sistema desenvolvido na disciplina de Engenharia de Software do curso de Pós-graduação em Engenharia da Computação da UERJ – Geomática , Rio de Janeiro, 2007

RAPELLO, C. N. ; BERNARDO FILHO, O. ; WERNECK, V. M. B. . **Testes de sistemas de informações geográficas com lógica nebulosa.** In: XIV SBSR - Simpósio Brasileiro de Sensoriamento Remoto, 2009, Natal. Anais do XIV SBSR - Simpósio Brasileiro de Sensoriamento Remoto. São Paulo : Instituto Nacional de Pesquisas Espaciais (INPE), 2009. v. 1. p. 4949-4976.

RAMSEY, P. **PostGis Manual** [online]. Disponível em <http://postgis.refractory.net>, 2002.

RAVADA, S.; SHARMA, J. **Oracle8i Spatial: Experiences with Extensible Databases**. In: 6th International Symposium on Advances in Spatial Databases. Springer-Verlag, London, UK, 1999. p. 355-359.

SALMASO, F.V.; BERNARDO FILHO, O.; RIBEIRO, J.A. **Sistema de Informação Geográfica para Apoio à Tomada de Decisão no Setor de Cobrança – SIGATDSC**. Anais do XXIII Congresso Brasileiro de Cartografia, 2007.

SANDRI, S.; CORREA, C. **Lógica Nebulosa**. Anais da V Escola de Redes Neurais. São José dos Campos, 1999, pp 073-090.

SOMMERVILLE, I. **Engenharia de Software**. 8ª Ed., Pearson Addison-Wesley, 2007.

SOUZA, F.J. **Apostila de aula de Inteligência Artificial do Programa de Pós-Graduação em Engenharia de Computação da UERJ - Área de concentração Geomática**. Rio de Janeiro, 2001.

TANSCHKEIT, R. **Fundamentos de Lógica Fuzzy e Controle Fuzzy**. Apostila de aula do Departamento de Engenharia Eletrônica da PUC-RJ, Rio de Janeiro, 1999.

TERRALIB [online]. Disponível em: [www.terralib.org](http://www.terralib.org), 2003.

ZADEH, L.A. **Fuzzy Sets, Information and Control**. No.8, pp 338-353, 1965.

Vilfredo Pareto – Wikipedia, a enciclopédia livre [homepage na internet] . [Acesso em 2009 Abr 2]. Disponível em [http://pt.wikipedia.org/wiki/Vilfredo\\_Pareto](http://pt.wikipedia.org/wiki/Vilfredo_Pareto).

WERNECK, V.M.B.; MORAES, E.A. **Uma abordagem de Avaliação de Qualidade de Aplicações Web**. In: Cadernos do IME; 2003, pp. 71-78.

## **APÊNDICE A – Descrição dos Casos do Uso**

### **1 Cadastrar Sistema**

Nome: Cadastrar Sistema

Ator: Usuário

Descrição: Cadastra um novo sistema.

Fluxo Normal

1. Usuário informa nome do novo sistema;
2. Sistema exibe mensagem confirmando o cadastro;

Fluxos Alternativos

Passo 1: Caso o sistema informado já esteja cadastrado

- Sistema exibe mensagem de sistema já cadastrado;
- Retornar ao passo 1;

### **2 Excluir Sistema**

Nome: Excluir Sistema

Ator: Usuário

Descrição: Exclui um sistema cadastrado.

Fluxo Normal

1. Usuário informa o número do sistema a excluir;
2. Sistema exibe mensagem confirmando a exclusão;

Fluxos Alternativos

Passo 1: Caso o sistema informado não esteja cadastrado

- Sistema exibe mensagem de sistema não cadastrado;
- Retornar ao passo 1;

### **3 Cadastrar Entrada**

Nome: Cadastrar Entrada



Ator: Usuário

Descrição: Cadastra uma nova entrada.

Fluxo Normal

1. Usuário informa nome da nova entrada;
2. Sistema exibe mensagem confirmando o cadastro;

Fluxos Alternativos

Passo 1: Caso a entrada informada já esteja cadastrada

- Sistema exibe mensagem de entrada já cadastrada;
- Retornar ao passo 1;

#### **4 Excluir Entrada**

Nome: Excluir Entrada

Ator: Usuário

Descrição: Exclui entradas cadastradas.

