PREDIÇÃO, ESTIMAÇÃO E MEDIÇÃO DA CONFIABILIDADE DURANTE O CICLO DE VIDA DO SOFTWARE

Rogério Nesi Pereira Cardoso

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Ana Regina C. da Rocha, D.Sc.
(Presidente)

Prof. Arndt Von Staa, Ph.D.

Prof. Paulo Afonso Lopes da Silva, Ph.D.

Prof. Jano Moreira de Souza, Ph.D.

Dr. Firmo Freire, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 1990

CARDOSO, ROGÉRIO NESI PEREIRA

Predição, Estimação e Medição da Confiabilidade Durante o Ciclo de Vida do Software (Rio de Janeiro) 1990.

XVII, 266 p. 29,7 cm (COPPE/UFRJ) M.Sc., Engenharia de Sistemas e Computação, 1990)

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1. Confiabilidade de Software

I. COPPE/UFRJ II. Titulo (série)

À Bárbara, Isabela e aos meus pais.

AGRADECIMENTOS

À Ana Regina Cavalcanti da Rocha pela eficiente orientação e incentivo dados a este trabalho;

À César José dos Santos pelas discussões, críticas e sugestões feitas;

Aos professores Jano Moreira de Souza, Arndt Von Staa, Paulo Afonso Lopes da Silva e ao Dr. Firmo Freire pela participação na banca examinadora;

À meu pai pela ajuda no trabalho de digitação e revisão de texto;

Aos colegas da COPPE e da COBRA pelo interesse e incentivo;

Ao CNPq e a CAPES pela ajuda financeira.

Resumo da Tese apresentada a COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PREDIÇÃO, ESTIMAÇÃO E MEDIÇÃO DA CONFIABILIDADE DURANTE O CICLO DE VIDA DO SOFTWARE

Rogério Nesi Pereira Cardoso

DEZEMBRO, 1990

Orientadora : Ana Regina C. Rocha

Programa : Engenharia de Sistemas e Computação

Este trabalho tem o objetivo de oferecer mecanismos que garantam a qualidade de programas através de um acompanhamento sistemático do seu comportamento ao longo de todo o ciclo de vida, desde as fases iniciais de concepção até sua operação final. É realizado um estudo da literatura no que se refere às proposições de métricas para a predição da confiabilidade de software, aos modelos estatísticos utilizados para a sua estimação e aos procedimentos necessários para a sua medição na fase operacional. Uma ferramenta automatizada é proposta com o objetivo de predizer a confiabilidade nas etapas de especificação, projeto e codificação, estimá-la na fase de testes e medi-la na fase operacional.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.)

PREDICTION, ESTIMATION AND MEASUREMENT OF THE RELIABILITY DURING THE LIFE-CYCLE OF SOFTWARE

Rogério Nesi Pereira Cardoso

DECEMBER, 1990

Thesis Supervisor : Ana Regina C. Rocha

Department : Engenharia de Sistemas e Computação

This thesis has the objective of providing mechanisms that assure the software quality by systematic accompanying of its behavior duringthe life-cycle of the software , from its conception to its final operation. A review of the literature is doing relativing to software reliability metrics, applicable statistical models to estimation and measurement during the operational phase. A CASE tool is proposed, whose objectives are three: first, to predict the software reliability during the specification, the design and the implementation phases; second, to estimate the reliability during tests and, third, to measure it during the operational phase.

ÍNDICE

Pag.

DEDICATÓRIA..... iii

AGRADECIMENTOS..... iv

RESUMO..... v

ABSTRACT..... vi

CAPÍTULO I - INTRODUÇÃO..... 1

1.1 - Ciclo de vida do software..... 3

1.2 - Qualidade de software..... 5

1.3 - Confiabilidade de software X confiabilidade de hardware..... 11

1.4 - Objetivo e organização deste trabalho..... 16

CAPÍTULO II - MÉTODO PARA AVALIAÇÃO DA QUALIDADE COM BASE NA PREDIÇÃO, ESTIMAÇÃO E MEDIÇÃO DA CONFIABILIDADE DE SOFTWARE..... 19

2.1 - Introdução..... 19

2.2 - Especificação da confiabilidade..... 22

2.3 - Predição da confiabilidade..... 27

2.3.1 - Esforços de padronização de métricas..... 28

2.3.1.1 - IEEE P.982 ("Measures to Produce Reliable Software")..... 30

2.3.1.2 - IEEE P.1061 ("Standard for Software Quality Metrics Methodology")..... 40

