**Gustavo Alexandre de Souza**

# *Utilizando Redes Neurais Artificiais para Modelar a Confiabilidade de Software*

Curitiba - Paraná

2004

**Gustavo Alexandre de Souza**

# *Utilizando Redes Neurais Artificiais para Modelar a Confiabilidade de Software*

Dissertação apresentada à Coordenação do Curso de Mestrado em Informática da Universidade Federal do Paraná para a obtenção do título de Mestre em Informática.

Orientadora: Silvia Regina Vergilio

MESTRADO EM INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA
SETOR DE CIÊNCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ

Curitiba - Paraná

2004

GUSTAVO ALEXANDRE DE SOUZA

# UTILIZANDO REDES NEURAIS ARTIFICIAIS PARA MODELAR A CONFIABILIDADE DE SOFTWARE

Dissertação apresentada à Coordenação do Curso de Mestrado em Informática, Departamento de Informática, Setor de Ciências Exatas da Universidade Federal do Paraná para a obtenção do título de Mestre em Informática.

Orientadora: Prof.ª Dr.ª Silvia Regina Vergilio

CURITIBA

2004

**Ministério da Educação**
**Universidade Federal do Paraná**
**Mestrado em Informática**

# PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno *Gustavo Alexandre de Souza*, avaliamos o trabalho intitulado, "*Utilizando Redes Neurais para Modelar a Confiabilidade de Software*", cuja defesa foi realizada no dia 22 de setembro de 2004, às treze horas, no Auditório do Departamento de Informática do Setor de Ciencias Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 22 de setembro de 2004.

Profª. Dra. Silvia Regina Vergilio
**DINF/UFPR** – Orientadora

Prof. Dr. Mario Jino
**UNICAMP** – Membro Externo

Profª. Dra. Aurora Ramirez Trinidad Pozo
**DINF/UFPR** – Membro Interno

# *Conteúdo*

# *Lista de Figuras*

# *Lista de Tabelas*

# *Agradecimentos*

Dedico meus sinceros agradecimentos para:

– a professora doutora Silvia Regina Vergilio, pela orienatação e incentivo;

– a professora doutora Aurora Trindad Ramirez Pozo, pela co–orienatação durante o curso de mestrado;

– o coordenador do curso de Mestrado em Informática, professor doutor Alexandre Direne, pelo apoio sempre manifestado;

– o professor doutor Eduardo Parente, por sua ajuda no aprendizado de redes neurais artificiais;

– a professora doutora Renate Sitte, pelos esclarecimentos quanto ao seu trabalho;

– o meu amigo Giorgio, pela revisão do trabalho;

– e todos os colegas do Mestrado em Informática da UFPR.

*Dedico esta dissertação a meus pais,*
*que sempre me impulsionaram a ir mais adiante,*
*e para minha companheira Romine, que tem*
*me dado apoio nos momentos mais difíceis.*

# *Resumo*

A confiabilidade de um sistema é uma característica fundamental pois ela significa a probabilidade de sucesso durante sua operação. Em outras palavras, a confiabilidade é a probabilidade de um sistema funcionar quando submetido a uma determinada condição de operação. Se a confiabilidade é baixa, significa que o sistema apresenta muitas falhas. Quando estamos falando de confiabilidade de software, estas falhas estão associadas a um conjunto de defeitos no código, introduzidos durante o desenvolvimento do programa.

Os Modelos de Crescimento de Confiabilidade de Software são modelos matemáticos utilizados para estimar a probabilidade de um programa falhar ao longo do tempo. Com estes modelos, gerentes de projetos podem estimar o tempo necessário para testar um programa e fazer a sua entrega com um nível de confiabilidade aceitável. Atualmente, existem vários modelos de confiabilidade com capacidades diferentes para modelar diferentes tipos de comportamento. Para aplicar um modelo, é necessário fazer uma pré análise dos dados de falha obtidos para descobrir a tendência dos dados para depois selecionar um modelo que melhor modele o comportamento observado. Porém, cada projeto tem um comportamento de crescimento de confiabilidade diferente e embora existam vários modelos analíticos diferentes, nenhum destes consegue ter um bom desempenho em bases de projetos com características diferentes das suposições consideradas por cada um. Estudos realizados com redes neurais artificiais comprovaram a eficiência desta técnica aplicada a modelagem da confiabilidade de software. As redes neurais artificiais são estruturas computacionais cujo comportamento é adaptado a partir de exemplos até que se obtenha uma solução ótima para um determinado problema. Podemos entender as redes neurais como funções matemáticas que se adaptam aos dados observados para se obter um modelo desejado.

O objetivo deste trabalho é explorar o uso de redes neurais para modelar a confiabilidade de software. Foram realizados dois experimentos. Um experimento utilizando modelos baseados em tempo aplicados a vários projetos diferentes para comparar a adaptabilidade e a capacidade de predição dos modelos neurais comparados aos modelos analíticos. No segundo experimento, foram utilizados modelos baseados em cobertura de teste para verificar a aplicabilidade dos modelos neurais para este tipo de dados de falha.

# *Abstract*

The system reliability is a fundamental quality characteristic. Reliability is the system's probability of successful operation when operated under a specific condition. If reliability is low, that means the system presents several failures. When considering software reliability, these failures are associated with a set of defects or faults in the code introduced during the development process.

The software reliability growth models are mathematical models used to estimate the probability of software's failure over time. Using these models, project managers are able to estimate the necessary time to test and release the software at an acceptable reliability level. Nowadays, there are several reliability models with different characteristics to model different behaviors. To select a model that better represents the observed behavior, it is necessary to analyze the previously obtained failure data to find out the data tendencies. However, each project presents a different reliability growth behavior and even though several models have been proposed, none was able to show a good performance in projects with different characteristics. Studies on artificial neural networks have shown its efficiency when applied to software reliability modeling. Artificial neural networks are computational structures whose behavior is adapted from examples until the best solution for a specific problem is found. It is possible to understand artificial neural networks as mathematical functions that adapt theirselves to the observed data to obtain a desired model.

The objective of this work is to explore the use of artificial neural networks for software reliability modeling. Two experiments were conducted. The first experiment using models based on time that were applied to various different projects. The results of this experiment allows evaluation of the adaptability and prediction ability of the neural models and, comparison against the analytic models. In the second experiment, test coverage based models were used to verify the applicability of the neural models to this type of failure data.

# 1 Introdução

Alcançar um alto nível de qualidade de software deve ser um dos objetivos das empresas que desejam competitividade em seus produtos. Para que o software tenha qualidade, todas as pessoas envolvidas com o seu desenvolvimento devem estar comprometidas com ela. Sob o ponto de vista do cliente, a noção de qualidade é que o produto desenvolvido esteja conforme a sua especificação. Porém, o conceito de qualidade é muito amplo e vai além do que o cliente pode perceber. Na realidade, o cliente geralmente percebe apenas aspectos que dizem respeito à funcionalidade do software. Outros aspectos como a manutenibilidade, testabilidade, complexidade, modularidade, não são diretamente percebidos pelo usuário final do software mas estão associados à qualidade final do sistema. A engenharia de software busca sistematizar e controlar o desenvolvimento de software para garantir sua qualidade em todos os seus aspectos [41].

Antes da entrega do produto, é necessário realizar tarefas de validação e verificação para certificar a qualidade do software a ser entregue. Durante estas atividades há a necessidade de se validar a qualidade sob o aspecto da capacidade de funcionamento sem erros. A probabilidade de que um sistema funcione sem falhas é chamada de confiabilidade [14]. Esta métrica indica o quanto se pode confiar que o software irá funcionar corretamente e é freqüentemente utilizada como critério para liberação de um programa.

Além disso, é muito importante que ainda durante o desenvolvimento seja realizada uma estimativa da confiabilidade na fase atual e uma projeção da confiabilidade final para que se possa alocar os recursos necessários para que o sistema seja entregue dentro do prazo determinado com a confiabilidade desejada.

A confiabilidade é estimada a partir de modelos de confiabilidade. Com estes modelos é possível estimar a confiabilidade atual do sistema, ou também pode-se fazer uma estimativa futura da confiabilidade, também chamada de predição da confiabilidade. Estes modelos utilizam dados de falha coletados durante os testes para realizar as estimativas de confiabilidade. Existem diversos tipos de modelos [21], cada qual baseado em um tipo de informação e/ou características dos testes.

Para qualquer sistema, de modo geral, a confiabilidade é definida como a probabilidade de um sistema funcionar sem falhas em um determinado ambiente por um período limitado de tempo. A confiabilidade de um sistema é determinada pela confiabilidade dos elementos que o compõem [37]. Uma maneira direta e mais utilizada para se determinar a confiabilidade de um sistema é colocando-o em funcionamento na fase de testes por um tempo limitado e observar

quantas falhas o sistema apresentou neste período. Após um tratamento estatístico nestes dados determina-se a confiabilidade. Os testes devem ser realizados com as entradas selecionadas aleatoriamente, de acordo com um perfil operacional [21], ou seja, da mesma maneira como o sistema será operado em sua utilização final. A confiabilidade de software é definida da mesma maneira que a confiabilidade de sistemas.

Até o final da década de 60 a maior preocupação no desenvolvimento de sistemas era relativa ao desempenho do hardware. Isso porque, naquele tempo, o hardware era mais caro que o software e menos confiável que o hardware atualmente. Os programas eram bem simples, com poucas linhas de código e baixa complexidade. Depois do início da década de 70, os custos relacionados ao software começaram a aumentar enquanto o hardware começou a se tornar mais barato e confiável. Por esta razão, o interesse em determinação da confiabilidade de software cresceu muito desde então [16].

Sistemas de software, diferentemente dos sistemas de hardware, não se desgastam com o tempo. Os únicos fatores que determinam a saída de um programa são os dados de entrada. Os fatores externos, como umidade e temperatura, não influenciam diretamente o resultado de um programa como ocorre com sistemas de hardware. Mesmo que estes fatores façam parte dos dados de entrada, a confiabilidade do software não deve mudar, pois a grandeza é primeiramente transformada em um sinal elétrico, depois transformada em um sinal digital, e só então é utilizada pelo programa como dado de entrada.

Uma outra característica importante é que se por um lado os componentes de hardware degradam com o tempo fazendo com que a confiabilidade do sistema diminua, por outro lado a confiabilidade do software aumenta à medida que vão surgindo as falhas e vão sendo corrigidos os defeitos associados a ela. É importante notar que nem sempre isso ocorre. A confiabilidade do software só aumenta se a correção do defeito não causar nenhum efeito colateral sobre o resto do código, ou seja, se durante a correção não for inserido nenhum outro defeito. Por este motivo, os modelos de confiabilidade de software são conhecidos como modelos de crescimento de confiabilidade de software (SRGM, Software Reliability Growth Model).

Por estas razões, os mesmos modelos utilizados para estimar a confiabilidade de hardware não são adequados para estimar a confiabilidade de software.

A engenharia de confiabilidade de software é o estudo quantitativo da adequação do comportamento operacional do software aos requisitos do sistema. Isto significa assegurar que o desempenho do sistema esteja dentro do esperado com um determinado grau de segurança.

A engenharia de confiabilidade é uma disciplina bastante ampla que cobre não apenas a determinação da confiabilidade de um sistema, mas também como e onde os defeitos são introduzidos no sistema durante o desenvolvimento e como evitar que isso aconteça. A engenharia de confiabilidade envolve o estudo de maneiras de reduzir o número de defeitos introduzidos em cada estágio do desenvolvimento e minimizar a propagação do número de falhas de uma fase do desenvolvimento para a outra [27].

O desenvolvimento de um sistema confiável envolve uma série de atividades, entre elas: especificação da confiabilidade do sistema, aplicação de metodologias que garantam a confia-

bilidade durante o desenvolvimento, testes para determinação da confiabilidade, modelagem de estimativa de crescimento de confiabilidade e estimativa da confiabilidade em si.

A confiabilidade de um sistema é uma grandeza que não se pode medir diretamente. Por ser uma característica dinâmica do sistema, é necessário que se observe o seu funcionamento, e, por estas observações, estimar o valor da confiabilidade. A atividade de observar o comportamento dinâmico do sistema é uma atividade de testes em si, porém com uma finalidade diferente da dos testes convencionais, que tem por finalidade encontrar possíveis defeitos.

Considera-se que a ocorrência de uma falha é um evento isolado e não fornece informações sobre outras falhas que possam ocorrer. Em outras palavras, a simples ocorrência de uma falha não é o suficiente para se determinar a confiabilidade do software. Por esta razão, técnicas de regressão estatística e modelos de confibilidade são utilizados em dados obtidos a partir de falhas observadas durante os testes ou durante a operação em campo para medir a confiabilidade do software.

Modelos de confiabilidade são utilizados para se determinar o comportamento da confiabilidade à medida que os defeitos são removidos. O software é testado utilizando uma abordagem estatística e a confiabilidade é medida com base nas falhas observadas. Os defeitos descobertos são reparados e o software é testado novamente até que um certo número de medidas de confiablidade são realizadas para poderem ser comparadas com o modelo de crescimento e então estimativas futuras de confiabilidade poderão ser realizadas.

Existem diferentes tipos de modelos de confiabilidade de software. As principais diferenças entre eles são as suposições que cada modelo faz para a sua formulação. Apesar do grande número de modelos existentes, a maioria é apenas uma variação dos modelos mais conhecidos [37]. A escolha de um modelo em particular é muito importante na estimativa do crescimento de confiabilidade de software porque tanto a data de entrega do sistema quanto as decisões para alocação de recursos dependem da precisão da estimativa [1].

A confiabilidade de um software depende da maneira como este será utilizado, ou seja, depende do seu perfil operacional. Os modelos que relacionam a confiabilidade em função do tempo realizam testes baseados no perfil operacional do sistema, e assim estão sujeitos a possíveis erros no levantamento do perfil. Em alguns casos, os processos do software são ativados por eventos físicos cuja freqüência de ocorrência durante a operação do sistema não pode ser determinada [38], por se tratarem de eventos aleatórios. Nestes casos o perfil operacional não pode ser determinado com exatidão e, conseqüentemente, a determinação da confiabilidade é prejudicada.

Os modelos de confiabilidade baseados em cobertura utilizam dados obtidos por meio de técnicas de teste onde os casos de teste são escolhidos para satisfazerem algum ou alguns critérios. Cada critério define um tipo de elemento do software e exige que se obtenha a maior cobertura possível para este elemento, ou seja, todos os elementos definidos pelo critério devem ser exercitados pelo menos uma vez. Os modelos baseados em cobertura de teste ([4], [24], [3], [22], [15], entre outros) assumem que a cobertura de elementos está associada à cobertura de defeitos. Assim, a determinação da confiabilidade não depende mais do tempo, apenas da eficiência do critério de teste escolhido.

Os modelos de confiabilidade são determinados realizando-se ajustes de uma curva determinada aos dados observados. Em sua formulação analítica, existem parâmetros que devem ser determinados para se realizar este ajuste. Estes parâmetros são determinados pela análise estatística de dados de falha e representam características da confiabilidade do modelo, como a taxa de falhas, o tempo médio para ocorrer uma falha e a intensidade de falhas. O problema é que cada modelo assume que cada uma destas características apresenta um comportamento específico ao longo do tempo que nem sempre é uma hipótese válida. A influência destes parâmetros e as peculiaridades de cada modelo podem ser eliminadas se tivermos um modelo que se adapte a cada problema a partir dos dados do comportamento de falhas durante as fases iniciais de teste [19]. Uma das maneiras de se realizar isto é utilizando técnicas de aprendizado de máquina, tais como redes neurais artificiais, para a determinação do modelo de confiabilidade.

A maior parte dos modelos de confiabilidade existentes são modelos analíticos ([33], [12], [13], [32], entre outros), isto é, assumem que o crescimento de confiabilidade possui um comportamento definido por uma função (normalmente logarítmica ou exponencial) e o problema de modelamento resume–se a determinar os parâmetros desta função. Estes modelos têm algumas desvantagens, pois eles assumem uma série de suposições para que sua formulação matemática seja possível, e estas suposições podem não ser válidas para todos os casos. Determinar os parâmetros do modelo pode ser uma tarefa custosa, uma vez que não é tão simples encontrar uma curva que se ajuste aos dados obtidos, sem que seja feito uma análise estatística rigorosa.

As redes neurais são estruturas computacionais cujos parâmetros são automaticamente ajustados até que a solução ao problema proposto seja encontrado. Podem ser utilizadas para "aprender" funções ou encontrar padrões em um conjunto de dados. E é exatamente nisso que consiste a modelagem da confiabilidade. Encontrar um padrão nos dados observados no início dos testes para que se possa prever o comportamento da confiabilidade do sistema à medida que os defeitos são removidos. Os modelos de confiabilidade que utilizam redes neurais são referidos como modelos neurais.