Fluxo Normal

1. Usuário seleciona uma ou mais entradas a excluir;
2. Sistema exibe mensagem confirmando a exclusão;

Fluxos Alternativos

Passo 1: Caso nenhuma entrada tenha sido selecionada

- Sistema exibe mensagem de nenhuma entrada selecionada;
- Retornar ao passo 1;

#### **5 Gerar Testes**

Nome: Gerar Testes

Ator: Usuário

Descrição: Gera os testes para as entradas selecionadas.

Fluxo Normal

1. Usuário seleciona um sistema;
2. Usuário seleciona uma ou mais entradas;
3. Sistema exibe os testes gerados;

#### Fluxos Alternativos

Passo 1: Caso não haja nenhum sistema cadastrado

- Chama o caso de uso Cadastrar Sistema;
- Retornar ao passo 1;

Passo 1: Caso não haja nenhuma entrada cadastrada para o sistema

- Chama o caso de uso Cadastrar Entrada;
- Retornar ao passo 2;

Passo 2: Caso nenhuma entrada tenha sido selecionada

- Sistema exibe mensagem de nenhuma entrada selecionada;
- Retornar ao passo 2;

## 6 Listar Regras

Nome: Listar Regras

Ator: Usuário

Descrição: Lista as regras dos Sistemas de Inferência Nebulosos.

Fluxo Normal

1. Usuário solicita a lista das regras;
2. Sistema exibe as regras de todos os SIN;

Fluxos Alternativos

Não há.

## APÊNDICE B – Código fonte da DLL

### 1 clsFuzzy.cls

```

Private oBD As clsFuncoesBD
Private antecedente1() As tipoAntecedenteConsequente
Private antecedente2() As tipoAntecedenteConsequente
Private consequente() As tipoAntecedenteConsequente

Private consequenteSelecionado() As tipoAntecedenteConsequente

Private regrasFAM() As tipoRegrasFAM

Private Sub Class_Initialize()
    'inicializa a classe funcoesBD
    Set oBD = New clsFuncoesBD
End Sub

Public Function inicia(      ByVal dEntradaTermo1 As Double, _
                            ByVal dEntradaTermo2 As Double, _
                            ByVal dEntradaTermo3 As Double, _
                            ByVal dEntradaTermo4 As Double) As Double

    Dim i As Integer, j As Integer
    '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    'conecta - retorna true/false
    If oBD.conectaBanco Then
    End If

    Dim dValorSaidaSIF1 As Double, dValorSaidaSIF2 As Double, dValorSaidaSIF3 As Double

    dValorSaidaSIF1 = calculaValorSaidaSIF(1, dEntradaTermo1, dEntradaTermo2)
    dValorSaidaSIF2 = calculaValorSaidaSIF(2, dEntradaTermo3, dEntradaTermo4)
    dValorSaidaSIF3 = calculaValorSaidaSIF(3, dValorSaidaSIF1, dValorSaidaSIF2)

    'desconecta
    Call oBD.desconectaBanco

    inicia = Round(dValorSaidaSIF3, 4)
End Function

Private Function calculaValorSaidaSIF(ByVal iSIF As Integer, _
                                      ByVal dEntradaTermo1 As Double, _
                                      ByVal dEntradaTermo2 As Double) As Double

    Dim dMi1 As Double, dMi2 As Double, dMiSelecionado As Double
    Dim dSomatorioCentroide As Double, dSomatorioArea As Double, dCentroideRegra As Double,
    dAreaRegra As Double
    Dim dValorFinal As Double

