ODAIR JACINTO DA SILVA

SENSIBILIDADE A VARIAÇÕES DE PERFIL OPERACIONAL DE DOIS MODELOS DE CONFIABILIDADE DE SOFTWARE BASEADOS EM COBERTURA

CAMPINAS
2014

UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

ODAIR JACINTO DA SILVA

SENSIBILIDADE A VARIAÇÕES DE PERFIL OPERACIONAL DE DOIS MODELOS DE CONFIABILIDADE DE SOFTWARE BASEADOS EM COBERTURA

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica na Área de Engenharia de Computação.

Orientador: Prof. Dr. Mario Jino
Co-orientador: Prof. Dr. Adalberto Nobiato Crespo

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL
DISSERTAÇÃO DEFENDIDA PELO ALUNO ODAIR JACINTO DA SILVA
E ORIENTADO PELO PROF. DR. MARIO JINO

_____

CAMPINAS
2014

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

Si38s

Silva, Odair Jacinto da, 1967-
Sensibilidade a variações de perfil operacional de dois modelos de confiabilidade de software baseados em cobertura / Odair Jacinto da Silva. – Campinas, SP : [s.n.], 2014.

Orientador: Mario Jino.
Coorientador: Adalberto Nobiato Crespo.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Confiabilidade de software. 2. Perfil operacional. 3. Confiabilidade (Engenharia) - Modelos matémáticas. I. Jino, Mario,1943-. II. Crespo, Adalberto Nobiato. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Sensitivity to variations in the operational profile of two software reliability models based on coverage
Palavras-chave em inglês:
Software reliability
Operational profile
Reliability (Engineering) - Mathematical models
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Mario Jino [Orientador]
Marcos Lordello Chaim
Romis Ribeiro de Faissol Attux
Data de defesa: 28-01-2014
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Odair Jacinto da Silva

Data da Defesa: 28 de janeiro de 2014

Título da Tese: "Sensibilidade a Variações de Perfil Operacional de Dois Modelos de Confiabilidade de Software Baseados em Cobertura"

Prof. Dr. Mario Jino (Presidente): _____

Prof. Dr. Marcos Lordello Chaim: _____

Prof. Dr. Romis Ribeiro de Faissol Attux: _____

RESUMO

Diversos estudos publicados indicam que a capacidade preditiva dos modelos de confiabilidade de software, que utilizam a informação da cobertura de teste, é melhor do que a capacidade preditiva dos modelos baseados no domínio do tempo. E, por isso, esses modelos têm sido propostos por pesquisadores da área como uma alternativa aos modelos baseados no domínio do tempo. Entretanto, para chegar a uma conclusão sobre a superioridade desta classe de modelos, é necessário avaliar a sua sensibilidade a variações do perfil operacional. Uma qualidade desejável dos modelos de confiabilidade de software é a de que sua capacidade preditiva não seja afetada por variações no perfil operacional de um software.

Esta dissertação avalia, por meio de um experimento, o comportamento de dois modelos de confiabilidade de software que se baseiam na informação de cobertura do código: "Modelo Binomial Baseado em Cobertura" (MBBC) e "Modelo de Falhas Infinitas Baseado em Cobertura" (MFIBC). O experimento aplica os modelos nos dados de falhas observados durante a execução de um programa em três perfis operacionais estatisticamente distintos. Adicionalmente, seis modelos de confiabilidade de software tradicionais são utilizados para estimar a confiabilidade do software utilizando os mesmos dados de falhas. Os modelos escolhidos foram: Musa-Okumoto, Musa Básico, Littlewood-Verral Linear, Littlewood-Verral Quadrático, Jelinski-Moranda e Geométrico.

Os resultados mostram que a capacidade preditiva dos modelos MBBC e MFIBC não é afetada com a variação do perfil operacional do software. O mesmo resultado não foi observado nos modelos de confiabilidade de software baseados no domínio do tempo, ou seja, a alteração do perfil operacional afeta a capacidade preditiva desses modelos. Um resultado observado, por exemplo, é de que nenhum dos modelos tradicionais pôde ser utilizado para estimar a confiabilidade do software aplicando os dados de falhas gerados segundo um dos perfis operacionais.

Palavras-chave: Confiabilidade de software, teste de software, perfil operacional.

ABSTRACT

Several published studies indicate that the predictive ability of software reliability models using test coverage information observed is better than the predictive ability of models based on time domain. Hence, they have been proposed by researchers as an alternative to models based on time domain. However, to reach a conclusion about the superiority of this class of models it is necessary to evaluate their sensitivity to variations in operational profile. A desirable quality of software reliability models is that their predictive ability is not affected by variations in the operational profile of a program.

This thesis evaluates by means of an experiment, the sensitivity of two software reliability models based on code coverage information: "Binomial Model Based on Coverage" (BMBC) and "Infinite Failure Model Based on Coverage" (IFMBC). The experiment applies the models to data failures observed during the execution of a program according to three statistically distinct operational profiles. Additionally, six traditional software reliability models were used to estimate the reliability using the same software failure data. The models selected were: Musa-Okumoto, Musa Basic, Littlewood-Verrall Linear, Quadratic Littlewood-Verrall, Jelinski-Moranda and Geometric.

The results show that the predictive ability of the models BMBC and IFMBC is not affected by variations in the operational profile of the software. The same result was not observed for software reliability models based on time domain, i.e., changing the operational profile affects the predictive ability of these models. A result observed for example is that none of the traditional models could be used to estimate the software reliability using the fault data set generated by one of the operational profiles.

Keywords: software reliability, software test, operational profile.

SUMÁRIO

AGRADECIMENTOS

Alguém já deve ter dito que “escrever uma dissertação de Mestrado é uma experiência enriquecedora e de plena superação”. Além dos desafios inerentes ao projeto, não é fácil conduzir uma pesquisa e em paralelo manter outras atividades. É preciso acreditar e ter muita força de vontade. Em certo sentido, este é um trabalho bastante solitário, investimos muito tempo procurando artigos, depois lendo e tentando entender estes mesmos artigos, depois refletindo sobre como utilizar algumas das ideias no projeto...ah! o projeto!? Que projeto? E o tempo que passamos buscando um projeto? É muito tempo pensando, refletindo, planejando, programando, testando etc. Parece interminável. Estes períodos de solidão só são superados com o apoio de pessoas que direta ou indiretamente nos suportam. E utilizo aqui o verbo “suportar” no sentido de dar apoio e de aguentar. Mesmo correndo o risco de esquecer alguns nomes quero agradecer às pessoas que sempre estiveram próximas.

Em primeiro lugar ao meu pai, Guisolfo, e à minha mãe Neusa, que infelizmente não está mais aqui para ver este momento. Saudades.

À minha esposa Cláudia e meus filhos Caio e Júlia que sempre entenderam minhas ausências. Amo vocês!

À minha avó, Dona Maria, que me levava para a escola mesmo nos dias em que chovia muito em Caraguatatuba.

Aos meus irmãos Rogério e Elizabete.

Aos tio Emílio e tia Tereza.

Aos amigos de longa data Borjão, Lú, Bicudo, Sassá, Bela, Azul e Soninha.

Ao Paulo e Regina Akiyama.

Aos colegas professores da Faculdade Metrocamp: Carla, Suzana, Kesede, Portella, Oswaldo, Armando.

Ao pessoal do CTI: Clenio e Miguel.

Ao Professor Doutor Marcos L. Chaim pela disponibilidade em ajudar com a POKE-TOOL.

A todos que, direta ou indiretamente, colaboraram para que este trabalho de pesquisa fosse realizado.

Em especial, quero agradecer aos meus orientadores Professor Doutor Mario Jino e Doutor Adalberto Nobiato Crespo. Obrigado pela oportunidade, experiência, colaboração e paciência. Obrigado pela presença. Vocês estiveram presente em todos os momentos que foram necessários.

LISTA DE FIGURAS

LISTA DE TABELAS

"All truths are easy to understand once they are discovered;
the point is to discover them."
Galileo Galilei

CAPÍTULO 1

INTRODUÇÃO

Este capítulo contextualiza a importância do estudo da confiabilidade de software; destaca os primeiros modelos de previsão da confiabilidade de software, que utilizam o tempo de falhas em sua forma funcional; apresenta, rapidamente, uma crítica a esses modelos, ditos tradicionais, e apresenta outra abordagem, onde os modelos probabilísticos utilizam a cobertura de código em sua forma funcional. O capítulo finaliza apresentando os objetivos desta dissertação e como ela está organizada.

Visão Geral sobre Confiabilidade de Software

O tamanho e a complexidade das aplicações de software têm crescido muito ao longo dos anos. A alta demanda por software de apoio às atividades econômicas e científicas tem resultado no desenvolvimento de aplicações cada vez mais complexas. A velocidade do crescimento desta complexidade é, por vezes, maior do que a velocidade do desenvolvimento da tecnologia para construí-los. Dados indicam que o tamanho das aplicações tem crescido exponencialmente nos últimos quarenta anos. Estima-se que o tamanho das aplicações cresça a uma taxa de dez vezes a cada cinco anos [Shoo84], [Lit00] e [Lyu07].

Software permeia o dia a dia do mundo atual. Existem aplicações de software em elevadores, nos aparelhos de TV, nos automóveis, em aparelhos celulares, sem contar as aplicações de missão crítica já conhecidas, tais como software de controle de usinas nucleares, software de apoio à navegação em aviões, software embutido em mísseis, radares, etc. Estima-se que cerca de 9% do custo de fabricação de um carro médio seja devido ao software e esta estimativa deve aumentar para 15% nos próximos anos [Lom07]. O ônibus espacial da NASA voa com aproximadamente 500.000 linhas de código de

software a bordo e mais cerca de 3,5 milhões de linhas de código nos sistemas de controle em terra [Lyu96].

Esta forte dependência de aplicações de software pode levar a situações de grandes perdas financeiras e até mesmo colocar em risco vidas humanas. A existência de diversos relatos publicados corrobora esta afirmação:

a) Perda da Mars Polar Orbiter: Em 1999, o satélite foi destruído devido a uma confusão no uso do sistema de medida; a equipe da terra fez uso do sistema inglês para calcular os parâmetros de inserção do dispositivo na atmosfera marciana enquanto os sistemas da sonda espacial realizavam cálculos no sistema métrico [Wood03];
b) Sistema de controle de bagagens do aeroporto de Denver, EUA: Problemas no desenvolvimento do sistema levaram ao atraso de mais de um ano na abertura do aeroporto causando um prejuízo de cerca de US$ 1,1 milhão/dia [Wood03];
c) Sistema de serviço de atendimento de ambulâncias de Londres: Em 26 de outubro de 1992, o sistema Computer Aided Dispatch do serviço de ambulâncias de Londres (LASCAD) ficou inoperante logo após sua instalação, paralisando um dos maiores serviços de ambulância do mundo, com cerca de 5000 solicitações diárias para transporte de pacientes em situação de emergência [Lyu96];
d) Airbus A320 derrubado por um míssil lançado do USS Vincennes em 1998: O software confundiu o A320 com um F-14 [Huc08];

A lista anterior é apenas uma pequena amostra do que já foi divulgado, e reforça a importância do desenvolvimento e aplicação de métodos que permitam a criação de software com qualidade. A engenharia de software busca a qualidade aplicando métodos e medidas técnicas sólidas, conduzindo revisões técnicas formais e efetuando testes de software bem planejados [Press09].

A confiabilidade é uma das características de qualidade de software que têm sido extensivamente consideradas na análise da qualidade do software, pois, se um software não é confiável, pouco importa se outras características da qualidade são aceitáveis. Por outro lado, medir a confiabilidade de um software tem-se mostrado uma tarefa desafiadora [Dela07].

Um dos primeiros esforços para a modelagem da confiabilidade de software foi realizado por Hudson em 1967 [Hud67] utilizando um processo de Markov. Os primeiros modelos de confiabilidade foram desenvolvidos, independentemente, por Jelinski-Moranda [Jeli72] e Shooman [Shoo72]. O modelo de Jelinski-Moranda foi criado para o projeto Apollo, na McDonnell Douglas Astronautics Company. Ambos foram publicados em 1971 [Nik98]. Desde então, diversos modelos para avaliação do crescimento da confiabilidade de software têm sido propostos.

Alguns modelos são sempre citados na literatura sobre confiabilidade de software: Jelinski-Moranda (1972), Shooman (1972), Schick-Wolveron (1973), Musa (1975), Moranda (1975), Schneidewind (1975), Goel-Okumoto (1978), Goel-Okumoto (1979b), Musa (1979a) [Xie91], [Lyu96].

Abordagem Baseada na Cobertura de Código

A maioria dos modelos existentes sobre crescimento da confiabilidade de software é baseada no domínio de tempo, que utiliza o tempo decorrido entre falhas do software ou o número de falhas ocorridas durante um determinado período de tempo. Entretanto, um crescente número de pesquisadores tem argumentado que o tempo de execução não deve ser o único fator que afeta o comportamento de falhas de um software. A qualidade de predição dos modelos de confiabilidade de software pode ser melhorada se forem utilizados outros fatores importantes que a afetam. A cobertura de código alcançada durante o teste, por exemplo, tem sido utilizada como um indicador de completude e eficácia do teste de software e pode ser utilizada para melhorar a capacidade de predição dos modelos de confiabilidade de software tradicionais [An10], [Chen01], [Crespo96] e [Crespo97a].

Diversos estudos têm mostrado a existência de uma correlação entre o crescimento da cobertura, obtida durante os testes, e a confiabilidade do software. Esses estudos baseiam-se na suposição de que, quanto maior for a cobertura obtida, maior será a confiabilidade do software [Kris96]. Frate, Garg, Mathur e Pasquini realizaram um experimento utilizando cinco programas diferentes para investigar a relação entre confiabilidade do software e a cobertura dos elementos requeridos pelo critério de teste. Os resultados mostraram evidências da existência de correlação entre a confiabilidade do software e a cobertura [Frat95].

Objetivos

Os estudos realizados, as evidências empíricas e os resultados dos experimentos motivaram o desenvolvimento de modelos de confiabilidade nos quais a estrutura do software deve ser considerada no processo de modelagem. Nesta linha de pesquisa, Crespo et al. [Crespo97b], [Crespo08], [Crespo09] propuseram dois modelos de confiabilidade de software que utilizam a informação da cobertura medida durante os testes: a) Modelo Binomial Baseado em Cobertura (MBBC) e b) Modelo de Falhas Infinitas Baseado em Cobertura (MFIBC). Ambos utilizam em sua forma funcional a informação de cobertura dos elementos requeridos pelo critério de teste.

Crespo et al. [Crespo97b] realizou um experimento para avaliar a capacidade preditiva dos modelos, comparando as estimativas obtidas pelos modelos MBBC e MFIBC com as obtidas pelos modelos tradicionais baseados no domínio de tempo. Em todas as comparações, os modelos MBBC e MFIBC mostraram-se superiores aos modelos tradicionais.

O objetivo desta dissertação é dar continuidade à linha de pesquisa iniciada por Crespo, investigando a capacidade preditiva dos modelos MBBC e MFIBC em situações diferentes daquelas já avaliadas por ele. Outras variáveis relacionadas com a confiabilidade devem ser consideradas, como o perfil operacional do software. O perfil operacional de um software define a forma como o software é utilizado por um tipo de usuário em um determinado ambiente e está fortemente relacionado com a medida da confiabilidade do

software. Um software para edição de textos, por exemplo, possui diferentes perfis de uso: uma secretária de um escritório de advocacia o utiliza de maneira diferente de um engenheiro que precisa criar fórmulas e equações matemáticas.

Uma qualidade desejada para modelos de confiabilidade de software é que eles sejam robustos à variação do perfil operacional. Ser robusto à variação do perfil operacional significa que os modelos podem ser utilizados para estimar a confiabilidade do software sob diversas condições de uso, reduzindo a necessidade de utilização de diversos modelos diferentes.

Neste sentido, este trabalho visa a avaliar a sensibilidade dos modelos de confiabilidade de software MBBC e MFIBC sob diferentes perfis operacionais. Para ampliar a análise, modelos baseados no domínio de tempo também foram selecionados para o experimento. Assim, além de estudar o comportamento dos modelos MBBC e MFIBC em diferentes perfis operacionais, também será possível compará-los com os resultados obtidos pelos modelos tradicionais.

Organização da Dissertação

Este capítulo contextualizou o tema desta dissertação e destacou sua relevância no escopo do estudo sobre a confiabilidade de software. Foram apresentadas as justificativas de pesquisa sobre o tema, bem como o objetivo desta dissertação.

O Capítulo 2 apresenta conceitos básicos de confiabilidade de software: taxa de falhas, função valor médio, intensidade de falhas e tempo médio entre falhas; além disso, apresenta a classificação dos modelos de confiabilidade de software, aborda o processo de modelagem da confiabilidade de software e define perfil operacional.

O Capítulo 3 explica a utilização da informação da cobertura dos elementos requeridos por critérios de teste em modelos de confiabilidade de software; além disso apresenta dois modelos que utilizam a informação de cobertura observada nos testes como parâmetro do modelo – o Modelo Binomial Baseado em Cobertura (MBBC) e o Modelo de Falhas Infinitas Baseado em Cobertura (MFIBC).

O Capítulo 4 descreve o experimento realizado para avaliar a sensibilidade dos modelos baseado na informação de cobertura sob três perfis operacionais diferentes. Inicialmente, são descritos os procedimentos realizados em cada etapa do experimento e a seguir, são apresentados os resultados obtidos.

No Capítulo 5, são analisados os resultados de cada etapa, para cada perfil e, em seguida, no Capítulo 6, são apresentadas as conclusões gerais do experimento e sugestões para trabalhos futuros.

Todos os programas e "scripts" desenvolvidos para o experimento são apresentados no Apêndice A. Foram utilizados programas em Bash Linux, para automatizar algumas etapas do experimento; rotinas em Matlab para estimar parâmetros dos modelos por métodos de otimização matemática; e "scripts" para a ferramenta POKE-TOOL, utilizada para executar o programa com os dados de teste e medir a cobertura dos elementos requeridos pelos critérios de teste.

CAPÍTULO 2

CONCEITOS BÁSICOS

Este capítulo tem a finalidade de apresentar os conceitos necessários ao entendimento da teoria da confiabilidade de software.

Confiabilidade de software é definida como a probabilidade de execução sem falhas de um programa em um determinado intervalo de tempo, em um dado ambiente. A confiabilidade de software é um dos atributos da qualidade de software, que inclui outras características, tais como funcionalidade, eficiência, portabilidade, usabilidade e manutenibilidade [ISO9126]. Entretanto, a confiabilidade de software é aceita como a característica mais importante da qualidade de software, uma vez que quantifica falhas de software.

A medida da confiabilidade de software é uma abordagem analítica para a quantificação da qualidade, do ponto de vista do usuário. Dentre todas as características da qualidade de software ela é a única que pode ser medida e estimada utilizando-se dados históricos, qualificando, assim, as falhas do software.

A confiabilidade representa a qualidade do ponto de vista do usuário. Sabe-se que uma confiabilidade de 100%, mesmo para programas de baixo nível de complexidade, é praticamente impossível de ser obtida. Do ponto de vista das organizações que desenvolvem software, a confiabilidade é uma referência para a avaliação do software. Nesse contexto, a indústria analisa o software para garantir que o produto atingiu um certo nível de confiabilidade, como um critério para sua liberação. Para isso, o teste é conduzido tanto para remover defeitos como para determinar o nível de confiabilidade do software.

De uma maneira geral, um sistema de software é dito confiável se desempenha corretamente suas funções especificadas por um longo período de execução e em uma variedade de ambientes operacionais. Essencialmente, existem três maneiras de se alcançar um alto nível de confiabilidade:

- Evitar a introdução de defeitos durante o projeto e o desenvolvimento do software.
- Fazer uso de estruturas tolerantes a defeitos.
- Remover defeitos durante a fase de testes e depuração.

As duas primeiras maneiras enquadram-se na abordagem construtiva: consistem em desenvolver software utilizando os métodos, as técnicas e as ferramentas disponíveis. Finalmente, a confiabilidade do software pode ser melhorada pelo teste e depuração, abordagem em que o software é submetido a um intenso processo dinâmico que objetiva a detecção e a remoção dos defeitos restantes. Durante os testes são coletados os dados de falhas, que compreendem: a) contagem de falhas por unidade de tempo e b) tempo médio entre falhas. Estes dados são utilizados em modelos probabilísticos para estimação e previsão da confiabilidade do software

A ocorrência de falhas em um software é um evento totalmente imprevisível e, desse modo, é considerado um processo aleatório [Hoel78]. Por isso, o estudo da confiabilidade de um software resume-se em aplicar a teoria da probabilidade na modelagem do processo de falhas do software e na predição da probabilidade de ocorrência de uma falha.

Para se atribuir uma confiabilidade a um software ele deve ser examinado em termos de suas falhas. À medida que os defeitos são detectados e removidos, sem a inserção de novos defeitos, o software passa a ter uma menor intensidade de falhas e, por consequência, há um aumento na confiabilidade.

Existem quatro formas gerais de se caracterizar a ocorrência de falhas no software em função do tempo [Goel85], [Musa98]:

- Tempo para a ocorrência da falha.
- Intervalo de tempo entre falhas.
- Falhas acumuladas até um determinado momento no tempo.
- Número de falhas num intervalo de tempo.

As quatro formas acima representam variáveis aleatórias, ou seja, não é possível saber, com antecedência, qual será seu exato valor. Estimativas da confiabilidade de software são, usualmente, expressadas em relação à variável tempo, embora possam ser utilizadas outras variáveis [Musa98].

A probabilidade de que ocorra uma falha durante um tempo de execução *t* é dada pela seguinte equação:

$$F(t) = P[0 \leq T \leq t] = \int_0^t f(x)\,dx$$

onde f(t) é a função densidade de probabilidade (densidade de falhas) e F(t) é a função distribuição de probabilidade acumulada (probabilidade de falhas).

Da expressão acima conclui-se que a probabilidade de que não ocorram falhas no software até o tempo *t* é dada por:

$$1 - F(t) = 1 - P[0 \leq T \leq t] = P[T > t]$$

2.1 Função Confiabilidade

A função confiabilidade ou função sobrevivência, R(t), definida como a probabilidade de operação livre de falhas de um software, em um tempo e ambiente especificados [Dela07], é dada por:

$$R(t) = P[T > t] = 1 - F(t) = \int_{t}^{+\infty} \mathrm{f(x)\,dx}$$

As falhas em um software podem ser expressas por diversas funções, como: função taxa de falhas, função de falhas acumuladas, função intensidade de falhas, tempo médio entre falhas (MTBF) e tempo médio para a próxima falha (MTTF). Para determinar o comportamento de falhas no software basta observar o comportamento de uma destas funções. Alguns modelos de confiabilidade de software são determinados por suposições no comportamento da taxa de falhas, outros por suposições na intensidade de falhas.

2.2 Função Taxa de Falhas

A taxa de falhas é utilizada para descrever o processo de falhas de um software. É definida como a probabilidade de que uma falha por unidade de tempo ocorra num intervalo [t, t + Δt], dado que o sistema não falhou até o tempo t. Na prática, a taxa de falhas é a razão entre o incremento do número de falhas e o incremento de tempo correspondente [Crespo97b].

Assim, usando probabilidade condicional, tem-se que:

$$\textit{Taxa de Falhas} \equiv \frac{\mathrm{P[t \le T \le t+\Delta t \mid T > t]}}{\Delta \mathrm{t}} = \frac{\mathrm{P[t \le T \le t+\Delta t]}}{\Delta \mathrm{t P[T > t]}} = \frac{\mathrm{F(t+\Delta t) - F(t)}}{\Delta \mathrm{t R(t)}}$$

A taxa de falhas instantânea, Z(t), também denominada taxa de risco associada à variável aleatória T, é definida como:

$$Z(t) = \lim_{\Delta t \to 0} \frac{F(t+\Delta t) - F(t)}{\Delta t R(t)} = \frac{f(t)}{R(t)}$$

Ou seja, a taxa de risco é definida como o limite da taxa de falhas quando o intervalo Δt tende a zero ($\Delta t \rightarrow 0$).

A taxa de risco é uma taxa de falhas instantânea no tempo t, dado que o sistema não falhou até o tempo t. Embora haja uma pequena diferença entre taxa de falhas e taxa de risco, normalmente se usa Z(t) como a taxa de falhas [Lyu96].

A taxa de falhas se altera ao longo do tempo de vida do sistema. A Figura 1 ilustra o comportamento da taxa de falhas de alguns sistemas.

A ***Região I***, conhecida como fase de depuração, representa as falhas iniciais do sistema. Nesta região a taxa de falhas tende a decrescer com o tempo.

A ***Região II***, conhecida como período de vida útil do sistema ou fase de operação, representa as falhas causadas por eventos aleatórios ou por condições de uso intenso do sistema. Nesta região a taxa de falhas permanece constante com o tempo.

A ***Região III*** representa a fase de desgaste do sistema que é caracterizada pelo crescimento na taxa de falhas em função do tempo.

A confiabilidade de software é semelhante à confiabilidade de hardware no sentido em que ambas são processos probabilísticos e podem ser descritas por distribuições de probabilidades. Contudo, a confiabilidade de software é diferente da confiabilidade de hardware no sentido que o software não se desgasta com o tempo, ou seja, a confiabilidade não decresce com o tempo. Consequentemente, a Região III não se aplica à confiabilidade de software. No software, geralmente a confiabilidade cresce na fase de teste e na fase de operação, desde que as falhas sejam removidas quando detectadas. No entanto, pode

ocorrer um decréscimo na confiabilidade devido a alterações abruptas no ambiente operacional ou modificações incorretas na manutenção.

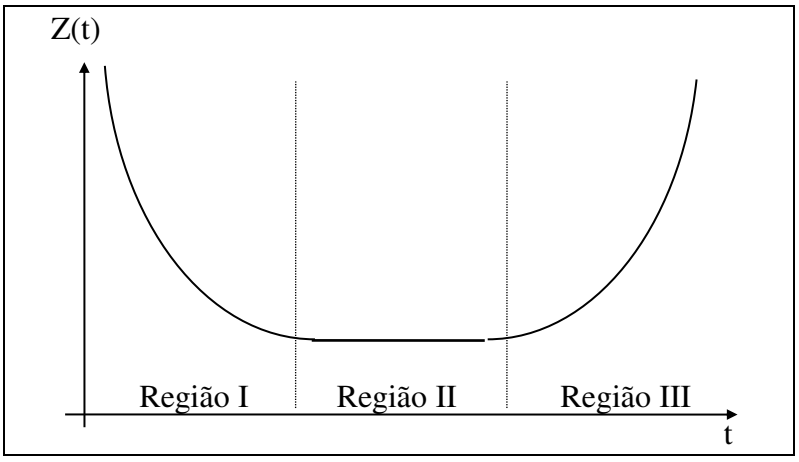

Figura 1 - Taxa de falhas de um sistema

Diferentemente dos defeitos de hardware que, na maioria das vezes, são defeitos físicos, os defeitos de software são defeitos de projeto, difíceis de se visualizar, classificar, detectar e remover. Como resultado, a confiabilidade de software é muito mais difícil de medir e analisar. Normalmente, a teoria de confiabilidade de hardware está fundamentada na análise de processos estacionários porque somente os defeitos físicos são considerados. Pressupõe-se que o projeto e a fabricação geram produtos sem defeitos. Contudo, com o crescimento da complexidade dos sistemas e a introdução de defeitos no projeto, a teoria de confiabilidade de software baseada em processos estacionários torna-se inconveniente. Isto torna a confiabilidade de software um problema desafiador que requer o emprego de vários métodos.

A taxa de falhas ideal do software é decrescente, e a Figura 2 dá uma ideia de seu comportamento em função do tempo pressupondo que os defeitos geradores de falhas são corrigidos sem inserir novos defeitos. Observando-se o comportamento da taxa de falhas em algum ponto, por meio de evidência estatística, é possível prever o comportamento da taxa num tempo futuro. Com isso, os modelos de confiabilidade de software podem prever

o tempo extra necessário de teste do software até que se atinja o objetivo especificado - a taxa de falhas desejada. Pode-se também predizer a confiabilidade ao término do teste.

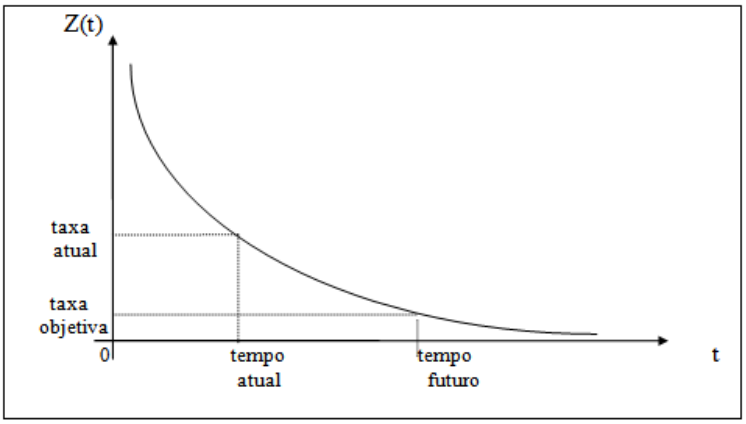

Figura 2 - Taxa de falhas do software

Quando se considera crescimento da confiabilidade, uma medida usual de confiabilidade é a confiabilidade condicional [Musa87]. Dado que o sistema teve *n-1* falhas, a confiabilidade condicional é a função de sobrevivência associada à *n-ésima* falha do sistema. A confiabilidade condicional é de interesse quando o sistema está em fase de desenvolvimento, onde se observa o tempo para a próxima falha. Quando o sistema está liberado e em fase operacional, o interesse passa a ser no intervalo de tempo livre de falhas e, neste caso, os instantes de falhas não são necessariamente condicionados às falhas anteriores. Neste caso, o interesse é a confiabilidade num dado intervalo de tempo, independentemente do número de falhas ocorridas anteriormente.

2.3 Função de Falhas Acumuladas ou Função Valor Médio

A função de falhas acumuladas, também denominada função valor médio, denota a média de falhas acumuladas associada a um ponto no tempo.

A função valor médio é uma maneira alternativa de representar o processo de falhas no software e o comportamento futuro pode ser predito com uma análise estatística sobre o

comportamento desta curva de crescimento. Teoricamente, o número de falhas acumuladas até o tempo *t* pode ser representado por uma variável aleatória M(t) com um valor médio μ(t). Isto é, μ(t) representa a função valor médio do processo aleatório.

Desta forma, tem-se que:

$$\mu(t) = E[\, M(t) \,]$$

A Figura 3 representa um comportamento típico da função valor médio.

O processo aleatório pode ser completamente especificado assumindo-se uma distribuição de probabilidades para a variável aleatória M(t), para algum *t*.

Uma vez que M(t) assume somente valores inteiros, as correspondentes distribuições de probabilidade devem ser discretas. As distribuições de probabilidade de Poisson e Binomial [Hoel78] são muito utilizadas para descrever o processo aleatório M(t).

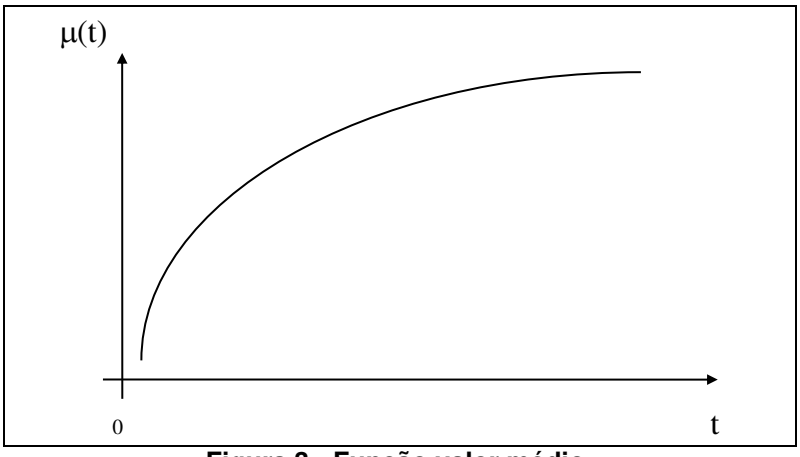

Figura 3 - Função valor médio

Assim, pode-se calcular a probabilidade de que o processo tenha um certo número *k* de falhas acumuladas num determinado tempo *t*, da seguinte forma:

$$P[M(t) = k] = \frac{[\mu(t)]^k}{k!} \exp[-\mu(t)]$$

se a distribuição de probabilidade assumida for de Poisson, ou então:

$$P[M(t) = k] = \binom{N}{k} [p(t)]^k [1 - p(t)]^{N-k}$$

onde N é o número de defeitos no software e p(t) é a probabilidade de falhas, se a distribuição de probabilidades do número de falhas assumida for a binomial. Neste caso, tem-se que:

$$E[M(t)] = Np(t).$$

A função valor médio $\mu(t)$ deve ser não decrescente com *t*. Alguns modelos de confiabilidade de software admitem uma função valor médio $\mu(t)$ limitada. Outros modelos admitem a suposição de que na remoção de um defeito existe a possibilidade da introdução de novos defeitos e, assim, admitem que a função valor médio $\mu(t)$ seja ilimitada.

2.4 Função Intensidade de Falhas

A função intensidade de falhas $\lambda(t)$ representa a taxa de variação instantânea da função valor médio, isto é, representa o número de falhas por unidade de tempo. Por exemplo, pode-se dizer que houve 0,01 falhas/hora ou então que houve uma falha a cada 100 horas. Esta função representa a derivada da função valor médio e é um valor instantâneo. A Figura 4 ilustra um comportamento típico dessa função.