Um modelo neural é proposto em [19]. Neste trabalho os autores propõem um modelo que utiliza redes neurais artificiais para a determinação da confiabilidade de um software. A aplicação das redes neurais é explorada utilizando vários tipos de redes – *feed forward*, recorrentes (redes de Elman) [8] e semi-recorrentes (redes de Jordan), regimes de treinamento e métodos de representação dos dados. Os modelos neurais ainda são capazes de desenvolver modelos de complexidades diferentes.

O mesmo estudo é realizado por Sitte em [40]. Neste estudo, a autora utilizou duas das bases disponíveis em [34] e compara os resultados obtidos com modelos neurais e modelos de recalibração paramétrica[1]. A autora conclui que os modelos neurais não são apenas mais simples de utilizar do que os métodos de recalibração paramétrica, mas ainda são iguais ou melhores para a estimativa de confiabilidade. Este trabalho complementa o trabalho de Malaiya et al. em [19].

[1]Este modelo consistem em recalcular os parâmetros do modelo ao longo do processo e não são abordados em nossos experimentos.

Sultan et. al utilizam redes neurais em seu trabalho [1] com uma arquitetura um pouco diferente das arquiteturas propostas em modelos anteriores. Os autores utilizaram uma rede neural *feed forward* com uma camada de entrada, uma camada intermediária e uma camada de saída. O número de neurônios na camada de entrada representa o número de atrasos aplicados à entrada. Neste trabalho os autores utilizaram 4 atrasos, representando o número de falhas encontradas nos dias anteriores. Este trabalho difere um pouco dos dois anteriores pois utiliza dados de falha acumulados durante um determinado tempo (normalmente durante um dia).

## 1.1 Motivação

Os principais pontos que constituem a motivação deste trabalho são os seguintes:

- A importância da confiabilidade na qualidade final de um software;
- A utilização de redes neurais é uma técnica de aprendizado de máquina capaz de realizar a regressão ou o ajuste de praticamente qualquer função, sem que seja necessário o conhecimento do comportamento da curva;
- Em trabalhos anteriores [40] [1] [19], as redes neurais já provaram ter uma ótima capacidade de estimação e predição. Porém, apesar de utilizarem diversas bases de dados de falha e analisarem os resultados para cada base individualmente, estes trabalhos não apresentam nenhum resultado com relação a flexibilidade do uso dos modelos neurais em várias bases diferentes.
- Existem trabalhos na literatura que exploram a utilização das redes neurais artificiais para obter modelos baseados em tempo, mas até o momento da escolha do tema, não havia publicações a respeito da utilização destas técnicas para se obter modelos baseados em cobertura de teste.

## 1.2 Objetivos do Trabalho

O objetivo deste trabalho é desenvolver modelos de crescimento de confiabilidade de software baseados em tempo e baseados em cobertura de teste utilizando redes neurais e comparar os resultados com modelos analíticos conhecidos, tais como o modelo de Jelinski–Moranda [32], o modelo Geométrico [31], baseados em tempo e o modelo binomial (MBBC) [4], baseado em cobertura.

Este trabalho é dividido em duas etapas. Na primeira etapa foram realizados experimentos com modelos baseados em tempo aplicados a vários projetos diferentes (16 no total) cujo objetivo é comparar a adaptabilidade e a capacidade de predição dos modelos neurais comparados aos modelos de Jelinski–Moranda e o modelo Geométrico. Como este experimento considera vários tipos de projetos diferentes, resultados com referência à flexibilidade do uso dos modelos neurais puderam ser obtidos.

Na segunda etapa, foram realizados experimentos com modelos neurais baseados em cobertura de teste, cujo objetivo é verificar se os modelos neurais são adequados para estes tipos de dados através do teste de Kolmogorov–Smirnov, e comparar os resultados com o modelo binomial baseado em cobertura.

## 1.3 Organização do Trabalho

Neste capítulo foi apresentado o contexto associado ao trabalho proposto, a motivação e os objetivos para a sua realização.

O Capítulo 2 apresenta os princípios de redes neurais artificiais que servem de embasamento para os experimentos realizados neste trabalho.

No Capítulo 3 são introduzidos conceitos relacionados à confiabilidade, como a qualidade de software e testes de software. Também são apresentadas as métricas utilizadas para a confiabilidade de software e a terminologia utilizada nesta área de estudo.

O Capítulo 4 aborda a modelagem da confiabilidade de software. Os principais modelos de confiabilidade serão descritos, bem como trabalhos relacionados existentes que também exploram redes neurais para avaliar a confiabilidade de software.

O Capítulo 5 contém a descrição dos experimentos realizados na utilização de redes neurais para modelar o crescimento de confiabilidade de software. Na primeira parte é descrito um modelo baseado no domínio do tempo e na segunda parte é descrito um modelo baseado em cobertura de dados.

No Capítulo 6 estão as conclusões, contribuições e trabalhos futuros.

# 2 *Redes Neurais Artificiais*

Neste capítulo será introduzida a teoria sobre redes neurais artificiais. Estas redes são utilizadas como uma técnica de aprendizado de máquina. O aprendizado de máquina é uma área de inteligência artificial cujo objetivo é o desenvolvimento de modelos computacionais de aprendizado bem como a construção de sistemas capazes de adquirir conhecimento de forma automática. Um sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas por meio da solução bem–sucedida de problemas anteriores. Os diversos sistemas de aprendizado de máquina possuem características particulares e comuns que possibilitam sua classificação quanto à linguagem de descrição, modo, paradigma e forma de aprendizado utilizado [30].

## 2.1 Estrutura e Funcionamento

As redes neurais artificiais são modelos computacionais baseados na estrutura e no funcionamento do sistema nervoso humano e têm se mostrado eficientes na resolução de problemas de difícil solução para a computação convencional. Ao contrário dos modelos computacionais comumente utilizados, em que um programa precisa ser escrito para resolver um dado problema, as redes neurais aprendem com o meio externo, de maneira semelhante à que ocorre no aprendizado dos seres humanos.

Esta técnica fundamenta-se nos estudos realizados sobre a estrutura do cérebro humano para tentar simular a sua forma de aprendizado. A rede neural consiste em uma estrutura de processamento de informação distribuída e paralela. Ela é formada por unidades de processamento, conhecidas como neurônios. Os neurônios possuem memória local e podem realizar operações de processamento de informações localizadas. Cada nó possui uma única saída, que pode ser ligada a outros neurônios. Todo processamento que se realiza em cada unidade deve ser completamente local, isto é, deve depender apenas dos valores correntes dos sinais de entrada que chegam dos neurônios através das conexões. Estes valores atuam sobre os valores armazenados na memória local do neurônio.

A Figura 1 ilustra a estrutura básica de um neurônio inicialmente proposto por Mark Rosenblatt [28]. Esta estrutura também é conhecida como *perceptron*. As entradas do *perceptron* estão representadas por $x_1, x_2, ..., x_n$ e são multiplicadas por um fator conhecido como *peso* $w_i$ de cada entrada. O parâmetro $x_0$ é igual a 1 e é somente representado no modelo para simplificar a notação [29].

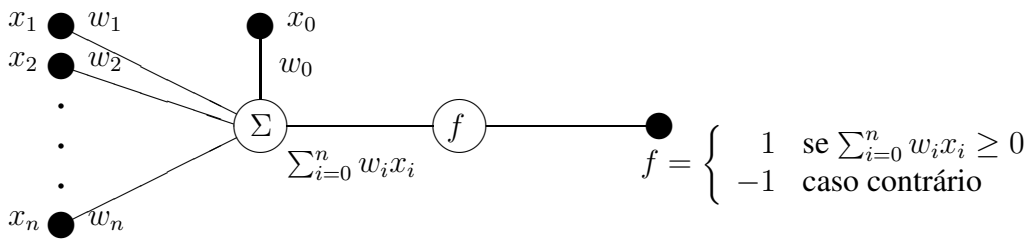



Figura 1: Estrutura básica de um neurônio

Um *perceptron* raramente é utilizado isoladamente. Normalmente eles são interconectados em estruturas conhecidas como *perceptrons multicamadas*. Em princípio, o *perceptron multicamadas* consiste de uma camada de neurônios de entrada, uma ou mais camadas de neurônios ocultas e uma camada de neurônios de saída [28]. A Figura 2 ilustra uma rede neural com dois neurônios na camada de entrada, dois na camada intermediária e um neurônio na camada de saída.

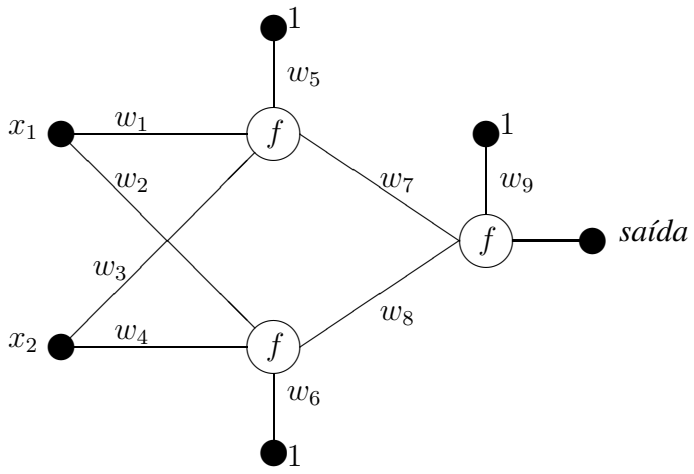



Figura 2: *Perceptron* multicamadas

A função inicialmente proposta para o modelo do *perceptron* não é adequada para os algoritmos de aprendizado mais utilizados, como por exemplo o algoritmo de retropropagação, pois estes algoritmos necessitam de funções diferenciáveis em todos os pontos [29]. Ao invés desta função, utiliza-se a função *sigmóide*, definida como:

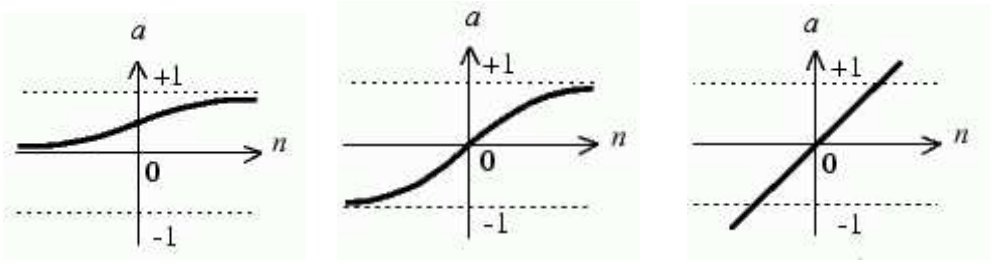
$$f(x) = \frac{1}{1 + e^{-y}} \tag{2.1}$$

Analisando a função, percebe-se que $\lim_{x \to -\infty} f(x)$ é igual a $0$ e que $\lim_{x \to +\infty} f(x)$ é igual a $1$. O comportamento desta função difere do comportamento da função degrau do *perceptron* em dois pontos:

1. a função degrau inicialmente proposta para o *perceptron* mapeia os valores de saída em $-1$ e $1$ somente e com uma transição em $x = 0$.

2. a função sigmóide mapeia os valores de saída entre $0$ e $1$, apresentando uma transição suave perto de $x = 0$, permitindo que os valores de saída possam assumir qualquer um dos valores entre $0$ e $1$.

Note que a definição do intervalo de saída não faz diferença na resolução do problema, uma vez que os dados terão que ser normalizados de maneira a estarem contidos neste intervalo. A função sigmóide, além de facilitar a convergência dos algoritmos, é também mais tolerante a ruído, sendo assim, mais apropriada para o uso de aprendizado de máquina. Existem outras funções também utilizadas como funções de transferência de neurônios. A Figura 3 apresenta algumas funções de transferência que podem ser utilizadas pelos neurônios [8]. A Figura 3.a é uma função sigmóide conhecida como função Log–Sigmóide, que tem a sua imagem limitada entre $0$ e $1$. A Figura 3.b é uma função sigmóide conhecida como tan–sigmóide, e tem o seu conjunto imagem limitado entre $-1$ e $1$. A Figura 3.c é uma função linear, e tem o seu conjunto imagem limitado entre $-\infty$ e $+\infty$.




(a) (b) (c)

Figura 3: Funções de tranferência.

A propriedade mais importante das redes neurais é a habilidade de adaptar sua estrutura para resolver um determinado problema. Isso é feito através de um processo iterativo de ajustes aplicados a seus pesos. Este processo também é conhecido como treinamento da rede neural. No treinamento da rede neural, são utilizadas técnicas que definem de que maneira os pesos entre as conexões serão ajustados de acordo com o desempenho atual da rede. Ou seja, as redes neurais passam por um regime de treinamento para aprenderem através de exemplos a resolver um dado problema.

O algoritmo de retropropagação (*Backpropagation*) [29] tem sido muito utilizado como algoritmo de aprendizado supervisionado. O algoritmo consiste em apresentar um exemplo à rede e calcular o erro entre a saída da rede e a saída esperada. Calcula-se então o gradiente deste erro em relação aos pesos das camadas de saída. Esta camada é então ajustada de forma a minimizar o erro. O mesmo processo é repetido para as camadas anteriores, propagando o erro para trás, originando daí o nome desta técnica de aprendizado. A cada novo exemplo o processo se repete. O processo de redução gradativa do erro, que acompanha a minimização chama–se convergência. À medida que a rede aprende, o valor do erro converge para um valor estável, normalmente irredutível.

## 2.2 Topologias de Redes Neurais Artificiais

Os neurônios de uma rede neural artificial podem se conectar de diversas maneiras e a maneira como é realizada esta interconexão define a topologia da rede. Cada topologia confere uma característica diferente à rede neural artificial. Redes com uma camada única de nodos, por exemplo, só conseguem resolver problemas linearmente separáveis. Redes recorrentes, por sua vez, são mais apropriadas para resolver problemas que envolvem processamento temporal. Os seguintes parâmetros definem a arquitetura de uma rede:

- Número de camadas da rede.
- Número de neurônios em cada camada.
- Tipo de conexão entre os neurônios.

As redes podem ser classificadas quanto ao número de camadas:

- Redes de camada única: são redes onde só existe um neurônio entre qualquer entrada e qualquer saída da rede (Figuras 4.a e 4.e).
- Redes de múltiplas camadas: são redes onde existe mais de um neurônio entre alguma entrada e alguma saída da rede (4.b, 4.c e 4.d).

As redes neurais podem também ser classificadas de acordo com o tipo de conexões:

- redes *feed–forward*: onde a saída de um neurônio em qualquer camada da rede nunca é utilizada como entrada de neurônios de camadas anteriores (4.a, 4.b e 4.c).
- redes *feedback*: onde a saída de algum neurônio de uma camada da rede é utilizada como entrada de um neurônio de alguma camada anterior (4.d e 4.e).

As redes neurais ainda podem ser classificadas conforme a sua conectividade:

- rede fracamente conectada: quando nem todos os neurônios de uma camada estão conectados aos neurônios da próxima camada (4.b e 4.d).
- rede completamente conectada: quando todos os neurônios de uma camada estão conectados aos neurônios da próxima camada (4.a, 4.c e 4.e).

A Figura 4 apresenta alguns exemplos de arquiteturas diferentes de redes.

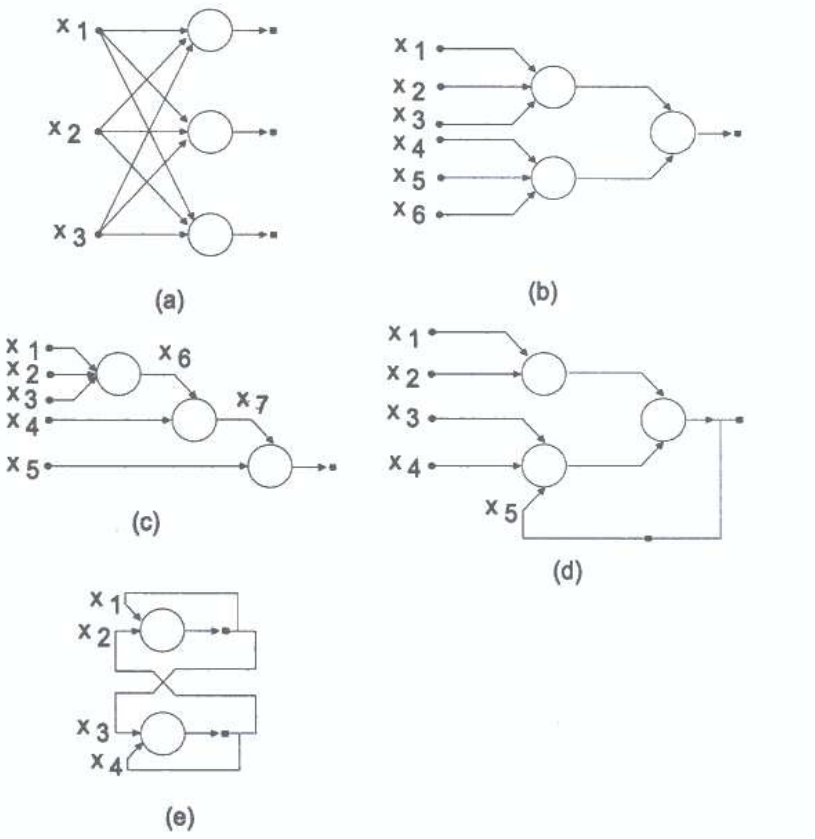



Figura 4: Arquiteturas de redes neurais.