```

```

'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'carrega partes dos termos linguisticos - retorna true/false
If carregaPartesTermosLinguisticos(iSIF, "1", antecedente1) Then
End If

If carregaPartesTermosLinguisticos(iSIF, "2", antecedente2) Then
End If

If carregaPartesTermosLinguisticos(iSIF, "C", consequente) Then
End If

'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'carrega regras FAM
If carregaRegrasFAM(iSIF) Then
End If

'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'Percorrer as regras e calcula o valor final pelo método do centróide
dSomatorioCentroide = 0
dSomatorioArea = 0

For i = 1 To UBound(regrasFAM)
'calcular Mi de cada termo
dMi1 = obterMi(dEntradaTermo1, antecedente1, regrasFAM(i).sTermoLingAntecedente1)
dMi2 = obterMi(dEntradaTermo2, antecedente2, regrasFAM(i).sTermoLingAntecedente2)

'Selecionar o menor Mi
dMiSelecioneado = IIf(dMi1 < dMi2, dMi1, dMi2)

'Se o Mi selecionado for maior que zero, acumulo o valor
If dMiSelecioneado > 0 Then
'obter consequente
Call obterConsequente(regrasFAM(i).sTermoLingConsequente, dMiSelecioneado)

'Calcular centroide
dCentroideRegra = 0
dAreaRegra = 0
Call calcularCentroideRegra(dCentroideRegra, dAreaRegra)

'Acumular
dSomatorioCentroide = dSomatorioCentroide + dCentroideRegra ' * dAreaRegra
dSomatorioArea = dSomatorioArea + dAreaRegra
End If
Next i

calculaValorSaidaSIF = dSomatorioCentroide / dSomatorioArea

```

End Function

```
Private Sub calcularCentroideRegra(ByRef dCentroideRegra As Double, ByRef dAreaRegra As Double)
```

```
    Dim i As Integer
```

```
    Dim a As Double, b As Double, m As Double, q As Double
```

```
    Dim dCentroideAux As Double, dAreaAux As Double
```

```
    dCentroideRegra = 0
```

```
    dAreaRegra = 0
```

```
    For i = 1 To UBound(consequenteSelecioneado)
```

```
        m = consequenteSelecioneado(i).dCoefAngular
```

```
        q = consequenteSelecioneado(i).dCoefLinear
```

```
        a = consequenteSelecioneado(i).dXIni
```

```
        b = consequenteSelecioneado(i).dXFim
```

```
        'Centróide =  $m(b^3)/3 + q(b^2)/2 - m(a^3)/3 - q(a^2)/2$ 
```

```
        dCentroideAux = ((m * (b ^ 3)) / 3) + ((q * (b ^ 2)) / 2) - _  
                        ((m * (a ^ 3)) / 3) - ((q * (a ^ 2)) / 2)
```

```
        'Área =  $m(b^2)/2 + qb - m(a^2)/2 - qa$ 
```

```
        dAreaAux = ((m * (b ^ 2)) / 2) + (q * b) - _  
                  ((m * (a ^ 2)) / 2) - (q * a)
```

```
        dCentroideRegra = dCentroideRegra + dCentroideAux
```

```
        dAreaRegra = dAreaRegra + dAreaAux
```

```
    Next i
```