A função intensidade de falhas, $\lambda(t)$, é então obtida a partir da função valor médio como:

$$\lambda(t) = \frac{d\,\mu(t)}{dt} = \frac{d(E[M(t)])}{dt}$$

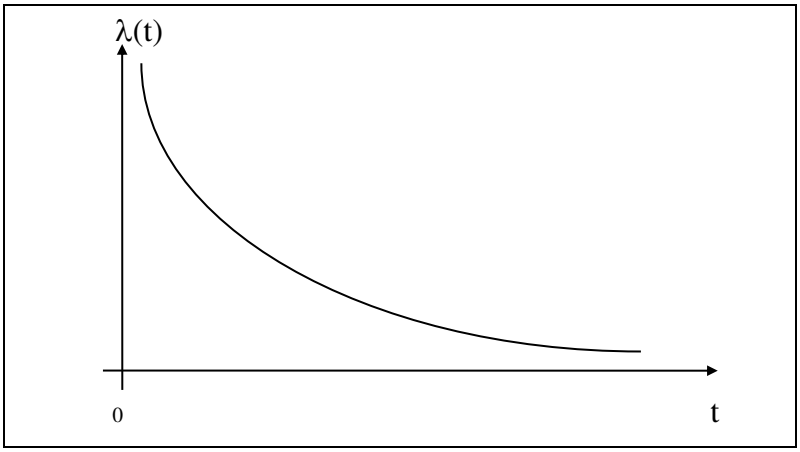

Figura 4 - Função intensidade de falhas

Para que haja crescimento de confiabilidade é necessário que $\frac{d\lambda(t)}{dt} < 0$, para qualquer $t \geq t_0$, para algum t_0.

2.5 Tempo Médio para Falhas

A função tempo médio para falhas, MTTF, representa o tempo esperado para a ocorrência da próxima falha, ou seja, é o tempo durante o qual o software funciona sem falhas. Como T é a variável aleatória que representa o tempo para uma falha ocorrer, então tem-se que:

$$MTTF = E[T] = \int_0^{\infty} tf(t)\,dt$$

onde f(t) é a função densidade de probabilidade do tempo de falha.

2.6 Modelos de Confiabilidade de Software

Para se modelar a confiabilidade de software, é necessário considerar os principais fatores que afetam a confiabilidade:

- **Introdução de defeitos**: depende das características do código desenvolvido e das características do processo de desenvolvimento, o que inclui técnicas, ferramentas e nível de experiência da equipe de desenvolvimento.
- **Remoção de defeitos**: depende do tempo de operação do software, do perfil operacional e da qualidade da atividade de reparos.
- **Ambiente no qual o software é executado**: depende diretamente do perfil operacional do usuário.

Por causa da natureza probabilística de alguns destes fatores e por operarem no tempo, os modelos de confiabilidade de software são geralmente formulados em termos de processos aleatórios. Em termos gerais, os modelos se distinguem pela distribuição de probabilidade do tempo entre falhas ou pela distribuição do número de falhas observadas e, também, pela natureza da variação do processo aleatório no tempo.

Um modelo matemático é chamado de modelo de confiabilidade de software se for utilizado para obter uma medida da confiabilidade. Todos os modelos matemáticos de confiabilidade de software são de natureza probabilística e, assim, tentam, de algum modo, especificar a probabilidade de ocorrência de falhas do software. O objetivo final é quantificar a confiabilidade do software de uma maneira tão precisa quanto possível.

Um modelo de confiabilidade de software especifica a forma funcional da dependência do processo de falhas em relação aos fatores mencionados, isto é, faz uma descrição probabilística precisa da confiabilidade baseada nas suposições *a priori* sobre os fatores que afetam a confiabilidade e também baseada nos resultados obtidos pelos dados experimentais. Várias são as formas matemáticas para se descrever o processo de falhas.

Uma forma específica pode ser determinada de uma maneira geral ao se estabelecerem os valores dos parâmetros por:

- **Estimação**: aplicação de procedimentos de inferência estatística aplicados aos dados de falhas.
- **Predição**: determinação das propriedades do software e do processo de desenvolvimento (pode ser feita antes da execução do software).

Sempre existe uma incerteza na determinação de uma forma específica; por isso, os parâmetros são, em geral, expressados em termos de intervalos de confiança. Uma vez que a forma específica tenha sido determinada, algumas características do processo de falhas podem ser identificadas. Alguns modelos apresentam uma expressão analítica para essas características, que são:

- Número médio de falhas observadas em algum ponto no tempo.
- Número médio de falhas em um intervalo de tempo.
- Intensidade das falhas em algum ponto no tempo.
- Distribuição de probabilidade do tempo entre falhas.

Segundo Musa [Musa98], para ser considerado um bom modelo de confiabilidade de software, deve-se ter as seguintes características importantes:

- Fornecer uma boa projeção do comportamento de falhas;
- Possuir expressões analíticas para quantidades úteis;
- Ter uma forma funcional simples;
- Ser amplamente aplicável;
- Estar baseado em suposições concretas.

A predição do comportamento futuro pressupõe que os valores dos parâmetros do modelo não se alteram nos próximos períodos. De uma forma geral, os modelos de confiabilidade de software estão baseados numa execução estável do software e num ambiente constante.

As medidas de confiabilidade de software podem ser de grande valor para a engenharia de software, e podem ser utilizadas para [Musa87]:

- Avaliar quantitativamente as tecnologias de engenharia de software;
- Avaliar o status do desenvolvimento de software durante as fases do projeto;
- Monitorar o desempenho operacional do software e controlar novas funcionalidades adicionadas e alterações de projeto realizadas no software;
- Um entendimento quantitativo da qualidade do software e os diversos fatores que influenciam e são afetados por ela, fornecendo subsídios para melhoria do produto de software e seus processos.

Por fim, deve-se destacar que muitos dos modelos existentes na literatura são testados com dados simulados ou com dados reais que, na maioria das vezes, não foram coletados para tal propósito. O resultado é que algumas suposições básicas desses modelos ou abordagens são violadas. Devido às controvérsias sobre qual é o melhor modelo, e por causa da incerteza sobre o desempenho das abordagens de confiabilidade de software, deve ser enfatizado que, dentre os modelos existentes, recomenda-se aplicar aquele que melhor se ajusta aos dados.

2.7 Classificação dos Modelos de Confiabilidade de Software

Na literatura, podem ser encontradas três abordagens distintas para a modelagem da confiabilidade de software, que resultam na classificação dos modelos aplicados em:

Modelos de implante de defeitos: essa abordagem, proposta primeiramente por Mills [Mill72], envolve implantar, em um dado programa, um certo número de defeitos. A suposição é que a distribuição dos defeitos implantados é a mesma dos defeitos inerentes ao programa. Assim, o programa é entregue a uma equipe de teste para validação e verificação. No procedimento de teste, alguns dos defeitos descobertos são defeitos implantados e outros são defeitos inerentes ao programa. Usando a contagem desses

defeitos, o número total de defeitos inerentes pode ser estimado. Em particular, suponha-se que 100 defeitos foram implantados num programa. Após um período de teste, 20 defeitos implantados e 10 defeitos inerentes foram detectados. Os defeitos implantados que foram descobertos representam 20% do total. Assume-se que os 10 defeitos inerentes descobertos também representam 20% do total. Portanto, o número total de defeitos inerentes do programa será estimado em 50.

Modelos baseados no domínio de dados: esta abordagem inclui procedimentos que estimam a confiabilidade corrente do software baseando-se estritamente no número observado de execuções com sucesso, em relação ao número total de execuções do programa. Nesta categoria, incluem-se também os procedimentos que usam casos de teste selecionados de acordo com a distribuição de probabilidades do perfil operacional de uso do programa. O domínio de entrada do software é dividido em classes e as probabilidades de cada classe são fixadas de acordo com o perfil de uso do programa. Como exemplo, pode-se supor um programa que possui como entrada apenas números inteiros positivos. Pode-se considerar também que os dados de entrada foram divididos em quatro classes, da seguinte forma: 25% dos dados de entrada estão na classe [0 – 1500], 35% na classe [1501 – 2500], 30% na classe [2501 – 3500] e 10% na classe [3501 ou superior]. Assim, numa amostra aleatória de 200 casos de teste, 25% deveriam ser provenientes da classe [0 – 1500], 35% da classe [1501 – 2500] e assim sucessivamente. Isso resulta em 50 casos de testes para a classe [0 – 1500], isto é, 25% de 200, 70 casos de testes para a classe [1501 – 2500], isto é, 35% de 200, e assim por diante. A confiabilidade do programa será obtida dividindo-se o número total de execuções com sucesso pelo número total de execuções, que, neste exemplo, será duzentos.

De forma geral, se N entradas são selecionadas de acordo com perfil operacional e S são as execuções com sucesso, isto é, não resultaram em falhas, então uma estimativa da confiabilidade é dada por:

$$\widehat{R} = \frac{S}{N}$$

Modelos baseados no domínio do tempo: esta abordagem utiliza o tempo de ocorrência entre falhas ou o número de falhas ocorridas num intervalo de tempo para se modelar o processo de falhas no software. Em geral, os modelos podem ser utilizados para predizer o tempo até a ocorrência da próxima falha ou o número esperado de falhas no próximo intervalo de tempo. Originalmente esses modelos foram motivados pelos conceitos sobre confiabilidade de hardware e, assim, muitos termos usados em confiabilidade de hardware são também utilizados em confiabilidade de software.

Nas últimas décadas, seguindo esta abordagem, muitos modelos foram propostos e várias extensões foram sugeridas. No entanto, um modelo pode ajustar-se bem a um dado conjunto de falhas e não a outro. A falta de um modelo universal implica a utilização de vários modelos para se analisar o ajuste a um conjunto de dados de falhas de um software. Deve-se utilizar técnicas estatísticas para se determinar o modelo que melhor explica os dados. Lembrando que o perfil operacional de um software pode mudar, um modelo de confiabilidade anteriormente selecionado pode não ser adequado quando um novo perfil de uso é adotado pelos usuários.

A classificação de Musa e Okumoto [Musa84] é tida como uma das mais completas e aceitas pelos pesquisadores. Eles classificaram os modelos de acordo com cinco atributos: domínio do tempo, categoria finita ou infinita, tipo, classe (somente para modelos de categoria finita), família (somente para modelos de categoria infinita).

As Tabelas 1 e 2 exibem os principais modelos de confiabilidade de acordo com a classificação de Musa e Okumoto.

Tabela 1 - Modelos de confiabilidade de categoria de falhas finitas

	Tipo		
Classe	**Poisson**	**Binomial**	**Outros**
Exponencial	Musa (1975) Moranda (1975) Schneidewind (1975) Goel-Okumoto (1979b)	Jelinski-Moranda (1972) Shooman (1972)	Goel-Okumoto (1978) Musa (1979a) Keiller-Littlewood (1983)
Weibull	-	Schick-Wolveton (1973) Wagoner (1973)	-
Pareto	-	Littlewood (1978)	-
Gamma	Yamada-Ohba-Osaki (1983)	-	-

Tipo: Distribuição do número de falhas observadas.
Classe: Forma funcional da intensidade de falhas em função do tempo.

Tabela 2 - Modelos de confiabilidade de categoria de falhas infinitas

	Tipo			
Família	**T1**	**T2**	**T3**	**Poisson**
Geométrica	Moranda (1975)	-	-	-
Linear Inversa		Littlewood-Verral (1973)		
Polinomial Inversa	-	-	Littlewood-Verral (1973)	-
Potência	-	-	-	Crow (1974)

Tipo: Distribuição do número de falhas observadas.
T1, T2, T3: outras distribuições.
Família: Forma funcional da intensidade de falhas em função do número esperado de falhas observadas.

2.8 Cálculo da Confiabilidade de Software

A seleção de um modelo de confiabilidade pode seguir um procedimento geral, compreendendo os passos descritos a seguir:

1. Teste do software: esta fase inicial não é trivial nem a mais rápida, pois trata da realização dos testes do software. Requer o planejamento dos testes, a realização dos testes e o registro das falhas.
2. Coleta dos dados: nesta fase, os resultados dos testes são coletados, anotados e armazenados. Normalmente, são anotados dados como: o tempo decorrido entre falhas ou o número de falhas num determinado período de tempo ou ainda a cobertura de código.
3. Seleção de modelos de confiabilidade: nesta fase, verificam-se os modelos que satisfazem as condições em que os testes foram realizados e estimam-se os parâmetros dos modelos.
4. Verificação dos modelos de confiabilidade: entre os modelos selecionados no passo anterior, verifica-se a adequação dos modelos com o uso de testes estatísticos.
5. Validação do modelo de confiabilidade: entre os modelos verificados, seleciona-se o modelo que melhor se ajusta aos dados, com base no valor crítico do teste estatístico utilizado no passo anterior.
6. Utilização do modelo de confiabilidade: nesta fase, utiliza-se o modelo de confiabilidade selecionado para a tomada de decisões sobre o software. Podem ser calculadas estimativas da confiabilidade corrente do software ou estimativas do tempo para a ocorrência das próximas falhas.

Em geral, a seleção de um modelo de confiabilidade do software requer o uso de uma ferramenta. Existem algumas ferramentas de domínio público que podem ser utilizadas para a seleção dos modelos e a estimação de seus parâmetros. Uma ferramenta bastante conhecida é a SMERFS (Statistical Modeling and Estimation of Reliability Functions for

Systems). O projeto SMERFS iniciou-se em 1981 e foi patrocinado pelo departamento de pesquisa naval do governos dos Estados Unidos (NSWC – Naval Surface Weapons Center). Atualmente é utilizada em diversos projetos de missão crítica da NASA, a agência espacial americana [Smer].

Outra ferramenta bastante conhecida é a CASRE (Computer Aided Software Reliability Estimation), desenvolvida em 1993 pelo Jet Propulsion Laboratories (JPL) a pedido da força aérea dos Estados Unidos. Caracterizada pela facilidade de uso e interfaces amigáveis para seus usuários, incorpora quase todos os modelos implementados no SMERFS [Casr].

Outras ferramentas são: AT&T Software Reliability Engineering Toolkit [Att], Software Reliability Estimation and Prediction Tool (SREPT) [Srep] e Software Reliability Program (SoRel) [Sore].

2.9 Perfil Operacional

A confiabilidade de um software depende de como ele será utilizado por seus usuários. Para se chegar a uma boa estimativa da confiabilidade de um software, é necessário levar em consideração a forma como ele é utilizado em campo. O perfil operacional é uma especificação quantitativa sobre como o software é utilizado e, portanto, é essencial para o estudo da confiabilidade de software. Ele é representado por um conjunto disjunto de alternativas chamadas elementos, cada uma com determinada probabilidade de ocorrência. Por exemplo, se um elemento "A" ocorre em 60% do tempo, "B" ocorre em 30% e "C" ocorre em 10%, então um perfil operacional é: {{"A"; 0,6}, {"B"; 0,3}, {"C"; 0,1}}. O soma total das probabilidades deve ser 1 (100%).

A Tabela 3 exibe outro exemplo de perfil operacional para um software de controle de um caixa eletrônico. Ela mostra as operações que são realizadas no software e sua

frequência de execução. A alteração dessas frequências define outro perfil operacional para o mesmo software.

A importância do perfil operacional para o estudo da confiabilidade de software reside no fato de que a maioria dos modelos de confiabilidade baseia-se na suposição de que o software sob avaliação é testado nas mesmas condições do ambiente operacional do usuário. Isto é, pressupõe-se que o perfil de teste é tão próximo quanto possível do perfil operacional do usuário.

Tabela 3 - Exemplo de perfil operacional

Operação	Probabilidade de ocorrência
Sacar dinheiro	0,53
Verificar Depósitos	0,15
Fazer depósito	0,14
Solicitação de extrato	0,06
Transferência de valores	0,08
Realimentar caixa	0,02
Coletar depósitos	0,02

No entanto, o desenvolvimento de perfis operacionais pode não ser uma tarefa simples. Em alguns tipos de aplicações, como centrais telefônicas, não seria difícil detectar a distribuição de probabilidades de chamadas de cada prefixo. Por outro lado, softwares para controle de processos químicos e controle de usinas nucleares são ativados por uma complexa combinação de eventos cuja frequência dificilmente pode ser estimada *a priori*.

Outro fator que dificulta a determinação de um perfil operacional é que o software sofre alterações com o tempo, isto é, suas funcionalidades são modificadas ou novas funcionalidades são implementadas. Adicionalmente, os usuários podem mudar sua maneira de utilizar um software, na medida em que aprendem mais sobre ele e passam a explorar melhor suas funcionalidades.

Chen, Mathur e Rego [Chen94b] utilizaram a TERSE, uma ferramenta de estimação da confiabilidade, para modelar um programa como um grafo de arcos e nós associados com probabilidades de transição. Erros na estimação do perfil operacional são simulados ao se modificar a probabilidade de transição em um arco. Os autores concluíram que:

a. Imprecisões no perfil operacional podem resultar em erros na estimativa da confiabilidade;
b. São necessários modelos que sejam robustos em relação a erros no perfil operacional.

Musa [Musa93], [Musa94] faz uma abordagem analítica e define o perfil operacional como um conjunto de probabilidades de ocorrência das funções do software. Assim, a soma dessas probabilidades de ocorrência deve completar o valor 1 e um erro em uma dessas probabilidades provoca um erro em outras probabilidades. Dessa observação, o autor sugere que múltiplos erros no perfil operacional podem ter um efeito compensador, em vez de um efeito cumulativo, sobre a intensidade de falhas no software.

2.10 Limitações dos Modelos de Confiabilidade

Apesar do grande esforço de pesquisa na área de modelagem, os modelos não fornecem uma estimativa confiável da confiabilidade do software. As principais aplicações dos modelos têm se limitado ao suporte à gerência dos projetos e ao seu uso como critério de parada para a atividade de teste.

No procedimento para a modelagem da confiabilidade de software, o programa é executado várias vezes usando os dados de teste selecionados aleatoriamente, de acordo com o perfil operacional. Quando uma falha ocorre, para-se o teste e, em seguida, inicia-se o procedimento de detecção e remoção do defeito que causou a falha. Novamente, inicia-se o teste até que uma nova falha ocorra. Em alguns modelos se utilizam os resultados do teste para se estimar o número de defeitos restantes e obter a taxa de falhas em função deste

número. Em outros modelos, utiliza-se a sequência do tempo entre falhas para se medir a confiabilidade corrente e, também para estimar o crescimento da confiabilidade com a execução de novos dados de teste. A informação sobre a cobertura dos elementos requeridos geralmente não é utilizada, mesmo sabendo-se que a ocorrência de falhas, na maioria da vezes, está associada ao exercício de elementos requeridos pelo critério de teste.

Uma outra restrição aos modelos de confiabilidade de software está relacionada aos critérios de teste. Todo critério de teste possui um limite na sua capacidade de gerar dados de teste distintos dos dados já gerados. Quando um critério de teste atinge este limite, o prosseguimento do teste não exercita novos elementos requeridos pelo critério, fazendo com que o tempo entre falhas aumente consideravelmente. Logo, a estimativa da confiabilidade, produzida pelos modelos baseados no domínio do tempo, cresce sem que haja a remoção de algum novo defeito. Os testadores que não estiverem conscientes da redução da capacidade do critério de teste podem obter uma superestimação da confiabilidade.

O perfil operacional, conforme dito anteriormente, pode ser difícil de ser estimado, principalmente em software de controle de processos. Em alguns casos, um único perfil operacional de um software pode não ser suficiente para os mais diferentes usuários. Sem perfis operacionais adequados, as estimativas dos modelos certamente são pouco úteis.

2.11 Resumo

Este capítulo apresentou os conceitos básicos de confiabilidade de software; as formas funcionais utilizadas para caracterizar as falhas no tempo: função taxa de falhas, função valor médio ou falhas acumuladas, função intensidade de falhas e tempo médio entre falhas. Apresentou a classificação dos modelos de confiabilidade e o processo de modelagem da confiabilidade de software. Finalmente, introduziu o conceito de perfil operacional de software e as limitações dos modelos de confiabilidade de software.

CAPÍTULO 3

MODELOS BASEADOS EM COBERTURA DE TESTE

3.1 Cobertura do Teste

Diversos modelos para estimação e previsão da confiabilidade de software têm sido propostos desde que Jelinski e Moranda apresentaram seu modelo de classe exponencial em 1972 [Jeli72]. Por ter tido origem na confiabilidade de hardware, os primeiros modelos propostos baseiam-se na suposição de que a confiabilidade do software depende do tempo de teste. Estes modelos utilizam em sua forma funcional a informação sobre o tempo até a ocorrência de uma falha ou tempo entre falhas. Deve-se ajustar, portanto, o histórico de falhas detectadas durante os testes, realizados de acordo com um perfil operacional selecionado.

Uma crítica feita a estes modelos é que eles não consideram a estrutura interna do programa testado [Chen97] e, portanto, geram estimativas otimistas para a confiabilidade do software. A superestimação da confiabilidade do software pode ocorrer quando um critério de teste atinge sua capacidade de selecionar dados de teste distintos dos já gerados. Se a variável de controle do teste é o tempo entre falhas, então os valores observados podem ser cada vez maiores, ou seja, o tempo entre falhas aumenta.

Qualquer que seja a variável de controle do teste, é importante entender que o número de unidades desta variável aumenta consideravelmente com o decorrer do teste. Os modelos de confiabilidade de software que utilizam o tempo diretamente em sua forma funcional tendem a superestimar a confiabilidade do software [Crespo97b].

Um indicador proposto para avaliar a eficiência de um dado de teste é a cobertura de código observada com a execução deste dado de teste. Esta medida pode indicar para a equipe de projeto quanto teste adicional deve ser necessário para melhorar a qualidade do software. A cobertura de código é medida como uma fração do código executado pelo menos uma vez durante o teste. A medida da cobertura é realizada em relação à quantidade de elementos requeridos pelo critério de teste estrutural [Dela07] utilizado para selecionar os dados de teste.

Os critérios de teste estrutural mais conhecidos dividem-se em duas classes:

a) Critérios baseados em fluxo de controle, que utilizam apenas características de controle da execução do programa, como comandos ou desvios, para determinar quais estruturas são necessárias. Os critérios mais conhecidos desta classe são: Todos-Nós, Todos-Arcos (ou Todas-Arestas) e Todos-Caminhos.

b) Critérios baseados em fluxo de dados, que baseiam-se na análise do fluxo de dados como fonte de informação para derivar os requisitos de teste. Uma característica comum aos critérios dessa categoria é que eles requerem o teste das interações que envolvam as definições de variáveis e subsequentes referências a essas definições. Portanto, para a derivação de casos de teste, tais critérios baseiam-se nas associações entre a definição de uma variável e seus possíveis usos subsequentes. Os critérios mais conhecidos desta classe são os critérios Todos-Usos, Todos-Potenciais-Usos (PU), Todos-Potenciais-Usos/Du (PUDU), Todos-Potenciais-Du-Caminhos (PDU).

Diversos experimentos têm sido realizados para mostrar a existência de uma correlação entre o crescimento da cobertura dos elementos requeridos por critérios de teste e a confiabilidade de software [Garg95], [Lyu03], [Chen96] e [Karc96]. A Tabela 4 apresenta alguns experimentos realizados para investigar a relação entre cobertura de código e cobertura de defeitos e suas conclusões [Lyu07].

A medida da cobertura é uma informação objetiva sobre o quanto do software foi exercitado; esta abordagem pode ser mais eficiente do que a utilizada nos modelos tradicionais que se baseiam no tempo, melhorando as estimativas dos modelos [Mal02]. Sob esta suposição, diversos modelos baseados na informação de cobertura de teste têm sido propostos pelos pesquisadores.

Tabela 4 – Investigação da relação entre a cobertura de código e defeitos

Experimento	Observação
Horgan (1994) Frankl (1988) Rapps (1988)	Quanto maior a cobertura maior a confiabilidade do software e menor a taxa de defeitos [Hor94].
Chen (1992)	Observada correlação entre confiabilidade de software e cobertura de código.
Wong (1994)	A correlação entre a eficiência do teste e cobertura de comandos é maior do que a correlação entre a eficiência do teste e o tamanho do conjunto de dados de teste.
Frate (1995)	Um aumento na confiabilidade é acompanhado de pelo menos um aumento na medida de cobertura e uma diminuição da confiabilidade é acompanhada de pelo menos uma diminuição da medida de cobertura [Frat95].
Cai (2005)	Cobertura de código contribui com o volume total de defeitos descobertos [Cai05].

Na abordagem de Jalote e Muralidhara [Jal94], um programa é representado como um grafo de fluxo de dados, onde cada nó representa um módulo ou um conjunto executável de comandos. Com o prosseguimento do teste, diferentes nós são executados um diferente número de vezes. Uma função exponencial é proposta para calcular a confiabilidade do nó baseada no número de vezes que ele foi executado. A confiabilidade do sistema é obtida com base na confiabilidade calculada de todos os nós. Jalote e Muralidhara assumem que cada defeito manifesta-se aleatoriamente em cada nó; que a taxa de falhas diminui com o aumento da cobertura; os defeitos em cada nó são independentes e que o teste é aleatório e representa o perfil operacional.

Malaiya et al. [Mal02] propuseram um modelo logarítmico-exponencial que relaciona a medida de cobertura observada durantes os testes com a quantidade de defeitos removidos. Tendo por base a informação de cobertura, o modelo pode prever o tempo para a ocorrência da próxima falha.

Pham e Zhang [Pham03] formularam, a partir de um modelo geral de crescimento da confiabilidade baseado no processo de Poisson não-homogêneo (NHPP), um modelo que possui as seguintes suposições: a) a ocorrência de defeitos segue um processo de Poisson; b) a intensidade da taxa de defeitos no software, em um dado momento, é proporcional à quantidade de defeitos existentes no software, naquele momento; c) quando um defeito é encontrado, ele é corrigido imediatamente. O modelo pressupõe que a taxa de introdução de defeitos no software está linearmente relacionada com o tempo.

Inoue e Yamada [Inou08] propuseram um modelo tipo Weibull de falhas infinitas. Eles consideram que o tempo de teste pode ser descrito segundo dois fatores: a) o tempo de teste propriamente dito, relacionado ao tempo de calendário e b) o esforço de teste, que pode ser, por exemplo, a informação de cobertura de código observada ou o número de casos de testes executados. Estes dois fatores foram incorporados ao modelo de crescimento da confiabilidade.

3.2 Modelo Binomial Baseado em Cobertura (MBBC)

O processo de falhas de um modelo tipo binomial é caracterizado pelo comportamento da taxa de ocorrência de falhas no software. Musa et al. [Musa87] classificam estes modelos em “tipo binomial”, onde a probabilidade de ocorrência de falhas segue um modelo de distribuição binomial de probabilidades.

A taxa de falhas em um software é definida como o número de falhas por unidade de medida do teste do software. No modelo tipo binomial, a suposição básica é que a taxa

de falhas do software possui uma determinada forma funcional, caracterizando o modelo de confiabilidade do software.

No modelo proposto por Crespo [Crespo97b], [Crespo08], a variável tempo é substituída pelo número de dados de teste, com a suposição de que a aplicação de um dado de teste equivale a uma unidade de tempo de execução do software, isto é, a unidade de medida do teste do software será o dado de teste. Assim, a taxa de falhas no software será dada pelo número de falhas observadas por dado de teste.

A execução do software com certa quantidade de dados de teste implica a cobertura do código. A cobertura obtida dependerá do critério de teste utilizado. Assim, a taxa de falhas no software estará relacionada com a medida da cobertura do critério de teste.

Deve-se destacar que, no início dos testes, a taxa de falhas é alta e a cobertura baixa. No final dos testes a taxa de falhas é baixa e a cobertura alta. Ou seja, o complemento da cobertura também é uma variável relacionada diretamente com a taxa de falhas do software.

Suposições básicas do Modelo Binomial Baseado em Cobertura:

1. O software é testado nas mesmas condições em que é utilizado pelo usuário;
2. Cada defeito tem a mesma chance de ser detectado dentro de uma mesma classe de dificuldade;
3. O defeitos 1, 2, 3, ...,k detectados, respectivamente, em cada um dos intervalos $(0 ; n_1)$, $(n_1 ; n_2)$, $(n_2 ; n_3)$, ...$(n_{k-1} ; n_k)$ são independentes;
4. Existe um número N de defeitos no software no início do teste;
5. 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; e
6. Para um defeito "a", a taxa de falhas, $Z_a(n)$, tem a seguinte forma funcional:

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

onde n é o número de dados de teste aplicados na detecção do defeito "a" e α é o complemento da cobertura alcançada com a aplicação dos n dados de teste, ou seja, $0 \leq \alpha \leq 1$.

A Suposição 1 garante que as estimativas obtidas pelo modelo, usando-se os dados coletados no ambiente de teste, são válidas quando utilizadas no ambiente de operação do software.

A Suposição 2 garante que todos os defeitos têm as mesmas propriedades em suas distribuições.

A Suposição 3 permite que a função densidade de probabilidade conjunta, do método de máxima verossimilhança, seja obtida pela multiplicação das funções densidades de cada uma das variáveis aleatórias. Se a hipótese de independência for válida, o método da máxima verossimilhança garante que os estimadores sejam não viciados e robustos [Mood74].

A Suposição 4 indica que o número de falhas detectadas é finito e que, portanto, o modelo deve ser de categoria de falhas finitas, segundo a classificação de Musa et al. [Musa87]. Essa suposição está fundamentada no fato de que o processo de remoção de defeitos é perfeito. Isto é, nenhum novo defeito é inserido no software ao se remover os defeitos existentes.

A Suposição 5 é uma característica do procedimento do teste e indica que a cobertura deve ser medida a cada ocorrência de falha.

A Suposição 6 caracteriza a forma funcional da taxa de falhas por defeito no software. Observe-se que a variável aleatória "dados de teste" é do tipo discreta e a taxa de falhas, tal como definida na Suposição 6, é a taxa de falhas do modelo de Weibull, expressão 3.1, com o parâmetro $\beta = 1$.

$$f(t) = \alpha\beta t^{\alpha-1}\exp(-\beta t^{\alpha}), \text{ onde } t \geq 0,\ \alpha > 0 \text{ e } \beta > 0 \qquad (3.1)$$

Assim, na Suposição 6, está implícito que a variável aleatória número de dados de teste tem aproximadamente uma distribuição de Weibull. A suposição de que a aplicação de um dado de teste equivale a uma unidade tempo de execução do software possibilita essa aproximação. O modelo de Weibull é muito aplicado no estudo da confiabilidade pela amplitude de sua utilização. Esse modelo tem a vantagem de se ajustar a qualquer tipo de variação polinomial na taxa de falhas, de acordo com os valores dos parâmetros α e β. Isto é, o modelo se ajusta a uma taxa de falhas decrescente, constante ou crescente. Uma taxa de falhas com esses três tipos de comportamento é assumida em modelos de confiabilidade de hardware. No modelo que se propõe para a confiabilidade de software, a taxa de falhas decresce com o progresso do teste tendendo ao valor zero, de acordo com o crescimento da cobertura do critério utilizado no teste. A Figura 5 mostra o processo de teste do software para avaliação da confiabilidade.

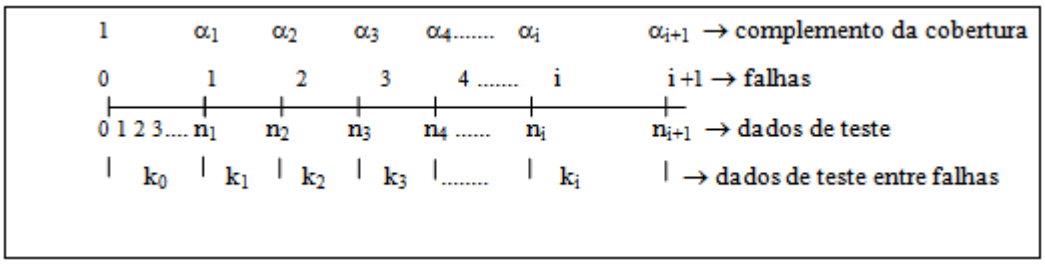

Figura 5 - Processo de teste do software

onde:

$n_0 = 0$

$n_1 = k_0$

$n_2 = n_1+k_1$

$n_3 = n_2+k_2$

$n_{i+1} = n_i+k$

e $\alpha_0 = 1$

observe-se que k_i é o número de dados de teste adicionais que serão aplicados para a ocorrência da (i+1)-ésima falha.

Portanto, conforme as Suposições 4 e 6 a taxa de falhas do software, condicionada aos defeitos restantes no mesmo é definida como:

$$Z(k_i \mid n_i) = [N - i].Z_a(n_i+k_i)$$

ou seja,

$$Z(k_i|n_i) = [N-i]\alpha_i(n_i+k_i)^{\alpha_i-1}$$

onde:

α_i: é o complemento da cobertura atingida com a aplicação dos n_i dados de teste, $0 \leq \alpha_i \leq 1$.

N: é o número de defeitos no software no início do teste.

i: é a ordem de falhas ocorridas, portanto i = 0, 1, 2, 3, 4, ..., N.

A Figura 6 mostra o comportamento da função taxa de falhas do software.