## 2.3 Considerações Finais

Uma característica importante das redes neurais é que elas são capazes de lidar com imperfeições nos dados de entrada, o que as tornam bastante apropriadas para aplicações em engenharia de software, tais como a estimativa de confiabilidade, onde frequentemente os dados possuem uma certa quantidade de subjetividade.

Por esta razão, as redes neurais artificiais foram utilizadas neste trabalho como uma técnica de aprendizado de máquina para encontrar uma função que descreva o comportamento dos dados de falha observados e, assim, determinar um modelo de confiabilidade para estes dados.

# 3 *Fundamentação Básica*

Neste capítulo serão abordados conceitos relacionados à confiabilidade tais como a qualidade de software, técnicas e critérios de teste, as métricas mais conhecidas de confiabilidade de software e a terminologia utilizada ao longo deste trabalho.

## 3.1 Garantia da Qualidade

A qualidade é um conceito muito amplo e ao mesmo tempo subjetivo. Pressman [39] define qualidade de software como sendo a conformidade aos requisitos de funcionalidade e desempenho, metodologias de desenvolvimento documentadas e características implícitas que são esperadas de um programa de computador desenvolvido profissionalmente.

A qualidade de um software começa antes mesmo do seu desenvolvimento. Ainda na fase de definição do sistema, os requisitos devem ser claramente especificados de acordo com a qualidade final desejada. As metodologias de desenvolvimento servem para definir os procedimentos adequados para se atingir os requisitos de funcionalidade e desempenho com a qualidade desejada.

As características implícitas são características que nem sempre são definidas como um requisito do sistema, tais como complexidade, testabilidade, reusabilidade, manutenibilidade e confiabilidade. Devido a complexidade dos sistemas e limitações de projeto é muito importante que estas características sejam claramente especificadas, principalmente para que sejam alocados adequadamente os recursos necessários para a realização do projeto.

## 3.2 Terminologia

A **confiabilidade de software** é definida como a probabilidade de operação livre de falhas de um software em um ambiente determinado por um dado período de tempo [37]. Uma **falha** é o desvio do comportamento do software do comportamento esperado. Este termo deve ser distingüido da definição de **defeito** no código, o qual pode causar uma falha assim que é ativado durante a execução de um código defeituoso no programa. A ativação de um defeito é caracterizada pela ocorrência de uma falha.

Uma vez que a confiabilidade de software diz respeito a falhas e defeitos, é importante que

estes termos sejam bem definidos para que o conceito de confiabilidade possa ser corretamente entendido. O conceito de falha e defeito é definido pela IEEE [27] da seguinte maneira:

**Falha (failure):** Comportamento não esperado do sistema diferente do comportamento especificado nos requisitos do sistema.

**Defeito (fault):** Característica do programa, ou um problema textual no código fonte que pode produzir um comportamento não esperado do sistema (falha).

Uma vez que o software não se desgasta da mesma maneira que o hardware, a confiabilidade do software permanece constante se nenhuma alteração for feita no código ou nas condições de operação do software. Entretanto, se a cada falha ocorrida o defeito associado for encontrado e corrigido, então a confiabilidade do software irá aumentar. Esta é a situação típica encontrada durante as atividades de testes. À medida que cada ocorrência de falha implica na remoção do defeito associado, o número de falhas observadas durante um tempo $t$, representado por $M(t)$, pode ser considerado como uma função de crescimento de confiabilidade. Modelos que procuram descrever o comportamento de $M(t)$ são conhecidos como Modelos de Crescimento de Confiabilidade (Software Reliablity Growth Models, SGRM), ou simplesmente Modelos de Confiabilidade. Utilizando o tempo de falhas $t_1, t_2, ..., t_n$ ou o tempo entre as falhas $\Delta t_1, \Delta t_2, ..., \Delta t_n$ observados durante os testes, os parâmetros dos modelos podem ser estimados. O **valor médio** de $M(t)$ é uma função $\mu(t)$ que representa o número esperado de falhas observadas até o tempo $t$. A derivada da função valor médio em termos da variável tempo, $d\mu(t)/dt$, é chamada de **intensidade de falhas** $\lambda(t)$. Esta variável representa o limite do número de falhas esperadas no intervalo de tempo $[t, t + \Delta t)$ quando $\Delta t$ se aproxima de zero.

A intensidade de falhas não deve ser confundida com a **taxa de falhas** do sistema, $z(\Delta t|t_{i-1})$, que é a densidade da probabilidade para a falha $i$ ser observada em $t_{i-1} + \Delta t$ dado que a falha $(i-1)$ ocorreu em $t_{i-1}$. Em alguns tipos de modelos, a taxa de falhas $z(\Delta t|t_{i-1})$ é de fato igual à intensidade de falhas no tempo $t_{i+1} + \Delta t$ [14].

Muitos modelos de confiabilidade especificam, além da taxa de falhas do sistema, a taxa de falhas para um defeito individualmente. Se cada defeito tem a mesma probabilidade de ser ativado, então o tempo até que ocorra uma falha tem a mesma densidade de probabilidade $f_a(1)$. A taxa de falhas para cada defeito é dada, então, por

$$z_a(t) = \frac{f_a(t)}{1 - \int_0^t f_a(x)d(x)} \tag{3.1}$$

Assim, a taxa de falhas por defeito é uma medida da probabilidade de um defeito em particular causar uma falha, dado que ele não foi ativado ainda [14].

## 3.3 Atividades de Teste

Teste de software é um elemento crucial na garantia da qualidade do software e representa a revisão final da especificação, do projeto e do código gerado [39]. A atividade de teste de

software envolve: planejamento dos testes, projeto de casos de teste (ou seja, a especificação dos dados de entrada e da saída esperada), execução e avaliação dos resultados de teste [26]. Ela é uma atividade de verificação e validação, ou seja, é uma atividade que consiste em verificar se o sistema está sendo desenvolvido corretamente, através das metodologias de desenvolvimento, e validar o produto desenvolvido segundo as especificações de seus requisitos. A atividade de teste fornece indicativos para a determinação da qualidade do software.

Sommerville [41] classifica as atividades de teste em duas abordagens diferentes:

- *Testes Estatísticos*
  Testes estatísticos são geralmente utilizados para testar o desempenho e a confiabilidade do sistema. Estes testes são normalmente projetados baseados na maneira como o sistema será utilizado pelo usuário final. Os dados de entrada são escolhidos estatisticamente com a mesma freqüência que eles aparecerão na operação do sistema em campo, segundo o perfil operacional do sistema. Após a execução dos testes é possível fazer uma estimativa da confiabilidade do sistema. O desempenho do programa pode ser determinado medindo–se o tempo de execução dos testes.

- *Testes Orientados a Defeitos*
  Testes orientados a defeitos são testes que têm como propósito descobrir regiões do programa que não estão de acordo com a especificação do sistema. Os testes são projetados para revelar a presença de defeitos no sistema. Os dados de entrada são escolhidos de acordo com a sua capacidade de causar falhas durante a execução do programa. Esta abordagem de testes utiliza técnicas específicas para buscar o maior número de defeitos possível. Estas técnicas serão vistas na seção seguinte.

### 3.3.1 Técnicas de Teste de Software

Os testes de software devem ser projetados segundo alguma técnica baseada em um critério para que se possa ter certeza que o software foi suficientemente testado. Existem basicamente três técnicas de teste de software: funcional, estrutural e baseada em erros, sendo que esta última pode ser considerada como técnica estrutual. A técnica funcional e a estutural são técnicas complementares entre si pois são utilizadas para detectar defeitos em fases diferentes do desenvolvimento.

Na técnica funcional, também conhecida como *caixa preta*, os testes são projetados de acordo com a especificação de uso do sistema. Os casos de teste devem ser escolhidos de maneira a executar todas as funcionalidades do sistema, inclusive o tratamento de entradas incorretas. Um exemplo de uma técnica funcional é o particionamento do domínio de entrada em classes de equivalência, onde os dados de entrada são classificados (inclusive os valores inválidos) e são escolhidos alguns representantes de cada classe para testar o software [39].

Na técnica estrutural, também conhecida como *caixa branca*, os testes são projetados baseados na estrutura interna do programa. Os dados de teste são escolhidos de maneira a executar os componentes da estrutura do programa. Esta estrutura é representada por um grafo, conhecido

por grafo de fluxo de controle, conforme mostra a Figura 5. Nesta figura, podemos identificar alguns elementos utilizados em critérios de teste. No nó 1 temos a definição de uma variável $x$. No arco que une o nó 2 ao 3 e no arco que une o nó 2 ao 4 temos um uso desta variável $x$ definida no nó 1 para um controle de fluxo no programa. Este uso é definido como um uso *predicativo*, também conhecido como *p–uso*, pois o conteúdo da memória não é modificado. No nó 3 temos a definição da variável $y$ e o uso da variável $x$. Desta vez, o uso da variável $x$ é um uso computacional, também conhecido como *c–uso*, pois ele altera o conteúdo da memória mapeado pela variável $y$.

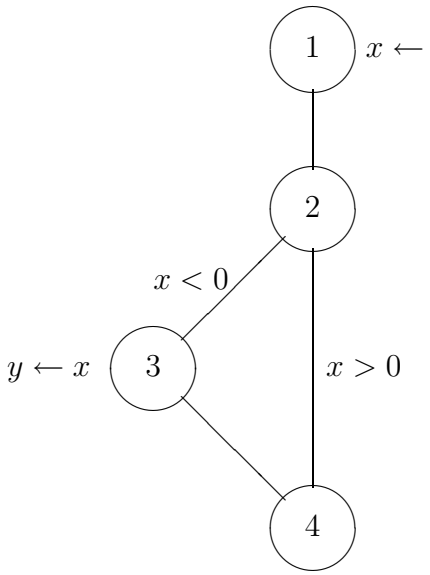



Figura 5: Grafo de fluxo de controle de um programa.

Existem diversos critérios que definem quais são os componentes que devem ser testados, que são chamados de elementos do critério. A medida que avalia quantos elementos requeridos por um critério foram testados chama–se cobertura de elementos. Um exemplo de critério estrutural são os critérios baseados em fluxo de controle. Estes critérios associam os programas a grafos que representam os comandos de desvio de fluxo de controle. O critério todos os nós é um dos critérios baseados em fluxo de controle e exige que cada nó do grafo seja exercitado pelo menos uma vez. Há também os critérios baseados em fluxo de dados, que exercitam as definições e usos das variáveis do programa [25].

Neste trabalho foram utilizados dados de testes estruturais segundo os seguintes critérios baseados em fluxo de controle:

- Todos Nós: requer a execução de todos os nós executáveis do grafo de fluxo de controle, isto é, de todos os comandos do programa.
- Todos Arcos: requer a execução de todos os arcos executáveis, ou seja, os dados de teste devem praticar todas as transições de nós possíveis.

Também foram utilizados dados de teste estrutural segundo os seguintes critérios baseados em fluxo de dados [25]:

- Todos Potenciais Usos (PU): requer o exercício de todas as associações definição potencial–uso, onde potencial–uso representa a possibilidade de uso predicativo ou computacional de uma variável.
- Todos Potenciais Usos/DU (PUDU): requer que todas as associações definição potencial–uso de uma variável sejam exercidas pela execução de potenciais du–caminhos. Um potencial du–caminho é uma seqüência de nós livre de laços entre a definição e um potencial uso de uma variável.
- Todos Potenciais DU Caminhos (PDU): requer a execução de todos potenciais du–caminhos do programa.

Os critérios PU, PDU e PUDU são critérios baseados no conceito de Potencial Uso. Os critérios Potenciais Usos requerem associações definição–uso independentemente da ocorrência explicíta de um uso de uma determinada variável. Ou seja, se um uso de uma definição pode existir, a potencial associação é requerida.

A técnica baseada em defeitos parte do princípio de que o software desenvolvido, apesar de conter defeitos, está muito próximo do programa correto. Esta técnica baseia-se em enganos típicos cometidos por programadores, normalmente por falta de atenção, como por exemplo a troca de uma operação lógica ou aritmética. O critério Análise de Mutantes [6] é um dos mais conhecidos critérios baseados em defeitos, o qual consiste em gerar versões do programa a ser testado com pequenas alterações, através de operadores de mutação, e avaliar o resultado gerado pela versão original e sua versão modificada (mutante).

Na prática, os critérios estruturais e baseados em defeitos somente são viáveis de serem aplicadas mediante o uso de uma ferramenta computacional. A ferramenta PokeTool [2], por exemplo, foi desenvolvida na Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas (UNICAMP) e é uma ferramenta de testes baseados em critérios estruturais. A PokeTool pode avaliar os seguintes critérios: PU, PUDU, PDU, Todos-Nós e Todos-Arcos [26] mencionados anteriomente. Ao executar um caso de teste, a ferramenta verifica quais foram os elementos cobertos pelo conjunto de casos de teste já executados para cada critério.

A ferramenta PROTEUM [7] foi desenvolvida no Instituto de Ciências Matemáticas de São Carlos (ICMC–USP) e permite ao usuário avaliar a adequação ou gerar um conjunto de casos de teste para um programa, segundo o critério análise de mutantes. A PROTEUM suporta 71 operadores de mutação, classificados em: mutação de comandos, mutação de operadores, mutação de variáveis e mutação de constantes. A ferramenta gera os mutantes, executa os casos de teste e compara os resultados dos programas mutantes com o programa original.

## 3.4 Métricas de Software

Durante o desenvolvimento de software é necessário que se estabeleçam métricas para avaliar a qualidade alcançada. Uma métrica é uma medida quantitativa de algum atributo do soft-

ware.

As métricas podem ser divididas em duas classes: métricas de controle e métricas de planejamento [41]. As métricas de controle são aquelas utilizadas para gerenciamento e controle do processo de desenvolvimento do software. Exemplos deste tipo de métrica são: esforço despendido e tempo transcorrido. Métricas de planejamento são medidas de um atributo do software que podem ser usadas para estimar algum fator de qualidade.

As métricas de confiabilidade de software derivaram-se das métricas de confiabilidade de hardware. Entretanto, as métricas de confiabilidade de hardware nem sempre são aplicáveis em projetos de software devido à natureza das falhas em ambos os casos. As falhas de hardware ocorrem devido à falha mecânica de algum dispositivo normalmente ocorrida pela sua degradação devido ao tempo de uso. O sistema permanecerá fora de operação até que o dispositivo seja reparado ou substituído. Por outro lado, as falhas de software ocorrem apenas para algumas entradas; para outras o sistema pode continuar operando normalmente.

Uma métrica bastante utilizada em sistemas de software e que derivou–se de sistemas de hardware é o tempo médio entre falhas (MTBF – Mean Time Between Failures). Esta métrica é composta por duas outras métricas: tempo médio de ocorrência de falha (MTTF – Mean Time To Failure) e tempo médio para o reparo (MTTR – Mean Time To Repair). O MTBF pode ser expresso matematicamente como [41]:

$$MTBF = MTTF + MTTR \tag{3.2}$$

Existem outras métricas de confiabilidade como, por exemplo, defeitos por mil linhas de código (defeitos/KLOC) ou defeitos por pontos de função (defeitos/FP). Porém, muitos pesquisadores alegam que MTBF é uma medida mais eficiente do que defeitos/KLOC ou defeitos/FP [39]. Isso se deve ao fato dos defeitos contidos em um programa não apresentarem a mesma probabilidade de serem revelados. Ou seja, que não há uma relação direta entre o tempo entre falhas e a quantidade de defeitos em um programa.