End Sub

```
Private Sub obterConsequente(ByVal sTermoLing As String, dMiSelecioneado As Double)
```

```
    Dim i As Integer, bAlterou As Boolean
```

```
    'Selecionar consequentes de acordo com o termo lingüístico
```

```
    ReDim consequenteSelecioneado(0)
```

```
    For i = 1 To UBound(consequente)
```

```
        If Trim(consequente(i).sTermoLing) = Trim(sTermoLing) Then
```

```
            ReDim Preserve consequenteSelecioneado(UBound(consequenteSelecioneado) + 1)
```

```
            consequenteSelecioneado(UBound(consequenteSelecioneado)) = consequente(i)
```

```
        End If
```

```
    Next i
```

```
    'Atenuar consequente selecionado
```

```
    For i = 1 To UBound(consequenteSelecioneado)
```

```
        bAlterou = False
```

```
        If consequenteSelecioneado(i).dYIni > dMiSelecioneado Then
```

```
            bAlterou = True
```

```
            consequenteSelecioneado(i).dYIni = dMiSelecioneado
```

```
        End If
```

```

    If consequenteSelecionado(i).dYFim > dMiSelecionado Then
        bAlterou = True
        consequenteSelecionado(i).dYFim = dMiSelecionado
    End If

    If bAlterou Then
        'Calcula os novos coeficientes da reta
        Call calculaCoeficientesReta(consequenteSelecionado, i)
    End If
Next i

End Sub

Private Function obterMi(ByVal dEntrada As Double, _
                        antecedente() As tipoAntecedenteConsequente, _
                        sTermoLing As String)

    Dim i As Integer, dMi As Double, dAux As Double

    dMi = 0
    For i = 1 To UBound(antecedente)
        'se for o termo linguístico
        If Trim(antecedente(i).sTermoLing) = Trim(sTermoLing) Then
            'se estiver dentro do intervalo
            If (dEntrada >= antecedente(i).dXIni) And dEntrada <= antecedente(i).dXFim Then
                'Está dentro do intervalo
                'Y = mX + q
                dAux = (antecedente(i).dCoefAngular * dEntrada) + antecedente(i).dCoefLinear

                If dMi < dAux Then
                    dMi = dAux
                End If
            End If
        End If
    Next i
    obterMi = dMi

End Function

Private Function carregaPartesTermosLinguisticos(ByVal iSIF As Integer, _
        ByVal sTipoAntecedente As String, _
        ByRef antecedente() As tipoAntecedenteConsequente) As Boolean
    ReDim antecedente(0)
    carregaPartesTermosLinguisticos = False

    sql = "Select" & Chr(10)
    sql = sql & "S01.S01_TX_DESCRICAO As ANTECEDENTE," & Chr(10)
    sql = sql & "S02.S02_SG_TERMOS As TERMOLING," & Chr(10)
    sql = sql & "S03.S03_NR_PARTE As PARTE," & Chr(10)
    sql = sql & "S03.S03_NR_X_INI As XINI," & Chr(10)
    sql = sql & "S03.S03_NR_X_FIM As XFIM," & Chr(10)
    sql = sql & "S03.S03_NR_Y_INI As YINI," & Chr(10)

```

```

sql = sql & "S03.S03_NR_Y_FIM As YFIM" & Chr(10)

sql = sql & "From" & Chr(10)
sql = sql & "SIT_S01_ANTECEDENTES S01," & Chr(10)
sql = sql & "SIT_S02_TERMOS_LINGUISTICOS S02," & Chr(10)
sql = sql & "SIT_S03_PARTES_TERMOS_LINGUISTICOS S03" & Chr(10)

sql = sql & "Where" & Chr(10)
sql = sql & "S01.S01_NR_ANTECEDENTE      = S03.S01_NR_ANTECEDENTE" & Chr(10)
sql = sql & "And S02.S01_NR_ANTECEDENTE = S03.S01_NR_ANTECEDENTE" & Chr(10)
sql = sql & "And S02.S02_NR_TERMOS      = S03.S02_NR_TERMOS" & Chr(10)
sql = sql & "And S03.S01_NR_ANTECEDENTE = (Select distinct S01_NR_ANTECEDENTE_" &
sTipoAntecedente & Chr(10) '1/2/C
sql = sql & "
                                From SIT_S04_REGRAS_FAM" & Chr(10)
sql = sql & "
                                Where S04_NR_SIF = " & iSIF & ")" & Chr(10)

If oBD.sqlConsulta(sql) Then
    Set rs = oBD.getRs
    While Not rs.EOF
        ReDim Preserve antecedente(UBound(antecedente) + 1)
        antecedente(UBound(antecedente)).sNome = rs("ANTECEDENTE")
        antecedente(UBound(antecedente)).sTermoLing = rs("TERMOLING")
        antecedente(UBound(antecedente)).iParte = rs("PARTE")
        antecedente(UBound(antecedente)).dXIni = rs("XINI")
        antecedente(UBound(antecedente)).dXFim = rs("XFIM")
        antecedente(UBound(antecedente)).dYIni = rs("YINI")
        antecedente(UBound(antecedente)).dYFim = rs("YFIM")

        'Calcula os coeficientes da reta
        Call calculaCoeficientesReta(antecedente, UBound(antecedente))
        rs.MoveNext
    Wend
    carregaPartesTermosLinguisticos = True
End If
End Function

Private Function carregaRegrasFAM(ByVal iSIF As Integer) As Boolean
    carregaRegrasFAM = False
    ReDim regrasFAM(0)

    sql = "SELECT" & Chr(10)
    sql = sql & "S02A.S02_SG_TERMOS As TERMO1," & Chr(10)
    sql = sql & "S02B.S02_SG_TERMOS As TERMO2," & Chr(10)
    sql = sql & "S02C.S02_SG_TERMOS As TERMOC" & Chr(10)

    sql = sql & "From" & Chr(10)
    sql = sql & "SIT_S04_REGRAS_FAM S04," & Chr(10)
    sql = sql & "SIT_S02_TERMOS_LINGUISTICOS S02A," & Chr(10)
    sql = sql & "SIT_S02_TERMOS_LINGUISTICOS S02B," & Chr(10)
    sql = sql & "SIT_S02_TERMOS_LINGUISTICOS S02C" & Chr(10)