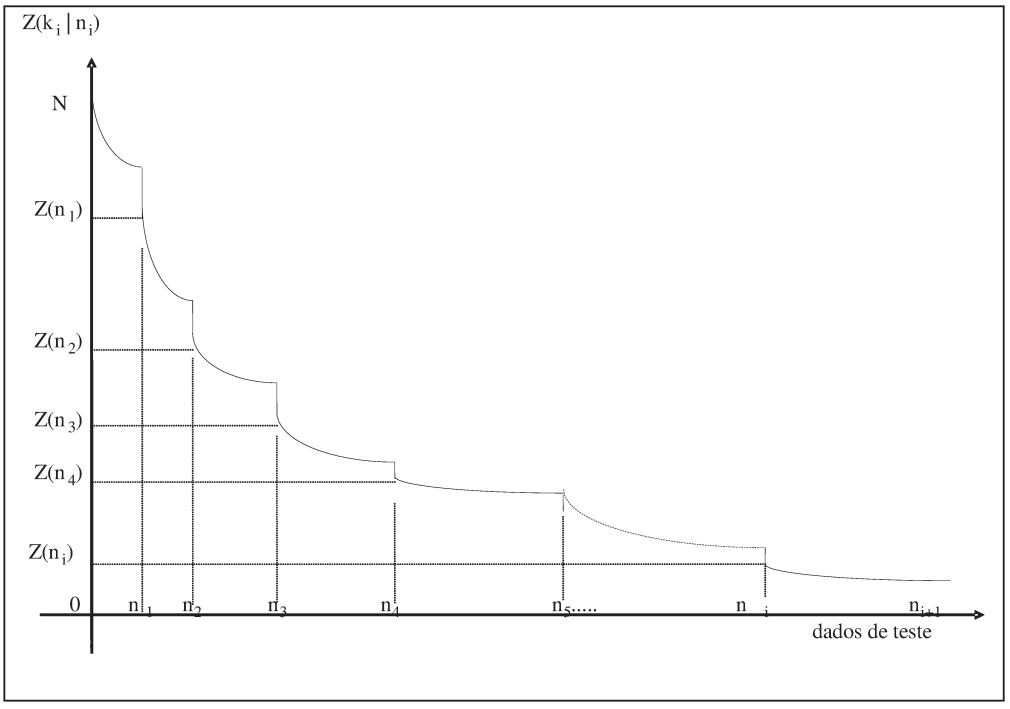

Figura 6 - Comportamento da função taxa de falhas

Para diferentes critérios de teste, obtém-se coberturas diferentes para os mesmos dados de teste. Contudo, para os mesmos dados de teste a confiabilidade estimada deve ser igual para todos os critérios de teste. Por isso, o modelo adota o uso da cobertura normalizada, no lugar da cobertura medida, para padronizar as estimativas de confiabilidade geradas pelo modelo independentemente de qualquer critério.

Assim, o parâmetro α representará a cobertura normalizada, isto é,

$$\alpha_i = g(c_i)$$

onde c_i representa a cobertura observada no teste, após a detecção da i-ésima falha g(.) é uma função crescente. Neste modelo, para se obter a cobertura normalizada, será adotada a transformação linear $\alpha_i = \alpha_0 + \alpha_1 c_i$. Logicamente, outras transformações podem ser utilizadas de acordo com as suposições ou de acordo com os dados obtidos no teste. Os parâmetros α_0 e α_1 podem ser estimados pelo método da máxima verossimilhança, conforme a Suposição 3. Observe-se que o complemento da cobertura é uma função decrescente e, por consequência, o complemento da cobertura normalizada será também decrescente. A restrição é que: $0 \leq \alpha_0 + \alpha_1 c_i \leq 1$.

Com a utilização da transformação linear para a normalização da cobertura, a função taxa de falhas condicional do software passa a ter a seguinte forma funcional:

$$Z(k_i | n_i) = [N - i]\alpha_i (n_i + k_i)^{\alpha_i - 1}$$

Qualquer que seja o critério de teste utilizado, a cobertura medida será sempre normalizada pelos parâmetros α_0 e α_1, garantindo a obtenção dos mesmos resultados nas previsões do modelo ao se utilizar qualquer um dos critérios de teste para medir a cobertura. Para cada um dos critérios selecionados, as estimativas dos parâmetros α_0 e α_1 serão distintas.

Após a remoção do i-ésimo defeito, a confiabilidade do software pode ser calculada por:

$$R(k_i|n_i) = \exp\{-[N-i][(n_i+k_i)^{\alpha_i} - (n_i)^{\alpha_i}]\}$$

onde:

- $\alpha_i = \alpha_0 + \alpha_1 c_i$ é a cobertura normalizada e c_i é o complemento da cobertura observada atingida com a aplicação dos n_i dados de teste, $0 \leq \alpha_i \leq 1$.
- N: é o número de defeitos no software no início do teste.
- n_i: é o número de dados de teste acumulado até a i-ésima falha observada, *i = 1, 2, 3,...,N*.
- k_i: é o número de dados de teste entre a i-ésima falha e a (i+1)-ésima falha, *i = 0,1,2,3,...N-1*.

Utiliza-se o método da máxima verossimilhança, aplicado à função de verossimilhança do modelo, para estimar seus parâmetros. A função de verossimilhança do modelo tem a seguinte forma:

$$L(K_0, K_2, ...,K_{r-1}; N, \alpha_0, \alpha_1) = f(k_0)f(k_1 \mid n_1)f(k_2 \mid n_2).....f(k_{r-1} \mid n_{r-1}) =$$

$$\prod_{i=0}^{r-1}[N-i]\alpha_i (n_i+k_i)^{\alpha_i-1} \exp\left\{-[N-i][(n_i+k_i)^{\alpha_i} - (n_i)^{\alpha_i}]\right\}$$

Tomando-se o logaritmo da função de verossimilhança e fazendo a substituição da cobertura normalizada $\alpha_i = \alpha_0 + \alpha_1 c_i$, tem-se que:

$$\ln[L(K_1, K_2, ...,K_r; N, \alpha_0, \alpha_1)] =$$

$$\sum_{i=0}^{r-1}\left\{\ln[N-i] + \ln(\alpha_0 + \alpha_1 c_i) + \ln[(n_i+k_i)^{(\alpha_0+\alpha_1 c_i)-1}]\right\}$$

$$-\sum_{i=0}^{r-1}\left\{[N-i][(n_i+k_i)^{(\alpha_0+\alpha_1 c_i)}-(n_i)^{(\alpha_0+\alpha_1 c_i)}]\right\} \quad (3.1)$$

Derivando (3.1) em relação ao parâmetro N e igualando a zero, tem-se:

$$\frac{\partial \ln(L)}{\partial N}=0$$

Assim, tem-se que:

$$\sum_{i=0}^{r-1}\frac{1}{[N-i]}-\sum_{i=0}^{r-1}\left\{[(n_i+k_i)^{(\alpha_0+\alpha_1 c_i)}-(n_i)^{(\alpha_0+\alpha_1 c_i)}]\right\}=0 \quad (3.2)$$

Derivando (3.1) em relação ao parâmetro α_0 e igualando a zero, tem-se:

$$\frac{\partial \ln(L)}{\partial \alpha_0}=0$$

Assim, tem-se que:

$$\sum_{i=0}^{r-1}\left[\frac{1}{\alpha_0+\alpha_1 c_i}+\ln(n_i+k_i)\right]-$$

$$\sum_{i=0}^{r-1}\left[-[N-i](n_i+k_i)^{(\alpha_0+\alpha_1 c_i)}\ln(n_i+k_i)-(n_i)^{(\alpha_0+\alpha_1 c_i)}\ln(n_i)\right]=0 \quad (3.3)$$

Derivando (3.1) em relação ao parâmetro α_1 e igualando a zero, tem-se:

$$\frac{\partial \ln(L)}{\partial \alpha_1}=0$$

Assim, tem-se que:

$$\sum_{i=0}^{r-1}\left[\frac{c_i}{\alpha_0+\alpha_1 c_i}+c_i \ln(n_i+k_i)\right]-$$

$$\sum_{i=0}^{r-1}\left[-[N-i](n_i+k_i)^{(\alpha_0+\alpha_1 c_i)}\, c_i \ln(n_i+k_i)-(n_i)^{(\alpha_0+\alpha_1 c_i)}\, c_i \ln(n_i)\right]=0 \qquad (3.4)$$

Observe-se que para se obter as estimativas de N, α_0 e α_1, é necessário resolver o sistema não linear (3.5) de três incógnitas, formado pelas Equações (3.2), (3.3) e (3.4).

Os valores de N, α_0 e α_1 que satisfazem simultaneamente o Sistema (3.5) são as estimativas obtidas pelos estimadores de máxima verossimilhança dos parâmetros. A solução desse sistema pode ser encontrada através de procedimentos numéricos.

Uma forma alternativa de se obter as estimativas dos parâmetros seria utilizar rotinas de otimização para se maximizar a função de verossimilhança (3.1), simultaneamente em função de N, α_0 e α_1.

$$\begin{cases}
\displaystyle\sum_{i=0}^{r-1}\frac{1}{[N-i]}=\sum_{i=0}^{r-1}\left\{[(n_i+k_i)^{(\alpha_0+\alpha_1 c_i)}-(n_i)^{(\alpha_0+\alpha_1 c_i)}]\right\} \\
\displaystyle\sum_{i=0}^{r-1}\left[\frac{1}{\alpha_0+\alpha_1 c_i}+\ln(n_i+k_i)\right]= \\
\displaystyle\sum_{i=0}^{r-1}\left[-[N-i](n_i+k_i)^{(\alpha_0+\alpha_1 c_i)}\ln(n_i+k_i)-(n_i)^{(\alpha_0+\alpha_1 c_i)}\ln(n_i)\right] \\
\displaystyle\sum_{i=0}^{r-1}\left[\frac{c_i}{\alpha_0+\alpha_1 c_i}+c_i\ln(n_i+k_i)\right]= \\
\displaystyle\sum_{i=0}^{r-1}\left[-[N-i](n_i+k_i)^{(\alpha_0+\alpha_1 c_i)}\, c_i \ln(n_i+k_i)-(n_i)^{(\alpha_0+\alpha_1 c_i)}\, c_i \ln(n_i)\right]
\end{cases} \qquad (3.5)$$

O artigo “A Binomial Software Reliability Model Based on Coverage of Structural Testing Criteria” apresenta o desenvolvimento matemático do modelo [Crespo08].

3.3 Modelo de Falhas Infinitas Baseado em Cobertura (MFIBC)

O Modelo Binomial Baseado em Cobertura, apresentado na seção anterior, é classificado como modelo de categoria de falhas finita, de acordo com Musa et al. [Musa87], ou seja, é suposto que num tempo infinito de teste o software teria todos os seus defeitos removidos.

Na prática sabe-se, que são altas as chances de inserir novos defeitos ao corrigir outro, principalmente em softwares grandes e complexos. Para considerar o efeito da depuração imperfeita, alguns modelos de confiabilidade de software tradicionais tiveram sua forma funcional alterada, como o modelo de Jelisnki-Moranda e o modelo logarítmico baseado na distribuição de Poisson [Bol02]. Crespo et al. [Crespo09] também propõem um modelo nesta categoria, o Modelo de Falhas Infinitas Baseado em Cobertura (MFIBC).

O MFIBC possui as seguintes pressuposições:

1. O processo de remoção de defeitos no software é imperfeito, isto é, o software nunca está isento de falhas.
2. Os defeitos não têm a mesma chance de serem detectados ;
3. As falhas, quando os defeitos são detectados, são independentes;
4. O software é testado nas mesmas condições em que é operado pelo usuário;
5. Os dados de teste são aplicados e a cobertura dos elementos requeridos do critério utilizado na seleção dos dados de teste é avaliada a cada ocorrência de uma falha; e
6. A taxa de falhas do software é constante entre falhas e decresce, a cada remoção de um defeito, em função da cobertura atingida no código com a seguinte forma funcional:

$$Z(n) = D(\phi_i)^i$$

onde:

- D é a taxa de falha inicial e ϕ_i, $(0 \leq \phi_i \leq 1)$ representa o complemento da cobertura normalizada atingida após a remoção do i-ésimo defeito, detectado após a aplicação de n_i dados de teste, sendo $n_i < n \leq n_{i+1}$.
- $\phi_i = \alpha_0 + \alpha_1 c_i$, onde c_i representa o complemento da cobertura observada no teste, após a detecção do i-ésimo defeito.
- α_0 e α_1 são parâmetros para normalizar a cobertura.
- A letra "i" define a ordem de remoção do defeito, por exemplo, *i = 0, 1, 2, 3...*

A Figura 7 ilustra o processo de teste no software.

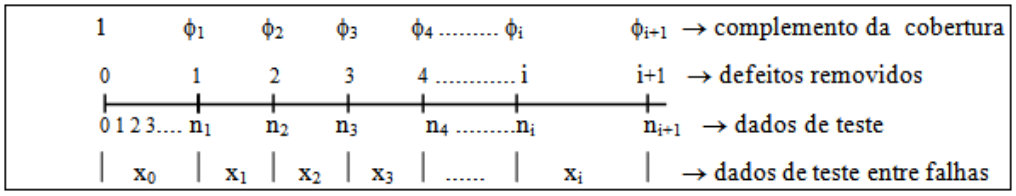

Figura 7 - Processo de teste do software

onde:

$n_0 = 0$

$n_1 = x_0$

$n_2 = n_1 + x_1$

.....

$n_{i+1} = n_i + x_i$

e $\phi_0 = 1$

A Suposição 1 indica que o processo de remoção de defeitos não é perfeito, isto é, a suposição implica que o número de falhas obtidas num tempo infinito poder ser infinito.

A Suposição 2 indica que os defeitos têm diferentes níveis de dificuldades de ser detectados.

A Suposição 3 permite que a função densidade de probabilidade conjunta, do método de máxima verossimilhança, seja obtida pela multiplicação das funções densidades de cada uma das variáveis aleatórias. Se as hipóteses são válidas, o método da máxima verossimilhança garante propriedades desejáveis para o estimador [Mood74].

A Suposição 4 permite que os parâmetros estimados no ambiente de teste sejam válidos para o ambiente de uso do software.

A Suposição 5 é uma característica do procedimento de teste e indica que a cobertura deve ser medida a cada ocorrência de falha.

A Suposição 6 caracteriza a forma funcional da taxa de falhas.

A suposição de que a aplicação de um dado de teste equivale a uma unidade de tempo de execução do software possibilita que a distribuição do número de dados de teste aplicados para a ocorrência de uma falha seja aproximada por uma distribuição contínua e pode ser utilizada como aproximação se a taxa de falhas é baixa.

Observe-se que a taxa de falhas, tal como definida, permanece constante entre falhas. Isto indica que a variável aleatória X, que representa o número de dados de teste entre falhas, tem uma distribuição de probabilidades que pode ser aproximada por uma distribuição exponencial com média:

$$E(X_i) = \frac{1}{D(\phi_i)^i}$$

Assim, o número de dados de teste entre falhas é representado por uma distribuição exponencial cuja média se altera a cada remoção de um defeito.

A Figura 8 ilustra o comportamento da função taxa de falhas do software.

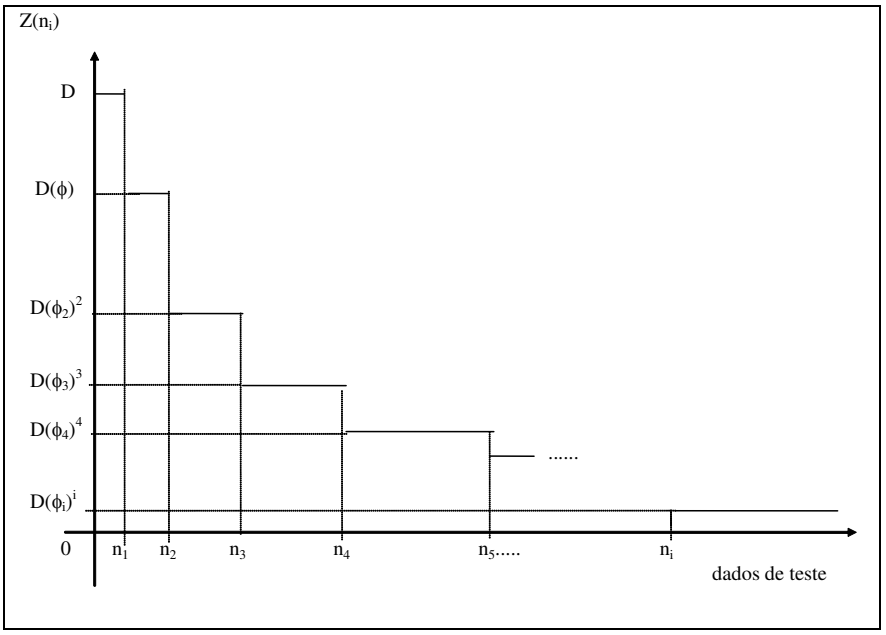

Figura 8 - Comportamento da taxa de falhas no software

Desde que tem-se a condição inicial que R(0) = 1, então c = 0. Logo a relação da função confiabilidade R(t) com a taxa de falhas Z(t) é dada por:

$$R(t) = \exp\left[-\int_0^t Z(x)\,dx \right]$$

Assim, tem-se que:

$$R(n) = \exp\left[-\int_0^n Z(n)\,dn \right] = \exp\left[-D(\phi_i)^i \int_0^n dx \right]$$

Desta forma, a função confiabilidade do software é definida como:

$$R(n) = \exp\left[-D(\phi_i)^i\, n \right], \text{ para } n_i \leq n$$

Utiliza-se o método da máxima verossimilhança, aplicado à função de verossimilhança do modelo, para estimar seus parâmetros. A função de verossimilhança do modelo tem a seguinte forma:

$$L(X_0,X_2,...,X_{k-1};\ D,\alpha_0,\alpha_1) \ = \ f(x_0)f(x_2)....f(x_{k-1}) =$$

$$D^k \prod_{i=0}^{k-1} \left[\alpha_0 + \alpha_1(1-c_i)\right]^i \exp\left\{-D\left[\alpha_0 + \alpha_1(1-c_i)\right]^i x_i\right\} \qquad (3.6)$$

Tomando o logaritmo da função de verossimilhança, tem-se:

$$\ln L = k\ln(D) + \sum_{i=0}^{k-1} i\ln\left[\alpha_0 + \alpha_1(1-c_i)\right] - D\sum_{i=0}^{k-1}\left[\alpha_0 + \alpha_1(1-c_i)\right]^i x_i \qquad (3.7)$$

Fazendo $\frac{\partial \ln L}{\partial D} = 0$, tem-se que: $\frac{k}{D} - \sum_{i=0}^{k-1}\left[\alpha_0 + \alpha_1(1-c_i)\right]^i x_i = 0$

Assim, o estimador de máxima verossimilhança do parâmetro D é a solução de:

$$\hat{D} = \frac{k}{\sum_{i=0}^{k-1}\left[\alpha_0 + \alpha_1(1-c_i)\right]^i x_i} \qquad (3.8)$$

Fazendo $\frac{\partial \ln L}{\partial \alpha_0} = 0$, tem-se que:

$$\sum_{i=0}^{k-1} \frac{i}{\left[\alpha_0 + \alpha_1(1-c_{i-1})\right]} - D\sum_{i=0}^{k-1} i\left[\alpha_0 + \alpha_1(1-c_i)\right]^{i-1} x_i = 0$$

Assim, o estimador de máxima verossimilhança para o parâmetro $\alpha 0$ é a solução de:

$$\sum_{i=0}^{k-1} \frac{i}{\left[\hat{\alpha}_0 + \alpha_1(1-c_i)\right]} = D\sum_{i=0}^{k-1} i\left[\hat{\alpha}_0 + \alpha_1(1-c_i)\right]^{i-1} x_i \qquad (3.9)$$

Fazendo $\frac{\partial \ln L}{\partial \alpha_1} = 0$, tem-se que:

$$\sum_{i=0}^{k-1} i \frac{(1-c_i)}{\left[\alpha_0 + \alpha_1(1-c_i)\right]} - D\sum_{i=0}^{k-1} i\left[\alpha_0 + \alpha_1(1-c_i)\right]^{i-1}(1-c_i)x_i = 0$$

Assim, o estimador de máxima verossimilhança para o parâmetro $\alpha 1$ é a solução de:

$$\sum_{i=0}^{k-1} i \frac{(1-c_i)}{\left[\alpha_0 + \hat{\alpha}_1(1-c_i)\right]} = D\sum_{i=0}^{k-1} i\left[\alpha_0 + \hat{\alpha}_1(1-c_i)\right]^{i-1}(1-c_i)x_i \qquad (3.10)$$

Observe-se que, para se obter as estimativas de D, α_0 e α_1, é necessário resolver um sistema não linear formado pelas Equações (3.8), (3.9) e (3.10). Os valores de D, α_0 e α_1 que satisfazem simultaneamente as três equações são as estimativas de máxima verossimilhança dos parâmetros. A solução desse sistema pode ser encontrada através de procedimentos numéricos.

Uma forma alternativa de se obter as estimativas dos parâmetros seria utilizar rotinas de otimização para se maximizar a função de verossimilhança, simultaneamente em função de D, α_0 e α_1.

O artigo "Applying Code Coverage Approach to an Infinite Failure Software Reliability Model" apresenta o desenvolvimento matemático do modelo [Crespo09].

3.4 Resumo

Neste capítulo foram apresentados estudos que investigaram a correlação entre cobertura de código e revelação de defeitos; alguns modelos de confiabilidade de software

que utilizam a informação de cobertura de código em sua forma funcional, com destaque para o Modelo Binomial Baseado em Cobertura (MBBC) e o Modelo de Falhas Infinitas Baseado em Cobertura (MFIBC), que são objetos do experimento descrito no próximo capítulo.

CAPÍTULO 4

AVALIAÇÃO DA ROBUSTEZ DOS MODELOS MBBC E MFIBC

Um processo geral para estimar a confiabilidade de software é apresentado pela Figura 9. Nele, dados de teste gerados a partir de um determinado perfil operacional são utilizados para testar um software. Durante o processo de teste, as informações sobre falhas são registradas, tais como tempo de ocorrência entre falhas ou o número de falhas num intervalo de tempo. Ao final do teste, estas informações são utilizadas para selecionar um modelo de confiabilidade de software que melhor se ajusta aos dados de falhas. A seleção do modelo de confiabilidade de software apropriado é feita por meio de testes estatísticos que analisam o ajuste aos dados das falhas observadas durante o processo de teste e as estimativas calculadas pelo modelo de confiabilidade de software. O modelo que apresentar o melhor ajuste é selecionado e pode ser utilizado tanto para estimar quanto para prever a confiabilidade do software.

No entanto, ao alterar-se o perfil operacional de um software, um novo conjunto de dados de falhas do software pode ser observado. Um modelo de confiabilidade pode ajustar-se bem para um conjunto de dados de falhas gerados a partir de um perfil operacional e ser rejeitado para outros conjuntos gerados a partir de diferentes perfis operacionais. Quando isso ocorre, é necessário realizar uma nova avaliação para seleção do modelo de confiabilidade que melhor se ajusta aos dados de falhas.

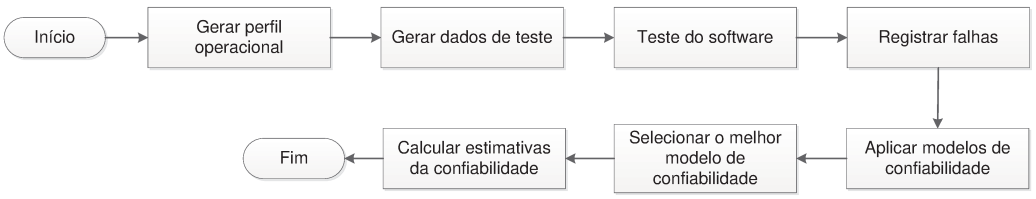

Figura 9 - Visão geral do processo de cálculo da confiabilidade de software

Uma qualidade desejável para um modelo de confiabilidade de software é que ele seja robusto às variações de perfil operacional, ou seja, um mesmo modelo pode ser utilizado para gerar estimativas precisas da confiabilidade do software em condições variadas de uso, reduzindo a necessidade de utilização de muitos modelos de confiabilidade [Cai05], [Frat95] e [Lyu07].

Lyu e Cai [Cai07] avaliaram um novo modelo de confiabilidade de software utilizando mutantes de um programa. Cotroneo et al. [Cot13] propõem um método de teste para obtenção de software com alta confiabilidade; eles validam o método utilizando diversos perfis operacionais de um mesmo software.

Os resultados dos modelos de confiabilidade de software MBBC e MFIBC, propostos por Crespo [Crespo08] e [Crespo09], foram comparados com os produzidos pelos modelos tradicionais e mostraram-se superiores a estes [Crespo97b]. No entanto, a investigação utilizou dados de teste gerados a partir de um único perfil operacional.

Um experimento foi realizado para analisar a robustez dos modelos MBBC e MFIBC para uma variação do perfil operacional. Três perfis operacionais estatisticamente distintos foram definidos e milhares de dados de teste foram gerados de acordo com cada perfil. A confiabilidade medida do software foi calculada para cada perfil operacional. A cobertura dos elementos requeridos pelos critérios de teste foi medida e utilizada como parâmetro dos modelos. A confiabilidade estimada pelos modelos foi comparada com a confiabilidade medida por um método estatístico. A Figura 10 mostra as etapas do procedimento utilizado para avaliar a robustez dos modelos de confiabilidade de software escolhidos para o experimento. Cada etapa é explicada detalhadamente nas seções seguintes. No Capítulo 5, serão apresentados os resultados deste experimento.

4.1 Geração do Perfil Operacional

Para avaliar a robustez dos modelos MBBC e MFIBC a variações no perfil operacional, três perfis operacionais estatisticamente diferentes para um software foram determinados para a realização do experimento. O software escolhido foi o **Space**, desenvolvido pela *European Space Agency* (ESA), e utilizado para calcular e gerar parâmetros de entrada para outro programa, utilizado para encontrar a melhor distribuição física de antenas utilizadas em aplicações espaciais.

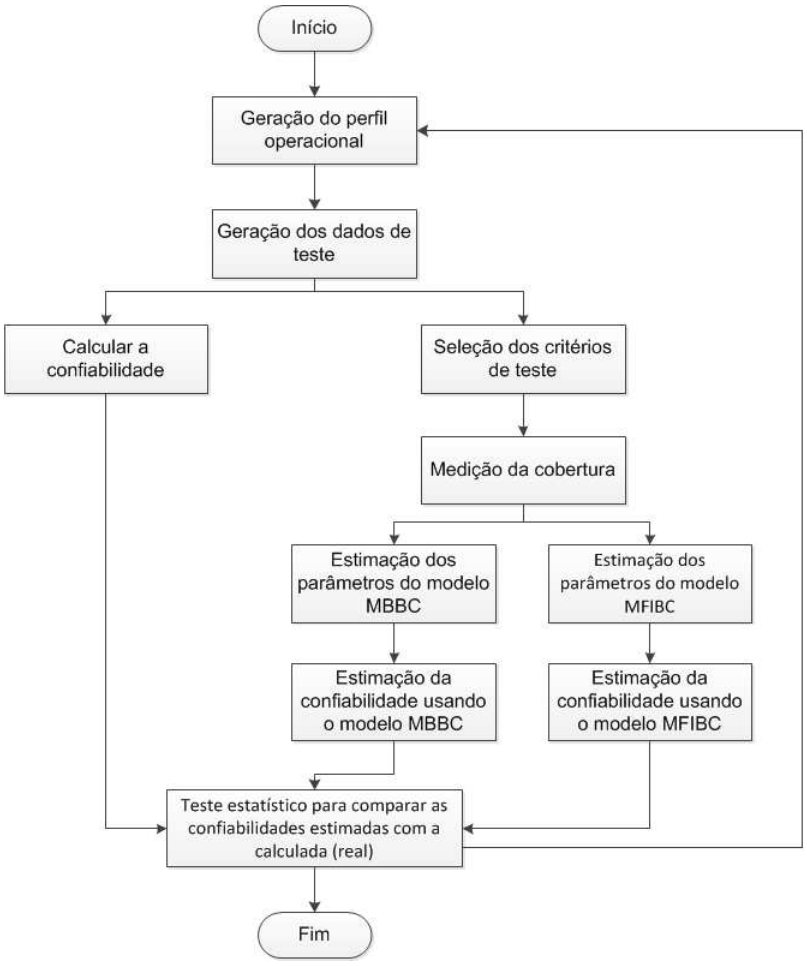

Figura 10 - Visão geral do experimento

O software **Space** foi desenvolvido em linguagem C e possui cerca de 10.000 linhas de código, sendo 6.100 de código executável; é estruturado em torno de um programa

principal e 134 módulos que totalizam 236 funções. O **Space** foi utilizado em experimentos anteriores que avaliaram a existência da correlação entre o aumento da cobertura dos elementos requeridos pelos critérios de teste e a revelação de defeitos; todos os 33 defeitos detectados durante o teste de sistema estão documentados [Garg95].

O perfil operacional para o software **Space** foi determinado a partir da atribuição de frequências de uso às suas funções. Um grafo foi gerado capturando a conectividade entre as funções, onde cada nó representa uma função. Dois nós, "A" e "B", estão conectados se o fluxo de controle pode ir da função "A" para a função "B". Existe um único nó de entrada e um único nó de saída. Um caminho do nó inicial ao nó final representa uma execução possível do software **Space**.

Cada arco conectando duas funções possui uma probabilidade de execução associada. Por exemplo, se um nó "A" está conectado aos nós "B", "C" e "D", e as probabilidades associadas aos arcos "A-B", "A-C" e "A-D" forem, respectivamente, 0,3, 0,6 e 0,1, então, após a execução da função "A", o programa poderá executar as funções "B", "C" ou "D" respectivamente com probabilidades 0,3, 0,6 e 0,1 – Figura 11.

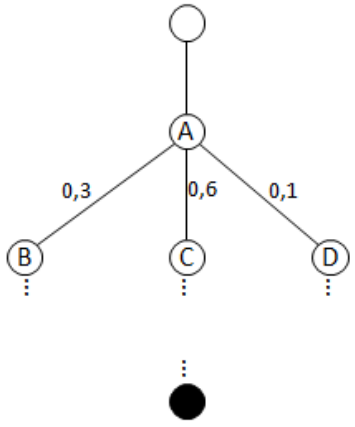

Figura 11 - Exemplo de grafo de fluxo de controle de Space

A Figura 12 mostra o grafo parcial do **Space** para um determinado perfil operacional. Nele, os nós 1, 2, 3, 4 e 5 possuem as seguintes frequências de uso: 35%, 15%,

35%, 14% e 1%. A sequência de execução do fluxo indica, por exemplo, que, caso o nó número três seja executado, então os próximos nós com chance de execução, segundo o perfil operacional, são os nós 6, 9 e 7, com frequências 1%, 70% e 29% de uso. Em cinza, destaca-se um possível caminho que seria executado pelo programa com a execução de um dado de teste.

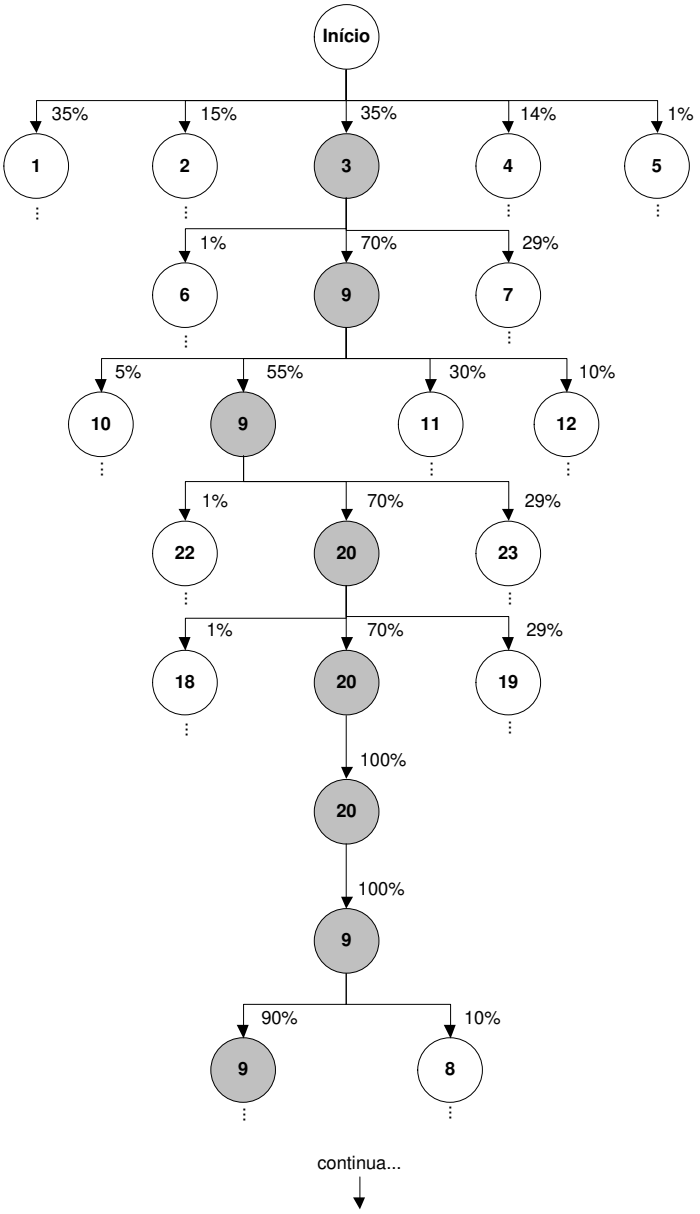

Figura 12 - Grafo parcial do programa Space para um perfil operacional

Para garantir que os três perfis operacionais selecionados para o experimento fossem distintos, as medidas da confiabilidade calculadas em cada perfil operacional foram comparadas entre si por um teste estatístico. Esta verificação é importante para o experimento, pois deseja-se analisar o comportamento dos modelos quando varia-se o perfil operacional do software. Isto é, considerando-se dois perfis operacionais de um mesmo software, o fato de duas funções possuírem frequências de execução diferentes pode não ser suficiente para alterar o comportamento da taxa de falhas do software. Para os objetivos do experimento, o software deve ser utilizado de acordo com perfis que geram taxas de falhas diferentes, implicando curvas de crescimento da confiabilidade distintas.