2.3.1.3 - DOD-STD-2167/2168 ("Defense System Development" e "Software Quality Evaluation Program")..... 41

2.3.1.4 - RADC-TR-85-37 ("Specification of Software Quality Atributes")..... 46

2.3.1.4.1 - Métricas de qualidade..... 48
2.3.2 - Método para a predição da confiabilidade de software..... 55
2.3.3 - Predição da confiabilidade de especificações..... 62
2.3.3.1 - Confiabilidade da representação de especificações..... 66
2.3.3.2 - Confiabilidade conceitual de especificações..... 70
2.3.3.3 - Confiabilidade da utilização de especificações..... 73
2.3.4 - Predição da confiabilidade de projetos... 79
2.3.4.1 - Confiabilidade da representação de projetos..... 83
2.3.4.2 - Confiabilidade da utilização de projetos..... 91
2.3.4.3 - Confiabilidade conceitual de projetos..... 102
2.3.5 - Predição da confiabilidade de programas.. 108
2.3.5.1 - Confiabilidade da representação de programas..... 113
2.3.5.2 - Confiabilidade conceitual de programas..... 121
2.3.5.3 - Confiabilidade da utilização de programas..... 126
2.4 - Estimação da confiabilidade..... 137
2.4. 1 - Modelo de disseminação..... 151
2.4. 2 - Modelo de Schick-Wolverton..... 153
2.4. 3 - Modelo modificado de Schick-Wolverton.... 155

2.4. 4 - Modelo de Poisson geométrico Jelinski-Moranda..... 156
2.4. 5 - Modelo De-Eutroficação Jelinsk-Moranda... 156
2.4. 6 - Modelo geométrico De-Eutroficação Jelinsk-Moranda..... 158
2.4. 7 - Modelo geométrico modificado De-Eutroficação..... 158
2.4. 8 - Modelo de crescimento da confiabilidade.. 159
2.4. 9 - Modelo experimental de Shooman..... 160
2.4.10 - Modelo de Shooman e Natarajan..... 165
2.4.11 - Modelo de Weibull..... 165
2.4.12 - Modelo bayesiano de Goel e Okumoto..... 166
2.4.13 - Modelo bayesiano de Littlewood e Verral.. 167
2.4.14 - Modelo de Shooman e Triverdi-Markov..... 167
2.4.15 - Modelo markoviano de Littlewood..... 168
2.4.16 - Modelo semi-markoviano de Littlewood..... 169
2.4.17 - Modelo de Moranda..... 169
2.4.18 - Micro modelo de Shooman..... 170
2.4.19 - Modelo de Nelson..... 170
2.4.20 - Modelo de Hecht..... 171
2.4.21 - Modelo Básico de Musa..... 173
2.4.22 - Modelo de Poisson Logaritmico de Musa-Okumoto..... 175
2.5 - Medição da confiabilidade..... 176
2.5.1 - Sistema de gerenciamento de falhas..... 179

CAPÍTULO III - PRESTIME - PREDIÇÃO, ESTIMAÇÃO E MEDIÇÃO DACONFIABILIDADE DE SOFTWARE..... 193
3.1 - Introdução..... 193

3.2 - Descrição geral..... 193
3.3 - Sistema de especificação da confiabilidade..... 197
3.3.1 - Descrição geral..... 197
3.3.2 - Ferramentas de apoio à especificação da metade confiabilidade..... 200
3.3.2.1 - Especificador de confiabilidade..... 200
3.3.2.2 - Especificador de ambientes de desenvolvimento..... 200
3.4 - Sistema de predição da confiabilidade..... 201
3.4.1 - Descrição geral..... 201
3.4.2 - Ferramentas de apoio à predição da confiabilidade..... 202
3.4.2.1 - Avaliador de especificações de requisitos de software..... 206
3.4.2.2 - Avaliador de projetos..... 206
3.4.2.3 - Avaliador de programas sensível à linguagem..... 206
3.5 - Sistema de estimação da confiabilidade..... 207
3.5.1 - Descrição geral..... 207
3.5.2 - Ferramentas de apoio à estimação da confiabilidade..... 209
3.5.2.1 - Analisador de falhas..... 209
3.5.2.2 - Gerador de gráficos de confiabilidade..... 210
3.5.2.3 - Simulador de custos e de cronograma. 212
3.5.2.4 - Gerenciador de falhas de teste..... 212
3.5.2.5 - Gerador de casos de teste aleatórios..... 212
3.6 - Sistema de medição da confiabilidade..... 213