```

```

sql = sql & "Where" & Chr(10)
sql = sql & "S02A.S01_NR_ANTECEDENTE = S04.S01_NR_ANTECEDENTE_1" & Chr(10)
sql = sql & "And S02A.S02_NR_TERMOS = S04.S02_NR_TERMOS_1" & Chr(10)
sql = sql & "And S02B.S01_NR_ANTECEDENTE = S04.S01_NR_ANTECEDENTE_2" & Chr(10)
sql = sql & "And S02B.S02_NR_TERMOS = S04.S02_NR_TERMOS_2" & Chr(10)
sql = sql & "And S02C.S01_NR_ANTECEDENTE = S04.S01_NR_ANTECEDENTE_C" & Chr(10)
sql = sql & "And S02C.S02_NR_TERMOS = S04.S02_NR_TERMOS_C" & Chr(10)
sql = sql & "And S04.S04_NR_SIF = " & iSIF & Chr(10)

sql = sql & "Order By" & Chr(10)
sql = sql & "S04.S02_NR_TERMOS_1, S04.S02_NR_TERMOS_2"

If oBD.sqlConsulta(sql) Then
    Set rs = oBD.getRs
    While Not rs.EOF
        ReDim Preserve regrasFAM(UBound(regrasFAM) + 1)
        regrasFAM(UBound(regrasFAM)).sTermoLingAntecedente1 = rs("TERMO1")
        regrasFAM(UBound(regrasFAM)).sTermoLingAntecedente2 = rs("TERMO2")
        regrasFAM(UBound(regrasFAM)).sTermoLingConsequente = rs("TERMOC")
        rs.MoveNext
    Wend

    carregaRegrasFAM = True
End If
End Function

Private Sub calculaCoeficientesReta(antecedente() As tipoAntecedenteConsequente, indice As Integer)
    Dim dAngular As Double, dLinear As Double
    Dim X1 As Double, X2 As Double, Y1 As Double, Y2 As Double

    X1 = antecedente(indice).dXIni
    X2 = antecedente(indice).dXFim
    Y1 = antecedente(indice).dYIni
    Y2 = antecedente(indice).dYFim

    dAngular = (Y1 - Y2) / (X1 - X2)
    dLinear = Y1 - (dAngular * X1)

    antecedente(indice).dCoefAngular = Round(dAngular, 5)
    antecedente(indice).dCoefLinear = Round(dLinear, 5)
End Sub

```



## 2 clsFuncoesBD.cls

```

Private sMensagem As String
Private oConn As ADODB.Connection
Private rs As ADODB.Recordset

Public Property Get getRs() As ADODB.Recordset
    'retorna o último recordset
    Set getRs = rs.Clone()
End Property

Public Property Get getMensagem() As String
    'retorna mensagem informativa
    getMensagem = sMensagem
    sMensagem = ""
End Property

Public Function conectaBanco() As Boolean
    Dim sCaminho As String

    conectaBanco = False
    sCaminho = App.Path & "\" & CONST_NOME_BANCO

    If Trim(CONST_NOME_BANCO) = "" Then
        sMensagem = "Banco de dados não informado"
    Else
        Set oConn = New ADODB.Connection

        On Error Resume Next
        oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & sCaminho & ";"
        If Err.Number = 0 Then
            conectaBanco = True
        Else
            'Trata erro
            sMensagem = "Erro ao conectar o banco de dados '" & sNomeBanco & "'"
            Err.Clear
        End If
    End If

    'MsgBox "oConn.State:" & oConn.State '1=conectado

End Function

Public Sub desconectaBanco()