4.2 Geração dos Dados de Teste

Os dados de teste foram gerados com o auxílio de um programa criado para este fim. O programa gera dados de teste aleatoriamente, de acordo com um perfil operacional determinado. O programa, denominado COPIA, carrega as configurações do perfil operacional, armazenada no arquivo TABELLA, e gera os dados de teste que são posteriormente utilizados como dados de entrada para o programa Space. A relação entre os programas e arquivos é ilustrada pela Figura 13.

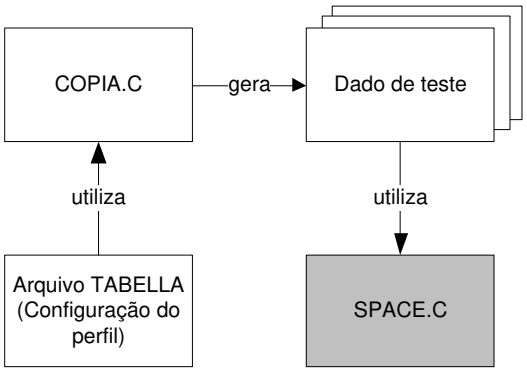

Figura 13 - Programas e arquivos utilizado na geração de dados de teste

4.3 Medição da Confiabilidade do Software

Para avaliar a precisão das estimativas da confiabilidade calculadas pelos modelos, é necessário compará-las com a confiabilidade medida do software. A confiabilidade medida pode ser obtida pelo Método de Nelson [Nels78], que estima a confiabilidade de um software por meio da expressão:

$$R = 1 - \frac{n_f}{n}$$

onde, n é o número total de execuções do software e n_f é o número de execuções com falhas durante as n execuções.

O software é testado com milhares de dados de teste, gerados pelo aplicativo COPIA com base no perfil operacional, definido em TABELLA. A cada defeito removido do software, o método é aplicado para se calcular a confiabilidade. Com este procedimento, tem-se uma estimativa do comportamento do crescimento da confiabilidade do software em função da remoção dos defeitos. A confiabilidade é sempre calculada após a remoção do defeito. Neste processo também é registrada a quantidade de dados de teste executados até a ocorrência de uma falha.

Para o cálculo da confiabilidade, foram consideradas 33 versões do software Space: P_0, P_1, P_2, ..., P_{33}, lembrando que 33 defeitos foram registrados durante o teste de sistema. Eles são:

P_0: **Space** com 33 defeitos.

P_1: **Space** com o primeiro defeito removido.

P_2: **Space** com o segundo defeito removido.

P_j: **Space** com o j-ésimo defeito removido.

P_{33}: **Space** com todos os defeitos removidos.

O algoritmo de alto nível a seguir foi utilizado para calcular a confiabilidade do software:

1. Selecionar a versão P_i de **Space**, sendo $0 \leq i \leq 33$.
2. Selecionar um dado de teste gerado aleatoriamente.
3. Executar P_i com o dado de teste.
4. Se o programa falhar
   i. Registrar a quantidade de dados de teste necessários até a falha.
   ii. Remover o defeito.
   iii. Calcular a confiabilidade de acordo com a expressão: $R = 1 - \frac{n_f}{n}$
   iv. Ir para o passo 1.
5. Se o programa não falhar
   i. Registrar o número de execuções sem falhas.
   ii. Ir para o passo 3.

No Apêndice A, são apresentados os programas utilizados para calcular a confiabilidade do programa **Space** e registrar a quantidade de dados de teste necessários para a revelação de cada defeito.

4.4 Seleção dos Critérios de Teste

Para o experimento, foram selecionados critérios estruturais baseados em fluxo de controle e fluxo de dados [DELA07], incluindo os critérios da família Potenciais-Usos. Os critérios escolhidos foram:

- **Critérios baseados em fluxo de controle** - são critérios de teste fundamentados nas estruturas de controle do código. Foram selecionados os seguintes critérios deste tipo para o experimento: *Todos-Nós* e *Todos-Arcos (Todas-Arestas)*.

- **Critério baseado em fluxo de dados** - São critérios de teste fundamentados nos tipos de ocorrências de uma variável, basicamente, a *definição* e o *uso* de uma variável. A *definição* de uma variável ocorre quando um valor é armazenado em uma posição de memória. O *uso* de uma variável ocorre quando seu valor é lido da posição de memória correspondente. Uma *associação* é o exercício da definição da variável em um nó e o uso desta variável em um outro nó, através de um dado de entrada. Foram selecionados os seguintes critérios deste tipo para o experimento: *Todos-Usos.*
- **Critérios baseado em fluxo de dados da família Potenciais-Usos -** Na família de critérios de fluxo de dados baseados no uso de uma variável identificam-se os pontos de um programa onde há definições e usos de variáveis, e os elementos requeridos desses critérios são associações envolvendo esses pontos. A família de critérios Potenciais-usos identifica somente os pontos onde há a definição de uma variável e todos os outros pontos onde é possível o uso dessa variável, daí o nome "potenciais-usos". Foram selecionados os seguintes critérios deste tipo para o experimento: *Todos-Potenciais-Usos (PU)*, *Todos-Potenciais-Usos/Du (PUDU)*, *Todos-Potenciais-Du-Caminhos (PDU).*

4.5 Medição da Cobertura do Software

A medição da cobertura dos elementos requeridos do critério de teste pela aplicação de um conjunto de dados de teste só é possível por meio de um processo automatizado, ou seja, com a utilização de um software desenvolvido para tal propósito.

A **POKE-TOOL** (*Potential Uses Criteria Tool for Program Testing*) suporta a aplicação dos critérios de teste de software baseados em informação de código do programa, tanto para a seleção de dados de teste como também para a análise da cobertura

atingida, segundo os critérios de teste selecionados. POKE-TOOL já foi utilizada para análise da cobertura em diversos experimentos [Chaim91], [Crespo97a], [Crespo97b].

O experimento foi realizado utilizando-se uma versão da POKE-TOOL compilada para o sistema operacional Fedora 5 (Linux); também disponível para outras versões do sistema operacional Linux.

Para melhor entendimento do procedimento de cálculo da cobertura dos elementos requeridos, os passos estão resumidos no seguinte algoritmo de alto nível:

1. Executar o módulo **poketool** em cada uma das 134 rotinas do programa **Space**. Este passo instrumentará as rotinas, preparando-as para a execução dos dados de teste.
2. Para cada uma das rotinas r_1, r_2, r_3, ..., r_{134} calcula-se a seguinte soma: $S = Sr_1 + Sr_2 + ... + Sr_{134}$, onde Sr_i é o número de elementos requeridos da rotina r_i (este procedimento é efetuado uma única vez).
3. Executa-se o módulo **pokeexec** com todos os dados de teste, em cada uma das 134 rotinas. Neste passo, aplicam-se os dados de teste.
4. Escolhe-se um dos critérios de teste disponíveis na **POKE-TOOL**.
5. Executa-se o modulo **pokeaval**, em cada uma das 134 rotinas, informando o intervalo dos dados de teste: ***dado$_1$ a dado$_x$***, onde ***dado$_x$*** é o dado de teste que ocasionou a ocorrência de uma falha.
6. Para cada uma das rotinas r_1, r_2, ..., r_{134} anotam-se os valores sr_1, sr_2, ..., sr_{134}, que representam o total de elementos requeridos exercitados com a aplicação dos dados de teste ***dado$_1$ a dado$_x$***.
7. Calculam-se os valores

   a) $s = sr_1 + sr_2 + ... + sr_{134}$; soma dos elementos requeridos exercitados com a aplicação dos dados de teste referente ao intervalo ***dado$_1$ a dado$_x$***.

   b) Calcula-se Cob = $\frac{s}{S}$, cobertura do programa **Space** atingida com a aplicação dos dados de teste referente ao intervalo ***dado$_1$ a dado$_x$***.

A Figura 14 ilustra o procedimento de cálculo da cobertura dos elementos requeridos do programa **Space**. A **POKE-TOOL** executa o programa **Space** com os dados de teste e para cada rotina do programa, R_1, R_2, ... , R_{134}, é contabilizado o total de elementos requeridos pelo critério de teste e o total de elementos requeridos que foram exercitados pela execução do programa com os dados de teste.

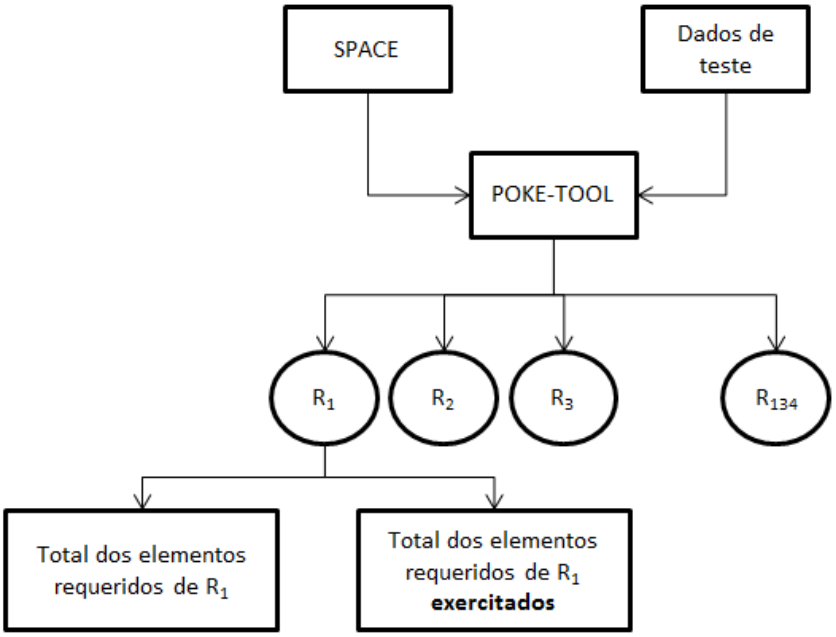

Figura 14 - Procedimento de cálculo da cobertura

No Apêndice A, são apresentados todos os programas utilizados para medir a cobertura atingida pelos elementos requeridos dos critérios de teste selecionados.

4.6 Estimação da Confiabilidade pelo Modelo MBBC

Conforme apresentado no Capítulo 3, a estimação da confiabilidade do software por meio do Modelo Binomial Baseado em Cobertura, MBBC, utiliza a seguinte função densidade de probabilidade:

$$R(k_i|n_i) = \exp\{-[N-i][(n_i + k_i)^{\alpha_i} - (n_i)^{\alpha_i}]\}$$

onde,

- $\alpha_i = \alpha_0 + \alpha_1 c_i$: representa a cobertura normalizada e c_i o complemento da cobertura atingida com a aplicação dos n_i dados de teste, $0 \leq \alpha_i \leq 1$.
- *N*: é o número de defeitos no software no início do teste.
- n_i: é o número de dados de teste acumulado até a i-ésima falha observada, *i = 1, 2, 3,...,N*.
- k_i: é o número de dados de teste entre a i-ésima falha e a (i+1)-ésima falha, *i = 0,1,2,3,...N-1*.

Os valores das variáveis n_i e k_i são obtidos diretamente dos dados gerados pelo processo de teste do software.

Os parâmetros *N*, α_0 e α_1 são estimados pelo método da máxima verossimilhança aplicado à função de verossimilhança da função densidade de probabilidade do modelo MBBC:

$$\ln[L(K_1, K_2, \dots, K_r; N, \alpha_0, \alpha_1)] = \sum_{i=0}^{r-1} \{\ln[N-1] + \ln(\alpha_0 + \alpha_1 c_i) + \ln[(n_i + k_i)^{(\alpha_0 + \alpha_1 c_i) - 1}]\} - \sum_{i=0}^{r-1} \{[N-1](n_i + k_i)^{(\alpha_0 + \alpha_1 c_i)} - (n_i)^{(\alpha_0 + \alpha_1 c_i)}]\}$$

As estimativas dos parâmetros são obtidas por meio da função CONSTR do software MATLAB. CONSTR determina o mínimo local de uma dada função em uma vizinhança pré-determinada por parâmetros informados. Os valores obtidos maximizam a função de máxima verossimilhança do modelo MBBC, isto é, determinam os valores prováveis dos estimadores da função, com base nos dados de cobertura dos critérios de teste.

No Apêndice B são apresentadas as rotinas MATLAB utilizadas.

4.7 Estimação da Confiabilidade pelo Modelo MFIBC

Conforme apresentado no Capítulo 3, a estimação da confiabilidade do software por meio do Modelo de Falhas Infinitas Baseado em Cobertura, MFIBC, utiliza a seguinte função densidade de probabilidade:

$$R(n) = \exp\left[-D\ (\emptyset_i)^i n\right]$$

Após a remoção do i-ésimo defeito, a confiabilidade do software pode ser calculada utilizando-se a seguinte função densidade de probabilidade condicional:

$$R(k_i|n_{i-1}) = \exp[-D\ \left[(\emptyset_i)^{(i-1)}\right]]$$

onde,

- $\emptyset_i = \alpha_0 + \alpha_1 c_i$: representa a cobertura normalizada após a remoção do i-ésimo defeito e c_i o complemento da cobertura atingida com a aplicação dos n_i dados de teste, $0 \leq \emptyset_i \leq 1$.
- *D*: representa a taxa de falhas inicial.
- n_i: é o número de dados de teste acumulado até a i-ésima falha observada, *i = 1, 2, 3,...,N*.
- k_i: é o número de dados de teste entre a i-ésima falha e a (i+1)-ésima falha, *i = 0,1,2,3,...N-1*.
- *i*: representa a ordem do defeito removido, ou seja, *i* = 0, 1,2,3,...

Os valores das variáveis n_i, k_i e *i* são obtidos diretamente dos dados gerados pelo processo de teste:

- $\boldsymbol{n_i}$ será obtido a partir dos dados de teste acumulados para a ocorrência de cada falha.
- $\boldsymbol{k_i}$ é o número de dados de teste entre as ocorrências de falhas.
- $\boldsymbol{i}$ será representado pela ordem de remoção do defeito.

Os parâmetros D, α_0 e α_1 são estimados pelo método da máxima verossimilhança aplicado à função de verossimilhança da função densidade de probabilidade do modelo MFIBC:

$$\ln[L(X_0, X_1, \dots, K_{k-1}; D, \alpha_0, \alpha_1)] = k\ln(D) + \sum_{i=0}^{k-1} i \ln[\alpha_0 + \alpha_1(1 - c_i)] - D\sum_{i=0}^{k-1}[\alpha_0 + \alpha_1(1 - c_i)]^i x_i$$

As estimativas dos parâmetros são obtidas por meio da função CONSTR do software MATLAB. CONSTR determina o mínimo local de uma dada função em uma vizinhança pré-determinada por parâmetros informados. Os valores obtidos maximizam a função de máxima verossimilhança do modelo MFIBC, isto é, determinam os valores prováveis dos estimadores da função, com base nos dados de cobertura dos critérios de teste.

No Apêndice B são apresentadas as rotinas MATLAB utilizadas.

4.8 Comparação das Estimativas de Confiabilidade

Nesta etapa foram comparadas as confiabilidades estimadas pelos modelos MBBC e MFIBC com a confiabilidade medida do software. As comparações foram realizadas para os resultados obtidos de acordo com cada um dos perfis operacionais selecionados, ou seja, OP1, OP2 e OP3.

Para comparar a confiabilidade medida com a confiabilidade estimada pelos modelos aplicou-se o teste de Kolmogorov-Smirnov. O teste é utilizado para verificar se duas amostras provêm da mesma população. A realização do teste é feita sob a hipótese nula, H_0, de que os dados são provenientes da mesma população. Se esta hipótese não puder ser comprovada, então ela é rejeitada em favor da hipótese alternativa, H_1, ou seja, as amostras provêm de populações diferentes [Con99].

4.9 Resumo

Este capítulo apresentou a visão geral do experimento: geração dos perfis operacionais, geração dos dados de teste, algoritmo para medir a confiabilidade medida do software utilizado no experimento, seleção dos critérios de teste utilizados para medir-se a cobertura dos elementos requeridos pelos critérios, algoritmo para medição da cobertura, estimação da confiabilidade utilizando os modelos MBBC e MFIBC e, por fim, a comparação das estimativas geradas pelos modelos com a confiabilidade medida do software.

CAPÍTULO 5

RESULTADOS DO EXPERIMENTO

Neste capítulo, são apresentados os resultados do experimento realizado para avaliar a robustez dos modelos de confiabilidade de software MBBC e MFIBC sob diferentes perfis operacionais.

5.1 Cálculo da Confiabilidade Observada

Os resultados do processo de teste do software **Space** e consequente medição da confiabilidade, para cada perfil operacional, são apresentados nas Tabelas 5, 6 e 7. A coluna "Ordem do defeito removido" refere-se à ordem em que o defeito foi descoberto durante os testes em decorrência de uma falha apresentada com a execução de um dado de teste. A coluna "Código do defeito" identifica um defeito do programa **Space**. Os defeitos estão identificados como E1, E2,.., E33. A coluna "Dados de teste entre falhas" refere-se ao número de dados de teste que foram aplicados entre duas falhas consecutivas. A coluna "Dados de teste acumulados" refere-se à quantidade de dados de testes executados até a ocorrência de uma falha. A coluna "Confiabilidade medida" refere-se à medida da confiabilidade, calculada pelo método da força-bruta [Nels78], após a ocorrência de uma falha e consequente remoção do defeito que a ocasionou.

A influência do perfil operacional está refletida no número de ocorrências de falhas e a quantidade de dados de teste necessários para revelá-las. De acordo com o perfil OP1, foram necessários 1191 dados de testes para a ocorrência de 24 falhas no Space. O perfil OP2 necessitou de 2428 dados de testes para a ocorrência de 17 falhas no software. A

execução de Space com os dados de testes gerados segundo o perfil OP3 causou 23 falhas com um total de 1634 dados de testes executados.

Os resultados mostram o impacto do perfil operacional no programa **Space**. A execução do programa de acordo com os diferentes perfis operacionais resulta em sequências diferentes de ocorrências de falhas e mesmo algumas falhas ocorrem em um perfil mas não em outro. Nota-se também que a quantidade de dados de teste executados para a revelar uma falha é diferente entre os perfis operacionais. A coluna "Dados de teste entre falhas" apresentada nas Tabelas 5, 6 e 7 aumenta com a remoção dos defeitos encontrados. Este comportamento é verificado nos três perfis. Por fim, conforme pode ser observado na Figura 15, observa-se o crescimento da confiabilidade com a remoção dos defeitos, um efeito esperado.

Tabela 5 - Defeitos revelados e confiabilidade medida – Perfil OP1

Ordem do defeito removido	Código do defeito	Dados de teste entre falhas	Dados de teste acumulados	Confiabilidade medida
1	E6	1	1	0,007800
2	E10	1	2	0,007800
3	E15	1	3	0,010200
4	E16	1	4	0,010200
5	E20	1	5	0,010200
6	E21	1	6	0,010200
7	E24	1	7	0,010200
8	E28	1	8	0,046400
9	E9	2	10	0,127800
10	E11	2	12	0,158500
11	E19	2	14	0,210100
12	E25	2	16	0,514700
13	E13	6	22	0,587900
14	E5	7	29	0,838400
15	E14	7	36	0,873400
16	E29	19	55	0,899500
17	E23	20	75	0,916500
18	E17	31	106	0,946300
19	E7	63	169	0,971000
20	E8	63	232	0,986900
21	E22	216	448	0,992700
22	E27	243	691	0,996100
23	E12	250	941	0,996100
24	E18	250	1191	0,998700

Tabela 6 - Defeitos revelados e confiabilidade medida – Perfil OP2

Ordem do defeito removido	Código do defeito	Dados de teste entre falhas	Dados de teste acumulados	Confiabilidade medida
1	E31	1	1	0,517587
2	E11	2	3	0,517587
3	E26	2	5	0,540610
4	E15	6	11	0,753358
5	E7	11	22	0,780004
6	E14	11	33	0,780004
7	E16	11	44	0,808996
8	E23	11	55	0,808996
9	E8	18	73	0,840546
10	E29	20	93	0,865487
11	E24	37	130	0,908975
12	E20	100	230	0,968237
13	E21	100	330	0,968237
14	E22	136	466	0,978256
15	E27	136	602	0,983586
16	E17	216	818	0,996163
17	E33	1610	2428	0,999147

Tabela 7 - Defeitos revelados e confiabilidade medida – Perfil OP3

Ordem do defeito removido	Código do defeito	Dados de teste entre falhas	Dados de teste acumulados	Confiabilidade medida
1	E4	1	1	0,294574
2	E10	2	3	0,325926
3	E14	2	5	0,334442
4	E15	3	8	0,363636
5	E16	3	11	0,363636
6	E23	3	14	0,365653
7	E11	4	18	0,423055
8	E17	5	23	0,440782
9	E26	6	29	0,474541
10	E19	8	37	0,590761
11	E7	13	50	0,606989
12	E8	13	63	0,619681
13	E29	13	76	0,650122
14	E13	14	90	0,651462
15	E31	14	104	0,899123
16	E24	32	136	0,945055
17	E20	38	174	0,953705
18	E21	38	212	0,953705
19	E27	64	276	0,980702
20	E12	187	463	0,983602
21	E18	187	650	0,983602
22	E22	276	926	0,990801
23	E32	708	1634	0,998800

Uma análise visual indica que as curvas de crescimento da confiabilidade são distintas para os três perfis operacionais selecionados. O teste de Kolmogorov-Smirnov para a hipótese de igualdade entre as distribuições apresentadas foi rejeitado para todos os pares comparados, mostrando que os três perfis operacionais utilizados no experimento são diferentes entre si. A obtenção de três perfis operacionais estatisticamente distintos não foi um processo fácil. Dezenas de perfis foram definidos e o resultado de seu impacto no crescimento da curva de confiabilidade foram comparados. Os resultados são apresentados na Tabela 8.

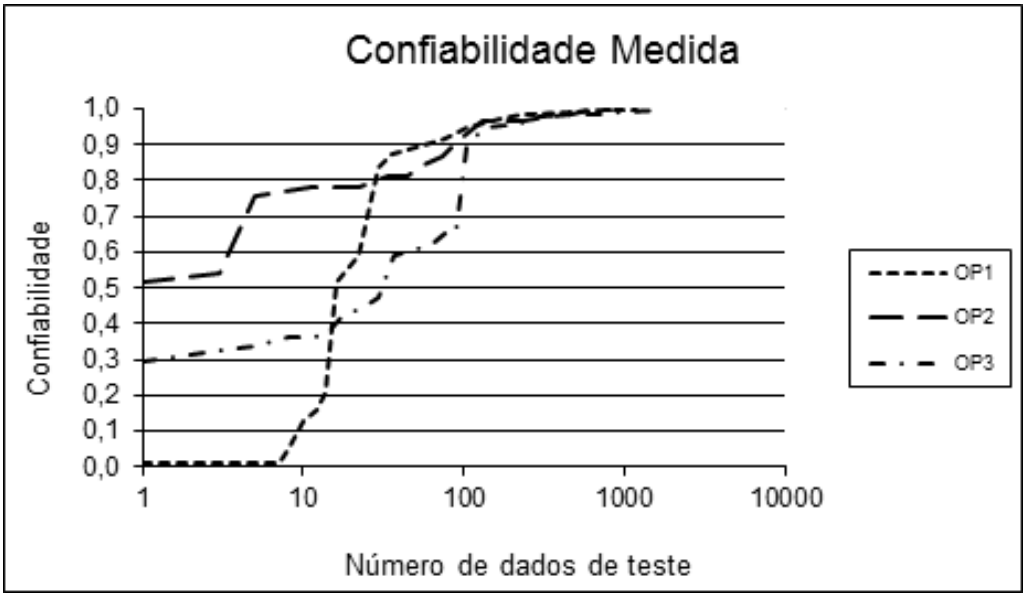

Figura 15 - Crescimento da confiabilidade para os perfis OP1, OP2 e OP3

Tabela 8 - Comparação entre os perfis operacionais utilizados

Hipótese (H_0)	Nível de Significância	Valor do Teste (p-Value)	Rejeita a hipótese de igualdade (H_0) entre as distribuições?
OP1 = OP2	1%	0,0084	Sim
OP1 = OP3	1%	0,0092	Sim
OP2 = OP3	5%	0,0358	Sim

5.2 Medição da Cobertura Atingida pelos Critérios de Teste

A cobertura dos critérios de teste observada durante a execução do software **Space** com dados de testes gerados segundo o perfil operacional OP1 é apresentada na Tabela 9. A coluna "Defeito Removido" refere-se à identificação dos defeitos detectados no **Space**, utilizando-se o perfil operacional OP1. A coluna "Dados de Teste Acumulados" refere-se ao número acumulado de dados de teste. A coluna "Dados de Teste Entre Falhas" refere-se ao número de dados de teste que foram executados no intervalo entre duas falhas consecutivas. As colunas "Nós", "Arcos", "PU", "PUDU", "PDU" e "Usos" referem-se às medidas de cobertura atingidas segundo os critérios de teste "Todos-Nós", "Todos-Arcos", "Todos Potenciais-Usos", "Todos Potenciais-Usos/DU", "Todos Potenciais-DU-Caminhos" e "Todos-Usos", respectivamente.

Os valores medidos da cobertura obedecem à relação de inclusão existente entre os critérios utilizados; o critério Todos-Arcos inclui o critério Todos-Nós, por isso, para uma mesma quantidade de dados de teste, provenientes do mesmo conjunto, observa-se que o índice de cobertura atingido para o critério Todos-Arcos é menor do que o índice atingido para o critério Todos-Nós, pois a quantidade de elementos requeridos exercitados é menor. A mesma relação de inclusão é observada para a família Potenciais-Usos.

O crescimento da cobertura para os diferentes critérios de teste selecionados para o perfil operacional OP1 pode ser analisado graficamente na Figura 16. A Figura 17 exibe a cobertura alcançada para os cem primeiros dados de teste utilizados, para uma análise mais detalhada sobre o comportamento do crescimento da cobertura no início do teste.

Observa-se que, no início dos testes, todos os critérios de cobertura utilizados no experimento apresentam uma taxa de crescimento maior. No entanto, após certa quantidade de dados de testes executados ela diminui. Isso ocorre porque a capacidade do critério de teste para selecionar novos dados de teste que exercitem elementos requeridos distintos dos já exercitados diminui com o prosseguimento do teste. Nas Figuras 16 e 17, pode-se

observar que, após executar a primeira centena de dados de teste, a taxa de crescimento da cobertura estabiliza-se para todos os critérios de teste selecionados.

Tabela 9 - Cobertura medida para o perfil OP1

Defeito Removido	Dados de Teste Acumulados	Dados de Teste Entre Falhas	Cobertura dos Critérios						Confiabilidade Medida
			Nós	Arcos	PU	PUDU	PDU	Usos	
E6	1	1	0,221836	0,124000	0,076937	0,069976	0,047175	0,193496	0,007800
E10	2	1	0,221836	0,124000	0,076937	0,069976	0,047175	0,193496	0,007800
E15	3	1	0,303894	0,165333	0,096171	0,089943	0,056509	0,193496	0,010200
E16	4	1	0,303894	0,165333	0,096171	0,089943	0,056509	0,193496	0,010200
E20	5	1	0,303894	0,165333	0,096171	0,089943	0,056509	0,193496	0,010200
E21	6	1	0,303894	0,165333	0,096171	0,089943	0,056509	0,193496	0,010200
E24	7	1	0,301926	0,164021	0,096031	0,089812	0,056395	0,193496	0,010200
E28	8	1	0,328748	0,186508	0,148710	0,128224	0,067472	0,193496	0,046400
E9	10	2	0,403026	0,257937	0,190232	0,168100	0,094159	0,311577	0,127800
E11	12	2	0,403026	0,257937	0,190232	0,168100	0,094159	0,311577	0,158500
E19	14	2	0,402893	0,258278	0,190119	0,167978	0,094002	0,311577	0,210100
E25	16	2	0,459366	0,295364	0,245170	0,201830	0,113407	0,311577	0,514700
E13	22	6	0,488981	0,316556	0,259286	0,216102	0,126008	0,340175	0,587900
E5	29	7	0,519284	0,344371	0,274890	0,230417	0,137131	0,382380	0,838400
E14	36	7	0,522039	0,340397	0,272328	0,229136	0,136879	0,382380	0,873400
E29	55	19	0,592975	0,411921	0,354136	0,288433	0,170910	0,458026	0,899500
E23	75	20	0,623967	0,439735	0,395681	0,321376	0,179985	0,490314	0,916500
E17	106	31	0,651515	0,470199	0,489934	0,385615	0,198639	0,543819	0,946300
E7	169	63	0,667353	0,491436	0,519737	0,410562	0,209915	0,562269	0,971000
E8	232	63	0,668033	0,492792	0,521461	0,414503	0,207632	0,562269	0,986900
E22	448	216	0,693669	0,521569	0,541180	0,435988	0,219101	0,601937	0,992700
E27	691	243	0,694350	0,522876	0,541350	0,436158	0,219551	0,602399	0,996100
E12	941	250	0,699115	0,528105	0,546930	0,441062	0,222697	0,606780	0,996100
E18	1191	250	0,699115	0,528105	0,546930	0,441062	0,222697	0,606780	0,998700

Observa-se, também nas Figuras 16 e 17, que, para uma mesma quantidade de dados de teste, o índice de cobertura alcançado por um critério considerado fraco é maior do que o índice de cobertura de um critério considerado forte. A "força" de um critério de teste está associada à dificuldade de alcançar uma boa cobertura.

A Tabela 10 apresenta os resultados do processo de medição da cobertura dos critérios de teste utilizando os dados de teste gerados segundo o perfil operacional OP2. O crescimento da cobertura para os diferentes critérios de teste selecionados para o perfil operacional OP2 pode ser analisado graficamente na Figura 18. Para uma análise detalhada sobre o comportamento do crescimento da cobertura, para cada um dos critérios selecionados, a Figura 19 exibe a cobertura alcançada com a execução dos cem primeiros dados de teste utilizados.

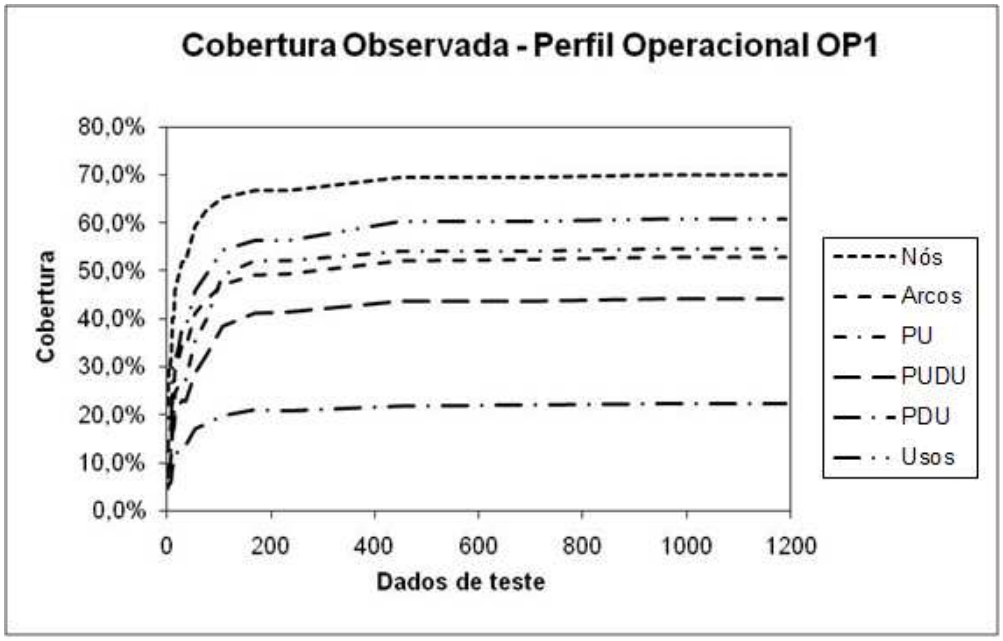

Figura 16 - Crescimento da cobertura – perfil OP1

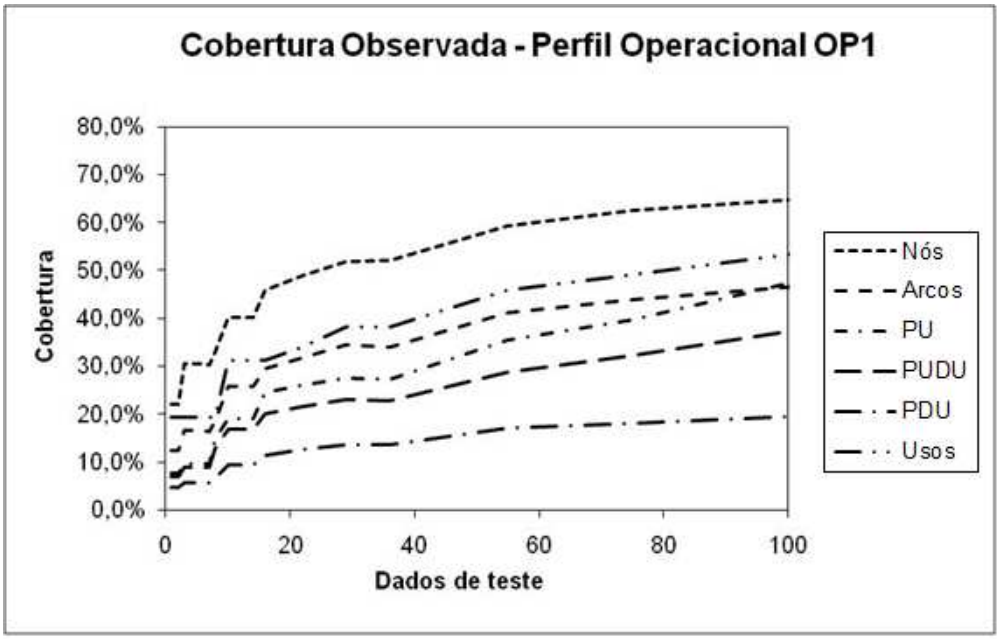

Figura 17 - Cobertura alcançada após 100 dados de teste

Também neste caso observa-se a repetição do efeito ocorrido no perfil operacional OP1, isto é, o crescimento da cobertura estabiliza-se após uma certa quantidade de dados de teste executados. O efeito é observado em todos os critérios de teste utilizados no

experimento. A estabilização é um indicativo de que o critério de teste atingiu seu limite de geração de dados de teste que poderiam exercitar novos elementos requeridos pelo critério.