## 3.5 Considerações Finais

Neste capítulo foram introduzidos os principais conceitos e a terminologia utilizada neste trabalho. Foram também apresentadas as principais técnicas de teste e introduzidos os critérios de teste e métricas de confiabilidade utilizadas pelos modelos apresentados no próximo capítulo e utilizados nos experimentos descritos no Capítulo 5.

# 4 *Modelos de Confiabilidade de Software*

Modelos de confiabilidade são utilizados para determinar o comportamento da confiabilidade à medida que os defeitos são removidos. Estes modelos são influenciados por 3 fatores: a natureza dos defeitos, a capacidade de detecção e remoção destes defeitos e a maneira como o programa é executado.

Os defeitos são introduzidos à medida que o programa é codificado ou seu código é modificado. Eles podem ocorrer por um erro do programador ou até mesmo por um erro de projeto. Qualquer que seja a origem do defeito, ele é inserido por razões essencialmente humanas. É praticamente impossível prever o comportamento de inserção de defeitos no código, mesmo porque este comportamento depende do nível de experiência de cada desenvolvedor envolvido no processo de planejamento e codificação de um sistema. Portanto, é bem provável que a confiabilidade de um sistema desenvolvido por uma equipe apresente uma curva de crescimento diferente da curva de um sistema desenvolvido por outra equipe. Se as características de desenvolvimento são diferentes, muitas vezes temos que utilizar modelos diferentes, que levem em consideração as características de cada caso.

O software é testado utilizando uma abordagem estatística e a confiabilidade é medida com base nas falhas observadas. Os defeitos descobertos são detectados e removidos e o software é testado novamente até que um número suficiente de medidas de confiabilidade é realizado para obter um modelo de confiabilidade e então estimativas futuras da confiabilidade poderão ser realizadas. Para que um defeito seja descoberto é necessário que o mesmo seja ativado, resultando em uma falha. Isto demanda um certo tempo, pois nem todas as execuções de um código defeituoso resultam em falha. O perfil operacional do sistema também tem influência direta nos resultados dos testes, pois os defeitos somente serão revelados se forem executados os dados de teste que levam o programa a um desvio do seu comportamento esperado (falha).

A confiabilidade de um software depende da maneira como este será utilizado, ou seja, depende do seu perfil operacional. Os modelos de confiabilidade baseados no domínio do tempo representam o comportamento da confiabilidade ao longo do tempo. Estes modelos presumem a execução de testes baseados no perfil operacional do sistema, e assim estão sujeitos a possíveis erros no levantamento deste perfil. Porém, em alguns casos, os processos do software são ativados por eventos físicos cujas freqüências de ocorrência durante a operação do sistema não podem ser determinadas [38], por se tratarem de eventos aleatórios. Nestes casos o perfil operacional não pode ser determinado com exatidão; conseqüentemente, a exatidão do modelo é

prejudicada.

Os modelos de confiabilidade baseados em cobertura representam a confiabilidade ao longo do progresso das atividades do software. Para isso, utilizam dados obtidos por técnicas de teste onde os casos de teste são escolhidos para satisfazer algum ou alguns critérios de teste. Cada critério define um tipo de elemento do software e procura–se obter a maior cobertura possível para este critério, ou seja, cada elemento definido pelo critério deve ser exercitado pelo menos uma vez. Os modelos baseados em cobertura de teste assumem que a cobertura de elementos está associada à cobertura de defeitos. Assim, a determinação da confiabilidade não depende mais do tempo, apenas da cobertura dos elementos do critério de teste escolhido.

Os modelos de confiabilidade de software podem ser classificados quanto à sua formulação matemática como modelos analíticos ou modelos neurais. Os modelos mais conhecidos são os modelos analíticos, que definem uma função matemática cujos parâmetros têm significado físico (por exemplo, a intensidade de falhas e o tempo entre falhas) e devem ter seus valores determinados de forma que a curva definida pela função se ajuste aos dados observados com o menor erro possível. Já os modelos neurais não possuem uma forma analítica. O modelo é definido por uma estrutura computacional conhecida como rede neural artificial (Capítulo 2). Os modelos neurais apresentam uma grande vantagem em relação aos modelos analíticos porque eles requerem apenas o histórico de falhas observadas e não partem de uma forma analítica pré definida para a curva de confiabilidade.

## 4.1 Modelos Baseados no Domínio do Tempo

Os modelos baseados no domínio do tempo procuram determinar a confiabilidade como uma função do tempo de execução do programa, normalmente utilizando o tempo de ocorrência entre falhas ou o número de falhas ocorridas em um determinado intervalo de tempo. A seguir serão descritos alguns modelos baseados no domínio do tempo que foram utilizados neste trabalho.

### 4.1.1 Modelo de Jelinski–Moranda

O modelo desenvolvido por Jelinski e Moranda [32] em 1972 foi um dos primeros modelos de confiabilidade propostos e ainda é amplamente utilizado e referenciado. O modelo de Jelinski–Moranda assume que o tempo entre falhas segue uma distribuição exponencial cujo parâmetro é proporcional ao número de defeitos restantes no software. O tempo médio entre o momento que ocorreu uma falha $t_{i-1}$ e o momento que ocorrerá a próxima falha $t_i$ é dado por:

$$\Delta t = \frac{1}{\varphi(N - i + 1)} \tag{4.1}$$

onde $N$ é o número de defeitos presentes no programa no início dos testes e o parâmetro $\varphi$ é uma constante de proporcionalidade.

O modelo parte das seguintes hipóteses:

- O número de defeitos $N$ no início dos testes é um valor fixo mas desconhecido.
- Cada defeito tem a mesma probabilidade de ser ativado, causando uma falha, e a taxa de falhas de cada defeito é constante ($z(t) = \varphi$).
- As falhas não estão correlacionadas. Ou seja, dado $N$ e $\varphi$, os tempos entre as falhas $\Delta t_1, \Delta t_2, ..., \Delta t_n$ são variáveis aleatórias independentes e distribuídas exponencialmente.
- A remoção dos defeitos após a ocorrência de uma falha é instantânea e não introduz novos defeitos no software.

Como conseqüência destas suposições, a taxa de falhas do sistema após a remoção do defeito $(i-1)$ é proporcional ao número de defeitos restantes no software. Assim, temos:

$$z(\Delta t|t_{i-1}) = \varphi\left(N - M(t_{i-1})\right) = \varphi\left(N - (i-1)\right) \tag{4.2}$$

O modelo de Jelinski–Moranda pertence a uma classe de modelos de tipo binomial. Para estes modelos, a função de intensidade de falhas é o produto do número de defeitos inicial do sistema e a densidade de probabilidade do tempo de ativação de uma falha $f_a(t)$ [27]. As funções valor médio e intensidade de falhas para este modelo são:

$$\mu(t) = N\left(1 - e^{-\varphi t}\right) \tag{4.3}$$

$$\lambda(t) = N\varphi e^{-\varphi t} = \varphi\left(N - \mu(t)\right) \tag{4.4}$$

A confiabilidade $R(t_i)$ antes da remoção do defeito $i$ é dada por:

$$R(t_i) = e^{-\varphi(N-(i-1))t_i} \tag{4.5}$$

### 4.1.2 Modelo Geométrico

O modelo geométrico [31] assume que o programa contém, inicialmente, um número ilimitado de defeitos e que os mesmos estão divididos em classes de dificuldade, com probabilidades diferentes de ativação. À medida que os defeitos que se encontram nas classes das mais fáceis de serem detectados são corrigidos, a probabilidade de se observar uma falha diminui. A distribuição das probabilidades dos tempos entre falhas é exponencial e sua média decresce exponencialmente.

O modelo parte das seguintes hipóteses:

- Existe um número infinito de defeitos no software.
- Cada defeito tem a mesma chance de ser encontrado dentro de uma mesma classe de dificuldade.

- A taxa de falhas forma uma progressão geométrica e é constante entre a detecção de defeitos, ou seja, $z(t) = D\varphi^{i-1}$.
- As falhas não estão correlacionadas.

As funções valor médio e intensidade de falhas para este modelo são:

$$\mu(t) = \frac{1}{\beta} ln\left[D\beta exp(\beta)t + 1\right] \tag{4.6}$$

$$\lambda(t) = \frac{Dexp(\beta)}{D\beta exp(\beta)t + 1} \tag{4.7}$$

onde: $\beta = -ln(\varphi)$ para $0 < \varphi < 1$.

### 4.1.3 Outros Modelos

Existem vários outros modelos além dos modelos utilizados neste trabalho. Entre eles podemos citar o modelo de Goel–Okumoto, o modelo ENHPP e os modelos Bayesianos, que serão descritos a seguir.

#### Modelo de Goel–Okumoto (G–O)

O modelo proposto por Goel e Okumoto [12] também é chamado de modelo de processo de Poisson não homogêneo (Non–Homogeneous Poisson Process, NHPP) e é baseado nas seguintes suposições:

- O número de falhas no software que ocorrem em $(t, t + \Delta T]$ com $\Delta t \to 0$ é proporcional ao número esperado de defeitos não encontrados, $N - \mu(t)$. A constante de proporcionalidade é $\varphi$.
- Os números de falhas detectadas em cada um dos intervalos $(0, t_1), (0, t_2), ..., (0, t_n)$ são valores independentes entre si.
- A taxa de falhas de cada defeito é uma constante $\varphi$.
- A correção dos defeitos não introduz novos defeitos.
- O número de falhas ocorridas durante um tempo $t$ segue uma distribuição de Poisson com valor médio $\mu(t)$.

Estas suposições resultam na seguinte formulação matemática para a média do número esperado de falhas observadas durante um tempo $t$ e a função de intensidade de falhas:

$$\mu(t) = N\left(1 - e^{-\varphi t}\right) \tag{4.8}$$

$$\lambda(t) = N\varphi e^{-\varphi t} = \varphi\left(N - \mu(t)\right) \tag{4.9}$$

Uma vez que cada defeito é removido sem a introdução de novos defeitos, o número de defeitos presentes no software no início dos testes é igual ao número de falhas que terão ocorrido após um tempo de testes infinito. De acordo com as suposições, M($\infty$) segue uma distribuição de Poisson com uma média igual a $N$. Então, $N$ é o número esperado de defeitos iniciais no software. E essa é uma das principais diferenças deste modelo comparado ao modelo de Jelinski–Moranda, onde $N$ é um valor fixo mas desconhecido.

### Modelo de Musa

Os dois modelos anteriores consideram apenas o tempo transcorrido e não fazem considerações sobre o esforço requerido nas atividades de teste. Com isto, assumem implicitamente que este esforço é constante durante todo o processo. Porém, na prática, isto não é verdadeiro. Faltam neste modelo algumas suposições que considerem este esforço.

O modelo de Musa foi o primeiro a incorporar o esforço requerido nos testes de software exigindo que as medidas de tempo sejam feitas baseadas no tempo de processamento ou o tempo de execução do programa, $\tau$. Por isso, este modelo também é conhecido como modelo de tempo de execução.

A intensidade de falhas do sistema é dada por:

$$\lambda(t) = NB\varphi e^{-B\varphi\tau} \tag{4.10}$$

onde $N$ tem o mesmo significado que no modelo G–O e $B$ é o fator de redução de defeitos, o qual é uma constante que relaciona a taxa de correção de defeitos com a taxa de risco de cada defeito. A equação anterior pode ser reescrita como:

$$\lambda(t) = \varphi\left(N - \mu(t)\right) = Kf\left(N - \mu(t)\right) \tag{4.11}$$

onde $\varphi$ é formulado como o produto da freqüência de execução linear $f$, pela taxa de redução de defeitos $K$, que pode ser interpretada como o número médio de falhas ocorrentes por defeito remanescente no código durante uma execução linear do programa [37].

Esta última equação sugere que o modelo do Musa seja matematicamente equivalente aos modelos de Jelinski–Moranda e Goel–Okumoto. Isto de fato é verdadeiro, sendo que a principal diferença é a formulação da taxa de falhas por defeito $\varphi$ [37].

### O Modelo NHPP Melhorado (ENHPP)

O modelo NHPP melhorado (Enhanced Non-homogeneous Poisson Process, ENHPP) incorpora dados de cobertura de teste em sua formulação analítica como uma medida de esforço de teste [13]. A cobertura de teste é definida como a razão entre o número de elementos execu-

tados pelo conjunto de testes e o número total de elementos requeridos pelo critério de teste.

O modelo faz as seguintes suposições:

- Os defeitos estão uniformemente distribuídos entre os elementos requeridos pelo critério de teste. Isto é, ao ser executado, cada elemento tem a mesma probabilidade de apresentar um defeito.
- A probabilidade de um elemento apresentar um defeito ao ser executado em um dado tempo $t$ é $c_d(t)$.
- Os defeitos são corrigidos sem efeitos colaterais.

A função valor médio dos defeitos encontrados em um dado tempo $t$ é:

$$\mu(t) = c(t)\check{N} \tag{4.12}$$

onde $c(t)$ é uma função da cobertura dos testes e $\check{N}$ é o número esperado de defeitos ao se atingir a cobertura total, diferindo dos outros modelos onde $N$ era o número total de defeitos presentes no início dos testes. A intensidade de falhas para o modelo ENHPP é expressa por:

$$\lambda(t) = z(t)(\check{N} - \mu(t)) \tag{4.13}$$

onde $z(t)$ é a função da taxa de risco por defeito. A confiabilidade é dada por:

$$R(t/s) = e^{-\check{N}K(c(s-t)-c(s))} \tag{4.14}$$

onde $s$ é o tempo em que ocorreu a última falha e $t$ é o tempo transcorrido desde a última falha.

### Modelos Bayesianos

O raciocínio bayesiano é baseado na suposição de que as variáveis de interesse são governadas por uma distribuição de probabilidade e que as decisões corretas podem ser realizadas analisando estas probabilidades junto com os dados observados [29].

Diferentemente dos modelos anteriores, os modelos bayesianos não necessitam de dados de falha para ser formulado. Uma formulação inicial pode ser realizada utilizando dados de outros projetos anteriores ou até pela experiência de um profissional em projetos anteriores [35].

À medida que o software é testado, o modelo é atualizado, mesmo quando não ocorrem falhas, pois o modelo bayesiano considera também o tempo de execução livre de falhas.

## 4.2 Modelos Baseados em Cobertura

A cobertura de teste é uma medida em termos de elementos, requeridos pelos critérios de teste, que foram exercitados. Quando um destes elementos é executado, é possível que um ou mais defeitos associados sejam encontrados. A cobertura de defeitos em software pode ser definida de maneira análoga; ela é a fração do número de defeitos inicialmente presentes que seria encontrado por um determinado conjunto de testes [24].

Muitos experimentos foram realizados para investigar a correlação entre cobertura de testes e confiabilidade de software [10] [11] [17] [3]. Estes trabalhos são motivados pela necessidade de se determinar a confiabilidade por medidas diretas [15].

Em [15] os autores propõem um modelo de confiabilidade baseados na cobertura estrutural do programa e em critérios baseados em fluxo de controle. O programa é representado como um grafo de fluxo de controle (Figura 5, Seção 3.3.1). À medida que os testes são executados, nós diferentes são executados um número diferente de vezes. O modelo fundamenta-se no fato de que quanto mais vezes um nó é executado sem apresentar falhas, maior será a probabilidade de que ele não esconda defeitos, aumentando assim sua confiabilidade.

A confiabilidade de um nó $i$ é dada por:

$$R_i(t) = e^{\lambda_i t} \tag{4.15}$$

onde $t$ é o tempo transcorrido e $\lambda_i$ é a intensidade de falhas do nó $i$ com respeito ao tempo, dada por:

$$\lambda_i = k_i e^{-\theta_i n_i} \tag{4.16}$$

onde $k_i$ é interpretada como a intensidade de falhas inicial do nó $i$. $\theta_i$ é uma constante que representa a redução da intensidade de falhas alcançada por uma execução adicional do nó $i$.

Dadas as confiabilidades dos nós, é então determinada a confiabilidade dos caminhos e a probabilidade destes caminhos serem percorridos de acordo com o perfil operacional do sistema. A confiabilidade do sistema é determinada a partir das confiabilidades dos caminhos e de suas probabilidades de serem executados.

No entanto, apesar deste modelo basear–se na cobertura dos testes, ele não é orientado a defeitos. Este modelo necessita do perfil operacional do sistema e está sujeito a erros no levantamento deste perfil. O modelo proposto por [24] é um modelo cuja determinação da confiabilidade depende apenas da cobertura obtida pelos critérios definidos pelas técnicas de teste estruturais. Na realidade, o modelo trata defeitos como se fossem elementos requeridos por um critério que exige que todos os defeitos sejam cobertos pelos testes. Para cada um destes critérios há um modelo logarítmico de crescimento de cobertura. Para distingüi-los, é utilizado um índice, onde o valor 0 é reservado para defeitos. Assim, $C^0$ refere-se à cobertura de defeitos. Os autores utilizam 4 critérios de cobertura de teste. São eles: $C^1$ – cobertura do critério Todos Nós, $C^2$ – cobertura do critério Todos Arcos, $C^3$ – cobertura do critério Todos C–Usos e $C^4$ – cobertura do critério Todos P–Usos.