```

```

        oConn.Close
        Set oConn = Nothing
    End Sub

Public Function sqlConsulta(sql As String) As Boolean
    sqlConsulta = False

    Set rs = New ADODB.Recordset
    rs.Open sql, oConn, adOpenKeyset, adLockOptimistic

    If Err.Number <> 0 Then
        'Trata erro
        sMensagem = "Erro ao executar a consulta SQL."
        Err.Clear
    Else
        sqlConsulta = True
    End If

End Function

Public Function sqlExecuta(sql As String) As Boolean
    Dim rs As ADODB.Recordset
    Dim regAfetados

    sqlExecuta = False
    '    intRegistros = 0

    Set rs = oConn.Execute(sql, regAfetados)
    If Err.Number <> 0 Then
        'Trata erro
        strMensagem = "Erro ao executar o comando SQL."
        Err.Clear
    Else
        'intRegistros = regAfetados
        sqlExecuta = True
    End If

    'destrói o objeto
    Set rs = Nothing
End Function

```

### 3 global.bas

```
Option Explicit
```

```
Public Const CONST_NOME_BANCO As String = "bdSTI.mdb"
```

```
Public sql As String
```

```
Public rsGbl As ADODB.Recordset
```

```
Type tipoAntecedenteConsequente
```

```
    sNome As String
```

```
    sTermoLing As String
```

```
    iParte As Integer
```

```
    dXIni As Double
```

```
    dXFim As Double
```

```
    dYIni As Double
```

```
    dYFim As Double
```

```
    dCoefAngular As Double
```

```
    dCoefLinear As Double
```

```
    sSelecioneado As String
```

```
End Type
```

```
Type tipoRegrasFAM
```

```
    'sAntecedente1 As String
```

```
    'sAntecedente2 As String
```

```
    'sConsequente As String
```

```
    sTermoLingAntecedente1 As String
```

```
    sTermoLingAntecedente2 As String
```

```
    sTermoLingConsequente As String
```

```
End Type
```

## APÊNDICE C – Regras de Inferência

N.º	SIN	Regra
01	1	Se Guarda é Fraca e Sensibilidade é Muito baixa Então Impacto é Pouco
02	1	Se Guarda é Fraca e Sensibilidade é Baixa Então Impacto é Medio
03	1	Se Guarda é Fraca e Sensibilidade é Media Então Impacto é Medio
04	1	Se Guarda é Fraca e Sensibilidade é Alta Então Impacto é Alto
05	1	Se Guarda é Fraca e Sensibilidade é Muito alta Então Impacto é Muito alto
06	1	Se Guarda é Media e Sensibilidade é Muito baixa Então Impacto é Muito pouco
07	1	Se Guarda é Media e Sensibilidade é Baixa Então Impacto é Pouco
08	1	Se Guarda é Media e Sensibilidade é Media Então Impacto é Medio
09	1	Se Guarda é Media e Sensibilidade é Alta Então Impacto é Medio
10	1	Se Guarda é Media e Sensibilidade é Muito alta Então Impacto é Alto
11	1	Se Guarda é Forte e Sensibilidade é Muito baixa Então Impacto é Muito pouco
12	1	Se Guarda é Forte e Sensibilidade é Baixa Então Impacto é Muito pouco
13	1	Se Guarda é Forte e Sensibilidade é Media Então Impacto é Pouco
14	1	Se Guarda é Forte e Sensibilidade é Alta Então Impacto é Medio
15	1	Se Guarda é Forte e Sensibilidade é Muito alta Então Impacto é Medio
16	2	Se Relevancia é Irrelevante e Perctabelas é Muito baixo Então Influencia é Muito pouca
17	2	Se Relevancia é Irrelevante e Perctabelas é Baixo Então Influencia é Muito pouca
18	2	Se Relevancia é Irrelevante e Perctabelas é Medio Então Influencia é Pouca
19	2	Se Relevancia é Irrelevante e Perctabelas é Alto Então Influencia é Media
20	2	Se Relevancia é Irrelevante e Perctabelas é Muito alto Então Influencia é Media
21	2	Se Relevancia é Opcional e Perctabelas é Muito baixo Então Influencia é Muito pouca
22	2	Se Relevancia é Opcional e Perctabelas é Baixo Então Influencia é Pouca
23	2	Se Relevancia é Opcional e Perctabelas é Medio Então Influencia é Media
24	2	Se Relevancia é Opcional e Perctabelas é Alto Então Influencia é Media
25	2	Se Relevancia é Opcional e Perctabelas é Muito alto Então Influencia é Alta
26	2	Se Relevancia é Desejavel e Perctabelas é Muito baixo Então Influencia é Pouca
27	2	Se Relevancia é Desejavel e Perctabelas é Baixo Então Influencia é Media
28	2	Se Relevancia é Desejavel e Perctabelas é Medio Então Influencia é Media
29	2	Se Relevancia é Desejavel e Perctabelas é Alto Então Influencia é Alta
30	2	Se Relevancia é Desejavel e Perctabelas é Muito alto Então Influencia é Muito alta
31	2	Se Relevancia é Obrigatorio e Perctabelas é Muito baixo Então Influencia é Media
32	2	Se Relevancia é Obrigatorio e Perctabelas é Baixo Então Influencia é Media
33	2	Se Relevancia é Obrigatorio e Perctabelas é Medio Então Influencia é Alta