Tabela 10 - Cobertura medida para o perfil OP2

Defeito Removido	Dados de Teste Acumulados	Dados de Teste Entre Falhas	Cobertura dos Critérios						Confiabilidade Medida
			Nós	Arcos	PU	PUDU	PDU	Usos	
E31	1	1	0,349217	0,211765	0,178928	0,142905	0,074157	0,227390	0,517587
E11	3	2	0,421375	0,271895	0,216980	0,178251	0,099775	0,304197	0,517587
E26	5	2	0,421375	0,271895	0,216980	0,178251	0,099775	0,304197	0,540610
E15	11	6	0,481280	0,326797	0,282767	0,226112	0,122921	0,366697	0,753358
E7	22	11	0,566372	0,386928	0,373922	0,296973	0,148764	0,450876	0,780004
E14	33	11	0,566372	0,386928	0,373922	0,296973	0,148764	0,450876	0,780004
E16	44	11	0,566372	0,386928	0,373922	0,296973	0,148764	0,450876	0,808996
E23	55	11	0,566372	0,386928	0,373922	0,296973	0,148764	0,450876	0,808996
E8	73	18	0,567052	0,388235	0,378488	0,301539	0,153708	0,452260	0,840546
E29	93	20	0,568414	0,392157	0,387959	0,309657	0,156404	0,455028	0,865487
E24	130	37	0,588155	0,418301	0,427194	0,338576	0,168989	0,491467	0,908975
E20	230	100	0,621511	0,460131	0,491121	0,386437	0,192584	0,528598	0,968237
E21	330	100	0,621511	0,460131	0,491121	0,386437	0,192584	0,528598	0,968237
E22	466	136	0,624234	0,462745	0,497040	0,392018	0,195281	0,533210	0,978256
E27	602	136	0,624234	0,462745	0,497040	0,392018	0,195281	0,533210	0,983586
E17	818	216	0,629680	0,469281	0,501776	0,396246	0,200000	0,541513	0,996163
E33	2428	1610	0,635807	0,478431	0,541011	0,423981	0,214607	0,558579	0,999147

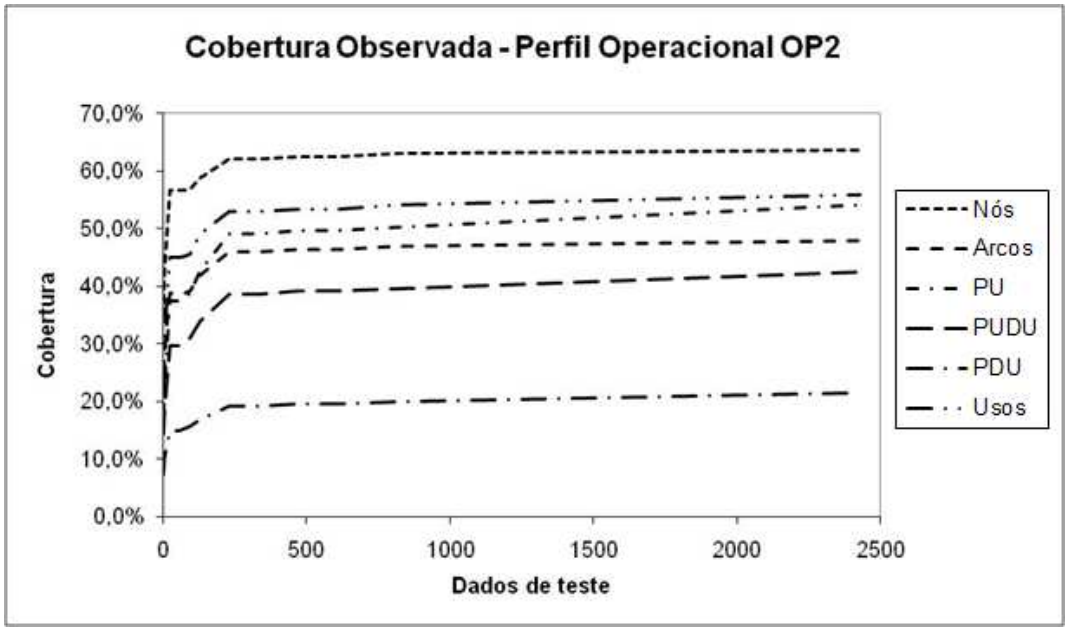

Figura 18 - Crescimento da cobertura – perfil OP2

A Tabela 11 apresenta os resultados do processo de medição da cobertura dos critérios de teste utilizando os dados de teste gerados segundo o perfil operacional OP3.

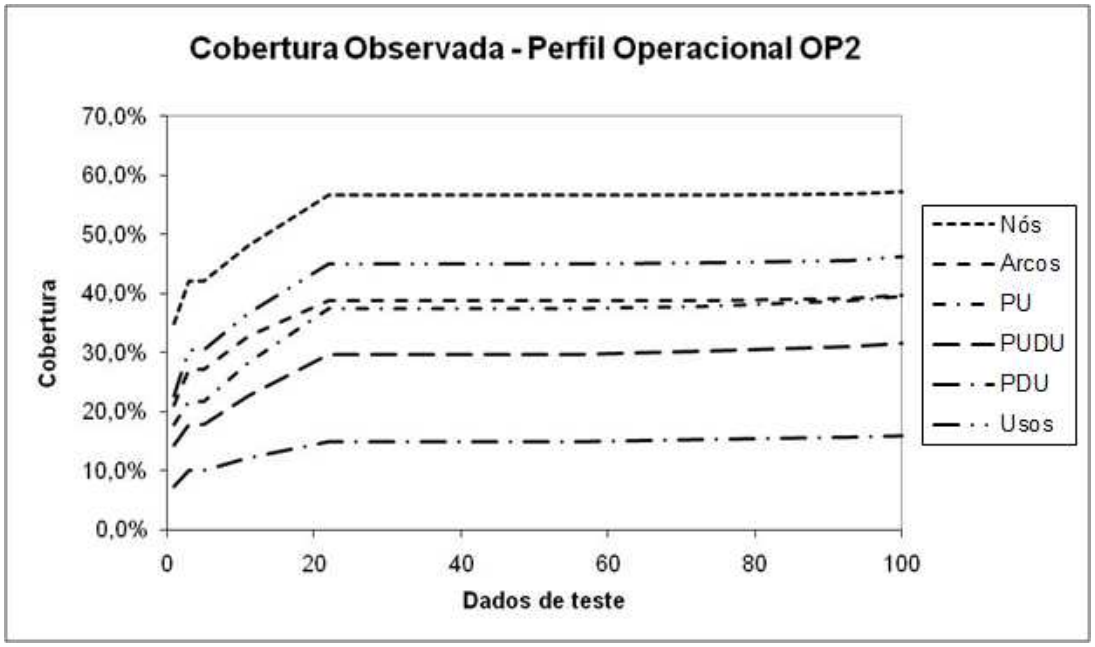

Figura 19 - Cobertura alcançada após 100 dados de teste

Tabela 11 - Cobertura medida para o perfil OP3

Defeito Removido	Dados de Teste Acumulados	Dados de Teste Entre Falhas	Cobertura dos Critérios						Confiabilidade Medida
			Nós	Arcos	PU	PUDU	PDU	Usos	
E4	1	1	0,317903	0,185621	0,146795	0,121258	0,062022	0,198339	0,294574
E10	3	2	0,388019	0,232679	0,239134	0,189075	0,080899	0,266144	0,325926
E14	5	2	0,388019	0,232679	0,239134	0,189075	0,080899	0,266144	0,334442
E15	8	3	0,464261	0,288889	0,290884	0,231186	0,095955	0,329336	0,363636
E16	11	3	0,464261	0,288889	0,290884	0,231186	0,095955	0,329336	0,363636
E23	14	3	0,464261	0,288889	0,290884	0,231186	0,095955	0,329336	0,365653
E11	18	4	0,524166	0,335948	0,322679	0,257737	0,113933	0,387223	0,423055
E17	23	5	0,533016	0,342484	0,327922	0,262642	0,118876	0,398524	0,440782
E26	29	6	0,590878	0,393464	0,391848	0,309488	0,141124	0,460793	0,474541
E19	37	8	0,605854	0,413072	0,398792	0,316421	0,151910	0,476476	0,590761
E7	50	13	0,611300	0,424837	0,421106	0,334348	0,160000	0,485701	0,606989
E8	63	13	0,611300	0,424837	0,421106	0,334348	0,160000	0,485701	0,619681
E29	76	13	0,611300	0,424837	0,421106	0,334348	0,160000	0,485701	0,650122
E13	90	14	0,622192	0,436601	0,426349	0,339591	0,162697	0,497232	0,651462
E31	104	14	0,622192	0,436601	0,426349	0,339591	0,162697	0,497232	0,899123
E24	136	32	0,659632	0,475817	0,486555	0,386268	0,181498	0,546587	0,945055
E20	174	38	0,665759	0,481046	0,491121	0,390157	0,184944	0,552122	0,953705
E21	212	38	0,665759	0,481046	0,491121	0,390157	0,184944	0,552122	0,953705
E27	276	64	0,671205	0,494118	0,516320	0,410959	0,200899	0,575646	0,980702
E12	463	187	0,701838	0,533333	0,561644	0,447150	0,229438	0,611624	0,983602
E18	650	187	0,701838	0,533333	0,561644	0,447150	0,229438	0,611624	0,983602
E22	926	276	0,703880	0,533595	0,567056	0,452055	0,235955	0,618008	0,990801
E32	1634	708	0,703880	0,535948	0,573313	0,455606	0,239101	0,620618	0,998800

O crescimento da cobertura para os diferentes critérios de teste selecionados para o perfil operacional OP3 pode ser analisado graficamente na Figura 20. Para uma análise detalhada sobre o comportamento do crescimento da cobertura, para cada um dos critérios selecionados, a Figura 21 exibe a cobertura para os cem primeiros dados de teste utilizados. Novamente observa-se o efeito da estabilização do crescimento da cobertura.

Os índices de cobertura, apresentados no perfil OP3, no início dos testes estão entre os medidos em OP1 e OP2. Nota-se, na Figura 21, que com cerca de quarenta dados de teste, a taxa de crescimento da cobertura diminui, estabilizando o índice de crescimento.

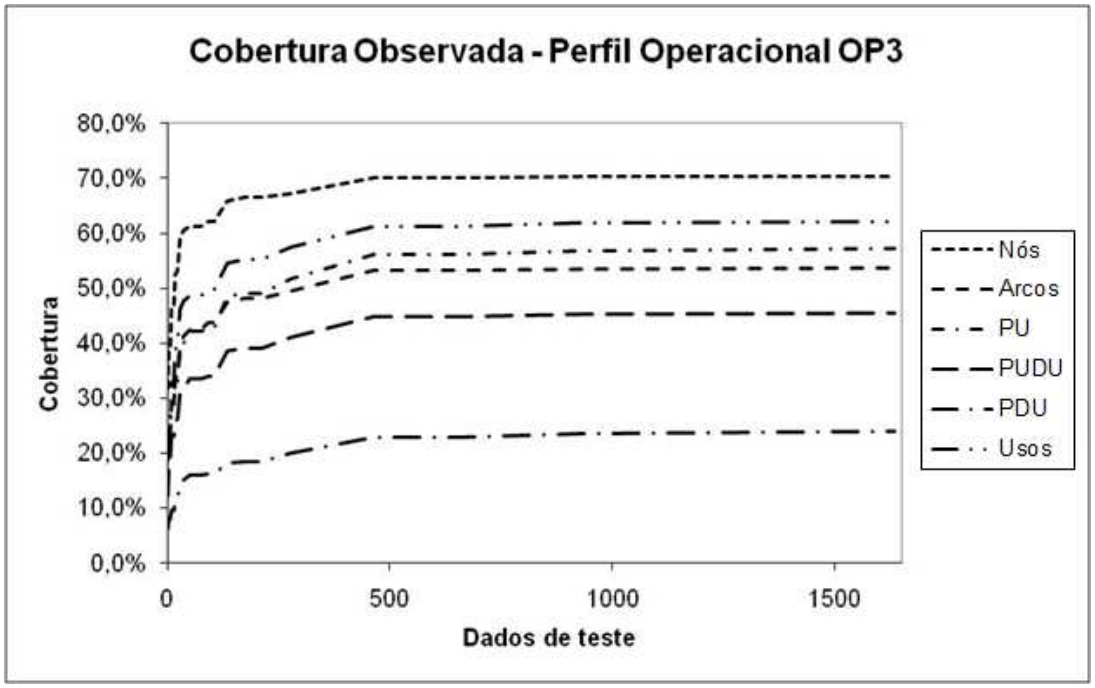

Figura 20 - Crescimento da cobertura – perfil OP3

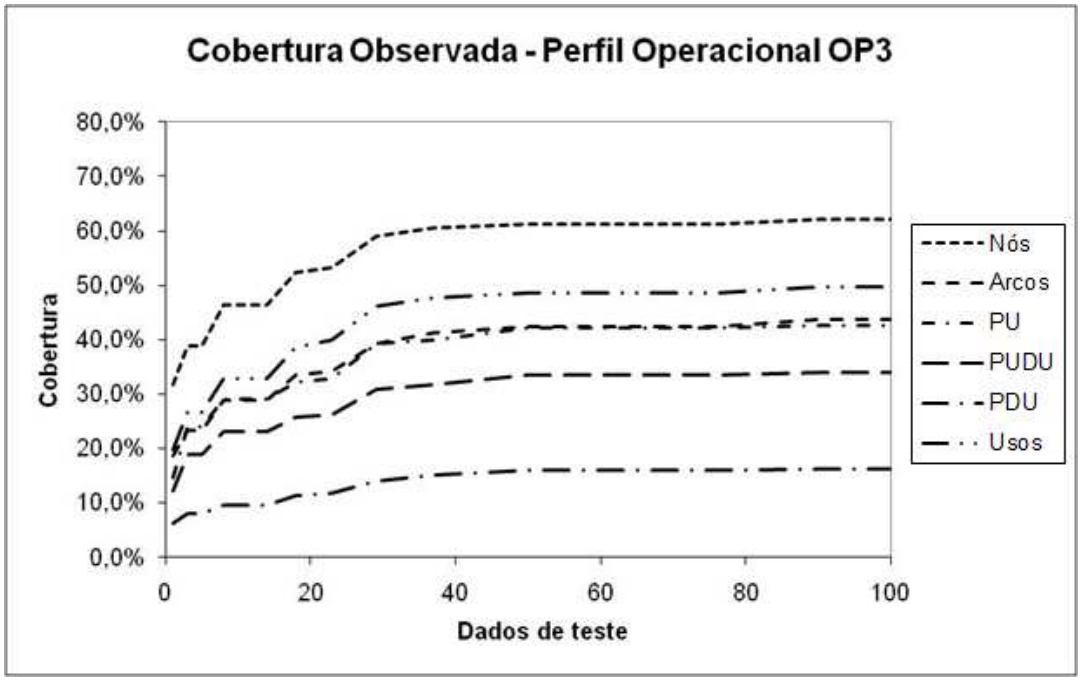

Figura 21 - Cobertura alcançada após 100 dados de teste

5.3 Estimação da Confiabilidade pelo modelo MBBC

O modelo MBBC estima a confiabilidade do software utilizando a informação de cobertura alcançada pelos elementos requeridos do critério de teste. A expressão a seguir, apresentada no Capítulo 3, representa a forma funcional do modelo MBBC:

$$R(k_i|n_i) = exp\{-[N-i][(n_i + k_i)^{\alpha_i} - (n_i)^{\alpha_i}]\}$$

A cobertura normalizada, α_i, é descrita por meio da função linear $\alpha_i = \alpha_0 + \alpha_1 c_i$. A variável c_i representa o complemento da cobertura atingida com a aplicação dos n_i dados de teste. Os parâmetros α_0 e α_1 são estimados por meio da otimização da função de verossimilhança da função densidade de probabilidade do modelo MBBC.

No Apêndice B, é apresentado o programa para estimar os parâmetros do modelo. Os resultados foram calculados separadamente para cada perfil operacional selecionado no

experimento. A função de otimização utilizada, CONSTR, do software MATLAB, obtém mínimos locais de funções, que, neste caso, maximizam os parâmetros da função de verossimilhança do modelo, determinando os parâmetros mais prováveis com base nos dados de cobertura observados durante os testes. A determinação de mínimos locais envolve a definição de intervalos de valores, mínimo e máximo, onde a função de otimização CONSTR será aplicada. Para alguns dos perfis operacionais, a definição inicial dos intervalos fez com que CONSTR convergisse rapidamente para os valores dos estimadores; em outros perfis foi necessário ampliar o tamanho do intervalo, pois a função não convergia para nenhuma solução; em alguns casos, foram encontrados valores melhores, para os estimadores, em outros intervalos.

5.3.1 Resultados para o Perfil Operacional OP1

Os valores dos parâmetros, que normalizam a cobertura são calculados para cada um dos critérios de teste utilizados, são apresentados na Tabela 12. A cobertura normalizada é apresentada na Tabela 13 para todos os critérios de teste utilizados.

Tabela 12 - Parâmetros do MBBC – Perfil OP1

Critérios de teste	Parâmetros do modelo **MBBC** - Perfil OP1		
	N	α0	α1
Nós	29,0010	0,2158	0,1201
Arcos	29,0010	0,1924	0,1371
PU	29,0010	0,1875	0,1388
PUDU	29,0010	0,1798	0,1434
PDU	29,0010	0,1649	0,1524
Usos	29,0090	0,1996	0,1325

A aplicação da cobertura normalizada resulta em valores estimados muito próximos, independentemente do critério de teste utilizado.

Tabela 13 - Confiabilidade estimada pelo MBBC – Perfil OP1

Critérios de Teste					
Nós		**Arcos**		**PU**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,309258	0,000000	0,312500	0,000000	0,315621	0,000000
0,309258	0,001238	0,312500	0,001145	0,315621	0,001062
0,299402	0,011453	0,306833	0,010785	0,312951	0,010176
0,299402	0,038785	0,306833	0,034689	0,312951	0,031601
0,299402	0,073092	0,306833	0,066471	0,312951	0,061389
0,299402	0,113015	0,306833	0,104062	0,312951	0,097097
0,299639	0,156023	0,307013	0,145077	0,312971	0,136472
0,296417	0,200081	0,303930	0,187629	0,305659	0,177934
0,287497	0,278640	0,294137	0,263802	0,299896	0,260445
0,287497	0,364406	0,294137	0,350048	0,299896	0,337770
0,287513	0,425118	0,294090	0,410492	0,299911	0,397906
0,280730	0,479780	0,289006	0,465384	0,292270	0,452676
0,277173	0,591362	0,286100	0,574234	0,290311	0,567434
0,273534	0,674458	0,282287	0,657790	0,288145	0,649812
0,273203	0,735743	0,282832	0,721268	0,288501	0,711362
0,264684	0,811200	0,273026	0,798424	0,277146	0,790656
0,260962	0,865394	0,269212	0,856768	0,271380	0,852372
0,257653	0,904726	0,265036	0,898224	0,258297	0,896463
0,255751	0,938914	0,262124	0,934878	0,254161	0,938570
0,255669	0,956506	0,261938	0,953911	0,253921	0,957135
0,252590	0,975812	0,257993	0,974273	0,251184	0,976227
0,252509	0,984856	0,257814	0,983983	0,251161	0,985077
0,251936	0,989466	0,257097	0,988851	0,250386	0,989618
0,251936	0,992466	0,257097	0,992027	0,250386	0,992594

Tabela 13 (cont.) - Confiabilidade estimada pelo MBBC – Perfil OP1

Critérios de Teste					
PUDU		**PDU**		**Usos**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,313165	0,000000	0,308205	0,000000	0,306462	0,000000
0,313165	0,001127	0,310111	0,001270	0,306462	0,001324
0,310302	0,010652	0,308688	0,011273	0,306462	0,012059
0,310302	0,032908	0,308688	0,033727	0,306462	0,034885
0,310302	0,063549	0,308688	0,064895	0,306462	0,066791
0,310302	0,100068	0,308688	0,101912	0,306462	0,104497
0,310321	0,140153	0,308705	0,142429	0,306462	0,145612
0,304813	0,182205	0,307017	0,184842	0,306462	0,188544
0,299095	0,262085	0,302950	0,257824	0,290816	0,258896
0,299095	0,339468	0,302950	0,331326	0,290816	0,357205
0,299112	0,399651	0,302974	0,391272	0,290816	0,417794
0,294258	0,454419	0,300017	0,446008	0,290816	0,472547
0,292211	0,563283	0,298096	0,551211	0,287027	0,570468
0,290158	0,646189	0,296401	0,634874	0,281435	0,656042
0,290342	0,707918	0,296440	0,697111	0,281435	0,722696
0,281839	0,788094	0,291253	0,779471	0,271412	0,800311
0,277115	0,847252	0,289870	0,836617	0,267133	0,858467
0,267903	0,891696	0,287027	0,880515	0,260044	0,899892
0,264325	0,933256	0,285309	0,921613	0,257599	0,937629
0,263760	0,952986	0,285657	0,943396	0,257599	0,955766
0,260679	0,973810	0,283909	0,967682	0,252343	0,975347
0,260655	0,983534	0,283841	0,979171	0,252282	0,984895
0,259952	0,988508	0,283361	0,985357	0,251702	0,989492
0,259952	0,991775	0,283361	0,989430	0,251702	0,992485

O exemplo a seguir mostra como calcular a confiabilidade utilizando o modelo MBBC. Para isso será utilizada a medida da cobertura observada para o critério Todos-Nós na ocorrência da quinta falha do programa **Space**. O cálculo é feito em duas etapas. Na primeira etapa calcula-se a cobertura normalizada e, a seguir calcula-se a confiabilidade utilizando o valor da cobertura normalizada calculada anteriormente.

1) Cálculo da cobertura normalizada:
    a. A cobertura normalizada é dada pela expressão: $\alpha_i = \alpha_0 + \alpha_1 * (1 - c_i)$.
    b. Onde α_0 e α_1 são parâmetros estimados que normalizam a cobertura e são calculados para cada critério de teste. O parâmetro c_i representa a cobertura medida para o critério na ocorrência de uma falha.
    c. Na Tabela 12 são apresentados os valores estimados para α_0 e α_1.
    d. Na Tabela 9 são apresentadas as medidas da cobertura observada para o critério Todos-Nós.
    e. Considerando o critério Todos-Nós e a ocorrência da 5ª. falha, a cobertura normalizada, α_5, será calculada pela expressão: $\alpha_5 = 0{,}2158 + 0{,}1201 * (1 - 0{,}303894)$.
    f. Ou seja, $\alpha_5 = 0{,}299402307$.
2) Cálculo da confiabilidade:
    a. Conforme apresentado na seção 3.2 do Capítulo 3, o MBBC estima a confiabilidade utilizando a expressão:

$$R(k_i|n_i) = exp\{-[N - i][(n_i + k_i)^{\alpha_i} - (n_i)^{\alpha_i}]\} \qquad (5.1)$$

    b. A Tabela 12 apresenta a estimativa para o parâmetro *N*, que é calculada para cada critério de teste.
    c. Aplicando os valores na expressão 5.1:

$$R(1/4) = exp\{-[29{,}001 - 4] * [5^{0{,}299402307} - 4^{0{,}299402307}]\}$$

d. Ou seja, a confiabilidade do software (R), após a execução de um dado de teste e tendo última falha ocorrida após a execução de quatro dados de teste, é R= 0, 073092.

A Figura 22 exibe o gráfico da confiabilidade medida versus a confiabilidade estimada pelo modelo MBBC para todos os critérios de teste segundo o perfil operacional OP1. Observa-se a diferença mínima entre as confiabilidades estimadas para os diferentes critérios de teste utilizados. Isto acontece devido à utilização da cobertura normalizada ao invés da cobertura observada. Nota-se também que a confiabilidade estimada possui um bom ajuste em relação à confiabilidade medida, ou seja, ela pode ser utilizada para estimar a confiabilidade do software.

Após o início dos testes, quando muitas falhas ocorrem e, consequentemente, muitos defeitos são corrigidos, o software tende a ficar mais estável, isto é, sua taxa de falhas diminui e aumenta o tempo entre falhas. O gráfico do crescimento da confiabilidade do software para o modelo MBBC - Figura 22 – mostra que, após a aplicação de cerca de dez dados de testes a curva de crescimento da confiabilidade do modelo MBBC apresenta estimativas da confiabilidade inferiores às calculadas. Esta é uma propriedade desejável para os modelos de confiabilidade de software; isto é, em se tratando de confiabilidade, é preferível um modelo que seja conservador em suas estimativas do que um modelo que superestime a confiabilidade. Por exemplo, quando se estima o tempo para a ocorrência da próxima falha, é melhor trabalhar com uma previsão de tempo de falha anterior ao tempo real, do que posterior, quando a falha já pode ter ocorrido. Será observado se o mesmo comportamento ocorre nos demais perfis operacionais utilizados no experimento.

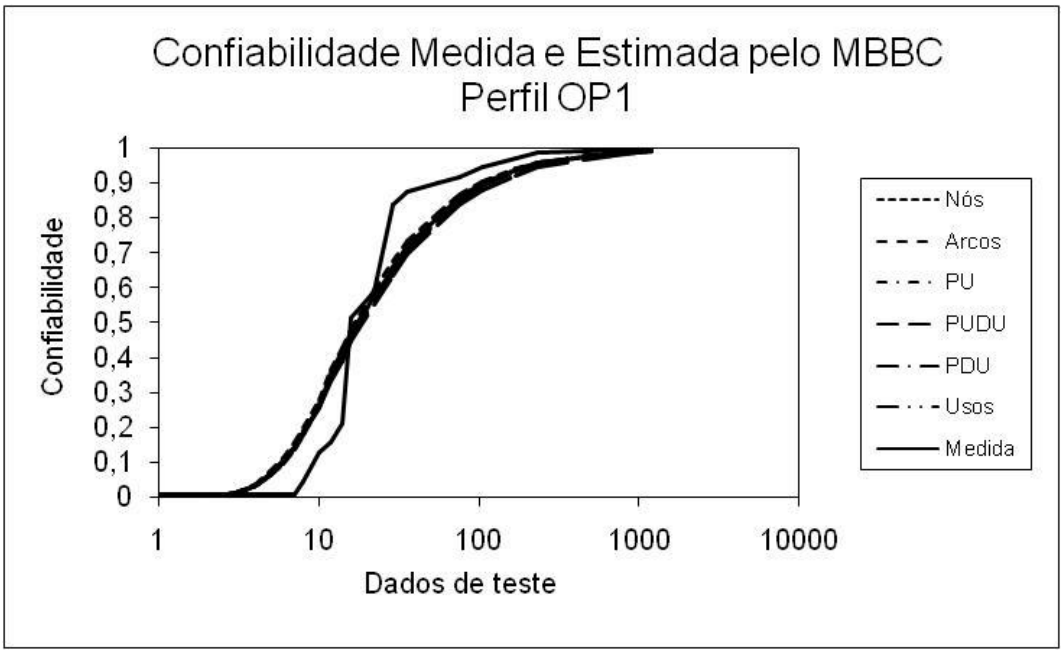

Figura 22 - Confiabilidade medida e estimada pelo MBBC - Perfil OP1

5.3.2 Resultados para o Perfil Operacional OP2

Os valores dos parâmetros que normalizam a cobertura são calculados para cada um dos critérios de teste utilizados. Os valores de α_0 e α_1 obtidos que normalizam a cobertura são apresentados na Tabela 14.

Tabela 14 - Parâmetros do MBBC – Perfil OP2

Critérios de teste	Parâmetros do modelo **MBBC** - Perfil OP2		
	N	α0	α1
Nós	40,0000	0,2000	-0,1500
Arcos	38,0000	0,2355	-0,1180
PU	40,0000	0,2250	-0,1020
PUDU	38,9500	0,6000	-0,5708
PDU	40,5000	1,2002	-1,1784
Usos	37,0000	0,1995	-0,0500

A Tabela 15 apresenta a confiabilidade estimada pelo modelo MBBC utilizando a cobertura normalizada. Nota-se que são mínimas as diferenças dos valores da cobertura normalizada entre os critérios de teste utilizados. Percebe-se que, devido ao uso da

cobertura normalizada, as confiabilidades estimadas são aproximadamente iguais entre os critérios utilizados.

Tabela 15 - Confiabilidade estimada pelo MBBC – Perfil OP2

Critérios de teste					
Nós		**Arcos**		**PU**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,0911002	0,0000000	0,0945882	0,0000000	0,0848217	0,0000000
0,0911002	0,2773144	0,1231503	0,2980047	0,1039237	0,3049825
0,0977768	0,4856201	0,1231503	0,3963955	0,1039237	0,4317147
0,1594737	0,7034193	0,1492287	0,6548873	0,1369489	0,6842992
0,1672012	0,7040458	0,1777908	0,7517960	0,1827087	0,7548570
0,1672012	0,7727286	0,1777908	0,7757113	0,1827087	0,7427709
0,1756088	0,8228156	0,1777908	0,8254335	0,1827087	0,7977572
0,1756088	0,8444523	0,1777908	0,8583491	0,1827087	0,8344627
0,1847583	0,8794286	0,1784117	0,8908807	0,1850010	0,8712682
0,1919913	0,8952552	0,1802745	0,9131620	0,1897553	0,8950161
0,2046027	0,9169046	0,1926928	0,9352803	0,2094515	0,9186831
0,2217888	0,9420989	0,2125621	0,9555465	0,2415428	0,9392352
0,2217888	0,9501484	0,2125621	0,9609622	0,2415428	0,9394606
0,2246943	0,9634651	0,2138039	0,9717572	0,2445143	0,9552676
0,2262398	0,9705481	0,2138039	0,9779063	0,2445143	0,9637431
0,2298872	0,9775430	0,2169085	0,9836788	0,2468914	0,9726333
0,2307527	0,9903858	0,2212549	0,9932385	0,2665877	0,9882241

A Figura 23 exibe o gráfico da confiabilidade medida versus a confiabilidade estimada pelo modelo MBBC para o perfil OP2. Nota-se que, para os diferentes critérios de teste utilizados, o modelo produz estimativas da confiabilidade do software muito próximas. Isso ocorre devido ao fato de ter sido utilizada a cobertura normalizada como parâmetro do modelo, ao invés da cobertura medida durante os testes. O comportamento conservador das estimativas calculadas pelo modelo também é observado para este perfil operacional.