Este modelo assume que a cobertura dos elementos segue um modelo logarítmico, da se-

guinte maneira:

$$C^i(t) = b_0^i \cdot log\left(1 + b_1^i t\right) \tag{4.17}$$

onde $i = 0, 1, 2, 3, 4$, $C^i(t) \leq 1$. As constantes $b_0$ e $b_1$ são parâmetros determinados estatisticamente para o ajuste da curva do modelo aos dados reais. A relação entre $C^0$ e $C^i(i = 1, 2, 3, 4)$ é:

$$C^0(C^i) = a_0^i \cdot log\left[1 + a_1^i\left(e^{a_2^i C^i} - 1\right)\right] \tag{4.18}$$

onde $a_0$, $a_1$ e $a_2$ são constantes determinadas para o ajuste das curvas do modelo.

O mais interessante neste último modelo é que, para dados reais (com $a_0$, $a_1$ e $a_2$ devidamente determinados), a curva tem um comportamento linear para valores suficientemente altos de cobertura de teste. Assim, o modelo pode ser utilizado para a estimação do número total de defeitos restantes do software [23].

### 4.2.1 Modelos Baseados em Cobertura de Crespo

Em seu trabalho, Crespo [4] faz um estudo sobre modelos de confiabilidade e propõe três novos modelos baseados em cobertura: o Modelo Tipo Binomial Baseado em Cobertura (MBBC), o Modelo Tipo Poisson Baseado em Cobertura (MPBC) e o Modelo de Categoria de Falhas Infinitas Baseado em Cobertura (MFIBC). Estes modelos são baseados na cobertura de elementos requeridos de testes estruturais, onde a informação sobre a cobertura entra diretamente na formulação da forma funcional.

Os resultados descritos pelo autor mostram que o modelo MBBC e o modelo MFIBC apresentam uma precisão melhor para as estimativas do que os modelos baseados em tempo utilizados para fazer a comparação. Os resultados obtidos com o modelo MPBC mostram que este modelo não conseguiu se ajustar aos dados. O modelo MBBC apresentou resultados ligeiramente melhores do que o modelo MFIBC. Portanto, escolhemos o modelo MBBC para comparar os resultados com o modelo neural baseado em cobertura (MNBC) desenvolvido em nosso trabalho. Na seção seguinte, este modelo é descrito com maiores detalhes.

### 4.2.2 Modelo Binomial Baseado em Cobertura – MBBC

O modelo tipo binomial (MBBC) proposto por Crespo [4] utiliza dados de critérios de cobertura baseados em fluxo de controle (Todos–Nós e Todos–Arcos) e de critérios baseados em fluxo de dados (Critérios Potenciais–Uso). Neste modelo, o processo de falhas no software é caracterizado pelo comportamento da taxa de ocorrência das falhas no software. Neste tipo de modelo o número de falhas observadas no software segue uma distribuição binomial de probabilidades.

A suposição básica deste modelo é que a taxa de falhas do software está diretamente relacionada com o complemento da cobertura atingida pelos dados de teste aplicados.

O modelo assume as seguintes hipóteses:

- Cada defeito tem a mesma chance de ser detectado dentro de uma mesma classe de dificuldade.
- As falhas não estão correlacionadas.
- Existe um número N limitado de defeitos no início dos testes.
- Os dados de teste são aplicados e a cobertura dos elementos requeridos do critério de seleção utilizado na avaliação dos dados é calculada a cada ocorrência de falha.
- Para cada defeito $a$, a taxa de falhas, $Z_a(n)$ tem a seguinte forma funcional:

$$Z_a(n) = \alpha n^{\alpha-1} \tag{4.19}$$

onde $n$ é o número de dados de teste aplicados na detecção do defeito $a$ e $\alpha$ é o complemento da cobertura alcançada com a aplicação dos $n$ dados de teste.

## 4.3 Modelos Neurais

Um modelo neural é proposto por Karunanithi *et al* [19]. Neste trabalho os autores propõem um modelo que utiliza redes neurais artificiais para a determinação da confiabilidade de um software. A aplicação das redes neurais é explorada utilizando vários tipos de redes – *feed forward*, recorrentes (redes de Elman)[8] e semi-recorrentes (redes de Jordan), regimes de treinamento e métodos de representação dos dados. Das bases utilizadas, uma delas está disponível em [34]. Os resultados apresentados sugerem que os modelos neurais podem se adaptar bem aplicados a diferentes conjuntos de dados e exibir uma precisão maior nas estimativas. Os modelos neurais ainda são capazes de desenvolver modelos de complexidades diferentes. Os autores também sugerem que as redes recorrentes de Jordan apresentam um desempenho melhor porque elas são capazes de capturar a correlação entre entradas consecutivas [19] devido à sua característica de recorrência. Esta característica também pode ser atribuída a uma rede quando acrescentamos mais uma entrada à rede que irá conter os dados posteriores da entrada anterior, ou seja, esta entrada contém um *atraso* com relação à entrada original.

O mesmo estudo é realizado por Sitte em [40]. Neste estudo, a autora utilizou duas das bases disponíveis em [34] e compara os resultados obtidos com modelos neurais e modelos de recalibração paramétrica[1]. A autora conclui que os modelos neurais não são apenas mais simples de utilizar do que os métodos de recalibração paramétrica, mas ainda são iguais ou melhores para a estimativa de confiabilidade. Este trabalho complementa o trabalho de Karunanithi et al. [19].

[1]Estes modelos consistem em recalcular os parâmetros do modelo ao longo do processo e não são abordados em nossos experimentos.

Sultan et. al [1] utilizam redes neurais com uma arquitetura um pouco diferente das arquiteturas propostas em modelos anteriores. Os autores utilizaram uma rede neural *feed forward* com uma camada de entrada, uma camada intermediária e uma camada de saída. O número de neurônios na camada de entrada representa o número de atrasos aplicados à entrada. Neste trabalho os autores utilizaram 4 atrasos, representando o número de falhas encontradas nos dias anteriores. Este trabalho difere um pouco dos dois anteriores pois utiliza dados de falha acumulados durante um determinado tempo (normalmente durante um dia). O problema destes tipos de modelos é que eles consideram que a atividade de testes e o processo de remoção de defeitos é uma atividade com um ritmo constante e igual em todos os dias. Não levam em cosideração se, por exemplo, durante um dia o período de testes foi menor ou maior do que nos outros.

## 4.4 Considerações Finais

Este capítulo apresentou os principais modelos de confiabilidade baseados no domínio do tempo e baseados em cobertura de testes e apresentou alguns trabalhos que utilizam modelos neurais. A precisão dos modelos baseados no domínio do tempo depende do perfil operacional do sistema e, portanto, estão sujeitos a possíveis erros no levantamento deste perfil. Ou seja, a qualidade destes modelos é afetada pela habilidade em estimar corretamente o perfil operacional do sistema [38]. O problema é que o tempo entre falhas não é uma medida de complexidade, mas sim de confiabilidade de software. Quando o perfil é incorretamente elaborado, ou a complexidade da estrutura do software é muito alta, os modelos baseados no domínio do tempo interpretarão errôneamente o alto tempo entre falhas como uma medida de confiabilidade. Por outro lado, os modelos baseados em cobertura de teste assumem que a cobertura de elementos está associada a cobertura de defeitos. Assim, a determinação da confiabilidade não depende mais do tempo entre falhas, mas sim, da cobertura dos elementos do critério de teste escolhido.

Os modelos neurais apresentam um vantagem significativa sobre os modelos analíticos pois eles requerem como entrada apenas o histórico de falhas e não fazem nenhuma suposição sobre o comportamento da confiabilidade. Com estas entradas, o modelo neural “aprende”o comportamento dos dados de falha através de algoritmos de aprendizado de redes neurais. Devido a esta característica, os modelos neurais são capazes de se ajustarem conforme a complexidade e características de cada base de dados.

No próximo capítulo serão apresentados os experimentos realizados para a determinação de um modelo neural baseado no domínio do tempo e outro baseado em cobertura bem como os resultados obtidos com cada um destes modelos.

# 5 *Experimentos Realizados*

Neste capítulo serão apresentados os dois experimentos realizados com a utilização de redes neurais para modelagem da confiabilidade de software. Cada experimento visa obter uma classe diferente de modelo. O primeiro visa a classe de modelos baseados no domínio do tempo e o segundo visa a classe de modelos baseados em cobertura de testes. Para a escolha dos modelos analíticos utilizados nos experimentos com modelos baseados no domínio do tempo, foi utilizado o software SMERFS (Software Modeling and Estimation of Reliability Functions for Software) [9]. Para cada base foi determinado um modelo analítico e escolhemos os dois modelos que apresentaram melhor ajuste aos dados. São eles: o Modelo de Jelinski–Moranda (Seção 4.1.1) e o Modelo Geométrico (Seção 4.1.2).

Para os experimentos realizados com modelos baseados em cobertura de teste utilizamos o modelo MBBC proposto por Crespo [4]. Como dito no capítulo anterior, na avaliação do autor, o modelo tipo binomial (MBBC) obteve melhor desempenho do que os outros dois modelos, o modelo de categoria de falhas infinitas baseado em cobertura (MFIBC) e o modelo tipo Poisson baseado em cobertura (MPBC). Por causa desses resultados, o modelo MBBC foi o escolhido para fazer a avaliação dos modelos utilizando redes neurais propostos neste trabalho.

O objetivo do primeiro experimento foi comprovar a capacidade dos modelos neurais de se ajustarem a projetos de características diferentes segundo a sua predictibilidade e, para isto, foi utilizada uma medida de avaliação que indica o erro do último ponto da estimativa, chamado de Erro de Predição. O segundo experimento foi realizado para comprovar a aplicabilidade das redes neurais para a obtenção de modelos de confiabilidade com dados de cobertura de teste, uma vez que estes tipos de modelos não tinham sido explorados anteriormente. Neste experimento não faria sentido calcular o Erro de Predição, pois o experimento não objetivava analisar esta característica e, portanto, não foram realizadas predições da confiabilidade. Foi utilizada então, uma medida chamada de Teste de Kolmogorov, também utilizada em [4], possibilitando a comparação dos resultados. Em ambos os casos ainda foram utilizadas as medidas de avaliação *Average Error*, *Average Bias* e Coeficiente de Correlação (r).

## 5.1 Modelo Neural Baseado no Domínio do Tempo – MNT

Esta seção descreve o modelo Neural Baseado no Domínio do Tempo e os experimentos realizados com os modelos baseados em tempo aplicados a vários projetos diferentes cujo objetivo é avaliar a adaptabilidade e a capacidade de predição dos modelos neurais comparada às

dos Modelos Jelinski–Moranda e do modelo Geométrico.

### 5.1.1 Base de Dados

As bases utilizadas neste trabalho foram obtidas por *John Musa* da *Bell Telephone Laboratories* [34] com o objetivo de coletar dados de intervalo entre falhas para auxiliar gerentes de projetos no acompanhamento das atividades de teste, estimativa de prazos, e em auxiliar pesquisadores na avaliação de modelos de confiabilidade de software. O conjunto de dados consiste em dados de intervalo entre falhas de 16 projetos relativos a diferentes aplicações, entre elas comando e controle em tempo real, processadores de texto e aplicações comerciais e militares.

As informações armazenadas em cada uma das bases são as seguintes:

- Identificação do Projeto;
- Número da Falha;
- Tempo entre Falhas;
- Dia da Ocorrência.

### 5.1.2 Adequação dos Dados

Para que os dados destas bases pudessem ser devidamente aplicados às técnicas utilizadas nos experimentos, os dados passaram por um pré–processamento que consistem em 3 etapas:

- Armazenamento dos dados em uma estrutura de dados adequada (vetores),
- Aplicar a média móvel nos dados de tempo entre falhas, e
- Transformação dos dados de tempo entre falhas em tempo transcorrido.

As etapas do pré–processamento são descritas a seguir.

Para cada projeto, as informações das bases foram armazenadas em dois vetores diferentes: um vetor contendo os números das falhas e outro contendo o intervalo entre as falhas. Os dados referentes aos dias em que as falhas ocorreram não foram armazenados porque não foram utilizados nos experimentos.

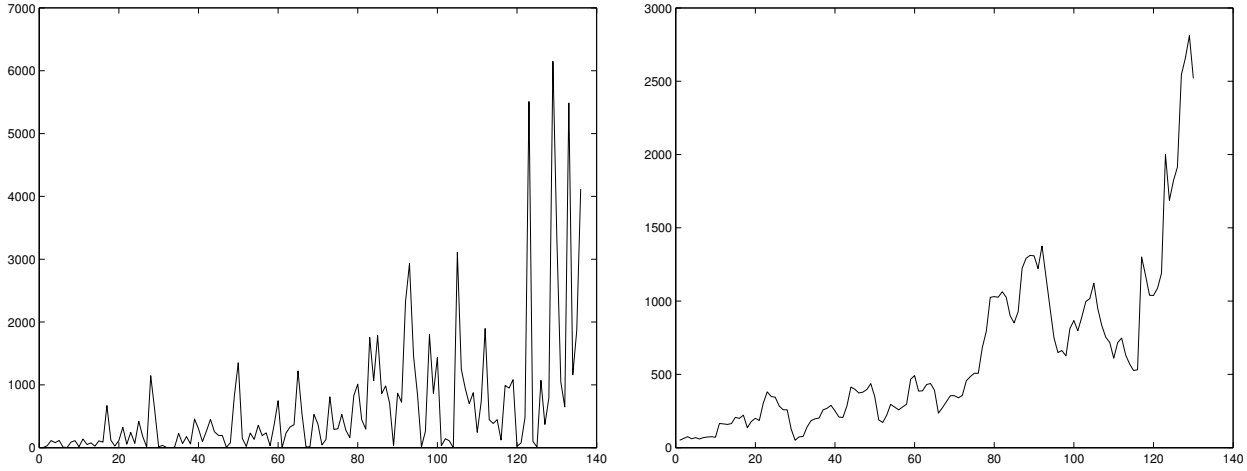
Dados de confiabilidade, como tempo entre falhas, podem apresentar ruído nos valores observados devido à natureza da operação do sistema [40]. Essa característica interfere negativamente no processo de convergência na obtenção de modelos que procuram descrever o comportamento destas variáveis. Para diminuir o ruído, foi utilizado o método das médias móveis, que é definida em [5] como:

Dado um conjunto de números: $Y_1, Y_2, Y_3, ..., Y_n$, define-se média aritmética móvel de ordem $K$, à sequência de médias:

$$\frac{Y_1, Y_2, ..., Y_K}{K}; \frac{Y_2, Y_3, ..., Y_{K+1}}{K}; \frac{Y_3, Y_4, ..., Y_{K+2}}{K} ... \frac{Y_{n+1-k}, Y_{n+2-k}, ..., Y_n}{K} \tag{5.1}$$

Quanto maior for a ordem $K$, maior será o efeito de suavização das variações.

A Figura 6 mostra o resultado da utilização da média móvel na base de dados 1. A Figura 6.a ilustra o gráfico do tempo entre falhas. A Figura 6.b ilustra o gráfico depois de aplicada a média aritmética de ordem 7 nos dados de falha.




(a) (b)

Figura 6: Utilização da média móvel para a suavização de ruído.

O cálculo da média móvel dos intervalos entre falhas ainda não é o suficiente para a adequação dos dados para os experimentos com redes neurais. Os dados continuam apresentando uma variação muito grande, e ainda é difícil que os modelos com redes neurais consigam convergir para uma solução ótima. Depois de calcular a média móvel, os dados dos intervalos entre falhas são somados de maneira a se obter um vetor com o tempo transcorrido absoluto, ou seja, o tempo transcorrido desde o início dos testes. A Figura 7 mostra o tempo em que cada uma das 136 falhas ocorreram na base de dados 1 e os intervalos entre estas. Note que as duas curvas não estão na mesma escala. Os intervalos entre as falhas foram aumentados por um fator de 25 para que se possa observar justamente a suavidade de cada curva.