3.6.1 - Descrição geral..... 214
3.6.2 - Ferramentas de apoio à medição da confiabilidade..... 216
3.6.2.1 - Gerenciador de falhas de operação.... 216
3.6.2.2 - Avaliador de comportamento operacional..... 216
3.7- Gerador de laudos de confiabilidade..... 217

CAPÍTULO IV - Especificação de Requisitos do protótipo do Sistema de Medição da Confiabilidade... 219
4.1- Introdução..... 219
4.1.1- Propósito..... 219
4.1.2- Escopo..... 219
4.1.3- Referências..... 220
4.1.4- Glossário..... 221
4.1.5- Visão geral da especificação..... 221
4.1.6- Interface do produto com outros produtos ou projetos..... 222
4.1.7- Funções do produto..... 223
4.2- Requisitos específicos..... 223
4.2.1- Requisitos de informação..... 223
4.2.2- Requisitos de interface..... 224
4.2.2.1- Interface do ambiente PRESTIME com o o usuário..... 224
4.2.2.2- Interface do SMC com o usuário..... 231
4.2.2.3- Interface com o software..... 240
4.2.2.4- Interface com o hardware..... 241
4.2.3- Requisitos de desempenho..... 241
4.2.4- Requisitos de qualidade..... 242

4.2.5- Requisitos funcionais..... 243
4.2.5.1- Diagrama de Fluxo de Dados..... 243
4.2.5.2- Descrição dos processos..... 243
4.2.5.2.1- Elaborar Relatório de Investigação..... 243
4.2.5.2.2- Analisar, Corrigir e Iniciar Alteração..... 243
4.2.5.2.3- Corrigir Procedimentos de Teste 244
4.2.5.2.4- Medir a Confiabilidade Operacional..... 244
4.2.5.2.5- Analisar Processos de Avaliação e Modelos..... 244
4.2.5.2.6- Certificar Alterações..... 245
4.2.5.3- Descrição dos Depósitos de Dados..... 245
4.2.5.3.1- Alterações de Projeto..... 245
4.2.5.3.2- Modelos de Medição da Confiabilidade..... 245
4.2.5.4- Modelo de Objetos..... 245
4.2.5.4.1- Gerenciador de Falhas de Operação..... 246
4.2.5.4.2- Analisador de Falhas de Operação..... 247
4.3- Aspectos da Implementação..... 247

CAPÍTULO V - CONCLUSÃO..... 250

REFERÊNCIAS BIBLIOGRÁFICAS..... 253

ÍNDICE DAS FIGURAS

Pag.

I.1 - Relação entre custos de hardware, desenvolvimento e manutenção de software..... 5

I.2 - Ciclo de vida do software - custo do erro por fase..... 6

I.3 - Taxa de falhas instantânea do hardware e taxa de detecção de erros de projeto de software ao longo do tempo..... 13

I.4 - Curva real de falhas do software..... 15

II. 1 - Modelo para avaliação da confiabilidade..... 21

II. 2 - Diagrama "V" do ciclo de desenvolvimento de software..... 43

II. 3 - Relacionamento entre áreas de interesse, fatores ecritérios de qualidade..... 50

II. 4 - Método para avaliação da qualidade de software. 56

II. 5 - Objetivos, fatores e subfatores da confiabilidade de especificações..... 65

II. 6 - Ciclo de desenvolvimento de software com e sem projeto..... 80

II. 7 - Objetivos, fatores e subfatores da confiabilidade de projetos..... 82

II. 8 - Objetivos, fatores e subfatores da confiabilidade de programas..... 112

II. 9 - Estimativa do número total de linhas de código no final do projeto ao longo do tempo..... 140

II.10 - Estimativa do número acumulado de linhas de código corrigidas ao longo do tempo..... 140

II.11 - Número acumulado de erros removidos ao longo do tempo..... 141
II.12 - Taxa de detecção de erros ao longo do tempo.... 142
II.13 - Curva de crescimento da confiabilidade..... 143
II.14 - Exemplos de taxa de erros normalizada..... 162
II.15 - Curva e(τ)..... 163
II.16 - Re-estimativa dos parâmetros do programa..... 174
II.17 - Formulário para acompanhamento de falhas (1ª parte)..... 180
II.18 - Formulário para acompanhamento de falhas (2ª parte)..... 181
II.19 - Erros acumulados por período mensal ao longo do tempo..... 186
II.20 - Número de erros ao longo do tempo..... 186
II.21 - Percentual de erros ao longo do tempo..... 187
II.22 - Taxa de falhas ao longo do tempo..... 190