Tabela 15 (cont.) - Confiabilidade estimada pelo MBBC – Perfil OP2

Critérios de teste					
PUDU		**PDU**		**Usos**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,0857704	0,0000000	0,0789870	0,0000000	0,0823253	0,0000000
0,1059458	0,3006511	0,1091751	0,3328570	0,1168888	0,3282315
0,1059458	0,4236137	0,1091751	0,4108995	0,1168888	0,3936700
0,1332647	0,6780163	0,1364505	0,6679922	0,1450138	0,6536216
0,1737120	0,7629325	0,1669035	0,7559536	0,1828943	0,7447383
0,1737120	0,7603263	0,1669035	0,7732896	0,1828943	0,7503435
0,1737120	0,8124866	0,1669035	0,8232814	0,1828943	0,8041608
0,1737120	0,8470654	0,1669035	0,8562529	0,1828943	0,8400339
0,1763184	0,8815561	0,1727293	0,8890125	0,1835170	0,8758741
0,1809520	0,9033605	0,1759069	0,9066851	0,1847624	0,9005644
0,1974592	0,9254368	0,1907363	0,9291333	0,2011600	0,9258003
0,2247780	0,9461348	0,2185412	0,9497413	0,2178690	0,9465437
0,2247780	0,9486340	0,2185412	0,9517542	0,2178690	0,9543104
0,2279636	0,9623090	0,2217189	0,9646889	0,2199446	0,9666616
0,2279636	0,9695042	0,2217189	0,9714727	0,2199446	0,9734661
0,2303769	0,9771138	0,2272800	0,9786371	0,2236808	0,9802063
0,2462084	0,9903287	0,2444925	0,9906846	0,2313607	0,9916063

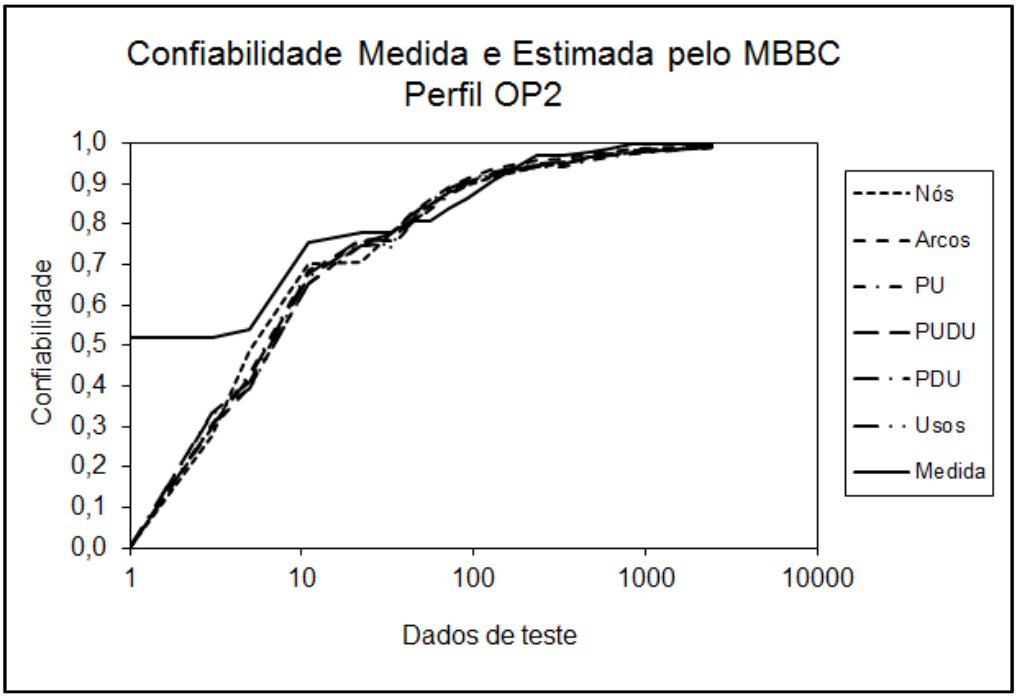

Figura 23 - Confiabilidade medida e estimada pelo MBBC - Perfil OP2

5.3.3 Resultados para o Perfil Operacional OP3

Os valores dos parâmetros que normalizam a cobertura são calculados para cada um dos critérios de teste utilizados e apresentados na Tabela 16.

Tabela 16 - Parâmetros do MBBC – Perfil OP3

Critérios de teste	Parâmetros do modelo **MBBC** - Perfil OP3		
	N	α0	α1
Nós	23,0028	0,4589	-0,3075
Arcos	22,6227	0,5752	-0,3848
PU	22,3553	0,5734	-0,3781
PUDU	22,5100	0,7399	-0,5895
PDU	22,9050	1,2535	-1,0756
Usos	22,7500	0,5657	-0,4110

A Tabela 17 apresenta a confiabilidade estimada pelo modelo MBBC utilizando as medidas de cobertura normalizada do perfil operacional OP3. Nota-se que são mínimas as diferenças dos valores da cobertura normalizada entre os critérios de teste utilizados. A coluna "Confiabilidade estimada" apresenta o cálculo da confiabilidade estimada utilizando-se o modelo MBBC. Percebe-se que, devido ao uso da cobertura normalizada, as confiabilidades estimadas são aproximadamente iguais para todos os critérios utilizados.

A Figura 24 exibe o gráfico da confiabilidade medida versus a confiabilidade estimada pelo modelo MBBC segundo o perfil operacional OP3. Nota-se que, para os diferentes critérios de teste utilizados, o modelo produz estimativas da confiabilidade do software muito próximas. Isso ocorre devido ao fato de ter sido utilizada a cobertura normalizada como parâmetro do modelo, em vez da cobertura medida durante os testes.

Observa-se, no final dos testes, quando o software tende a ficar mais estável devido à redução das falhas, que as estimativas da confiabilidade são inferiores à confiabilidade medida. Esta é uma característica importante para um modelo de confiabilidade de software, pois, quando o assunto é previsão de falhas, é melhor um estimador conservador, ou seja, que fornece uma estimativa de ocorrência de um evento de falha antes dele ocorrer. Esta característica também foi observada na aplicação dos modelos nos perfis operacionais

OP1 e OP2. A maioria dos modelos tradicionais superestima a confiabilidade do software e, por isso, são criticados pelos pesquisadores [Chen97], [Lyu07].

Tabela 17 - Confiabilidade estimada pelo MBBC – Perfil OP3

Critérios de teste					
Nós		**Arcos**		**PU**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,249155	0,000000	0,261827	0,000000	0,250803	0,000000
0,270716	0,062046	0,279935	0,054820	0,285717	0,065878
0,270716	0,149051	0,279935	0,140920	0,285717	0,136520
0,294160	0,287385	0,301564	0,275660	0,305283	0,269155
0,294160	0,345189	0,301564	0,337097	0,305283	0,334703
0,294160	0,430098	0,301564	0,421815	0,305283	0,419395
0,312581	0,515085	0,319673	0,507045	0,317305	0,504741
0,315302	0,555206	0,322188	0,548490	0,319287	0,558986
0,333095	0,620453	0,341805	0,614797	0,343458	0,625820
0,337700	0,654723	0,349350	0,646438	0,346083	0,648985
0,339375	0,717980	0,353877	0,705959	0,354520	0,716513
0,339375	0,767057	0,353877	0,752527	0,354520	0,756522
0,339375	0,806889	0,353877	0,794543	0,354520	0,798331
0,342724	0,840017	0,358404	0,829732	0,356502	0,833333
0,342724	0,863984	0,358404	0,854289	0,356502	0,860313
0,354237	0,896845	0,373494	0,889476	0,379266	0,894608
0,356121	0,915011	0,375506	0,906821	0,380993	0,906476
0,356121	0,934228	0,375506	0,928165	0,380993	0,928500
0,357796	0,953299	0,380536	0,949366	0,390521	0,950065
0,367215	0,972557	0,395627	0,969674	0,407658	0,969359
0,367215	0,981857	0,395627	0,979503	0,407658	0,979496
0,367843	0,990287	0,395727	0,989708	0,409704	0,990381
0,367843	0,996573	0,396633	0,997185	0,412070	0,998155

Tabela 17 (cont.) - Confiabilidade estimada pelo MBBC – Perfil OP3

Critérios de teste					
PUDU		**PDU**		**Usos**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,221882	0,000000	0,244611	0,000000	0,236217	0,000000
0,261860	0,094277	0,264915	0,066805	0,264085	0,076145
0,261860	0,169604	0,264915	0,159143	0,264085	0,162618
0,286684	0,314857	0,281109	0,301150	0,290057	0,305718
0,286684	0,370777	0,281109	0,375053	0,290057	0,358819
0,286684	0,456403	0,281109	0,460635	0,290057	0,444157
0,302336	0,540895	0,300446	0,544857	0,313849	0,528920
0,305227	0,586087	0,305763	0,582006	0,318493	0,557792
0,332843	0,649179	0,329692	0,640576	0,344086	0,619337
0,336930	0,665007	0,341295	0,662838	0,350532	0,639579
0,347498	0,728275	0,349996	0,713898	0,354323	0,701503
0,347498	0,763923	0,349996	0,753185	0,354323	0,749517
0,347498	0,804634	0,349996	0,794832	0,354323	0,791767
0,350589	0,838599	0,352896	0,829663	0,359063	0,827174
0,350589	0,863619	0,352896	0,855472	0,359063	0,851638
0,378105	0,897066	0,373119	0,890116	0,379347	0,887202
0,380398	0,905132	0,376826	0,903318	0,381622	0,900905
0,380398	0,926847	0,376826	0,923929	0,381622	0,923067
0,392660	0,948584	0,393987	0,945786	0,391290	0,945390
0,413995	0,967378	0,424684	0,963362	0,406077	0,965591
0,413995	0,976910	0,424684	0,970715	0,406077	0,976432
0,416886	0,988645	0,431693	0,984227	0,408701	0,987777
0,418980	0,997160	0,435077	0,994185	0,409774	0,996148

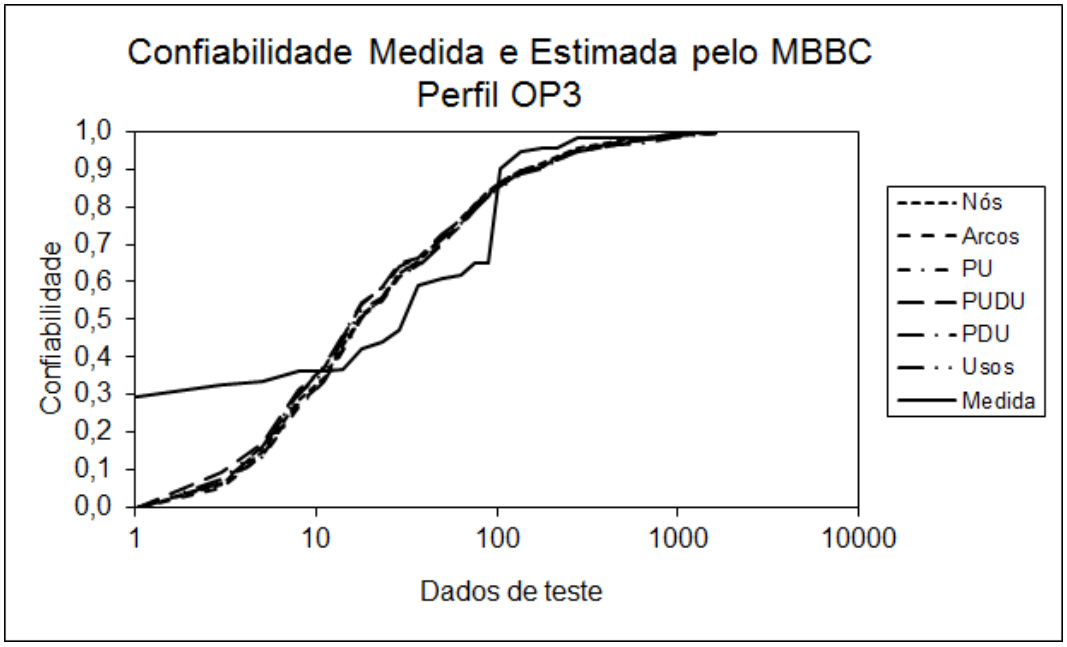

Figura 24 - Confiabilidade medida e estimada pelo MBBC - Perfil OP3

5.3.4 Análise dos Resultados Obtidos com a Utilização do MBBC

Os resultados obtidos com a aplicação do modelo MBBC para os perfis operacionais OP1, OP2 e OP3 indicam que o modelo é robusto à variação do perfil operacional do software. Conforme pode ser observado nas Figuras 22, 23 e 24, apresentadas anteriormente. Os valores das confiabilidade medida e estimada pelo MBBC são bastante próximos, indicando que o modelo ajusta-se bem aos dados.

A análise gráfica também mostra que são próximos os valores da confiabilidade estimada pelo MBBC utilizando os diferentes critérios de teste, a diferença é pequena entre eles; qualquer um dos seis critérios de teste aplicado poderia ser utilizado para se obter a cobertura do código utilizada pelo modelo. Este resultado é devido ao uso da cobertura normalizada no lugar da cobertura observada. Para qualquer um dos critérios de teste utilizados, os valores da confiabilidade estimada pelo modelo serão aproximadamente iguais.

Para confirmar a hipótese de igualdade entre as medidas da confiabilidade, é aplicado o teste de Kolmogorov-Smirnov. O teste é realizado sob a hipótese de igualdade entre elas: se esta hipótese não puder ser comprovada, então ela é rejeitada em favor da hipótese alternativa, ou seja, os valores estimados provêm de populações diferentes. Em outras palavras, o modelo não pode ser utilizado para estimar a confiabilidade do software.

Os resultados do teste, apresentados na Tabela 18, não rejeitam a hipótese de igualdade (H_0) entre os valores medido e estimado para todos os perfis utilizados. Isso indica que o modelo pode ser utilizado para estimar a confiabilidade do software e que a variação do perfil operacional não afetou a capacidade preditiva do modelo MBBC. Observa-se que a utilização da cobertura de código diretamente na forma funcional do MBBC resultou na sensibilidade do modelo à variação do perfil operacional do software, indicando que o uso da cobertura é importante na análise da confiabilidade de software por meio de modelos e não deve ser desprezada.

Tabela 18 - Teste de igualdade entre as medidas de confiabilidade

Perfil Operacional	Nível de Significância	Valor do Teste (p-value)	Conclusão
OP1	1%	0,6216	Não rejeita H_0
OP2	1%	0,9303	Não rejeita H_0
OP3	1%	0,8420	Não rejeita H_0

5.4 Estimação da Confiabilidade pelo MFIBC

O modelo MFIBC estima a confiabilidade do software utilizando a informação de cobertura alcançada pelos elementos requeridos do critério de teste. A expressão a seguir, apresentada no Capítulo 3, representa a forma funcional do modelo MFIBC:

$$R(k_i|n_{i-1}) = \exp[-D\left[(\emptyset_i)^{(i-1)}\right]$$

A cobertura normalizada, $\emptyset_i$, é descrita por meio da função linear $\emptyset_i = \alpha_0 + \alpha_1 c_i$, onde a variável c_i representa o complemento da cobertura atingida com a aplicação dos n_i

dados de teste. Os parâmetros α_0 e α_1 são estimados por meio da otimização da função de verossimilhança da função densidade de probabilidade do modelo MFIBC.

A seguir, são apresentados os resultados para os três perfis operacionais selecionados. O Apêndice B apresenta o programa desenvolvido para estimar os parâmetros do modelo. As mesmas situações observadas durante o processo de estimação dos parâmetros para o modelo MBBC, discutidas na seção 5.3, forma observadas para a determinação dos parâmetros do modelo MFIBC.

5.4.1 Resultados para o Perfil Operacional OP1

Os valores dos parâmetros que normalizam a cobertura são calculados para cada um dos critérios de teste utilizados. Os valores para α_0 e α_1 são apresentados na Tabela 19.

Tabela 19 - Parâmetros do MFIBC – Perfil OP1

Critérios de teste	Parâmetros do modelo **MFIBC** - Perfil OP1		
	D	α0	α1
Nós	3,9748	0,6521	0,3321
Arcos	3,8486	0,5786	0,3588
PU	3,9067	0,6375	0,2431
PUDU	3,8905	0,5649	0,3239
PDU	3,8504	0,1552	0,7634
Usos	3,8250	0,5350	0,4560

A Tabela 20 apresenta a confiabilidade estimada pelo modelo MFIBC utilizando as medidas de cobertura normalizada.

Nota-se que são mínimas as diferenças dos valores da cobertura normalizada entre os critérios de teste utilizados. A utilização da cobertura normalizada diretamente na forma funcional do modelo gera estimativas da confiabilidade muito próximas, independentemente do critério de teste utilizado, conforme é apresentado pela coluna "Confiabilidade estimada".

Tabela 20 - Confiabilidade estimada pelo MFIBC – Perfil OP1

Critérios de teste					
Nós		**Arcos**		**PU**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,910528	0,018783	0,892909	0,021310	0,861897	0,020107
0,910528	0,026805	0,892909	0,032179	0,861897	0,034487
0,883277	0,037055	0,878078	0,046494	0,857221	0,054905
0,883277	0,064629	0,878078	0,073861	0,857221	0,085361
0,883277	0,088978	0,878078	0,101480	0,857221	0,121298
0,883277	0,118011	0,878078	0,134129	0,857221	0,163931
0,883930	0,151444	0,878549	0,171355	0,857255	0,212222
0,875023	0,187144	0,870481	0,211238	0,844449	0,264697
0,850355	0,255110	0,844852	0,281182	0,834355	0,364156
0,850355	0,396889	0,844852	0,429997	0,834355	0,465085
0,850399	0,455750	0,844730	0,490154	0,834382	0,527963
0,831644	0,512424	0,831423	0,548017	0,820999	0,586772
0,821810	0,647214	0,823820	0,657098	0,817568	0,693244
0,811746	0,733458	0,813840	0,733573	0,813774	0,752127
0,810831	0,807037	0,815265	0,806378	0,814397	0,803960
0,787273	0,842730	0,789603	0,835438	0,794510	0,835588
0,776981	0,917080	0,779623	0,915880	0,784410	0,906204
0,767832	0,946971	0,768693	0,945642	0,761497	0,938984
0,762572	0,966372	0,761073	0,966770	0,754252	0,971453
0,762346	0,977214	0,760586	0,978731	0,753833	0,981776
0,753832	0,982679	0,750261	0,983976	0,749039	0,986375
0,753606	0,989533	0,749792	0,990822	0,748998	0,990996
0,752024	0,992152	0,747916	0,993200	0,747641	0,993256
0,752024	0,994358	0,747916	0,995182	0,747641	0,995150

Tabela 20 (cont.) - Confiabilidade estimada pelo MFIBC – Perfil OP1

Critérios de teste					
PUDU		**PDU**		**Usos**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,866135	0,020435	0,882587	0,021271	0,902766	0,021818
0,866135	0,034400	0,882587	0,033430	0,902766	0,031648
0,859667	0,054009	0,875461	0,049822	0,902766	0,044276
0,859667	0,084441	0,875461	0,075505	0,902766	0,059952
0,859667	0,119451	0,875461	0,104163	0,902766	0,078821
0,859667	0,160950	0,875461	0,138053	0,902766	0,100908
0,859710	0,207979	0,875548	0,176662	0,902766	0,126119
0,847268	0,259131	0,867092	0,219001	0,902766	0,154246
0,834353	0,355876	0,846719	0,292193	0,848921	0,184990
0,834353	0,466571	0,846719	0,422608	0,848921	0,416503
0,834392	0,529371	0,846839	0,482253	0,848921	0,475429
0,823427	0,588030	0,832025	0,538771	0,848921	0,531951
0,818804	0,685220	0,822405	0,654567	0,835880	0,585176
0,814168	0,748788	0,813914	0,738506	0,816635	0,689378
0,814583	0,803501	0,814106	0,806075	0,816635	0,798996
0,795376	0,835698	0,788127	0,838546	0,782140	0,832558
0,784706	0,905010	0,781200	0,918219	0,767417	0,927724
0,763899	0,938850	0,766959	0,943772	0,743018	0,958411
0,755819	0,969936	0,758351	0,968057	0,734605	0,981942
0,754542	0,981128	0,760094	0,980108	0,734605	0,989153
0,747583	0,986175	0,751338	0,984173	0,716517	0,992021
0,747529	0,991390	0,750995	0,990538	0,716306	0,996520
0,745940	0,993567	0,748593	0,992953	0,714308	0,997521
0,745940	0,995416	0,748593	0,995079	0,714308	0,998334

O exemplo a seguir mostra como calcular a confiabilidade utilizando o modelo MFIBC. Para isso será utilizada a medida da cobertura observada para o critério Todos-Nós na ocorrência da quinta falha do programa **Space**. O cálculo é realizado em duas etapas. Na primeira calcula-se a cobertura normalizada e, a seguir calcula-se a confiabilidade, utilizando o valor da cobertura normalizada calculada.

1) Cálculo da cobertura normalizada:
    a. A cobertura normalizada é dada pela expressão: $\emptyset_i = \alpha_0 + \alpha_1 * (1 - c_i)$.
    b. Tem-se que α_0 e α_1 são parâmetros estimados que normalizam a cobertura e são calculados para cada critério de teste. O parâmetro c_i representa a cobertura medida para o critério na ocorrência de uma falha.
    c. Na Tabela 19 são apresentados os valores estimados para α_0 e α_1.
    d. Na Tabela 9 são apresentadas as medidas da cobertura observada para o critério Todos-Nós.
    e. Considerando o critério Todos-Nós e a ocorrência da 5ª. falha, a cobertura normalizada, α_5, será calculada pela expressão: $\emptyset_5 = 0{,}6521 + 0{,}3321 * (1 - 0{,}303894)$.
    f. Ou seja, $\emptyset_5 = 0{,}883277$.
2) Cálculo da confiabilidade:
    a. Conforme apresentado na seção 3.3 do Capítulo 3, o MFIBC estima a confiabilidade utilizando a expressão:

$$R(k_i|n_{i-1}) = \exp[-D\left[(\emptyset_i)^{(i-1)}\right] \qquad (5.2)$$

    b. A Tabela 19 apresenta a estimativa para o parâmetro D, que é calculada para cada critério de teste.
    c. Aplicando os valores na expressão 5.2:

$$R(1/4) = exp[-3{,}9748 * [0{,}883277^4]].$$

d. Ou seja, a confiabilidade do software (*R*), após a execução de um dado de teste e tendo última falha ocorrida após a execução de quatro dados de teste, é *R*= 0, 088977.

A Figura 25 apresenta a confiabilidade medida versus a confiabilidade estimada pelo modelo MFIBC. Os dados de teste utilizados referem-se ao perfil operacional OP1. Na apresentação gráfica pode-se observar a proximidade entre as estimativas da confiabilidade, mesmo com a utilização de diferentes critérios.

Observa-se, no final dos testes, quando o software tende a ficar mais estável devido à redução das falhas, que as estimativas da confiabilidade são inferiores à confiabilidade medida. Esta é uma característica importante para um modelo de confiabilidade de software, pois quando o assunto é previsão de falhas, é melhor um estimador conservador, ou seja, que fornece uma estimativa de ocorrência de um evento de falha antes dele ocorrer. Esta característica também foi observada na aplicação dos modelos nos perfis operacionais OP1 e OP2. A maioria dos modelos tradicionais superestima a confiabilidade do software e, por isso, são criticados pelos pesquisadores [Chen97], [Lyu07]. O mesmo efeito foi observado nos resultados da aplicação do MBBC nos dados gerados pelos perfis operacionais OP1, OP2 e OP3.

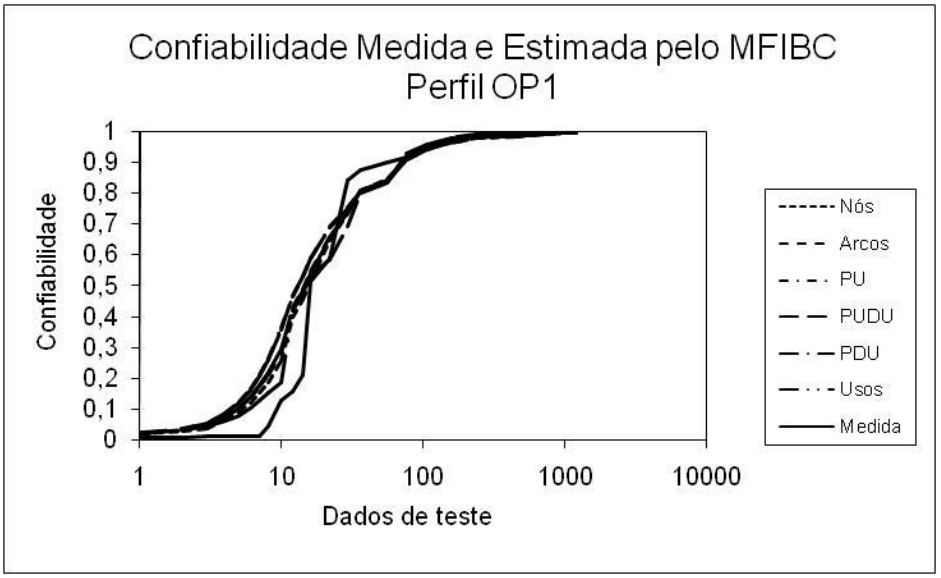

Figura 25 - Confiabilidade medida e estimada pelo MFIBC - Perfil OP1

5.4.2 Resultados para o Perfil Operacional OP2

Na Tabela 21 são apresentados os valores dos parâmetros D, α_0 e α_1, obtidos por meio da função de verossimilhança da função densidade de probabilidade do modelo MFIBC, aplicada às informações de cobertura apresentadas na Tabela 10, referentes ao perfil operacional OP2. A rotina utilizada para este cálculo é apresentada no Apêndice B. Os valores dos parâmetros, que normalizam a cobertura, são calculados para cada um dos critérios de teste utilizados.

Tabela 21 - Parâmetros do MFIBC – Perfil OP2

Critérios de teste	Parâmetros do modelo **MFIBC** - Perfil OP2		
	D	α0	α1
Nós	1,4500	0,6795	0,1350
Arcos	1,5236	0,6468	0,1120
PU	1,5236	0,5093	0,3234
PUDU	1,5236	0,6468	0,1305
PDU	1,5236	0,6400	0,1090
Usos	1,5400	0,6350	0,1590

A Tabela 22 apresenta a confiabilidade estimada pelo modelo MFIBC utilizando as medidas de cobertura normalizada do perfil operacional OP2.

Tabela 22 - Confiabilidade estimada pelo MFIBC – Perfil OP2

Critérios de teste					
Nós		**Arcos**		**PU**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,767356	0,234570	0,735082	0,217926	0,774875	0,217926
0,757614	0,328682	0,728348	0,326289	0,762569	0,307095
0,757614	0,435060	0,728348	0,445635	0,762569	0,412306
0,749527	0,532304	0,722199	0,555053	0,741293	0,508836
0,738040	0,632779	0,715464	0,660687	0,711814	0,631235
0,738040	0,727954	0,715464	0,751538	0,711814	0,756978
0,738040	0,791093	0,715464	0,815168	0,711814	0,820219
0,738040	0,841178	0,715464	0,863974	0,711814	0,868428
0,737948	0,880165	0,715318	0,900676	0,710337	0,904461
0,737764	0,910190	0,714878	0,928015	0,707274	0,932250
0,735099	0,933075	0,711950	0,948274	0,694585	0,953396
0,730596	0,952076	0,707265	0,964358	0,673911	0,972719
0,730596	0,967021	0,707265	0,976412	0,673911	0,986720
0,730228	0,975797	0,706973	0,983259	0,671997	0,991031
0,730228	0,982383	0,706973	0,988199	0,671997	0,994182
0,729493	0,987105	0,706241	0,991642	0,670466	0,996087
0,728666	0,990717	0,705216	0,994181	0,657777	0,997463

Tabela 22 (cont.) - Confiabilidade estimada pelo MFIBC – Perfil OP2

Critérios de teste					
PUDU		**PDU**		**Usos**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,758651	0,217926	0,740917	0,217926	0,757845	0,214381
0,754038	0,314780	0,738125	0,323401	0,745633	0,311274
0,754038	0,420513	0,738125	0,436006	0,745633	0,424777
0,747792	0,520374	0,735602	0,541877	0,735695	0,528135
0,738545	0,620999	0,732785	0,640114	0,722311	0,636901
0,738545	0,715497	0,732785	0,724753	0,722311	0,738755
0,738545	0,780947	0,732785	0,789859	0,722311	0,803556
0,738545	0,833098	0,732785	0,841252	0,722311	0,853871
0,737949	0,873837	0,732246	0,881022	0,722091	0,892162
0,736890	0,905851	0,731952	0,911913	0,721651	0,921092
0,733116	0,930598	0,730580	0,934961	0,715857	0,942715
0,726870	0,951142	0,728008	0,952925	0,709953	0,961790
0,726870	0,967403	0,728008	0,966796	0,709953	0,975065
0,726142	0,976199	0,727714	0,975716	0,709220	0,982233
0,726142	0,982883	0,727714	0,982361	0,709220	0,987534
0,725590	0,987541	0,727200	0,987133	0,707899	0,991143
0,721970	0,991047	0,725608	0,990725	0,705186	0,993894

Nota-se que são mínimas as diferenças dos valores da cobertura normalizada entre os critérios de teste utilizados. A utilização da cobertura normalizada diretamente na forma funcional do modelo gera estimativas da confiabilidade muito próximas, independentemente do critério de teste utilizado, a coluna "Confiabilidade estimada" apresenta estes valores.

Na Figura 26 são apresentadas as estimativas da confiabilidade calculadas pelo modelo MFIBC e a confiabilidade medida. Neste perfil operacional, também são próximas as estimativas calculadas utilizando-se os diferentes critérios de teste. A análise visual indica um bom ajuste entre as estimativas calculadas e a confiabilidade medida. Observa-se novamente uma característica conservadora na estimação da confiabilidade.

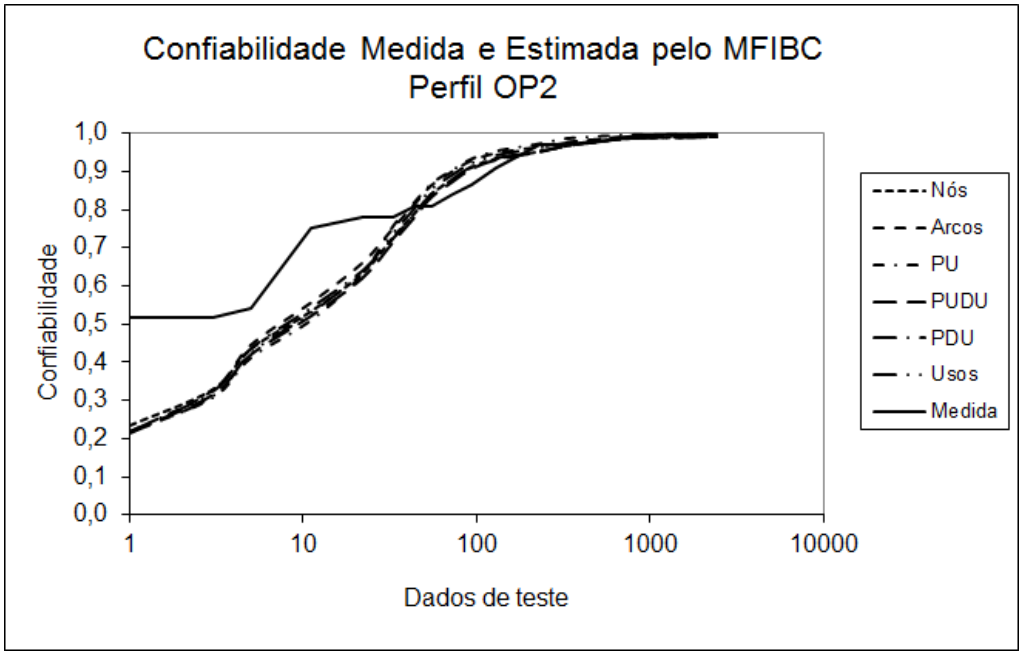

Figura 26 - Confiabilidade medida e estimada pelo MFIBC - Perfil OP2

5.4.3 Resultados para o Perfil Operacional OP3

A Tabela 23 apresenta os valores dos parâmetros que normalizam a cobertura. Eles são calculados para cada um dos critérios de teste utilizados no experimento.

Tabela 23 - Parâmetros do MFIBC – Perfil OP3

Critérios de teste	Parâmetros do modelo **MFIBC** - Perfil OP3		
	D	α0	α1
Nós	1,3747	0,8342	0,0387
Arcos	1,3925	0,7696	0,1357
PU	1,3491	0,7216	0,2012
PUDU	1,3486	0,7500	0,1543
PDU	1,3238	0,5911	0,3024
Usos	1,3918	0,7951	0,0911

A Tabela 24 apresenta a confiabilidade estimada pelo modelo MFIBC utilizando as medidas de cobertura normalizada do perfil operacional OP3. Nota-se que são mínimas as diferenças dos valores da cobertura normalizada entre os critérios de teste utilizados.

Tabela 24 - Confiabilidade estimada pelo MFIBC – Perfil OP3

Critérios de Teste					
Nós		**Arcos**		**PU**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,860597	0,252915	0,880111	0,248453	0,893265	0,259474
0,857884	0,306339	0,873725	0,293595	0,874686	0,299661
0,857884	0,363589	0,873725	0,345408	0,874686	0,356235
0,854933	0,419813	0,866098	0,395029	0,864274	0,405425
0,854933	0,479790	0,866098	0,456785	0,864274	0,471070
0,854933	0,533728	0,866098	0,507314	0,864274	0,521742
0,852615	0,584625	0,859712	0,555573	0,857877	0,569908
0,852272	0,637453	0,858825	0,616711	0,856822	0,630439
0,850033	0,682032	0,851907	0,662239	0,843960	0,675775
0,849453	0,727228	0,849246	0,719573	0,842563	0,745985
0,849243	0,764214	0,847650	0,762063	0,838073	0,784069
0,849243	0,796282	0,847650	0,797690	0,838073	0,824266
0,849243	0,824103	0,847650	0,825638	0,838073	0,850468
0,848821	0,848493	0,846053	0,850094	0,837019	0,873069
0,848821	0,870612	0,846053	0,874520	0,837019	0,894247
0,847372	0,889041	0,840732	0,892759	0,824905	0,910687
0,847135	0,907429	0,840022	0,916898	0,823986	0,939867
0,847135	0,921343	0,840022	0,930623	0,823986	0,951041
0,846924	0,932954	0,838248	0,941389	0,818916	0,959481
0,845739	0,943166	0,832927	0,952430	0,809797	0,970144
0,845739	0,952955	0,832927	0,964670	0,809797	0,980354
0,845660	0,960065	0,832891	0,970485	0,808708	0,984061
0,845660	0,966188	0,832572	0,975377	0,807449	0,987448

Tabela 25 (cont.) - Confiabilidade estimada pelo MFIBC – Perfil OP3

Critérios de Teste					
PUDU		**PDU**		**Usos**	
Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada	Cobertura normalizada	Confiabilidade estimada
0,885540	0,259603	0,874744	0,266122	0,868091	0,248627
0,875076	0,302934	0,869036	0,314118	0,861914	0,298732
0,875076	0,356044	0,869036	0,367966	0,861914	0,355595
0,868578	0,405072	0,864483	0,419441	0,856158	0,410168
0,868578	0,464139	0,864483	0,477425	0,856158	0,473402
0,868578	0,513402	0,864483	0,527739	0,856158	0,527164
0,864481	0,560415	0,859047	0,575488	0,850884	0,578019
0,863724	0,614713	0,857552	0,633164	0,849854	0,637987
0,856496	0,658547	0,850824	0,678974	0,844182	0,684728
0,855426	0,715687	0,847562	0,733960	0,842753	0,738569
0,852660	0,753558	0,845116	0,776278	0,841913	0,777614
0,852660	0,791702	0,845116	0,812270	0,841913	0,810860
0,852660	0,819422	0,845116	0,838854	0,841913	0,838186
0,851851	0,843823	0,844301	0,861998	0,840862	0,861905
0,851851	0,866857	0,844301	0,883541	0,840862	0,884309
0,844649	0,885402	0,838615	0,900740	0,836366	0,901782
0,844049	0,913464	0,837573	0,923838	0,835862	0,923315
0,844049	0,927250	0,837573	0,937026	0,835862	0,936086
0,840839	0,938237	0,832748	0,946978	0,833719	0,946289
0,835255	0,951179	0,824118	0,959938	0,830441	0,957003
0,835255	0,963835	0,824118	0,972734	0,830441	0,966702
0,834498	0,969702	0,822147	0,977475	0,829859	0,972269
0,833950	0,975124	0,821196	0,982346	0,829622	0,977265

Os resultados observados com a aplicação do modelo MFIBC nos perfis operacionais OP1 e OP2 repetem-se para o perfil OP3.