34	2	Se Relevancia é Obrigatorio e Perctabelas é Alto Então Influencia é Muito alta
35	2	Se Relevancia é Obrigatorio e Perctabelas é Muito alto Então Influencia é Muito alta
36	3	Se Impacto é Muito pouco e Influencia é Muito pouca Então Valoresatestar é Muito pouco
37	3	Se Impacto é Muito pouco e Influencia é Pouca Então Valoresatestar é Pouco
38	3	Se Impacto é Muito pouco e Influencia é Media Então Valoresatestar é Pouco
39	3	Se Impacto é Muito pouco e Influencia é Alta Então Valoresatestar é Medio
40	3	Se Impacto é Muito pouco e Influencia é Muito alta Então Valoresatestar é Medio
41	3	Se Impacto é Pouco e Influencia é Muito pouca Então Valoresatestar é Pouco
42	3	Se Impacto é Pouco e Influencia é Pouca Então Valoresatestar é Pouco
43	3	Se Impacto é Pouco e Influencia é Media Então Valoresatestar é Medio
44	3	Se Impacto é Pouco e Influencia é Alta Então Valoresatestar é Medio
45	3	Se Impacto é Pouco e Influencia é Muito alta Então Valoresatestar é Alto
46	3	Se Impacto é Medio e Influencia é Muito pouca Então Valoresatestar é Pouco
47	3	Se Impacto é Medio e Influencia é Pouca Então Valoresatestar é Medio
48	3	Se Impacto é Medio e Influencia é Media Então Valoresatestar é Medio
49	3	Se Impacto é Medio e Influencia é Alta Então Valoresatestar é Alto
50	3	Se Impacto é Medio e Influencia é Muito alta Então Valoresatestar é Alto
51	3	Se Impacto é Alto e Influencia é Muito pouca Então Valoresatestar é Medio
52	3	Se Impacto é Alto e Influencia é Pouca Então Valoresatestar é Medio
53	3	Se Impacto é Alto e Influencia é Media Então Valoresatestar é Alto
54	3	Se Impacto é Alto e Influencia é Alta Então Valoresatestar é Alto
55	3	Se Impacto é Alto e Influencia é Muito alta Então Valoresatestar é Muito alto
56	3	Se Impacto é Muito alto e Influencia é Muito pouca Então Valoresatestar é Medio
57	3	Se Impacto é Muito alto e Influencia é Pouca Então Valoresatestar é Alto
58	3	Se Impacto é Muito alto e Influencia é Media Então Valoresatestar é Alto
59	3	Se Impacto é Muito alto e Influencia é Alta Então Valoresatestar é Muito alto
60	3	Se Impacto é Muito alto e Influencia é Muito alta Então Valoresatestar é Muito alto