III. 1 - Diagrama de fluxo de dados do sistema de avaliação da confiabilidade de software (PRESTIME)..... 194
III. 2 - Diagrama de fluxo de dados do processo "Estabelecer meta de confiabilidade"..... 199
III. 3 - Diagrama de fluxo de dados do processo "Definir especificação de requisitos"..... 203
III. 4 - Diagrama de fluxo de dados do processo "Definir projeto"..... 204
III. 5 - Diagrama de fluxo de dados do processo "Definir programa"..... 205

III. 6 - Diagrama de fluxo de dados do sistema de estimação da confiabilidade..... 208
III. 7 - Relatório de falhas..... 210
III. 8 - Gráfico de intensidade de falhas..... 211
III. 9 - Diagrama de fluxo de dados do sistema de medição da confiabilidade..... 215
III.10 - Diagrama de fluxo de dados do processo "Elaborar laudo de confiabilidade"..... 218

IV. 1 - O Ambiente PRESTIME..... 224
IV. 2 - Menu da opção Arquivo..... 225
IV. 3 - Menu da opção Especificação..... 226
IV. 4 - Menu da opção Predição..... 227
IV. 5 - Menu da opção Estimação..... 228
IV. 6 - Menu da opção Medição..... 229
IV. 7 - Menu da opção Controles..... 230
IV. 8 - Gerenciador de Falhas de Operação..... 232
IV. 9 - Menu da opção Operações..... 232
IV.10 - Quadro de diálogo da opção Cadastramento..... 234
IV.11 - Quadro de diálogo da opção Configuração..... 234
IV.12 - Quadro de diálogo da opção Diagnóstico..... 235
IV.13 - Quadro de diálogo da opção Solução..... 236
IV.14 - Quadro de diálogo da opção Validação..... 237
IV.15 - Quadro de diálogo da opção Liberação..... 237
IV.16 - Menu da opção Erros..... 238
IV.17 - Quadro de diálogo da opção Origem..... 239
IV.18 - Analisador de Comportamento Operacional..... 240
IV.19 - Camada de assunto do Analisador de Comportamento Operacional..... 246

IV.20 - Camada de atributos do Analisador de Comportamento Operacional..... 246

IV.21 - Camada de assunto do Analisador de Falhas de Operação..... 247

IV.22 - Camada de atributos o Analisador de Falhas de Operação..... 247

IV.23 - Modelo de Desenvolvimento de Rotenberg..... 248

ÍNDICE DAS TABELAS

Pag.

II. 1 - Relação entre atividades da confiabilidade e seu valor..... 20

II. 2 - Fatores e subfatores de qualidade de maior importância em sistemas militares em tempo real..... 23

II. 3 - Fatores e subfatores de qualidade de maior importância em sistemas administrativos em tempo real..... 23

II. 4 - Fatores e subfatores de qualidade de maior importânciaem sistemas administrativos em "batch"..... 24

II. 5 - Interpretação dos níveis de confiabilidade..... 25

II. 6 - Formulário para avaliação de uma métrica especifica..... 59

II. 7 - Formulário para avaliação de um fator de qualidade..... 60

II. 8 - Formulário para totalização da confiabilidade de software..... 61

II. 9 - Modelos de categoria finita de falhas..... 149

II.10 - Modelos de categoria infinita de falhas..... 150

II.11 - Erros de software por categoria..... 184

II.12 - Percentual de erros de software por categoria.. 185

II.13 - Identificação dos tipos de erro de software.... 188

II.14 - Severidade dos erros de software..... 189

CAPÍTULO I

INTRODUÇÃO

Na década de 50 e início dos anos 60, as arquiteturas dos sistemas de processamento de dados se caracterizavam pela predominância dos custos de hardware sobre os custos de software. Naquela época, o estudo dos aspectos relacionados com a minimização dos custos de hardware era a preocupação dominante.

A medida em que a utilização e a potência dos computadores se ampliavam, observava-se uma grande demanda de software a fim de atender aos usuários que a cada dia vislumbravam novas aplicações. Esta rápida demanda associada a uma inicial falta de sistematização, a um aumento no volume e complexidade dos programas e à insuficiência de pessoal especializado para o desenvolvimento e manutenção, gerou uma situação que nos dias atuais é representada pela predominância dos custos de software.

A aplicação prática do conhecimento científico para o projeto e construção de programas junto à necessária documentação para desenvolvê-los, operá-los e mantê-los, se constitui atualmente numa ciência denominada Engenharia de Software, cuja finalidade é a de proporcionar um enfoque técnico e sistemático à sua produção e manutenção.