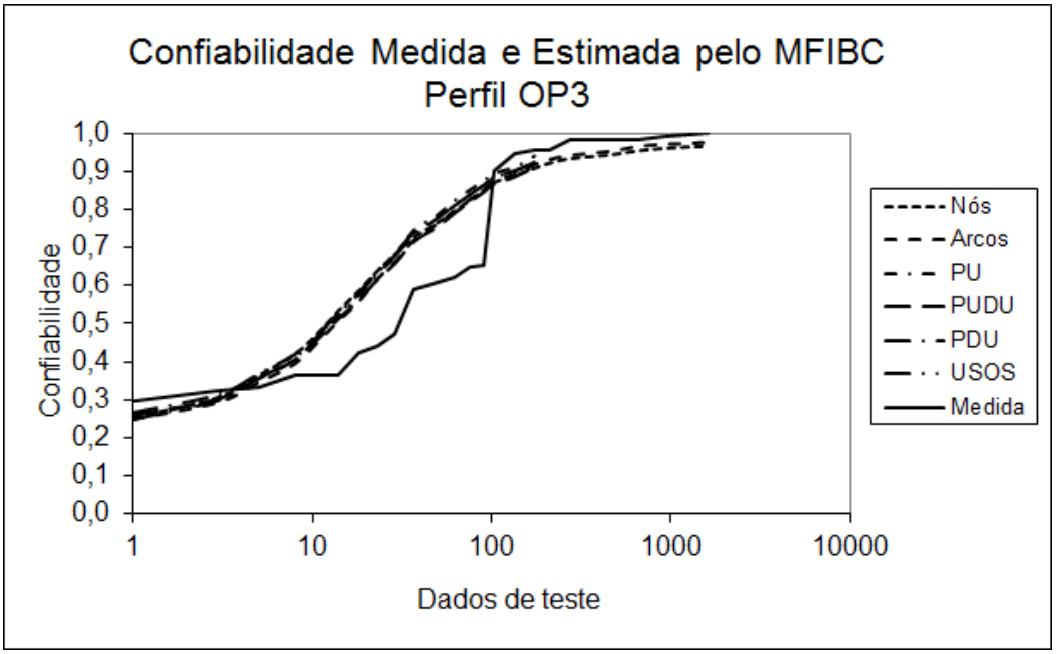

Figura 27 - Confiabilidade medida e estimada pelo MFIBC - Perfil OP3

5.4.4 Análise dos Resultados Obtidos com a Utilização do MFIBC

Os resultados obtidos com a aplicação do MFIBC para os perfis operacionais OP1, OP2 e OP3 indicam que o modelo é robusto à variação do perfil operacional do software. Conforme pode ser observado nas Figuras 25, 26 e 27, apresentadas anteriormente, os valores das confiabilidade medida e estimada pelo MFIBC são bastante próximos, indicando que o modelo ajusta-se bem aos dados.

A análise gráfica também mostra que são próximos os valores da confiabilidade estimada pelo MFIBC utilizando os diferentes critérios de teste, a diferença é pequena entre eles; qualquer um dos seis critérios de teste aplicado poderia ser utilizado para se obter a cobertura do código utilizada pelo modelo. Este resultado é devido ao uso da cobertura normalizada no lugar da cobertura observada. Para qualquer um dos critérios de teste

utilizados, os valores da confiabilidade estimada pelo modelo serão aproximadamente iguais.

Para confirmar a hipótese de igualdade entre as medidas da confiabilidade é aplicado o teste de Kolmogorov-Smirnov. O teste é realizado sob a hipótese de igualdade entre elas, se esta hipótese não puder ser comprovada, então ela é rejeitada em favor da hipótese alternativa, ou seja, os valores da confiabilidade, medida e estimada, provêm de populações diferentes.

Os resultados do teste, apresentados na Tabela 25, não rejeitam a hipótese de igualdade entre os valores medido e estimado para todos os perfis utilizados. Isso indica que o modelo pode ser utilizado para estimar a confiabilidade do software e que a variação do perfil operacional não afetou a capacidade preditiva do modelo MFIBC. Observa-se que a utilização da cobertura de código diretamente na forma funcional do MFIBC resultou na sensibilidade do modelo à variação do perfil operacional do software, indicando que o uso da cobertura é importante na análise da confiabilidade de software por meio de modelos e não deve ser desprezada.

Tabela 25 - Teste de igualdade entre as medidas de confiabilidade

Perfil Operacional	Nível de Significância	Valor do Teste (p-value)	Conclusão
OP1	1%	0,216	Não rejeita H_0
OP2	1%	0,9303	Não rejeita H_0
OP3	1%	0,3599	Não rejeita H_0

5.5 Comparação com Modelos Tradicionais

A comparação com os modelos tradicionais completa os resultados do experimento. Foram utilizados os modelos baseados no domínio do tempo: Musa-Okumoto, Musa Básico, Linear Littlewood-Verral, Quadrático Littlewood-Verral, Jelinski-Moranda e Geométrico. Esses modelos foram selecionados por estarem disponíveis no software CASRE (Computer Aided Software Reliability Estimation).

Cada modelo foi utilizado para estimar a confiabilidade do programa **Space** utilizando os dados de falhas observados com a execução do programa usando os perfis operacionais OP1, OP2 e OP3. Como os modelos são baseados no domínio de tempo considerou-se que um dado de teste é equivalente a uma unidade de tempo de execução [Musa98]. Os resultados foram comparados aos obtidos pelos modelos MBBC e MFIBC.

5.5.1 Aplicação dos Modelos Tradicionais ao Perfil Operacional OP1

Inicialmente, verificou-se quais modelos ajustavam-se bem aos dados de falhas observadas com a execução dos dados de testes referentes ao perfil operacional OP1, já apresentados na Tabela 9. A Tabela 26 apresenta os modelos que foram selecionados por apresentarem um bom ajuste aos dados.

Tabela 26 - Resultado da análise de ajuste aos dados de falhas

Modelo	Ajuste aos Dados
Musa-Okumoto	Não
Musa Básico	Não
Linear Littlewood-Verral	Sim
Quadrático Littlewood-Verral	Sim
Jelinski-Moranda	Não
Geométrico	Sim

Os modelos que tiveram um bom ajuste aos dados de falhas foram utilizados para estimar a confiabilidade do software. A Figura 28 mostra o gráfico do crescimento da confiabilidade medida e os gráficos do crescimento da confiabilidade estimada pelos modelos Linear Littlewood-Verral, Quadrático Littlewood-Verral e Geométrico, que ajustaram-se bem aos dados de falhas. A análise gráfica indica uma diferença grande entre as estimativas, principalmente no início da execução do programa. Ao final do processo se observa uma conversão das estimativas. A Tabela 27 apresenta o resultado do teste de Kolmogorov-Smirnov, que rejeita a hipótese de igualdade entre as estimativas da confiabilidade calculadas pelos modelos tradicionais e a confiabilidade medida.

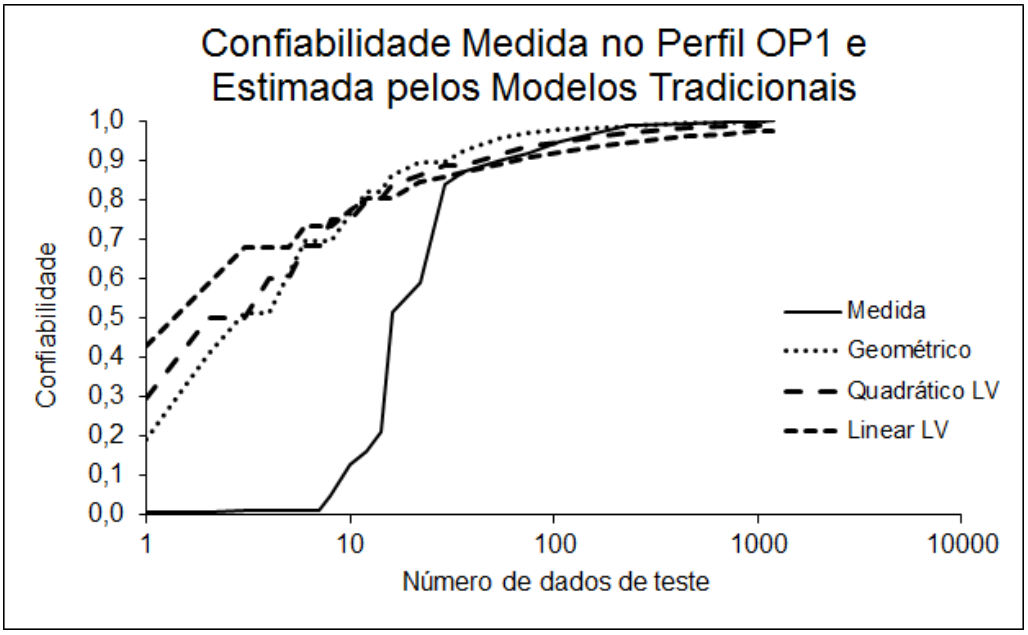

Figura 28 - Estimativas dos modelos tradicionais para o perfil OP1

Tabela 27 - Teste de igualdade entre as medidas de confiabilidade

Hipótese	**Nível de Significância**	**Valor do Teste (p-value)**	**Conclusão**
Confiabilidade estimada pelo modelo **Geométrico** é igual à confiabilidade medida	5%	0,3750	Rejeita a hipótese de igualdade
Confiabilidade estimada pelo modelo **Quadrático LV** é igual à confiabilidade medida	5%	0,3750	Rejeita a hipótese de igualdade
Confiabilidade estimada pelo modelo **Linear LV** é igual à confiabilidade medida	5%	0,3750	Rejeita a hipótese de igualdade

Concluindo, o teste indica que nenhum dos modelos tradicionais pode ser utilizado para estimar a confiabilidade do software quando o programa **Space** é executado conforme o perfil operacional OP1.

5.5.2 Aplicação dos Modelos Tradicionais ao Perfil Operacional OP2

Inicialmente verificou-se quais modelos ajustavam-se bem aos dados de falhas observadas com a execução dos dados de testes referentes ao perfil operacional OP2. Nesta

análise alguns dos modelos foram descartados. A Tabela 28 apresenta os modelos que foram selecionados por apresentarem um bom ajuste aos dados.

Tabela 28 - Resultado da análise de ajuste aos dados de falhas

Modelo	Ajuste aos Dados
Musa-Okumoto	Não
Musa Básico	Não
Linear Littlewood-Verral	Não
Quadrático Littlewood-Verral	Sim
Jelinski-Moranda	Não
Geométrico	Não

A comparação das estimativas da confiabilidade de software calculadas pelo modelo Quadrático de Littlewood-Verral, o único que apresentou um bom ajuste aos dados de falhas, com as medidas reais da confiabilidade do Space, Figura 29, apresentam diferenças no comportamento do crescimento da confiabilidade do software. Apesar delas convergirem no final da execução dos dados de teste, as tendências entre elas são bastante diferentes no início do processo.

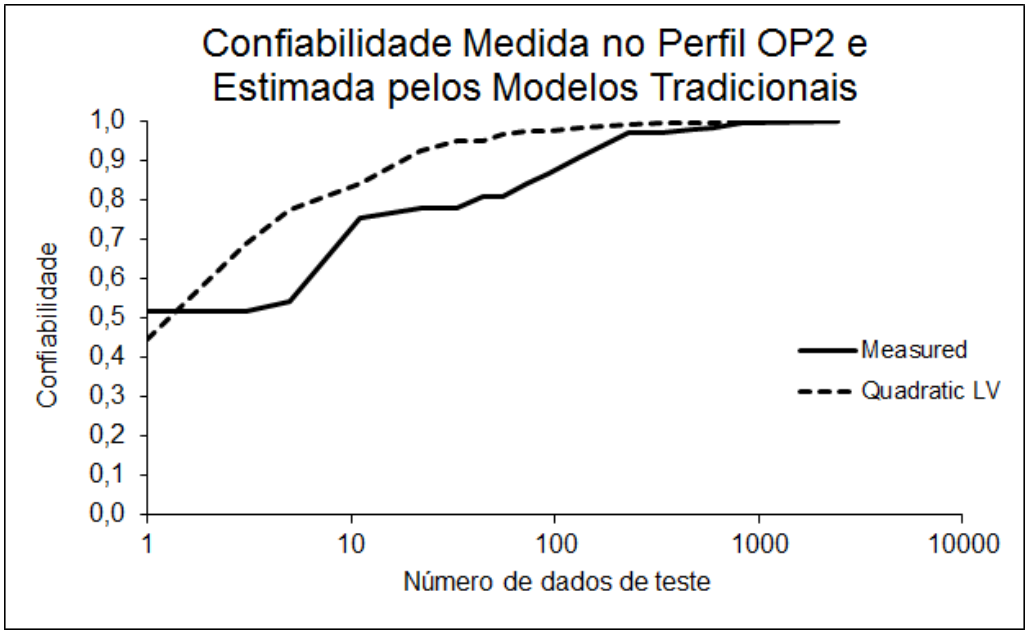

Figura 29 - Estimativas dos modelos tradicionais para o perfil OP2

A Tabela 29 apresenta o resultado do teste de Kolmogorov-Smirnov, que rejeita a hipótese de igualdade entre as estimativas da confiabilidade calculadas pelos modelos tradicionais e a confiabilidade medida.

Concluindo, o teste indica que nenhum dos modelos tradicionais selecionados pode ser utilizado para estimar a confiabilidade do software de acordo com o perfil operacional OP2.

Tabela 29 - Teste de igualdade entre as medidas de confiabilidade

Hipótese	**Nível de Significância**	**Valor do Teste (p-value)**	**Conclusão**
Confiabilidade estimada pelo modelo Quadrático LV é igual à confiabilidade medida	5%	0,4117	Rejeita a hipótese de igualdade

5.5.3 Aplicação dos Modelos Tradicionais ao Perfil Operacional OP3

Nenhum dos modelos tradicionais disponíveis ajustou-se aos dados de falhas gerados segundo o perfil operacional OP3 – Tabela 30. Desta forma, todos os modelos foram rejeitados.

Tabela 30 - Resultado da análise de ajuste aos dados de falhas

Modelo	**Ajuste aos Dados**
Musa-Okumoto	Não
Musa Básico	Não
Linear Littlewood-Verral	Não
Quadrático Littlewood-Verral	Não
Jelinski-Moranda	Não
Geométrico	Não

5.6 Resumo

Este capítulo descreveu a realização de um experimento que analisou a qualidade de ajuste dos modelos MBBC e MFIBC segundo três perfis operacionais diferentes. O capítulo apresenta todas as etapas realizadas no experimento, desde a criação de perfis operacionais, teste do programa com dados de testes gerados segundo cada perfil, seleção de critérios de teste, mensuração da cobertura de código, estimação dos parâmetros dos modelos e o cálculo da confiabilidade estimada utilizando os modelos MBBC e MFIBC.

Os resultados apresentados pelos modelos MBBC e MFIBC são comparados e analisados estatisticamente por meio do teste de Kolmogorov-Smirnov. Para concluir, realizou-se a comparação com os modelos tradicionais baseados no domínio do tempo: Musa-Okumoto, Musa Básico, Linear Littlewood-Verral, Quadrático Littlewood-Verral, Jelinski-Moranda e Geométrico. Nenhum destes modelos pôde ser utilizado para estimar a confiabilidade do programa **Space** nos três perfis operacionais utilizados.

CAPÍTULO 6

CONCLUSÕES E TRABALHOS FUTUROS

No presente capítulo são apresentadas as conclusões a partir dos resultados obtidos no experimento realizado e também a sugestão de temas para trabalhos futuros. Ao término deste trabalho foi possível concluir que os objetivos inicialmente propostos foram alcançados.

6.1 Conclusões

O objetivo desta dissertação foi avaliar, por meio de um experimento, o comportamento de dois modelos de confiabilidade de software que utilizam a informação de cobertura do código: "Modelo Binomial Baseado em Cobertura" (MBBC) e "Modelo de Falhas Infinitas Baseado em Cobertura" (MFIBC). Os modelos foram aplicados aos dados de falhas observados durante a execução de um programa segundo três perfis operacionais estatisticamente distintos. Para completar a análise comparativa, seis modelos de confiabilidade de software baseados no domínio do tempo também foram utilizados para estimar a confiabilidade do software utilizando os mesmos dados de falhas. Os modelos escolhidos foram: Musa-Okumoto, Musa Básico, Littlewood-Verral Linear, Littlewood-Verral Quadrático, Jelinski-Moranda e Geométrico.

O experimento realizado consistiu na execução de um software com milhares de dados de teste gerados de acordo com três perfis operacionais estatisticamente distintos. A cada falha observada, foram medidas as coberturas de código dos elementos requeridos para cada critério de teste estrutural utilizado: *Todos-Nós*, *Todos-Arcos, Todos-Usos* e critérios baseados em fluxo de dados da fam*ília Potenciais-Usos*. A medida da cobertura

foi utilizada para calcular os parâmetros dos modelos MBBC e MFIBC e, em seguida, estimar a confiabilidade do software de acordo com cada perfil operacional.

Para a realização do experimento, foi selecionado um software denominado **Space**, desenvolvido pela *European Space Agency* (ESA) e utilizado para obter a melhor distribuição espacial de antenas de transmissão de dados via satélite. O software foi escrito em linguagem C e possui 33 defeitos conhecidos distribuídos por 236 funções e cerca de 6.000 linhas de código. No início do experimento, mediu-se a confiabilidade do software para, posteriormente, ser comparada com a confiabilidade estimada pelos modelos MBBC, MFIBC e modelos tradicionais.

A comparação entre a confiabilidade medida e os valores das estimativas da confiabilidade, calculadas pelos modelos MBBC, MFIBC e tradicionais, foi realizada por meio do teste estatístico de Kolmogorov-Smirnov, com a suposição de que elas são estatisticamente iguais. A seleção desse teste deveu-se ao fato dele ser um teste não-paramétrico, ou seja, usado quando não se tem conhecimento prévio da função distribuição de probabilidade dos dados e também por ser um teste adequado para pequenas amostras.

Os seguintes resultados e conclusões foram obtidos:

- Para os modelos MBBC e MFIBC, os resultados não indicaram evidências para rejeitar a hipótese de igualdade entre a confiabilidade medida e a estimada. Esse resultado foi observado nos três perfis operacionais utilizados no experimento. Em outras palavras, os resultados indicaram que os modelos são robustos à variação do perfil operacional do software, isto é, a mudança de perfil operacional não afetou a capacidade preditiva de nenhum dos modelos baseados em cobertura.

- O mesmo resultado não foi observado para os modelos tradicionais. Nenhum deles foi capaz de apresentar um bom ajuste aos dados de falhas observadas durante a execução do software de acordo com os perfis operacionais utilizados. Alguns ajustaram-se bem aos dados de falhas produzidos segundo

o perfil operacional OP1, mas não ajustaram-se aos dados de falhas observados durante a execução do software conforme os perfis operacionais OP2 e OP3. De fato, nenhum modelo ajustou-se aos dados de falhas gerados segundo o perfil operacional OP3. Dentre os modelos tradicionais que ajustaram-se bem aos dados de falhas, nenhum deles gerou boas estimativas da confiabilidade do software. O teste de Kolmogorov-Smirnov rejeitou a hipótese de igualdade entre as estimativas calculadas pelos modelos tradicionais e a confiabilidade medida do software.

- Outro resultado, decorrente do uso da informação de cobertura de código exercitado durante o teste, é que não se observou o efeito da superestimação da confiabilidade do software. A superestimação da confiabilidade é observada nos modelos tradicionais, que utilizam o tempo entre falhas como parâmetro. A superestimação ocorre quando um critério de teste atinge o limite da sua capacidade de gerar dados de teste que exercitem elementos requeridos pelo critério de teste diferentes dos já exercitados. O tempo entre falhas aumenta, causando a falsa impressão de que a taxa de falhas está diminuindo. A informação da cobertura dos elementos requeridos do critério de teste diretamente na forma funcional dos modelos permite controlar o efeito da superestimação.

- Os modelos MBBC e MFIBC estimam a confiabilidade de acordo com a cobertura de código atingida no teste do software, eliminando o efeito da superestimação da confiabilidade existente nos modelos tradicionais de confiabilidade de software. Os modelos não apresentaram resultados com superestimação da confiabilidade em nenhum dos três perfis operacionais utilizados no experimento. Esse é um indicativo de que a cobertura de código tem um importante efeito na análise da confiabilidade de software por meio de modelos, e, desse modo não deve ser desprezada.

- São muito próximas as estimativas da confiabilidade do software calculadas em função da cobertura de cada critério de teste. Este resultado foi observado para os modelos MBBC e MFIBC nos três perfis operacionais utilizados. Isso ocorre devido ao uso da cobertura normalizada ao invés da cobertura observada. A normalização da cobertura padroniza os resultados, fazendo com que seja possível utilizar qualquer critério de teste para estimar a confiabilidade.

Os resultados obtidos devem ser utilizados com cautela. Não é possível afirmar que os modelos MBBC e MFIBC podem ser utilizados como modelos universais, isto é, para qualquer tipo de software ou perfil operacional. Deve-se levar em consideração que o experimento foi realizado para um tipo específico de software, desenvolvido para calcular o posicionamento de antenas baseado em parâmetros de entrada, ou seja, um aplicativo com muito processamento interno, mas com interfaces simples, por exemplo, *drivers* que são instalados em sistemas operacionais para interface com placas, impressoras, etc.

É impossível afirmar que os modelos se ajustarão aos dados de falhas observadas com a execução do software em qualquer perfil operacional, devido às infinitas possibilidades de se definir um perfil de uso.

6.2 Trabalhos Futuros

É necessário avaliar o comportamento dos modelos MBBC e MFIBC com outros tipos de aplicações e outros perfis operacionais, como, sistemas de informação desenvolvidos para operar na Internet, que fazem uso intensivo de interação humano-computador e de bancos de dados.

Outras variáveis devem ser estudadas para avaliar sua relação com a confiabilidade de software, por exemplo, o tamanho do software e a quantidade de defeitos revelados. Espera-se que a cobertura cresça mais rapidamente em aplicações menores. Isso terá algum

efeito na estimação dos parâmetros dos modelos? Qual será o comportamento dos modelos com poucos dados de falhas do software?

REFERÊNCIAS

[An10] An, J. and Zhu, J., "Software Reliability Modeling with Integrated Test Coverage", Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement, Singapore, 2010, pp. 106-112.

[Att] AT&T Software Reliability Engineering Toolkit. Disponível em: <http://www.cse.cuhk.edu.hk/~lyu/book/reliability/sretools.html>. Acesso em: 22 mar. 2014.

[Bol02] Boland, P. J., "Challenges in Software Reliability and Testing", Third International Conference on Mathematical Methods in Reliability, Trondheim, Norway, 2002.

[Broo80] Brooks, W. D. and Motley R. W., "Analysis of discrete software reliability models", IBM FEDERAL SYSTEMS DIV GAITHERSBURG MD, April, 1980.

[Cai05] Cai, X. and Lyu, M. R., "The Effect of Code Coverage on Fault Detection under Different Testing Profiles", International Workshop on Advances in Model-Based Software Testing, 2005.

[Cai07] Cai, X. and Lyu, M. R.," Software Reliability Modeling with Test Coverage: Experimentation and Measurement with A Fault-Tolerant Software Project", The 18th IEEE International Symposium on Software Reliability, 2007, pp. 17-26.

[Casr] Computer Aided Software, Version 3. Disponível em: <http://www.openchannelfoundation.org/projects/CASRE_3.0/>. Acesso em: 22 mar. 2014.

[Chaim91] Chaim, M. L., "POKE-TOOL – Uma Ferramenta para Suporte ao Teste Estruturado de Programas Baseado em Análise de Fluxo de Dado"s. Tese de Mestrado, DCA/FEE/UNICAMP – Campinas, SP, Abril, 1991.

[Chen94b] Chen, M; Mathur, P. and Rego, V., "A case study to investigate sensitivity of reliability estimates to errors in operational profile", Proceedings Fifth International Symposium of Software Reliability Engineering, 1994; IEEE Press.

[Chen96] Chen, M.; Lyu, M. R. and Wong, W. E., "An Empirical Study of the Correlation between Code Coverage and Reliability Estimation", Proceedings of the Third IEEE International Software Metrics Symposium, 1996, pp. 133-141.

[Chen97] Chen, M.; Lyu, M. R. and Wong, W. E., "Incorporating Code Coverage in the Reliability Estimation for Fault-Tolerant Software". Proc. of Seventeenth Intl. Symposium on Reliable and Distributed Systems, 1997, pp. 45-52.

[Chen01] Chen, M.; Lyu, M. R. and Wong, W. E., "Effect of Code Coverage on Software Reliability Measurement". IEEE Transactions on Reliability, Vol 50, nº 2, June 2001, pp. 165-170.

[Con99] Conover, W. J., "Practical Nonparametrics Statistics", Third Edition, Wiley, 1999.

[Cot13] Cotroneo, D.; Pitrantuono, R. and Russo, S., "Combining Operational and Debug Testing for Improving Reliability", IEEE Transactions on Reliability, Vol. 62, Issue 2, June 2013, pp. 408-423.

[Crespo96] Crespo, A. N.; Matrella P. and Pasquini, A., "Sensitivity of Reliability Growth Models to Operational Profile Errors vs Testing Accuracy", IEEE Transactions on Reliability, Vol. 45, Nº 4, December, 1996, pp. 531-540.

[Crespo97a] Crespo, A. N.; Pasquini, A.; Jino, M. and Maldonado, J. C., "Cobertura dos Critérios Potenciais-usos e Confiabilidade do Software", Anais do XI Simpósio Brasileiro de Engenharia de Software - Fortaleza, 1997, pp. 379-394.

[Crespo97b] Crespo, A. N., "Modelos de Confiabilidade de Software Baseados em Cobertura de Critérios Estruturais de Teste." Tese de Doutorado, DCA/FEE/UNICAMP - Campinas, SP, Dezembro 1997.

[Crespo08] Crespo, A. N.; Jino, M.; Pasquini, A. and Maldonado, J. C., "A Binomial Software Reliability Model Based on Coverage of Structural Testing Criteria", Empirical Software Engineering, Vol. 13, Issue 2, April 2008, pp. 185-209.

[Crespo09] Crespo, A. N.; Jino, M.; Pasquini, A. and Maldonado, J. C., "Applying Code Coverage Approach to an Infinite Failure Software Reliability Model", XXIII Simpósio Brasileiro de Engenharia de Software (SBES), 2009, Fortaleza-CE, Brasil, pp. 216-226.

[Dela07] Delamaro, M.E.; Maldonado, J.C.; Jino, M., "Introdução ao Teste de Software", Campus, Rio de Janeiro, 2007.

[Frat95] Frate, F. D.; Garg, P.; Mathur, A. P. and Pasquini, A., "Experiments to Investigate the Correlation Between Code Coverage and Software Reliability", SERC-TR-162-P, Software Engineering Research Center, Purdue University, West Lafayette, Indiana 47907, April 1995.

[Garg95] Garg, P., "Investigating Coverage - Reliability Relationship and Sensitivity of Reliability to Errors in Operational Profile", Technical Report - Department of Computer Sciences, Purdue University, West Lafayette, May 1994.

[Goel85] Goel, A. L., "Software Reliability Models: Assumptions, Limitations, and Applicability", IEEE Transactions on Software Reliability Engineering, Vol. SE-11, Nº12, December, 1985, pp. 1411-1423.

[Hoel78] Hoel, P. G.; Port, S. C. and Stone, C. J., "Introdução à Teoria da Probabilidade", Interciência, Rio de Janeiro, 1978.

[Hor94] Horgan, J.R.; London, S. and Lyu, M.R.; "Achieving software quality with testing coverage measures", IEEE Computer, Vol. 27, nº 9, September, 1994, pp. 60-69.

[Huc08] Huckle, T., "Collection of Software Bugs", Disponível em http://www5.informatik.tu-muenchen.de/~huckle/bugse.html, em 12/12/2008.

[Hud67] Hudson, A., "Program Errors as a Birth and Death Process", Technical Report SP - 3011, Santa Monica, Cal.: Systems Development Corporation, 1967.

[Inou08] Inoue, S. and Yamada, S., "Two-Dimensional Software Reliability Assessment with Testing-Coverage", Second International Conference on Secure System Integration and Reliability Improvement, July, 2008, pp. 150-157.

[ISO9126] NBR ISO/IEC TR 9126-1, "Engenharia de Software – Qualidade de Produto – Parte 1: Modelo de Qualidade". ABNT, Junho, 2003.

[Jal94] Jalote, P. and Muralidhara, Y, R., "A Coverage Based Model for Software Reliability Estimation", First International Conference on Software Testing, Reliability and Quality Assurance, New Delhi, India, December 1994, pp. 6-10.

[Jeli72] Jelinski Z. and Moranda P. B., "Software Reliability Research", Proceedings of the Statistical Methods for the Evaluation of Computer System Performance, Academic Press, 1972, pp. 465-484.

[Karc96] Karcich, R. M.; Skibbe, R.; Mathur, A. P. and Garg, P., "On Software Reliability and Code Coverage", IEEE, 1996, pp. 297-308.

[Kris96] Krishnamurthy, S. and Mathur, A. P., "On Predicting Reliability of Modules Using Code Coverage", Proceedings of the 1996 Conference of the Centre for Advanced Studies on Collaborative Research, August, 1996.

[Litt73] Littlewood, B. and Verral, J. L., "A Bayesian Reliability Growth Model for Computer Software", Applied Statistics, vol. 22, no. 3, 1973, pp. 332-346.

[Lit00] Littlewood, B. and Strigini, L., "Software Reliability and Dependability: a Roadmap", ICSE'00 Proceedings of the Conference on The Future of Software Engineering, Ireland, 2000, pp. 175-188.

[Lom07] Lombardi, C., "Body of a car, brains of a pc", http://news.cnet.com/Body-of-a-car,-brains-of-a-PC/2100-11389_3-6201752.html, em 11/12/2008.

[Lyu96] Lyu, M., "Handbook of Software Reliability Engineering", McGraw-Hill, 1996.

[Lyu03] Lyu, M. R.; Huang, Z.; Sze, S. K. and Cai, X., "An Empirical Study on Testing and Fault Tolerance for Software Reliability Engineering", Proceedings 14th IEEE International Symposium on Software Reliability Engineering, Denver, Colorado, USA, November, 2003, pp. 119-130.

[Lyu07] Lyu, M., "Software Reliability Engineering: A Roadmap", Proceedings ICSE'2007, 2007, pp. 153-170.

[Mal02] Malaiya, Y. K.; Li, N.; Bieman, J. and Karcich, R., "Sofware Reliability Growth With Test Coverage", IEEE Transactions on Reliability, December 2002, pp. 420-426.

[Mill72] Mills, H. D. "On the Statistical Validation of Computer Programs", FSC-72-6015, IBM Federal System Division, Gaithersburg, Maryland, 1972.

[Mood74] Mood, A.; Graybill, F. and Boes, D., "Introduction to the Theory of Statistics", 3d ed., McGraw-Hill, Inc., New York, 1974.

[Musa87] Musa, J. D.; Ianino, A. and Okumoto, K., "Software Reliability - Measurement, Prediction, Application", McGraw-Hill, New York, 1987.

[Musa93] Musa, J. D., "Operational profiles in software reliability engineering", IEEE Software, vol. 10, March 1993, pp. 14-32.

[Musa94] Musa, J. D., "Sensitivity of field failure intensity to operational profile errors", Proceedings Fifth International Symposium of Software Reliability Engineering, 1994, pp. 334-337.

[Musa98] Musa, J. D., "Software Reliability Engineering", McGrall-Hill, New York, 1998.