### 5.1.3 Avaliação dos Modelos

Para a avaliação dos modelos obtidos neste experimento, foram utilizadas 4 medidas:

- **Desvio Máximo [40]**: É a diferença máxima entre o valor observado e o valor estimado

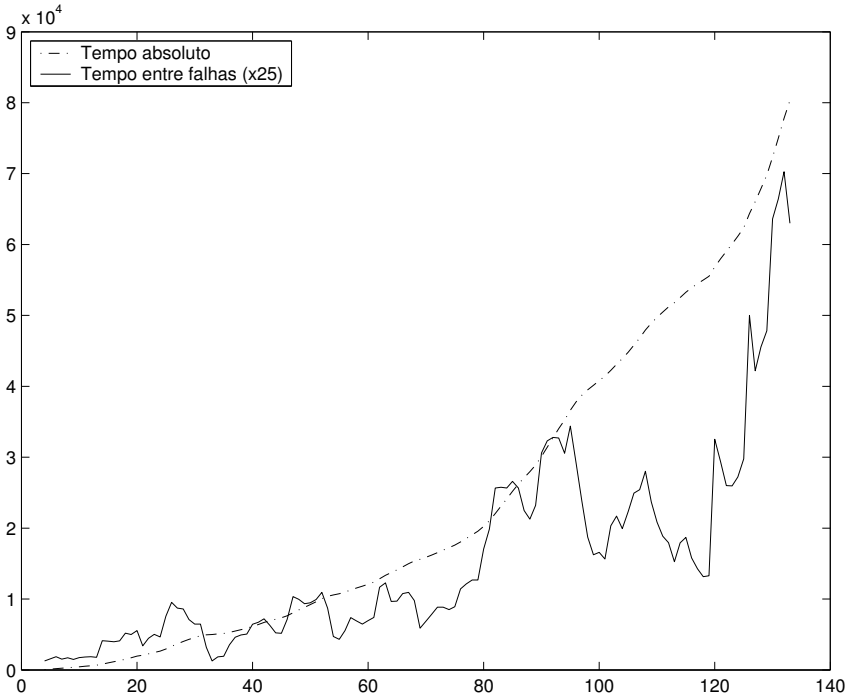



Figura 7: Falhas *x* Tempo.

pelo modelo. É expresso matematicamente por:

$$max\left(\frac{\|y-\hat{y}\|}{y}\right) \tag{5.2}$$

onde: $y$ é o valor observado e $\hat{y}$ é a estimativa deste valor.

- **Average Bias (AB) [18]**: É uma medida da tendência do modelo em superestimar ($AB < 0$) ou subestimar ($AB > 0$) o número de defeitos presentes no software. É dado pela fórmula:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{y_i-\hat{y}_i}{y_i} \tag{5.3}$$

onde: $y$ é o valor observado e $\hat{y}$ é a estimativa deste valor, e $n$ é o número total de estimativas realizadas pelo modelo.

- **Average Error (AE) [18]**: Mede a qualidade das estimativas realizadas pelo modelo e o quão bem o modelo se ajusta aos dados observados. É dado pela expressão:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{\|y_i-\hat{y}_i\|}{y_i} \tag{5.4}$$

onde: $y$ é o valor observado $\hat{y}$ é o valor estimado, e $n$ é o número total de estimativas realizadas.

- **Erro de Predição (EP) [40]**: É o erro da estimativa do último ponto do conjunto de dados estimados. É dado por:

$$\frac{y_n-\hat{y}_n}{y_n} \tag{5.5}$$

onde: $y$ é o valor observado, $\hat{y}$ é a estimativa deste valor, e $n$ é o número de estimativas realizadas.

- **Coeficiente de Correlação (r)**: É outra medida que representa o quão bem os dados estimados se relacionam com os dados reais. Quanto mais próximo de 1, melhor é o ajuste das estimativas do modelo.

### 5.1.4 Configuração das Redes Neurais

Foram utilizadas no experimento, redes de camadas múltiplas (*feed forward*) com 1 camada de entrada, 1 camada intermediária e 1 camada de saída, cuja estrutura está ilustrada na Figura 8. A camada de entrada tem um neurônio cuja função de transferência é $f_e(x) = x$. Pode-se dizer que a função deste neurônio é distribuir os dados para os neurônios da camada intermediária. A camada intermediária apresenta 2 neurônios com função de transferência $f_h$ do tipo sigmóide. A camada de saída apresenta 1 neurônio com função de transferência $f_s$ do tipo linear [8]. Esta configuração foi obtida experimentalmente. Várias configurações foram experimentadas antes de escolhermos a mais adequada. Quando aumentamos o número de neurônios na camada intermediária, por exemplo, o desempenho das redes não apresentava uma melhora significativa nos resultados. Decidiu-se que seria utilizada a configuração mais simples possível. Portanto os experimentos foram realizados com 2 neurônios na camada intermediária. O mesmo foi realizado com respeito ao número de camadas intermediárias, onde decidiu-se utilizar apenas uma.

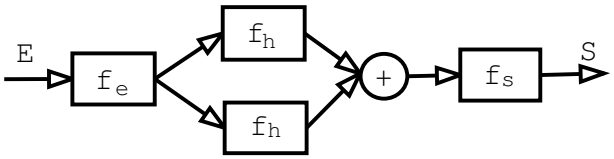



Figura 8: Estrutura da rede utilizada.

Na Figura 8, $E$ representa o vetor de entrada e $S$ representa o vetor de saída. Cada neurônio está caracterizado com sua função de transferência de acordo com a descrição anterior.

A utilização da função linear na camada de saída é para permitir um mapeamento contínuo entre as variáveis de entrada e saída. Experimentamos a utilização de uma função sigmóide na camada de saída e verificamos erros de predição muito baixos. Entretanto, isso não significa que o modelo esteja correto. O baixo erro ocorre porque a normalização dos dados (ver Seção 5.1.5) utiliza o último ponto como valor máximo. Como a função sigmóide apresenta uma característica de saturamento, sempre que o modelo saturar, a curva irá convergir para este valor. O problema é que, na prática, este valor não está disponível. Na realidade é justamente este o objetivo do problema. Portanto, neste caso, a função mais adequada é a função linear para a camada de saída da rede neural.

### 5.1.5 Treinamento das Redes Neurais Artificiais

As redes foram treinadas utilizando como entrada o tempo atual e o tempo que ocorreu a falha anterior. Ou seja, foram utilizados $t_i$ e $t_{i-1}$ para encontrar uma estimativa para $f(t_i)$. Esta é uma forma de recorrência *forçada*, e foi utilizada nos experimentos para obter a correlação entre entradas consecutivas descrita por Karunanithi *et al* [19] (ver Seção 4.3).

Antes de realizar o treinamento da rede, é necessário que os dados estejam dentro de uma faixa de valores limitados, de acordo com a função de transferência escolhida (Figura 3), para garantir a convergência do algoritmo de aprendizado da rede neural. Neste trabalho, foi utilizada a função *tan-sigmóide* nos neurônios da camada intermediária. Por isso, os dados foram normalizados entre $-1$ e $+1$. Assim, cada um dos vetores de entrada sofreram a seguinte tranformação:

$$p_n = 2\left(\frac{p - min(p)}{max(p) - min(p)}\right) - 1 \tag{5.6}$$

onde: $p_n$ é o vetor normalizado, $p$ é o vetor a ser normalizado, e $max(p)$ e $min(p)$ são os valores máximos e mínimos de $p$ respectivamente.

Cada conjunto de dados foi dividido em duas bases: uma base de treinamento e uma base de teste. A base de treinamento foi constituída neste experimento pelas primeiras $67\%$ das amostras. Esta base é utilizada para o treinamento das redes e, na prática, representam os dados coletados durante os testes. A base de testes foi constituída dos $33\%$ de dados restantes e é utilizada para avaliar a rede treinada segundo os critérios estabelecidos na Seção 5.1.3. Na prática, estes dados representam os valores futuros e que deseja–se estimar.

Os resultados obtidos com os modelos de confiabilidade utilizando redes neurais foram comparados com as estimativas de outros dois modelos bastante conhecidos: o Modelo de Jelinski–Moranda e o modelo Geométrico. Estes modelos foram obtidos utilizando o SMERFS (Statistical Modeling and Estimation of Reliability Functions of Softwre [9]).

### 5.1.6 Análise dos Resultados

As Tabelas 1, 2 e 3 mostram os resultados obtidos com modelos utilizando redes neurais artificiais (MNT), com o Modelo de Jelinski-Moranda (JAM) e com o Modelo Geométrico (GEO).

O modelo neural MNT descrito neste capítulo apresentou um melhor ajuste aos dados reais da maioria dos projetos utilizados no experimento. Uma alta correlação das estimativas (coluna "Coeficiente de Correlação" das Tabelas 1, 2 e 3) geralmente significa um melhor desempenho nos outros critérios. Porém, existem casos onde isso não acontece. Como é o caso da base 27, onde apesar das estimativas terem se ajustado melhor aos dados reais em comparação aos outros dois tipos de modelo, os erros das estimativas (coluna "Average Error" da Tabela 1) do modelo neural foram maiores do que os erros do modelo logarítmico (Tabela 3).

Podemos observar que dos 16 projetos, o modelo obtido utilizando redes neurais (MNT) alcançou um coeficiente de correlação maior de que $0,99$ em 10 destes projetos, isto é, em

Tabela 1: Modelo Utilizando Redes Neurais (MNT)

| *Base* | *Desvio máximo (%)* | *Erro da última predição (%)* | *Average Error (%)* | *Average Bias (%)* | *Coeficiente de correlação (%)* |
|---|---|---|---|---|---|
| 1 | 13,82 | -04,88 | 06,77 | 07,15 | 99,29 |
| 14C | 12,53 | 12,53 | 06,32 | 06,32 | 99,51 |
| 17 | 17,67 | -12,48 | -12,33 | 12,33 | 99,31 |
| 2 | 28,54 | 28,54 | 20,98 | 20,98 | 98,29 |
| 27 | 24,17 | 24,17 | 11,11 | 11,11 | 98,72 |
| 3 | 33,02 | 33,02 | 16,17 | 16,35 | 95,25 |
| 4 | 08,55 | 00,45 | -02,30 | 03,54 | 99,61 |
| 40 | 12,20 | 04,79 | 07,36 | 07,36 | 99,47 |
| 5 | 22,15 | -14,17 | -13,73 | 13,87 | 99,21 |
| 6 | 19,68 | 19,68 | 10,88 | 10,88 | 98,74 |
| SS1A | 11,60 | 11,60 | 03,23 | 05,17 | 99,47 |
| SS1B | 21,91 | 21,91 | 12,04 | 12,04 | 98,61 |
| SS1C | 11,32 | 11,32 | 01,95 | 05,70 | 99,33 |
| SS2 | 18,54 | -18,54 | 07,49 | 07,52 | 99,34 |
| SS3 | 08,21 | 08,21 | 03,41 | 03,62 | 99,80 |
| SS4 | 23,63 | 23,63 | 12,54 | 12,54 | 98,46 |

Tabela 2: Modelo Geométrico (GEO)

| *Base* | *Desvio máximo (%)* | *Erro da última predição (%)* | *Average Error (%)* | *Average Bias (%)* | *Coeficiente de correlação (%)* |
|---|---|---|---|---|---|
| 1 | 06,37 | 03,67 | 04,15 | 07,69 | 99,71 |
| 14C | 189,75 | 31,86 | -54,58 | 59,89 | 77,32 |
| 17 | 27,54 | -27,54 | -12,09 | 12,09 | 98,99 |
| 2 | 13,35 | 13,32 | 10,26 | 10,26 | 99,32 |
| 27 | 07,65 | 03,29 | -02,21 | 04,28 | 97,89 |
| 3 | 15,75 | -14,10 | -11,77 | 11,77 | 99,36 |
| 4 | 56,84 | -56,84 | -15,11 | 15,11 | 97,10 |
| 40 | 83,51 | -71,64 | -62,12 | 62,12 | 94,02 |
| 5 | 13,24 | 13,24 | 08,28 | 08,28 | 99,11 |
| 6 | 26,36 | 22,26 | 20,89 | 20,89 | 96,37 |
| SS1A | 37,28 | -37,28 | -25,91 | 25,91 | 98,40 |
| SS1B | 18,28 | 18,28 | 09,18 | 09,18 | 98,81 |
| SS1C | 17,86 | -07,81 | -10,74 | 10,74 | 99,50 |
| SS2 | 40,67 | 36,00 | 37,79 | 37,79 | 99,03 |
| SS3 | 22,31 | 11,78 | 16,65 | 16,65 | 99,31 |
| SS4 | 07,10 | -07,10 | -02,23 | 02,43 | 99,82 |

$62,5\%$ dos casos, contra $50\%$ (8 projetos) para o modelo Geométrico e $31,25\%$ (5 projetos) para o modelo de Jelinski–Moranda. Isto significa que o modelo MNT conseguiu se adaptar melhor à diversidade de projetos do que o modelo Geométrico que por sua vez se adaptou melhor de que o modelo de Jelinski–Moranda. Observa-se também que o modelo MNT obteve um erro médio (Average Error – AE) melhor do que os outros dois tipos de modelos em 8 projetos dos 16 analisados, ou seja, em $50\%$ dos casos. A Figura 9 mostra um gráfico comparativo dos módulos dos erros médios (*Average Error*) obtidos pelos modelos nos experimentos realizados em cada uma das bases.

Tabela 3: Modelo de Jelinski–Moranda (JM)

| *Base* | *Desvio máximo (%)* | *Erro da última predição (%)* | *Average Error (%)* | *Average Bias (%)* | *Coeficiente de correlação (%)* |
|---|---|---|---|---|---|
| 1 | 27,01 | 27,01 | 13,82 | 14,39 | 97,50 |
| 14C | 61,43 | 61,43 | 53,59 | 53,59 | 92,01 |
| 17 | 10,37 | -10,37 | -06,17 | 06,17 | 99,62 |
| 2 | 35,54 | 35,54 | 22,92 | 22,92 | 94,07 |
| 27 | 27,30 | 27,30 | 13,54 | 13,80 | 96,41 |
| 3 | 30,27 | 30,27 | 16,16 | 16,16 | 95,55 |
| 4 | 10,82 | -08,30 | -05,35 | 05,50 | 99,42 |
| 40 | 13,89 | 13,89 | -00,93 | 07,50 | 98,52 |
| 5 | 15,66 | 15,66 | 09,11 | 09,11 | 98,86 |
| 6 | 31,95 | 31,95 | 24,33 | 24,33 | 94,10 |
| SS1A | 64,86 | 64,86 | 57,50 | 57,50 | 91,37 |
| SS1B | 19,94 | 19,94 | 10,05 | 10,05 | 98,55 |
| SS1C | 15,99 | -02,36 | -07,75 | 07,75 | 99,53 |
| SS2 | 18,68 | 11,75 | 14,62 | 14,62 | 99,09 |
| SS3 | 26,08 | -25,45 | -09,30 | 11,44 | 98,79 |
| SS4 | 06,88 | -06,88 | -02,12 | 02,36 | 99,82 |

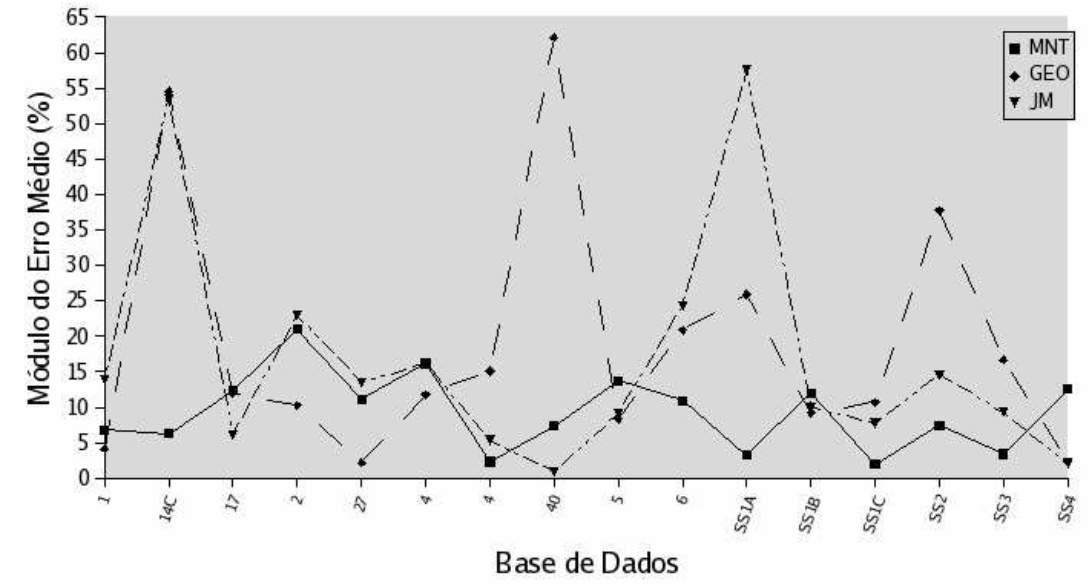



Figura 9: Erro médio dos modelos em cada uma das bases.