A fim de diminuir os custos e aumentar a qualidade do software, uma quantidade muito grande de técnicas e

ferramentas vêm sendo propostas na literatura. Muitas delas se apresentam como solução final para os problemas da Engenharia de Software. Todavia, tais propostas não mostram, na prática, evidências concretas de solução de tais problemas.

A avaliação de aspectos, tais como, metodologias de projeto, ferramentas de desenvolvimento, metodologias de teste e inspeção, são importantes no sentido de identificar o ambiente de desenvolvimento mais adequado às peculiaridades e características do tipo de programa que a organização desenvolve. É necessário, portanto, que medidas quantitativas avaliem a qualidade do software a partir das diferentes tecnologias empregadas.

Não existem, atualmente, definições amplamente aceitas ou medidas que descrevam e avaliem a qualidade de programas durante o seu ciclo de vida. Procedimentos para a avaliação dos dados obtidos variam de projeto para projeto, tornando difícil fazer uma comparação entre eles ou utilizar os benefícios das lições aprendidas com projetos passados.

Dobbins [1] advoga que a aplicação de um bem planejado conjunto de medidas, organizado de tal forma a atender as características e peculiaridades do tipo de produto que está sendo desenvolvido, pode fornecer um quadro consistente do seu estado real em qualquer ponto do processo de desenvolvimento. A visão deste quadro é de fundamental importância para que os fatores que influenciam direta ou indiretamente a qualidade

possam ser controlados e avaliados com base em níveis aceitáveis pré-estabelecidos.

Funcionalidade, desempenho, confiabilidade e custos são, provavelmente, as características mais importantes a serem avaliadas em um produto. O cálculo da confiabilidade durante o seu ciclo de vida é essencial, pois permite determinar os outros fatores, ou seja, o custo total de desenvolvimento e operação e se as necessidades e as expectativas do usuário estão sendo especificadas e atendidas corretamente.

1.1- Ciclo de vida do software

É habitual considerar o ciclo de vida do software como consistindo de um conjunto de etapas que ocorrem numa dada ordem durante o seu processo de desenvolvimento e de operação.

Para disciplinarmos o processo de desenvolvimento, devemos seguir uma série de etapas, cada uma das quais utilizando os resultados da anterior, detalhando e formalizando cada vez mais os resultados através de refinamentos sucessivos.

A necessidade de que o processo de desenvolvimento se dê de forma segura e confiável implica em que os produtos intermediários resultantes de cada etapa tenham a sua qualidade avaliada.

Embora na prática as etapas tendam a se sobrepor, é importante definir cada uma delas de modo a permitir especificar

precisamente as ferramentas de avaliação aplicáveis a cada etapa. A seguir, temos as fases propostas por Rocha [14] para o ciclo da vida tradicional do software.

1. DEFINIÇÃO
2. PROJETO
3. CONSTRUÇÃO
4. AVALIAÇÃO
5. OPERAÇÃO

A fase de definição visa estabelecer os objetivos, requisitos, hipóteses e restrições do sistema, assim como o planejamento inicial do desenvolvimento.

A fase de projeto tem como objetivo fornecer uma solução viável para o sistema definido da fase anterior.

A fase de construção é o período do ciclo de vida durante o qual o software é construído em conformidade com os documentos produzidos nas fases anteriores.

A fase de avaliação tem o objetivo de verificar e validar o software produzido, confrontando-o com as especificações e, além disso, examinar se está em conformidade com as normas e padrões estabelecidos.

A fase de operação é o período de tempo no ciclo de vida do software, durante o qual o produto é utilizado no seu ambiente

operacional e, quando necessário, corrigido, aprimorado, adaptado ou expandido [14].

1.2 - Qualidade de software

Devido ao extraordinário aumento na complexidade dos programas, conjugado a um lento aparecimento de novas metodologias e evolução das já existentes, é compreensível que cada vez mais os custos dos sistemas se concentrem nas atividades de manutenção de software (Figura I.1).