[Nels78] Nelson, F., "Estimating Software Reliability from Test Data", Microeletronics and Reliability, vol. 17, no. 1, 1978, pp. 67-73.

[Nik98] Nikora, Allen P., "Software System Defect Content Prediction from the Development Process and Product Characteristics", PhD Thesis, Faculty of the Graduate School University of Southern California, May, 1998.

[Pham03] Pham, H. and Zhang, X., "NHPP software reliability and cost models with testing coverage". European Journal of Operational Research 145, 2003, pp. 443-454.

[Press09] Pressman, Roger S., "Software Engineering: A Practitioner's Approach", McGraw-Hill Science, 7th edition, 2009.

[Shoo72] Shooman, M. L., "Probabilistic Models for Software Reliability Prediction," Statistical Computer Performance Evaluation, W. Freiberg, Ed. New York: Academic Press, 1972, pp. 485-502.

[Shoo84] Shooman, M. L., “Software Reliability: A Historical Perspective”, IEEE Transactions on Software Reliability, Vol. R-33, Nº 1, April, 1984, pp. 48-55.

[Smur] Statistical Modeling and Estimation of Reliability Functions for Systems. Disponível em: <http://www.slingcode.com/smerfs/>. Acesso em: 22 mar. 2014.

[Sore] Software Reliability Program. Disponível em: <http://homepages.laas.fr/surf4tst/sorel/sorel.html/>. Acesso em: 22 mar. 2014.

[Srep] Software Reliability Estimation & Prediction Tool. Disponível em: <http://srel.ee.duke.edu/>. Acesso em: 22 mar. 2014.

[Xie91] Xie, M., “Software Reliability Modelling”, World Scientific Publishing, Singapore, 1991.

APÊNDICE A

Script para medir a cobertura do critério de teste *Todos-Nós*

A mensuração da cobertura dos elementos requeridos pelo critério de teste *Todos-Nós*, utilizado no experimento, foi realizada utilizando-se os scripts abaixo.

ExecNos.sh: Este script orquestra as chamadas do script principal, analisaNos.sh. Três parâmetros, números inteiros, são passados ao script principal: o primeiro inteiro refere-a a ordem do defeito, os outros dois identificam a faixa de dados de teste que deve ser executada para a detecção do defeito.

Por exemplo, a linha `analisaNos.sh 9 1 18` indica que será analisada a cobertura do critério *Todos-Nós* até o momento da detecção do nono defeito que é revelado após a execução de dezoito dados de teste.

```
#Apaga arquivo que registra a cobertura em cada intervalo
#rm -f coberturaNos
#Executa o programa e analisa a cobertura
#analisaNos.sh p1 p2 p3
#p1 = sequencia do defeito; p2 = dado de teste inicial; p3 = dado #de
teste final
rm -f coberturaNos
analisaNos.sh 1 1 1
analisaNos.sh 2 1 2
analisaNos.sh 3 1 2
analisaNos.sh 4 1 6
analisaNos.sh 5 1 11
analisaNos.sh 6 1 11
analisaNos.sh 7 1 11
analisaNos.sh 8 1 11
analisaNos.sh 9 1 18
analisaNos.sh 10 1 20
analisaNos.sh 11 1 37
analisaNos.sh 12 1 100
analisaNos.sh 13 1 100
analisaNos.sh 14 1 136
analisaNos.sh 15 1 136
analisaNos.sh 16 1 216
analisaNos.sh 17 1 1610
```

analisaNos.sh: Este script mede a cobertura atingida pelo critério *Todos-Nós*. Recebe como parâmetro o seqüencial de execução do programa SPACE e o intervalo de dados de teste. Por exemplo, o seqüencial "1" indica a primeira execução do programa, com um defeito removido, o seqüencial "2" indica a segunda execução, com o segundo defeito removido. A ordem de remoção dos defeitos segue a seqüencia em que eles foram detectados no processo de teste, de acordo com o perfil operacional utilizado. O intervalo de dados de teste indicará a quantidade de dados que foram necessários para que a falha fosse apresentada.

```
IFS=$'\n'
sere=0
ser=0
i=$1
if [ $i -eq "1" ];then
   listaDefeitos="-DE1 -DE2 -DE3 -DE7 -DE8 -DE11 -DE12 -DE13 -DE14 -
DE15 -DE16 -DE17 -
DE18 -DE20 -DE21 -DE22 -DE23 -DE24 -DE26 -DE27 -DE29 -DE31 -DE32 -
DE33"

elif [ $i -eq "2" ];then
   listaDefeitos="-DE1 -DE2 -DE3 -DE7 -DE8 -DE11 -DE12 -DE13 -DE14 -
DE15 -DE16 -DE17 -
DE18 -DE20 -DE21 -DE22 -DE23 -DE24 -DE26 -DE27 -DE29 -DE32 -DE33"

elif [ $i -eq "3" ];then
   listaDefeitos="-DE1 -DE2 -DE3 -DE7 -DE8 -DE12 -DE13 -DE14 -DE15 -
DE16 -DE17 -DE18 -
DE20 -DE21 -DE22 -DE23 -DE24 -DE26 -DE27 -DE29 -DE32 -DE33"

elif [ $i -eq "4" ];then
   listaDefeitos="-DE1 -DE2 -DE3 -DE7 -DE8 -DE12 -DE13 -DE14 -DE15 -
DE16 -DE17 -DE18 -
DE20 -DE21 -DE22 -DE23 -DE24 -DE27 -DE29 -DE32 -DE33"

elif [ $i -eq "5" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE7 -DE8 -DE12 -DE13 -DE14 -DE16 -
DE17 -DE18 -DE20 -
DE21 -DE22 -DE23 -DE24 -DE27 -DE29 -DE32 -DE33"

elif [ $i -eq "6" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE8 -DE12 -DE13 -DE14 -DE16 -DE17
-DE18 -DE20 -DE21 -
DE22 -DE23 -DE24 -DE27 -DE29 -DE32 -DE33"

elif [ $i -eq "7" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE8 -DE12 -DE13 -DE16 -DE17 -DE18
-DE20 -DE21 -DE22 -
DE23 -DE24 -DE27 -DE29 -DE32 -DE33"
```

```
elif [ $i -eq "8" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE8 -DE12 -DE13 -DE17 -DE18 -DE20
-DE21 -DE22 -DE23 -
DE24 -DE27 -DE29 -DE32 -DE33"

elif [ $i -eq "9" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE8 -DE12 -DE13 -DE17 -DE18 -DE20
-DE21 -DE22 -DE24 -
DE27 -DE29 -DE32 -DE33"

elif [ $i -eq "10" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE12 -DE13 -DE17 -DE18 -DE20 -DE21
-DE22 -DE24 -DE27 -
DE29 -DE32 -DE33"

elif [ $i -eq "11" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE12 -DE13 -DE17 -DE18 -DE20 -DE21
-DE22 -DE24 -DE27 -
DE32 -DE33"

elif [ $i -eq "12" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE12 -DE13 -DE17 -DE18 -DE20 -DE21
-DE22 -DE27 -DE32 -
DE33"

elif [ $i -eq "13" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE12 -DE13 -DE17 -DE18 -DE21 -DE22
-DE27 -DE32 -DE33"

elif [ $i -eq "14" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE12 -DE13 -DE17 -DE18 -DE22 -DE27
-DE32 -DE33"

elif [ $i -eq "15" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE12 -DE13 -DE17 -DE18 -DE27 -DE32
-DE33"

elif [ $i -eq "16" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE12 -DE13 -DE17 -DE18 -DE32 -
DE33"

elif [ $i -eq "17" ];then
   listaDefeitos= "-DE1 -DE2 -DE3 -DE12 -DE13 -DE18 -DE32 -DE33"

fi

funcoes=$(cat listaFuncoes)
echo ""
echo "==================="
echo "Sequencia $i"
echo "==================="
limpa.sh

#Faz análise estática do programa
```

```
newpoketool prepro.c -L3 -cdf-pu -f_todas_

#Compilação condicional do programa - considera os defeitos da
variável listadefeitos
gcc testeprog.c -o testeprog $listaDefeitos -lm

#Execução do programa com os dados de teste
mypokeexec testeprog $2 $3

#Análise da cobertura - neste caso o critério é todos-nós
for v in $funcoes; do
   echo "Analisando $v ..."
   newpokeaval -d$v -nos $2 to $3 > temp
done

for v in $funcoes; do
   ere=0
   erne=0

   echo $(wc -w ./$v/exec_nos.tes) > contador
   linha=$(cat contador)

   ere=${linha:0:2}
   ere=$((ere-9))

   echo $(wc -w ./$v/nosoutput.tes) > contador
   linha=$(cat contador)

   erne=${linha:0:2}
   erne=$((erne-10))

   sere=$((sere+ere))
   ser=$((ser+ere+erne))
done

echo "scale=7;$sere/$ser" |bc
#Registra a cobertura de nós a cada defeito removido
echo "prepro$1" >> coberturaNos
echo "scale=7;$sere/$ser" |bc >> coberturaNos
```

Script para medir a cobertura do critério de teste Todos-Arcos

analisaArcos.sh: Este script mede a cobertura atingida pelo critério *Todo-Arcos*. Para evitar repetição de texto, será apresentada apenas a parte do código referente à instrumentação do programa, execução dos testes e análise da cobertura, realizada pela POKE-TOOL. O restante do código é igual ao apresentado no script analisaNos.sh. A execução deste script é realizada por outro script de nome ExecArcos.sh, semelhante ao ExecNos.sh, apresentado anteriormente.

```
funcoes=$(cat listaFuncoes)
echo "==================="
echo "Prepro$i"
echo "==================="
limpa.sh

newpoketool prepro.c -L3 -cdf-pu -f_todas_
gcc testeprog.c -o testeprog $listaDefeitos -lm

mypokeexec testeprog $2 $3

for v in $funcoes; do
   echo "Analisando $v ..."
   newpokeaval -d$v -arcs $2 to $3 > temp
done

for v in $funcoes; do
   ere=0
   erne=0

   if [ -f ./$v/exec_arc.tes ]; then
      rm -f contArc
      echo $(wc -l ./$v/exec_arc.tes) > contArc1
      linha=$(cat contArc1)
      ere=${linha%./$v/exec_arc.tes}
      ere=$((ere-7))
   fi
   if [ -f ./$v/arcoutput.tes ]; then
      rm -f contArc2
      echo $(wc -l ./$v/arcoutput.tes) > contArc2
      linha=$(cat contArc2)
      erne=${linha%./$v/arcoutput.tes}
      erne=$((erne-7))
   fi

   sere=$((sere+ere))
   ser=$((ser+ere+erne))
done
echo "prepro$1" >> coberturaArcos
echo "scale=7;$sere/$ser" |bc >> coberturaArcos
```

Script para medir a cobertura do critério de teste Potenciais-Usos

analisaPU.sh: Este script mede a cobertura atingida pelo critério *Todos-Potenciais-Usos*. Para evitar repetição de texto, será apresentada apenas a parte do código referente à instrumentação do programa, execução dos testes e análise da cobertura, realizada pela POKE-TOOL. O restante do código é igual ao apresentado no script analisaNos.sh. A

execução deste script é realizada por outro script de nome ExecPU.sh, semelhante ao ExecNos.sh, apresentado anteriormente.

```
funcoes=$(cat listaFuncoes)
echo "==================="
echo "Prepro$i"
echo "==================="
#remove estrutura criada no teste anterior
for v in $funcoes; do
   echo "Excluindo arquivos de $v..."
   rm -f ./$v/*
done

newpoketool prepro.c -L3 $listaDefeitos -cdf-pu -f_todas_
gcc testeprog.c -o testeprog -lm
mypokeexec testeprog $2 $3
for v in $funcoes; do
   echo "Analisando $v ..."
   newpokeaval -d$v -pu $2 to $3 > outPU
done
for v in $funcoes; do
   ere=0
   erne=0
   if [ -f ./$v/exec_pu.tes ]; then
      for LINHA in $(cat ./$v/exec_pu.tes); do
         if [ ${LINHA:0:1} = "<" ]; then
            ere=$((ere+1))
         fi
      done
   fi
   if [ -f ./$v/puoutput.tes ]; then
      for LINHA in $(cat ./$v/puoutput.tes); do
         if [ ${LINHA:0:1} = "<" ]; then
            erne=$((erne+1))
         fi
      done
   fi
   sere=$((sere+ere))
   ser=$((ser+ere+erne))
done
echo "prepro$1" >> coberturaPU
echo "scale=7;$sere/$ser" |bc >> coberturaPU
```

Script para medir a cobertura do critério de teste Potenciais-Usos/Definição-Uso

analisaPUDU.sh: Este script mede a cobertura atingida pelo critério *Todos-Potenciais-Usos/DU*. Para evitar repetição de texto, será apresentada apenas a parte do código referente à instrumentação do programa, execução dos testes e análise da cobertura,

realizada pela POKE-TOOL. O restante do código é igual ao apresentado no script analisaNos.sh. A execução deste script é realizada por outro script de nome ExecPUDU.sh, semelhante ao ExecNos.sh, apresentado anteriormente.

```
funcoes3=$(cat funcoes3)
echo ""
echo "==================="
echo "Sequencia $i"
echo "==================="

limpa.sh
newpoketool prepro.c -L3 -cdf-pu -f_todas_

gcc testeprog.c -o testeprog $listaDefeitos -lm
mypokeexec testeprog $2 $3

for v in $funcoes3; do
   echo "Analisando $v ..."
   newpokeaval -d$v -pudu $2 to $3 > outPUDU
done
for v in $funcoes3; do
   ere=0
   erne=0
   for LINHA in $(cat ./$v/exec_pudu.tes); do
      if [ ${LINHA:0:1} = "<" ]; then
         ere=$((ere+1))
      fi
   done

   for LINHA in $(cat ./$v/puduoutput.tes); do
      if [ ${LINHA:0:1} = "<" ]; then
         erne=$((erne+1))
      fi
   done

   sere=$((sere+ere))
   ser=$((ser+ere+erne))
done
echo "prepro$i" >> coberturaPUDU
echo "scale=7;$sere/$ser" |bc >> coberturaPUDU
```

Script para medir a cobertura do critério de teste Potenciais-Definições-Usos

analisaPDU.sh: Este script mede a cobertura atingida pelo critério *Todos-Potenciais-DU-Caminhos*. Para evitar repetição de texto, será apresentada apenas a parte do código referente à instrumentação do programa, execução dos testes e análise da cobertura,

realizada pela POKE-TOOL. O restante do código é igual ao apresentado no script analisaNos.sh. A execução deste script é realizada por outro script de nome ExecPDU.sh, semelhante ao ExecNos.sh, apresentado anteriormente.

```
funcoes=$(cat funcoes3)
echo ""
echo "==================="
echo "Sequencia $i"
echo "==================="

limpa.sh
newpoketool prepro.c -L3 -cpdu -f_todas_

gcc testeprog.c -o testeprog $listaDefeitos -lm
mypokeexec testeprog $2 $3

for v in $funcoes; do
   echo "Analisando $v ..."
   newpokeaval -d$v -pdu $2 to $3 > outPDU
done
for v in $funcoes; do
   ere=0
   erne=0
   if [ -f ./$v/exec_pdu.tes ]; then
      rm -f contPDU1
      echo $(wc -l ./$v/exec_pdu.tes) > contPDU1
      linha=$(cat contPDU1)
      ere=${linha%./$v/exec_pdu.tes}
      ere=$((ere-9))
   fi

   if [ -f ./$v/pduoutput.tes ]; then
      rm -f contPDU1
      echo $(wc -l ./$v/pduoutput.tes) > contPDU2
      linha=$(cat contPDU2)
      erne=${linha%./$v/pduoutput.tes}
      erne=$((erne-9))
   fi
   sere=$((sere+ere))
   ser=$((ser+ere+erne))
done
echo "prepro$i" >> coberturaPDU
echo "scale=7;$sere/$ser" |bc >> coberturaPDU
```

Script para medir a cobertura do critério de teste Todos-Usos

analisaUsos.sh: Este script mede a cobertura atingida pelo critério *Todos-Usos*. Para evitar repetição de texto, será apresentada apenas a parte do código referente à instrumentação do programa, execução dos testes e análise da cobertura, realizada pela

POKE-TOOL. O restante do código é igual ao apresentado no script analisaNos.sh. A execução deste script é realizada por outro script de nome ExecUsos.sh, semelhante ao ExecNos.sh, apresentado anteriormente.

```
funcoes3=$(cat funcoes3)
echo ""
echo "==================="
echo "Sequencia $i"
echo "==================="

limpa.sh
newpoketool prepro.c -L3 -cdf-pu -f_todas_

gcc testeprog.c -o testeprog $listaDefeitos -lm
mypokeexec testeprog $2 $3

for v in $funcoes3; do
   echo "Analisando $v ..."
   newpokeaval -d$v -uses $2 to $3 > outTodosUsos
done
for v in $funcoes3; do
   ere=0
   erne=0
   for LINHA in $(cat ./$v/exec_uses.tes); do
      if [ ${LINHA:0:1} = "<" ]; then
         ere=$((ere+1))
      fi
   done

   for LINHA in $(cat ./$v/usesoutput.tes); do
      if [ ${LINHA:0:1} = "<" ]; then
         erne=$((erne+1))
      fi
   done

   sere=$((sere+ere))
   ser=$((ser+ere+erne))
done
echo "scale=7;$sere/$ser" |bc

echo "prepro$1" >> coberturaTodosUsos
echo "scale=7;$sere/$ser" |bc >> coberturaTodosUsos
```

Script para eliminar arquivos temporários

limpa.sh: Este script elimina todos os arquivos temporários gerados pelo programa SPACE e pela POKE-TOOL. A cada defeito encontrado, diversos dados de saída são gerados pelo SPACE. A POKE-TOOL, por sua vez, também gerar diversos arquivos

temporários no processo de instrumentação e avaliação da cobertura. Eliminar estes arquivos temporários, entre os defeitos encontrados, garante que a avaliação da medição da cobertura de um defeito revelado não tenha interferência da avaliação do defeito revelado anteriormente.

```
#lista de funcoes de SPACE
#cada funcao gera uma pasta onde sao gravados
#arquivos temporarios de controle da POKE-TOOL
funcoes=$(cat listaFuncoes)

#remove estrutura criada no teste anterior
for v in $funcoes; do
   echo "Excluindo arquivos de $v..."
   rm -f ./$v/*
done

#remove arquivos temporarios gerados por SPACE
rm -f ./output/*
rm -f ./dados/*.dat
rm -f ./dados/outo*
```

Script para calcular a confiabilidade

calcconf.sh: Este script calcula a confiabilidade do programa SPACE segundo o perfil operacional 3 utilizado no experimento. Foram utilizadas 25 versões do SPACE. Os defeitos foram inseridos e removidos através de diretivas de compilação condicional, evitando-se assim a inserção de novos defeitos. A ordem dos defeitos é a mesma em que eles foram revelados.

```
#Funcao para calcular a confiabilidade
calcConf()
{
   i=1
   s=0
   f=0
   conf_old=0
   conf_new=0
   while [ "$i" -le 10000 ]; do
      prepro < ./dados/in$i > ./dados/outprepro$i
      oracolo2 < ./dados/in$i > ./dados/outoracolo2$i
      cmp ./dados/outprepro$i ./dados/outoracolo2$i -s
      r=$?
      if [ "$r" -eq "1" ] || [ "$r" -eq "2" ]; then
         f=$((f+1))
         if [ "$f" -eq "1000" ]; then
```

```
            conf_old=$(echo "scale=7;$f/$i" |bc)
         elif [ "$f" -gt "1000" ]; then
            conf_new=$(echo "scale=7;$f/$i" |bc)
            dif=$(echo "scale=7;$conf_old-$conf_new" |bc)
            #se dif for negativo
            if [ $(echo "$dif < 0"|bc) -eq 1 ]; then
               dif=$(echo "scale=7;$dif*-1" |bc)
            fi
            if [ $(echo "$dif < 0.001"|bc) -eq 1 ]; then
               conf_new=$(echo "scale=7;1.0-$conf_new" |bc)
               echo $1 $conf_new >> cresConfP3
               break
            fi
         fi
      else
         s=$((s+1))
      fi
      rm ./dados/outprepro$i
      rm ./dados/outoracolo2$i
      i=$((i+1))
   done
}

rm -f cresConfP3

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE5 -DE6 -DE7 -DE8 -DE9 -
DE10 -DE11 -DE12 -DE13 -DE14 -DE15 -DE16 -DE17 -DE18 -DE19 -DE20 -
DE21 -DE22 -DE23 -DE24 -DE25 -DE26 -DE27 -DE28 -DE29 -DE32 -DE33
calcConf "-"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE6 -DE7 -DE8 -DE9 -DE10 -
DE11 -DE12 -DE13 -DE14 -DE15 -DE16 -DE17 -DE18 -DE19 -DE20 -DE21 -
DE22 -DE23 -DE24 -DE25 -DE26 -DE27 -DE28 -DE29 -DE32 -DE33
calcConf "E5"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE9 -DE10 -DE11
-DE12 -DE13 -DE14 -DE15 -DE16 -DE17 -DE18 -DE19 -DE20 -DE21 -DE22 -
DE23 -DE24 -DE25 -DE26 -DE27 -DE28 -DE29 -DE32 -DE33
calcConf "E6"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE10 -DE11 -DE12
-DE13 -DE14 -DE15 -DE16 -DE17 -DE18 -DE19 -DE20 -DE21 -DE22 -DE23 -
DE24 -DE25 -DE26 -DE27 -DE28 -DE29 -DE32 -DE33
calcConf "E9"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE10 -DE11 -DE12
-DE13 -DE14 -DE15 -DE16 -DE17 -DE18 -DE19 -DE20 -DE21 -DE22 -DE23 -
DE24 -DE25 -DE26 -DE27 -DE29 -DE32 -DE33
calcConf "E28"
```

```
#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE10 -DE11 -DE12
-DE14 -DE15 -DE16 -DE17 -DE18 -DE19 -DE20 -DE21 -DE22 -DE23 -DE24 -
DE25 -DE26 -DE27 -DE29 -DE32 -DE33
calcConf "E13"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE10 -DE11 -DE12
-DE14 -DE15 -DE16 -DE17 -DE18 -DE20 -DE21 -DE22 -DE23 -DE24 -DE25 -
DE26 -DE27 -DE29 -DE32 -DE33
calcConf "E19"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE10 -DE11 -DE12
-DE14 -DE15 -DE16 -DE17 -DE18 -DE20 -DE21 -DE22 -DE23 -DE24 -DE26 -
DE27 -DE29 -DE32 -DE33
calcConf "E25"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE10 -DE11 -DE12
-DE14 -DE16 -DE17 -DE18 -DE20 -DE21 -DE22 -DE23 -DE24 -DE26 -DE27 -
DE29 -DE32 -DE33
calcConf "E15"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE10 -DE11 -DE12
-DE14 -DE17 -DE18 -DE20 -DE21 -DE22 -DE23 -DE24 -DE26 -DE27 -DE29 -
DE32 -DE33
calcConf "E16"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE10 -DE11 -DE12
-DE14 -DE17 -DE18 -DE20 -DE21 -DE22 -DE23 -DE26 -DE27 -DE29 -DE32 -
DE33
calcConf "E24"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE11 -DE12 -DE14
-DE17 -DE18 -DE20 -DE21 -DE22 -DE23 -DE26 -DE27 -DE29 -DE32 -DE33
calcConf "E10"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE8 -DE11 -DE12 -DE14
-DE17 -DE18 -DE20 -DE21 -DE22 -DE23 -DE27 -DE29 -DE32 -DE33
calcConf "E26"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE11 -DE12 -DE14 -
DE17 -DE18 -DE20 -DE21 -DE22 -DE23 -DE27 -DE29 -DE32 -DE33
calcConf "E8"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE12 -DE14 -DE17 -
DE18 -DE20 -DE21 -DE22 -DE23 -DE27 -DE29 -DE32 -DE33
calcConf "E11"
```

```
#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE12 -DE14 -DE17 -
DE18 -DE20 -DE21 -DE22 -DE23 -DE27 -DE32 -DE33
calcConf "E29"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE7 -DE12 -DE17 -DE18 -
DE20 -DE21 -DE22 -DE23 -DE27 -DE32 -DE33
calcConf "E14"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE12 -DE17 -DE18 -DE20 -
DE21 -DE22 -DE23 -DE27 -DE32 -DE33
calcConf "E7"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE12 -DE17 -DE18 -DE20 -
DE21 -DE22 -DE27 -DE32 -DE33
calcConf "E23"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE12 -DE17 -DE18 -DE21 -
DE22 -DE27 -DE32 -DE33
calcConf "E20"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE12 -DE17 -DE18 -DE22 -
DE27 -DE32 -DE33
calcConf "E21"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE12 -DE17 -DE18 -DE27 -
DE32 -DE33
calcConf "E22"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE12 -DE18 -DE27 -DE32 -
DE33
calcConf "E17"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE18 -DE27 -DE32 -DE33
calcConf "E12"


#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE27 -DE32 -DE33
calcConf "E18"

#prepro (SPACE) -lm com defeitos inseridos
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE32 -DE33
calcConf "E27"

#prepro (SPACE) -lm com defeitos inseridos
```

```
gcc prepro.c -o prepro -lm -DE1 -DE2 -DE3 -DE32
calcConf "E33"
```

APÊNDICE B

Programa Matlab para estimar os parâmetros do MBBC

Função FuncaoW2.M

```
function [total,G]=funcaow2(X);
total=1;
TESTE = [1  2     3     4     5     6     7     8     10    12    14    16
      22    29    36    55    75    106   169   232   448   691   941
      1191];

NOS = [0.2218358  0.2218358   0.3038942   0.3038942   0.3038942
      0.3038942   0.3019257   0.3287482   0.4030261   0.4030261
      0.4028925   0.4593663   0.4889807   0.5192837   0.5220385
      0.5929752   0.6239669   0.6515151   0.6673525   0.6680327
      0.6936691   0.6943498   0.6991150   0.6991150];

ARCOS = [0.1240000      0.1240000   0.1653333   0.1653333   0.1653333
      0.1653333   0.1640211   0.1865079   0.2579365   0.2579365
      0.2582781   0.2953642   0.3165562   0.3443708   0.3403973
      0.4119205   0.4397350   0.4701986   0.4914361   0.4927916
      0.5215686   0.5228758   0.5281045   0.5281045];

PU = [0.0769371   0.0769371   0.0961714   0.0961714   0.0961714
      0.0961714   0.0960307   0.1487104   0.1902323   0.1902323
      0.1901189   0.2451697   0.2592863   0.2748901   0.2723279
      0.3541361   0.3956808   0.4899341   0.5197368   0.5214607
      0.5411804   0.5413495   0.5469304   0.5469304];

PUDU = [0.0699761 0.0699761   0.0899432   0.0899432   0.0899432
      0.0899432   0.0898115   0.1282238   0.1680995   0.1680995
      0.1679780   0.2018298   0.2161024   0.2304172   0.2291361
      0.2884333   0.3213762   0.3856149   0.4105618   0.4145032
      0.4359884   0.4361576   0.4410620   0.4410620];

PDU = [0.0471745  0.0471745   0.0565085   0.0565085   0.0565085
      0.0565085   0.0563947   0.0674723   0.0941591   0.0941591
      0.0940020   0.1134072   0.1260080   0.1371313   0.1368792
      0.1709100   0.1799848   0.1986387   0.2099153   0.2076321
      0.2191011   0.2195505   0.2226966   0.2226966];

USES = [0.1934963 0.1934963   0.1934963   0.1934963   0.1934963
      0.1934963   0.1934963   0.1934963   0.3115774   0.3115774
      0.3115774   0.3115774   0.3401752   0.3823800   0.3823800
      0.4580258   0.4903136   0.5438191   0.5622693   0.5622693
      0.6019372   0.6023985   0.6067804   0.6067804];

COB = 1 - USES;
```

```
COB = 1 - NOS;

total =0;
TOTAL(1) = 0;

for i = 4:24

TOTAL(i) = log(X(1)-i) + log(X(2)+X(3)*COB(i)) +
log(TESTE(i+1)^(X(2)+X(3)*COB(i)-1)) - (X(1)-
i)*(TESTE(i+1)^(X(2)+X(3)*COB(i)) - TESTE(i)^(X(2)+X(3)*COB(i)));
total = total+ TOTAL(i);
end;
veross = total;
total = -total;
disp('log(veross)');disp(veross);
disp('total     veross     N     a0     a1');
disp([total,veross,X(1),X(2),X(3)]);
%-------------------------------------------------------------------
```

Função CalcW2.M

```
X =
constr('funcaow2',[29;0.02;0.01],[zeros(13,1);1000;zeros(4,1)],[29;0.0155
6;0.02580],[40;2;2])
disp('X');disp(X);
```

Programa Matlab para estimar os parâmetros do MFIBC

Função FuncaoG3.M

```
function [Vero,G]=funcaog3(X);
TESTEN = [1 2     3     4     5     6     7     8     10    12    14    16
      22    29    36    55    75    106   169   232   448   691   941
      1191];

TESTEX = [1 1     1     1     1     1     1     1     2     2     2     2
      6     7     7     19    20    31    63    63    216   243   250
      250];

NOS = [0.2218358  0.2218358   0.3038942   0.3038942   0.3038942
      0.3038942   0.3019257   0.3287482   0.4030261   0.4030261
      0.4028925   0.4593663   0.4889807   0.5192837   0.5220385
      0.5929752   0.6239669   0.6515151   0.6673525   0.6680327
      0.6936691   0.6943498   0.6991150   0.6991150];

ARCOS = [0.1240000      0.1240000   0.1653333   0.1653333   0.1653333
      0.1653333   0.1640211   0.1865079   0.2579365   0.2579365
      0.2582781   0.2953642   0.3165562   0.3443708   0.3403973
      0.4119205   0.4397350   0.4701986   0.4914361   0.4927916
      0.5215686   0.5228758   0.5281045   0.5281045];
```

```
PU = [0.0769371   0.0769371   0.0961714   0.0961714   0.0961714
      0.0961714   0.0960307   0.1487104   0.1902323   0.1902323
      0.1901189   0.2451697   0.2592863   0.2748901   0.2723279
      0.3541361   0.3956808   0.4899341   0.5197368   0.5214607
      0.5411804   0.5413495   0.5469304   0.5469304];

PUDU = [0.0699761 0.0699761   0.0899432   0.0899432   0.0899432
      0.0899432   0.0898115   0.1282238   0.1680995   0.1680995
      0.1679780   0.2018298   0.2161024   0.2304172   0.2291361
      0.2884333   0.3213762   0.3856149   0.4105618   0.4145032
      0.4359884   0.4361576   0.4410620   0.4410620];

PDU = [0.0471745  0.0471745   0.0565085   0.0565085   0.0565085
      0.0565085   0.0563947   0.0674723   0.0941591   0.0941591
      0.0940020   0.1134072   0.1260080   0.1371313   0.1368792
      0.1709100   0.1799848   0.1986387   0.2099153   0.2076321
      0.2191011   0.2195505   0.2226966   0.2226966];

USES=[0.2430811   0.2430811   0.2430811   0.2430811   0.2430811
      0.2691420   0.2896678   0.3341789   0.3341789   0.3341789
      0.3761531   0.3761531   0.3761531   0.3971402   0.4467250
      0.4686346   0.4940036   0.5041512   0.5142988   0.5142988
      0.5696494   0.5809501   0.5841789];



TESTE = TESTEX;

COB = 1-USES;

% USO DA COBERTURA
denom =0;
soma2 =0;
soma3 = 0;
ptermo2 =0;
ptermo3 =0;
smembro =0;
tmembro =0;


for  i = 1:21

TERMO(i) = ( X(1) + X(2)*COB(i) );

% definicao da 1a. equacao

denom = denom + (TESTE(i))*( TERMO(i) )^(i-1);
D = 24/denom;

% definicao da 2a. equacao

ptermo2 =  ptermo2 + (i-1)/TERMO(i);
soma2 =  soma2 + (i-1)*TESTE(i)*( TERMO(i) )^(i-2);
stermo2 = D*soma2;
```

```
% definicao da 3a. equacao

ptermo3 = ptermo3 + ( (i-1)*COB(i) )/TERMO(i);
soma3 = soma3 +  (i-1)*COB(i)*TESTE(i)*( TERMO(i) )^(i-2);
stermo3 = D*soma3;

% calculo da funcao de verossi/ca

smembro = smembro + (i-1)*log( TERMO(i) );
tmembro = tmembro + TESTE(i)*( TERMO(i)^(i-1) );

end

pmembro = 24*log( D );
Vero = pmembro + smembro - D*tmembro;
Vero = -Vero;

equa1 = D - 24/denom;
equa2 = ptermo2 - stermo2;
equa3 = ptermo3 - stermo3;

sistema = (equa1)^2 + (equa2)^2 + (equa3)^2;
%G(1) =  X(1) + X(2)*COB(5) -1;
%G(2) =  X(1) + X(2)*COB(24) -1;
G(1) = -Vero -60.;

G(2) = -X(1)*X(2);
disp('   equa1        equa2       equa3      Vero'); disp([equa1 equa2
equa3 -Vero]);
disp('     D       X(1)         X(2)');disp([D,X(1),X(2)]);
disp('restricoes');disp([G(1)]);
disp('  *****  fim do programa ****** ');
```

Função CalcG3.M

```
X =
constr('funcaog3',[0.5334;0.4618],[zeros(13,1);2000;zeros(4,1)],[0.1;0.1]
,[0.7;0.9]);

disp('X');
disp(X);
```