## 5.2 Modelo Neural Baseado na Cobertura de Critérios de Teste – MNC

Esta seção descreve o Modelo Neural Baseado em Cobertura (MNC) e os experimentos realizados com este modelo, cujo objetivo é verificar se os modelos neurais são adequados para estes tipos de dados através do teste de Kolmogorov–Smirnov, e comparar os resultados com o Modelo Binomial Baseado em Cobertura (MBBC).

### 5.2.1 Bases de Dados

A base de dados escolhida foi obtida por Crespo [4] a partir de um programa denominado "Space". Durante o procedimento de teste de integração e do uso operacional deste software, 33 defeitos foram descobertos e registrados. Após a remoção, os mesmos defeitos foram reimplantados no software com a finalidade de viabilizar a realização de experimentos baseados em teste. No experimento realizado por Crespo, dos 33 defeitos reimplantados, 28 foram detectados durante a fase de teste após a aplicação de 1240 dados de teste. Os 5 defeitos restantes não foram detectados, mesmo após a aplicação de 20.000 dados de teste.

A Tabela 4 contém as informações levantadas por Crespo e que serão utilizadas neste trabalho. Nesta tabela, a primeira coluna é numerada sequencialmente representando o número de defeitos removidos até o momento. A segunda coluna representa os dados de teste entre falhas, ou seja, quantos testes foram executados para a observação de uma falha desde a remoção do último defeito. A quarta coluna agrupa os dados de cobertura dos critérios de teste utilizados (Todos–Nós, Todos–Arcos, PU, PUDU e PDU). Por fim, a última coluna é a confiabilidade observada obtida pelo método de Nelson [36]. Destes dados, somente os dados sobre confiabilidade observada e a cobertura de cada um dos critérios foi utilizada. Os dados de teste acumulados foram utilizados para plotar os resultados.

### 5.2.2 Medida de Avaliação

Neste experimento, adotamos as seguintes medidas:

**Teste de Kolmogorov–Smirnov** O teste de Kolmogorov–Smirnov procura determinar se duas bases de dados diferem significativamente. Assim, dados coletados em uma ocasião podem ser comparados com dados coletados em outra ocasião ou comparados com dados obtidos por um modelo matemático, com o objetivo de verificar se as duas amostragens apresentam a mesma distribuição. Em caso afirmativo dizemos que as duas amostras foram retiradas da mesma população e podemos concluir que elas representam a mesma realidade [42].

Para fazer o teste de Kolmogorov–Smirnov, primeiramente medimos a diferença entre as duas amostras. Multiplica-se o maior valor obtido ($D_{max}$) pelo tamanho da amostra ($N$). Se este valor estiver abaixo do valor crítico ($D_{critico}$), com um nível de significância $\alpha$, obtido na Tabela 5 (extraída de [42]), então pode-se dizer que existe uma igualdade na distribuição das amostras.

Além do teste de Kolmogorov, foram utilizadas outras medidas de avaliação como *Average Bias (AB)*, *Average Error (AE)* e o Coeficiente de Correlação *(r)*, mencionados na Seção 5.1.3, cujos resultados encontram–se na Tabela 8

Tabela 4: Cobertura e Confiabilidade em função dos Dados de Teste e Defeitos Removidos

| Falha | Dados de teste entre falhas | Dados de teste acumulados | Cobertura dos critérios de teste | | | | | Confiabilidade Observada |
|---|---|---|---|---|---|---|---|---|
| | | | NÓS | ARCOS | PU | PUDU | PDU | |
| 1 | 1 | 1 | 0,30684 | 0,21296 | 0,16709 | 0,15177 | 0,07406 | 0,04652 |
| 2 | 1 | 2 | 0,33769 | 0,22341 | 0,17012 | 0,15913 | 0,07812 | 0,04694 |
| 3 | 1 | 3 | 0,35792 | 0,23576 | 0,17975 | 0,16221 | 0,08145 | 0,04783 |
| 4 | 1 | 4 | 0,38931 | 0,24863 | 0,18756 | 0,16712 | 0,08432 | 0,04691 |
| 5 | 1 | 5 | 0,39931 | 0,25396 | 0,19517 | 0,17361 | 0,08897 | 0,05131 |
| 6 | 1 | 6 | 0,40127 | 0,26011 | 0,19689 | 0,17914 | 0,09232 | 0,04370 |
| 7 | 1 | 7 | 0,41739 | 0,27391 | 0,19912 | 0,18523 | 0,09434 | 0,17720 |
| 8 | 1 | 8 | 0,42132 | 0,28113 | 0,20680 | 0,19978 | 0,09891 | 0,29733 |
| 9 | 1 | 9 | 0,44567 | 0,28843 | 0,22178 | 0,20342 | 0,10236 | 0,32400 |
| 10 | 2 | 11 | 0,46743 | 0,30163 | 0,23167 | 0,21987 | 0,11068 | 0,35916 |
| 11 | 1 | 12 | 0,47397 | 0,32407 | 0,24028 | 0,22184 | 0,11736 | 0,43435 |
| 12 | 1 | 13 | 0,47956 | 0,32874 | 0,24354 | 0,22456 | 0,11876 | 0,48067 |
| 13 | 1 | 14 | 0,48172 | 0,33113 | 0,24562 | 0,22678 | 0,11987 | 0,47350 |
| 14 | 1 | 15 | 0,48493 | 0,33333 | 0,24794 | 0,22921 | 0,12091 | 0,56916 |
| 15 | 2 | 17 | 0,49012 | 0,33984 | 0,24913 | 0,23156 | 0,12356 | 0,62400 |
| 16 | 1 | 18 | 0,49765 | 0,34128 | 0,25212 | 0,23687 | 0,12565 | 0,87067 |
| 17 | 5 | 23 | 0,50132 | 0,34987 | 0,25987 | 0,24145 | 0,12918 | 0,91267 |
| 18 | 1 | 24 | 0,50893 | 0,35052 | 0,26269 | 0,24539 | 0,13274 | 0,92333 |
| 19 | 1 | 25 | 0,51231 | 0,36791 | 0,26987 | 0,25670 | 0,13776 | 0,91000 |
| 20 | 1 | 26 | 0,52687 | 0,37189 | 0,27187 | 0,26921 | 0,14567 | 0,93600 |
| 21 | 6 | 32 | 0,53835 | 0,38359 | 0,28670 | 0,28543 | 0,15286 | 0,95467 |
| 22 | 39 | 71 | 0,57465 | 0,42328 | 0,31829 | 0,29531 | 0,16919 | 0,96467 |
| 23 | 20 | 91 | 0,64041 | 0,47751 | 0,32680 | 0,30354 | 0,17628 | 0,97867 |
| 24 | 35 | 126 | 0,68082 | 0,51587 | 0,35773 | 0,32936 | 0,19309 | 0,98933 |
| 25 | 60 | 186 | 0,68356 | 0,51984 | 0,40992 | 0,36453 | 0,21060 | 0,99067 |
| 26 | 253 | 439 | 0,71575 | 0,58068 | 0,43163 | 0,41134 | 0,24917 | 0,99933 |
| 27 | 400 | 839 | 0,71575 | 0,58068 | 0,45418 | 0,41163 | 0,25011 | 0,99733 |
| 28 | 401 | 1240 | 0,71781 | 0,58465 | 0,46212 | 0,41560 | 0,25224 | 0,99867 |

### 5.2.3 Configuração das Redes Neurais

Foram utilizadas no experimento, redes de camadas múltiplas (*feed forward*) com 1 camada intermediária ou oculta e 1 camada de saída, cuja estrutura está ilustrada na Figura 8. A camada intermediária apresenta 2 neurônios com função de transferência do tipo sigmóide. A camada de saída apresenta 1 neurônio com função de transferência também do tipo sigmóide [8]. Constatamos ainda que esta configuração era suficiente e que, aumentando o número de neurônios ou o número de camadas, a rede não apresenta melhorias significativas.

### 5.2.4 Análise dos Resultados

A Tabela 6 apresenta a confiabilidade real (extraída da Tabela 4) e a confiabilidade obtida com um modelo neural, utilizando a cobertura obtida com cada um dos critérios de teste (Todos–Nós, Todos–Arcos, PU, PUDU, PDU).

Com estes dados, pode–se calcular a distância de Kolmogorov e aplicar o teste de Kolmogorov–Smirnov. Estes resultados são comparados com os resultados do modelo MBBC. Este

Tabela 5: Tabela de D crítico do teste de Kolmogorov–Smirnov para duas amostras pequenas e independentes ($n_1 = n_2 \leq 40$).

| | $D_{critico}$ | | | $D_{critico}$ | |
|---|---|---|---|---|---|
| N | $\alpha = 5\%$ | $\alpha = 1\%$ | N | $\alpha = 5\%$ | $\alpha = 1\%$ |
| 3 | - | - | 18 | 9 | 10 |
| 4 | 4 | - | 19 | 9 | 10 |
| 5 | 5 | 5 | 20 | 9 | 11 |
| 6 | 5 | 6 | 21 | 9 | 11 |
| 7 | 6 | 6 | 22 | 9 | 11 |
| 8 | 6 | 7 | 23 | 10 | 11 |
| 9 | 6 | 7 | 24 | 10 | 12 |
| 10 | 7 | 8 | 25 | 10 | 12 |
| 11 | 7 | 8 | 26 | 10 | 12 |
| 12 | 7 | 8 | 27 | 10 | 12 |
| 13 | 7 | 9 | 28 | 11 | 13 |
| 14 | 8 | 9 | 29 | 11 | 13 |
| 15 | 8 | 9 | 30 | 11 | 13 |
| 16 | 8 | 10 | 35 | 12 | - |
| 17 | 8 | 10 | 40 | 13 | - |

modelo foi proposto por Crespo [4] e é um modelo analítico baseado em cobertura de critérios de teste.

A comparação dos resultados pode ser vista na Tabela 7. Esta tabela apresenta a distância máxima entre o valor estimado e o valor real (Dmáx), a distância de Kolmogorov (KD) e o resultado da hipótese bilateral de igualdade de amostras (H0) encontrado para cada um dos critérios em cada modelo (MNC e MBBC). Numa amostra de tamanho 28 com um nível de significância $\alpha = 0,01$, a hipótese bilateral $H_0$ de igualdade entre a confiabilidade observada e a confiabilidade estimada pelo modelo é aceita para um valor de $KD < 13$ [42]. Na Tabela 7 podemos verificar que a hipótese de igualdade é aceita para os dois modelos em todos os critérios.

Ainda podemos avaliar os modelos de acordo com os erros médios e com o coeficiente de correlação, como é mostrado na Tabela 8. Podemos constatar que o erro médio para o modelo MNC ficou mais de $30\%$ abaixo do erro médio do modelo MBBC. Este fato se deve à capacidade da rede neural de ajustar-se a pontos fora da curva.

Por uma análise visual, observamos que o modelo MNC tende a superestimar a confiabilidade, conforme a Figura 10. Esta não é uma característica desejável em um modelo de confiabilidade. Ainda assim vemos que o erro médio (AB, na Tabela 8) não acusou uma polarização negativa dos dados (superestimativa). Ao contrário, acusou uma leve polarização positiva (subestimação) no conjunto de dados como um todo. O modelo MBBC também acusou uma polarização positiva, embora um pouco mais forte. Isto apenas indica uma maior consistência nesta subestimação, ou seja, de todas as estimativas, o modelo MBBC subestimou a confiabilidade com mais freqüência que o modelo MNC.

O coeficiente de correlação (r) de ambos os modelos também acusa um maior ajuste aos dados observados para as estimativas do modelo MNC. Para todos os critérios, o coeficiente de

Tabela 6: Confiabilidade Observada e Confiabilidade Obtida pelo Modelo MNC.

| Conf. | NOS | ARCOS | PU | PUDU | PDU |
|---|---|---|---|---|---|
| 0,0465 | 0,0183 | 0,0669 | 0,0680 | 0,0699 | 0,0677 |
| 0,0469 | 0,0199 | 0,0732 | 0,0751 | 0,0722 | 0,0733 |
| 0,0478 | 0,0243 | 0,0831 | 0,0823 | 0,0814 | 0,0762 |
| 0,0469 | 0,0614 | 0,0975 | 0,0900 | 0,0921 | 0,0816 |
| 0,0513 | 0,0946 | 0,1052 | 0,1059 | 0,1065 | 0,0909 |
| 0,0437 | 0,1028 | 0,1157 | 0,1210 | 0,1104 | 0,1012 |
| 0,1772 | 0,1841 | 0,1477 | 0,1320 | 0,1161 | 0,1162 |
| 0,2973 | 0,2045 | 0,1710 | 0,1640 | 0,1409 | 0,1788 |
| 0,3240 | 0,3014 | 0,2008 | 0,1969 | 0,2273 | 0,2040 |
| 0,3592 | 0,3813 | 0,2766 | 0,3222 | 0,3330 | 0,4062 |
| 0,4344 | 0,4264 | 0,4950 | 0,4864 | 0,4711 | 0,4428 |
| 0,4807 | 0,4835 | 0,5545 | 0,5276 | 0,5342 | 0,4978 |
| 0,4735 | 0,5117 | 0,5861 | 0,5615 | 0,5766 | 0,5458 |
| 0,5692 | 0,5605 | 0,6156 | 0,5939 | 0,6249 | 0,6005 |
| 0,6240 | 0,6566 | 0,7022 | 0,6769 | 0,6498 | 0,6544 |
| 0,8707 | 0,8117 | 0,7208 | 0,7399 | 0,7113 | 0,7709 |
| 0,9127 | 0,8763 | 0,8220 | 0,8336 | 0,8497 | 0,8551 |
| 0,9233 | 0,9593 | 0,8288 | 0,9045 | 0,8880 | 0,9099 |
| 0,9100 | 0,9767 | 0,9526 | 0,9634 | 0,9538 | 0,9846 |
| 0,9360 | 0,9975 | 0,9670 | 0,9946 | 0,9651 | 0,9989 |
| 0,9547 | 0,9992 | 0,9900 | 0,9994 | 0,9971 | 1,0000 |
| 0,9647 | 0,9997 | 1,0000 | 1,0000 | 1,0000 | 1,0000 |
| 0,9787 | 0,9997 | 1,0000 | 1,0000 | 1,0000 | 1,0000 |
| 0,9893 | 0,9997 | 1,0000 | 1,0000 | 1,0000 | 1,0000 |
| 0,9907 | 0,9997 | 1,0000 | 1,0000 | 1,0000 | 1,0000 |
| 0,9993 | 0,9997 | 1,0000 | 1,0000 | 1,0000 | 1,0000 |
| 0,9973 | 0,9997 | 1,0000 | 1,0000 | 1,0000 | 1,0000 |
| 0,9987 | 0,9997 | 1,0000 | 1,0000 | 1,0000 | 1,0000 |

correlação do modelo MNC ficou acima de $0,98$ chegando a $0,9957$ para o critério "Todos os Nós". Para o modelo MBBC, o coeficiente de correlação não chegou a $0,98$ chegando a um máximo de $0,9787$ para o critério "Todos os Nós".

## 5.3 Considerações Finais

Este capítulo apresentou os dois experimentos realizados neste trabalho. No primeiro experimento comprovamos a flexibilidade dos modelos neurais para o uso em projetos com diferentes características e graus de complexidade apresentando uma capacidade de adaptação melhor do que os modelos analíticos.

No segundo experimento verificamos que os modelos neurais apresentam um bom resultado também com dados de cobertura de testes, quando comparados com o Modelo Binomial Baseado em Cobertura (MBBC).