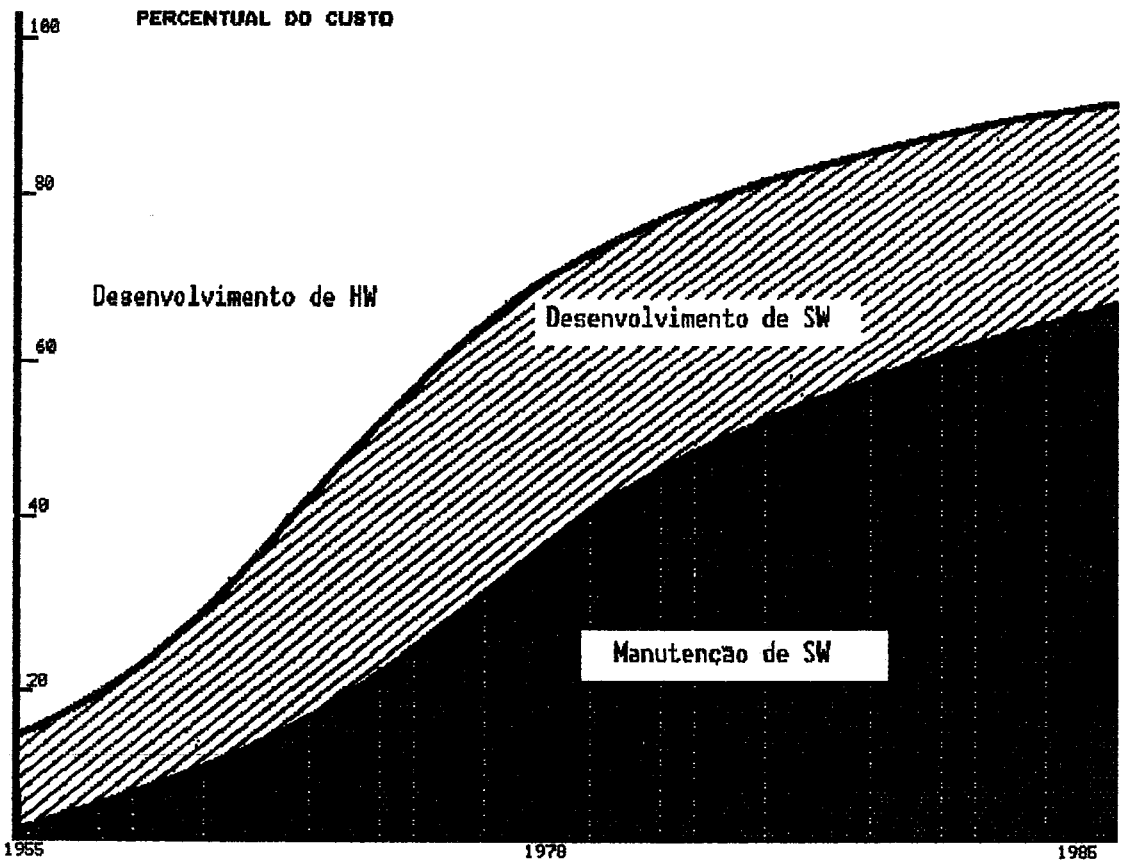


Fig. I.1

Relação entre custos de hardware, desenvolvimento e manutenção de software.

Fonte: [16]

É muito difícil garantir que durante o ciclo de vida não ocorram problemas. Mesmo com a utilização de métodos que auxiliem o processo de desenvolvimento, problemas de maior ou menor seriedade acabam aparecendo, às vezes comprometendo drasticamente seu custo, principalmente nas etapas finais, quando o custo de correção do erro é mais alto. A Figura I.2 mostra este comportamento.

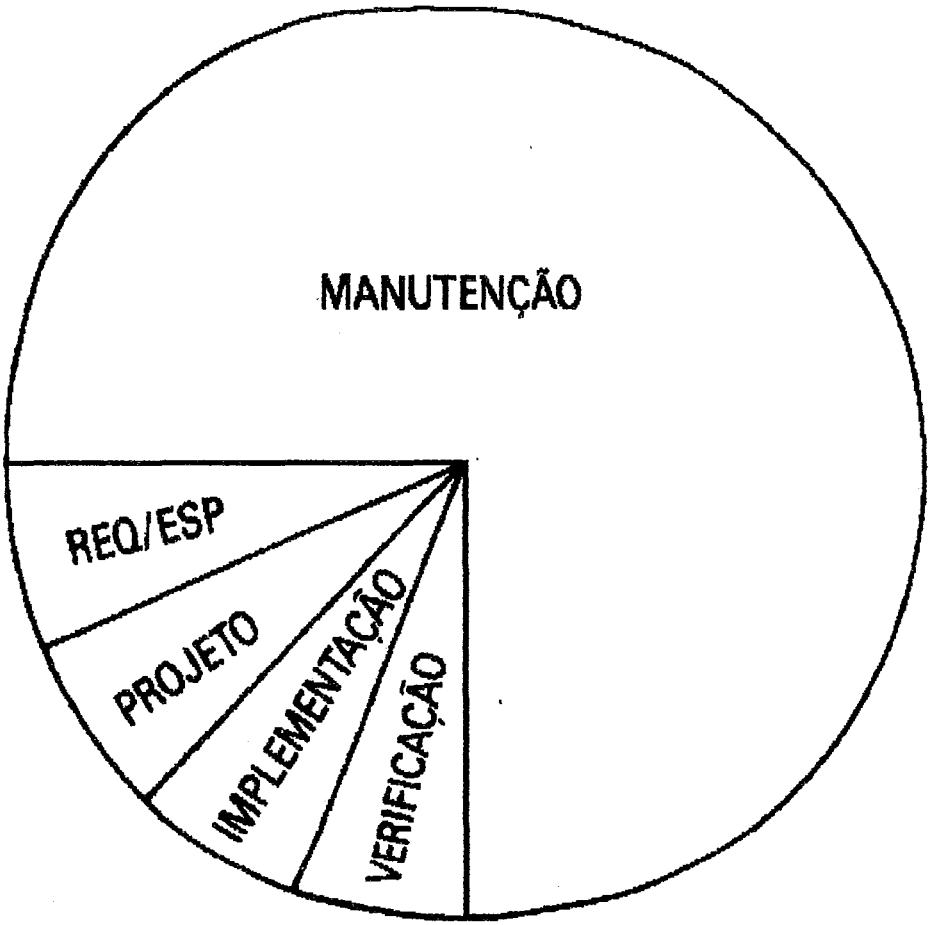


Fig. I.2

Ciclo de vida do software - custo do erro por fase

Fonte: [30]

Muitas vezes estes erros são originados nas etapas iniciais de desenvolvimento e só serão detectados quando o produto já estiver na fase de operação.

Segundo Cavano [3] uma série de questões permanecem ainda não suficientemente esclarecidas quando se procura implantar um trabalho de garantia da qualidade de software:

Que técnicas de desenvolvimento devem ser necessárias a fim de melhorar a qualidade do projeto?

De que forma o gerente deve proceder no estabelecimento de níveis de qualidade e de requisitos desejáveis que garantam uma alta confiabilidade do software?

Que relações devem ser consideradas entre confiabilidade, custos, cronograma e desempenho?

Como pode ser predita e estimada a confiabilidade do software entre as diversas etapas do ciclo de desenvolvimento (durante a revisão da especificação do software, revisão do projeto, revisão do código e dos testes)?

Qual a situação atual, em termos da aplicação prática, da confiabilidade de software?

Que tipos de ações corretivas devem ser tomadas, como resultado dos dados obtidos, a fim de aumentar a confiabilidade?

Durante quanto tempo, e quais testes devem ser realizados a fim de alcançar-se os níveis de confiabilidade especificados?

Como pode o usuário final ter garantias que a meta de confiabilidade especificada inicialmente foi mantida durante todo o desenvolvimento?

Até que estas e outras perguntas tenham sido respondidas, será muito difícil desenvolver programas altamente confiáveis, necessários principalmente em aplicações críticas.

Para que tais programas possam ser desenvolvidos, determinados procedimentos poderão ser adotados, tais como: melhorar a utilização das práticas de Engenharia de Software, o qual precisa ser melhor inspecionado e não somente melhor construído; descobrir as causas de problemas, a fim de que se possa prevení-los e não tratar seus efeitos; identificar que condições contribuem para uma alta ou baixa qualidade; avaliar mais cedo o software no seu processo de desenvolvimento, tomando ações corretivas quando necessárias.

A afirmativa de Tom de Marco "não podemos controlar o que não podemos medir" [86] nos mostra a necessidade da existência de medidas quantitativas que possam avaliar os atributos de

qualidade de um programa, de forma a oferecer subsídios que auxiliem o gerenciamento do projeto, tornando possível, desta forma, verificar se o mesmo está progredindo ou não, durante o seu ciclo de vida, de acordo com o que foi pré-estabelecido.

Rocha [14] define qualidade de software como "o conjunto de propriedades a serem satisfeitas em determinado grau, de modo que o software satisfaça às necessidades de seus usuários". Portanto, avaliar se estas necessidades estão sendo ou não atendidas durante o processo de desenvolvimento é fundamental para a tomada de decisões por parte da gerência que deve determinar se o produto prosseguirá para a próxima fase ou permanecerá na fase atual.