Tabela 7: Estatística de Kolmogorov.

| | Modelo MNC | | | | | Modelo MBBC | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | NOS | ARCOS | PU | PUDU | PDU | NOS | ARCOS | PU | PUDU | PDU |
| Dmáx | 0,09 | 0,15 | 0,13 | 0,16 | 0,12 | 0,17 | 0,16 | 0,17 | 0,17 | 0,17 |
| KD | 2,60 | 4,20 | 3,73 | 4,46 | 3,36 | 4,73 | 4,48 | 4,74 | 4,73 | 4,78 |
| H0 | Aceita | Aceita | Aceita | Aceita | Aceita | Aceita | Aceita | Aceita | Aceita | Aceita |

Tabela 8: Erros médios e Coeficiente de Correlação.

| | Modelo MNC | | | | | Modelo MBBC | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | NOS | ARCOS | PU | PUDU | PDU | NOS | ARCOS | PU | PUDU | PDU |
| AE | 0,1874 | 0,2865 | 0,2775 | 0,2714 | 0,2309 | 0,4023 | 0,4228 | 0,4098 | 0,4051 | 0,4173 |
| AB | 0,0304 | 0,1740 | 0,1736 | 0,1620 | 0,1377 | 0,1795 | 0,1585 | 0,1946 | 0,1811 | 0,2058 |
| r | 0,9957 | 0,9839 | 0,9875 | 0,9867 | 0,9906 | 0,9787 | 0,9761 | 0,9784 | 0,9778 | 0,9783 |

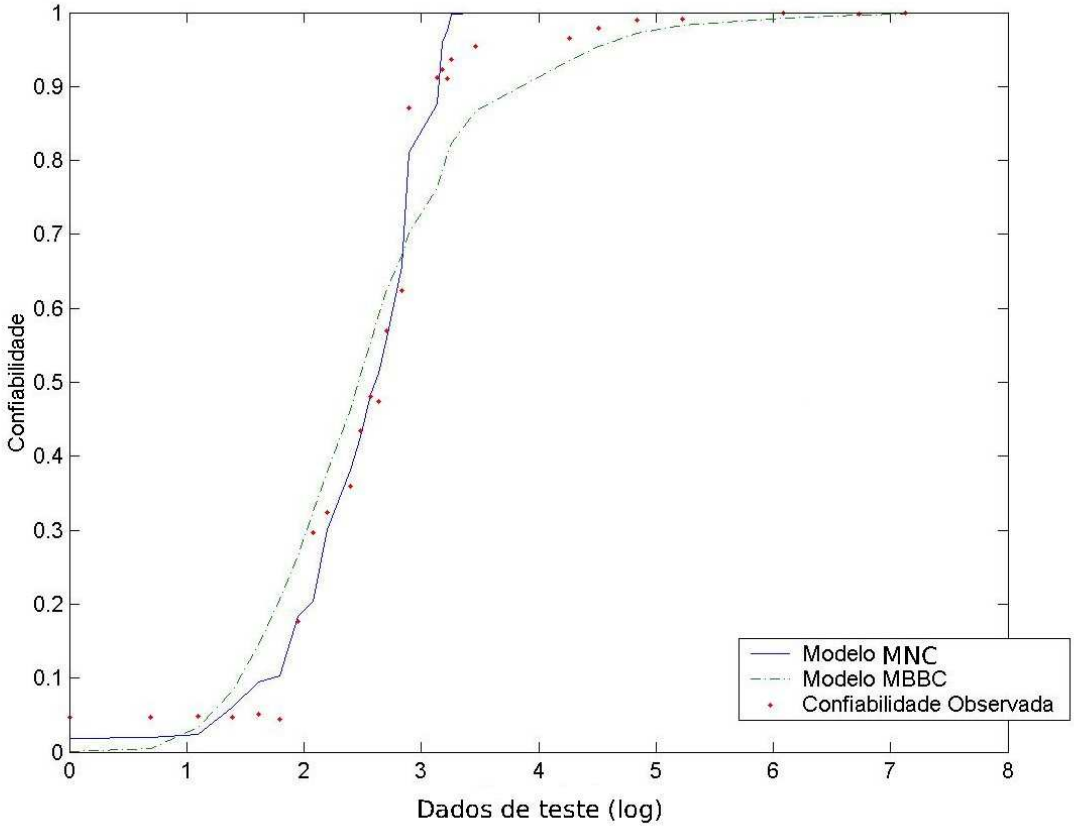



Figura 10: Gráfico dos resultados obtidos pelos dois modelos.

# 6 Conclusões e Trabalho Futuros

A qualidade de um software é determinada por diversas características que podem ou não ser percebidas pelo usuário final do sistema. O usuário do software quer que ele satisfaça suas necessidades da melhor maneira possível. Dentre todas as características, percebidas ou não pelo usuário do software, uma das mais importantes é a confiabilidade. De fato, quando o software falha, principalmente quando está executando uma tarefa crítica, todos os outros aspectos de qualidade são ignorados.

A confiabilidade de um software pode ser estimada a partir de modelos que descrevem o comportamento do seu crescimento ao longo das atividades de teste. Este trabalho abordou dois tipos de modelos de confiabilidade: os modelos baseados no domínio do tempo e os modelos baseados em cobertura de critérios de teste. Os modelos baseados no domínio do tempo procuram determinar a confiabilidade como uma função do tempo. Estes partem do princípio de que a confiabilidade cresce automaticamente ao longo do tempo e, que a probabilidade de uma falha ocorrer depende da probabilidade da porção do programa que contém o defeito correspondente a esta falha ser executada e da probabilidade de ativação deste defeito, resultando em uma falha. Isto significa dizer que a confiabilidade do software depende da maneira como ele será operado, ou seja, do seu perfil operacional. Desta forma, para determinar corretamente a confiabilidade do software é necessário conhecer exatamente como ele será operado. Existe uma grande dificuldade no levantamento do perfil operacional de um software porque o comportamento do sistema é de certa forma aleatório e não é possível prevê–lo com exatidão. Há também casos onde diferentes usuários poderão operar o software de diferentes maneiras e o custo para a determinação do perfil de operação de cada usuário é muito alto. Estes motivos ocasionam erros no levantamento do perfil operacional e estes erros levarão a uma determinação incorreta da confiabilidade. Os modelos de confiabilidade baseados em cobertura de critérios de teste, por outro lado, partem do princípio que a confiabilidade do software cresce à medida que são testados elementos que fazem parte da sua estrutura e que a probabilidade de uma falha ocorrer está diretamente relacionada ao quão exaustivamente foi testada cada porção de código.

Neste trabalho foram realizados dois experimentos com as duas classes de modelos apresentados: um baseado no domínio do tempo e outro baseado em cobertura de testes. Para a escolha dos modelos analíticos a serem utilizados nos experimentos com modelos baseados no domínio do tempo, foi utilizado o software SMERFS (Software Modeling and Estimation of Reliability Functions for Software) [9]. Para cada base foi determinado um modelo analítico e escolhemos os dois modelos que apresentaram melhor ajuste aos dados. São eles: o modelo de Jelinski–Moranda (Seção 4.1.1) e o Modelo Geométrico (Seção 4.1.2). Para os experimentos

realizados com modelos baseados em cobertura de teste utilizamos o modelo MBBC proposto por Crespo [4].

O objetivo do primeiro experimento foi analisar a adaptabilidade dos modelos neurais aplicando o modelo em diversos projetos diferentes e comparamos os modelos neurais com modelos analíticos conhecidos. Foi possível constatar que embora o modelo neural não tenha apresentado o menor erro em todos os casos, ele foi capaz de encontrar uma solução satisfatória para todos os projetos da base de dados. De maneira geral, o modelo neural conseguiu detectar a tendência do comportamento de crescimento de confiabilidade para todos os projetos em que foi aplicado. Por outro lado, os modelos analíticos obtiveram êxito em apenas alguns projetos com um bom desempenho nos projetos onde as suposições do modelo eram válidas. Portanto, quando todas as suposições de algum modelo analítico são válidas para um projeto específico, é possível que este modelo se ajuste melhor aos dados observados do que um modelo neural. Porém, para todos os casos onde estas suposições não correspondem à realidade do projeto ou quando as suposições não são conhecidas, os modelos neurais podem ser aplicados pois estes têm a capacidade de "aprender" o comportamento dos dados sem a necessidade de nenhuma suposição.

No segundo experimento verificamos que os modelos neurais apresentam um bom resultado também com dados de cobertura de testes, quando comparados com o Modelo Binomial Baseado em Cobertura (MBBC). A medida de avaliação utilizada indicou que as estimativas realizadas pelo modelo neural representam os dados observados melhor do que o modelo analítico MBBC. O erro médio (*Average Error*) das estimativas do modelo neural também ficou abaixo do erro médio obtido com o modelo MBBC. Como os dados de cobertura estão compreendidos entre 0 e 1, pudemos nos valer do efeito de saturação de uma rede neural e limitar a saída da rede entre esta mesma faixa de valores. Porém, esta propriedade deve ser utilizada com cautela porque ela tende a superestimar os dados próximos a 1.

Este trabalho representa um estudo inicial da aplicação de redes neurais artificiais para realizar a estimativa da confiabilidade de software. Os resultados apresentados neste trabalho indicam que os modelos neurais apresentam a capacidade de se adaptar bem a projetos com complexidades diferentes. Os modelos neurais também apresentam uma boa precisão quando são utilizados com dados de cobertura de teste. Porém, a tendência de superestimar os dados próximos a 1 deve ser controlada com técnicas específicas aplicadas ao aprendizado das redes neurais. É necessário estudar que técnicas seriam estas e como elas poderiam ser aplicadas no modelo neural.

Os modelos neurais devem ser também avaliados com outras bases de dados de cobertura de critérios de teste. Neste trabalho, foi utilizada apenas a base de dados levantada por Crespo [4]; porém, uma análise semelhante à realizada no primeiro experimento é necessária para podermos concluir que os modelos neurais baseados em cobertura apresentam a mesma característica de flexibilidade e capacidade de adaptação dos modelos neurais baseados no domínio do tempo.

Outras técnicas de aprendizado de máquina também podem ser exploradas para a modelagem da confiabilidade de software. Uma destas é a programação genética. Esta técnica foi inicialmente proposta por John Koza [20] e baseia–se na teoria da evolução e seleção natural

de Darwin para gerar soluções que evoluam automaticamente para uma solução ótima. Desta maneira, poderíamos utilizar a programação genética para encontrar uma função matemática que melhor represente os dados de falha. A programação genética foi pouco explorada para este tipo de problema até agora, mas elas poderiam ser muito úteis para encontrar relações implicítas entre os parâmetros de confiabilidade em diferentes projetos.

# *Bibliografia*


[1] Sultan H. Aljahdali, Alaa Sheta, and David Rine. Prediction of softwre reliability: A comparison between regression and neural network non-parametric models. In *ACS/IEEE International Conference on Computer Systems and Aplications*, pages 470–473, 2001.

[2] M. L. Chaim. Poke tool – uma ferramenta para suporte ao teste estrutural de programas baseados em análise de fluxo de dados. Master's thesis, Universidade Estadual de Campinas, 1991.

[3] Nei-Hwa Chen, Micheal R. Lyu, and W. Eric Wong. And empirical study of the correlation between code coverage and reliability estimation. In *IEEE Proceedings of Metrics*, pages 133–141, 1996.

[4] A. N. Crespo. *Modelos de Confiabilidade de Software Baseados em Cobertura de Critérios Estruturais de Teste*. PhD thesis, FEEC – UNICAMP, 1997.

[5] Elio Medeiros da Silva and Ermes Medeiros da Silva. *Matemática e Estatística Aplicada*. Editora Atlas, 1999.

[6] R.A. De Millo, R.J. Lipton, and F.G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, Vol. C-11:34–41, April 1978.

[7] Márcio Eduardo Delamaro and José Carlos Maldonado. *PROTEUM – A Tool for the Assessment of Test Adequacy for C Programs*. ICM–SC/USP, Março 1996. versão 1.1 - C.

[8] Howard Demuth and Mark Beale. *Neural Network Toolbox User's Guide*. Mathworks, Inc., 7th printing edition, Março 2001.

[9] W. H. Farr and O. Smith. Statistical modeling and estimation of reliability functions for software (smerfs) user's guide. Technical report, Naval Surface Warfare Center Dahlgren Division, 1993.

[10] F. Del Frate, P. Garg, A. P. Mathur, and A. Pasquini. On the correlation between code coverage and software reliability. In *Sixth International Symposium on Software Reliability Engineering*, pages 124–132, 1995.

[11] P. Garg. Investigating coverage-reliability relationship and sensitivity of reliability to errors in the operational profile. In *First International Conference on Software Testing, Reliability and Quality Assurance*, pages 21–35, 1994.

[12] A. L Goel and K. Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, 28:206–211, 1979.

[13] Swapna S. Gokhale and Kishor S. Trivedi. A time/structure based software reliability model. *Annals of Softwre Engineering*, 8:85–121, 1999.

[14] M. Grottke. Software reliability model study. Pets project technical report a.2, University of Erlangen-Nuremberg, 2001.

[15] Pankaj Jalote and Y. R. Muralidhara. A coverage based model for software reliability estimation. In *First International Conference on Software Testing, Reliability and Quality Assurance*, pages 6–10, 1994.

[16] Fedon Kalifeli. A reliability model for a large scale software system. Master's thesis, Universidade de Bogazici, Turquia, 1989.

[17] R. M. Karcich, R. Skibbe, A. P. Mathur, and P. Garg. On software reliability and code coverage. In *Aerospace Applications Conference*, volume 4, pages 297–308, 1996.

[18] Nachimuthu Karunanithi, Pradeep Verma, and Yashwant K. Malaiya. Predictability of software reliability models. *IEEE Transactions on Reliability*, 41(4):539–546, December 1992.

[19] Nachimuthu Karunanithi, Darrel Whitley, and Yashwant K. Malaiya. Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering*, 18(7):563–574, July 1992.

[20] John R. Koza. *Genetic Programming: On the Programming os Computers by Means of Natural Selection*. MIT Press, 1992.

[21] Peter B. Lakey and Ann Marie Neufelder. System and software reliability assurance notebook. Produced for Rome Laboratory, 1997.

[22] Michael Naichin Li, Yashwat K. Malaiya, and Jason Denton. Estimating the number of defects: A simple and intuitive approach. In *Proc. 7th Int'l Symposium on Software Reliability Engineering (ISSRE)*, pages 307–315, 1998.

[23] Yashwant K. Malaiya and Jason Denton. Estimating defect density using test coverage. Technical report, Colorado State University, 1998.

[24] Yashwant K. Malaiya, Michael Naixin Li, James M. Bieman, and Rick Karcich. Software reliability growth with test coverage. *IEEE Transactions on Reliability*, 51(4):420–426, December 2002.

[25] José Carlos Maldonado. *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. PhD thesis, Universidade Estadual de Campinas - UNICAMP, julho 1991.

[26] José Carlos Maldonado. Aspectos teóricos e empíricos de teste de cobertura de software. Notas didáticas do Instituto de Ciências Matemáticas de São Carlos, Junho 1998.

[27] John J. Marciniak. *Encyclopedia of Software Engineering*, volume 2. John Winsley & Sons, 1994.

[28] Ryszard S. Michalski, Ivan Bratko, and Miroslav Kubat. *Machine Learning and Data Mining*. Wiley, 1998.

[29] Tom Mitchell. *Machine Learning*. McGraw–Hill, 1997.

[30] Maria Carolina Monard and José Augusto Baranauskas. Redes neurais artificiais. In Solange Oliveira Rezende, editor, *Sistemas Inteligentes: Fundamentos e Aplicações*, chapter 4. Manole, 2002.

[31] P. L. Moranda. Predictions of software reliability during debugging. In *Proceedings of the Annual Reliability and Maintainability Symposium*, 1975.

[32] P. L. Moranda and Z. Jelinski. Final report on software reliability study. Technical Report MADC Report, No, 63921, McDonnell Douglas Astronautics Company, 1972.

[33] John D. Musa. A theory of software reliability and its application. *IEEE Transactions on Software Engineering*, pages 312–327, 1975.

[34] John D. Musa. Software reliability data. Data & Analysis Center for Software, January 1980.

[35] Dinesh D. Narkhede. Bayesian model for software reliability. Technical report, Indian Institute of Technology, 2001.

[36] F. Nelson. Estimating software reliability from test data. *Microelectronics and Reliability*, 17(1):67–73, 1978.

[37] Ganesh J. Pai. A survey of softwre reliability models. Technical report, Department of DCE, University of Virginia, 2002.

[38] A. Pasquini, A. N. Crespo, and P. Matrella. Sensitivity of reliability growth models to operational profile erros vs testing accuracy. *IEEE Transactions on Reliability*, 45(4):531–540, 1996.

[39] Roger Pressman. *Software Engineering. A Practitioner's Approach*. McGraw–Hill, 5th edition, 2001.

[40] Renate Sitte. Comparison of software reliability growth predictions: Neural networks vs parametric recalibration. *IEEE Transactions on Software Engineering*, 48(3):285–291, September 1999.

[41] Ian Sommerville. *Software Engineering*. Addison–Wesley, 5th edition, 2002.

[42] Kishor S. Trivedi. *Probability and Statistics with Reliability Queuing and Computer Science Applications*. Wiley, 2nd edition, 2001.