Juran [23] define confiabilidade como "a probabilidade de um produto realizar, sem falhas, suas especificadas funções sob determinadas condições e por um especificado período de tempo" (as diferenças entre a confiabilidade de software e de hardware são descritas no item I.3).

Segundo Hecht [2], para que a confiabilidade do software seja alcançada devem ser realizadas três atividades: Predição, Estimação e Medição. A "Predição" da confiabilidade baseia-se na avaliação das características do produto (com base na aplicação de métricas) antes do início da fase de teste, isto é, durante as etapas de especificação, projeto e codificação. A "Estimação" da confiabilidade se baseia no estudo das falhas ocorridas no ambiente de teste, podendo ser empregada para estimar a presente e a futura confiabilidade. A "Medição" da

confiabilidade baseia-se no estudo das falhas ocorridas durante a execução do programa no seu ambiente real de operação (estas três atividades serão vistas com mais detalhes no capítulo II).

Portanto, quanto mais alto o valor alcançado para a confiabilidade durante a atividade de predição (figura-de-mérito), maior a certeza de que as necessidades dos usuários foram correta e completamente especificadas e que as propriedades que software deve possuir estarão nos níveis pré-determinados, como também, quanto mais alta a probabilidade atingida durante a atividade de estimação, maior a certeza de que ele irá operar com o nível de intensidade de falhas especificado para a sua fase operacional. Sob esta ótica, o cálculo da confiabilidade é o mecanismo que possibilita a determinação do nível de qualidade atingido pelo produto em qualquer fase do ciclo de vida.

Assumindo que a confiabilidade de software pode ser avaliada em todas as fases do ciclo de vida, devemos determinar a que propósito esta medida se destina e que benefícios podemos esperar desta avaliação.

A confiabilidade de software pode ser de grande utilidade caso se deseje avaliar as tecnologias empregadas no processo de desenvolvimento ou mesmo a substituição da atualmente utilizada por outra, que pode ser realizada comparando-se os resultados obtidos através da aplicação de um conjunto de medidas pré-estabelecidas. Este procedimento é uma forma

eficaz de determinar o método que melhor atenda às características desejadas para o desenvolvimento.

Outro benefício importante e fruto da aplicação das técnicas de confiabilidade se refere à identificação de anomalias no processo de desenvolvimento. Muitas vezes, o planejamento especificado não é cumprido, já que quase sempre ocorrem atrasos, principalmente devido a estudos de viabilidade irreais ou gerenciamento de projeto ineficiente, com conseqüentes efeitos negativos para o seu sucesso. Portanto, devem existir mecanismos que detectem estes e outros problemas, já que tais informações são fundamentais, não só para se medir o progresso do desenvolvimento, como também para identificar áreas problemáticas ou deficiência de recursos.

1.3- Confiabilidade de software X confiabilidade de hardware

Muitos trabalhos sobre confiabilidade de software têm enfocado apenas o seu aspecto estimativo [34], intimamente relacionado com a fase de testes do programa. Entretanto, sua utilização pode ser mais abrangente, não só durante a fase de testes, como também para as primeiras fases do processo de desenvolvimento e durante a fase operacional do produto.

Uma visão mais abrangente da aplicação da confiabilidade de software pode fazer com que muitas pessoas tenham dificuldade em assimilá-la, em parte devido ao conceito e práticas históricas relacionadas com a medida da confiabilidade de hardware, que segundo Schneidewind [40] se baseia na

ocorrência de falhas segundo um modelo estatístico, enquanto que a confiabilidade de software está baseada na detecção de erros (processo também modelado estatisticamente).

Para melhor entender a confiabilidade de software é vantajoso compará-la com a confiabilidade de hardware. Um componente de hardware pode ficar inoperante por causa de três tipos de problemas: erro de projeto, erro de fabricação ou falha. Um erro de projeto é um erro que está inicialmente presente em qualquer cópia do produto. Este erro pode acontecer se, por exemplo, uma parte da memória não é endereçada pela CPU, significando erro no projeto lógico do circuito de endereçamento da memória. Um erro de fabricação é um erro que está presente em uma ou mais cópias do produto devido a falhas de produção daquelas cópias especificas. Por exemplo, um erro de fabricação ocorre quando existe uma solda mal feita de um componente ou que conecte duas ou mais trilhas que não devem estar conectadas. Falhas são problemas que não estão inicialmente presentes no produto, mas acontecem durante sua geração devido a um fenômeno físico, tal como a deterioração causada pela umidade, calor, fricção, radiação, ou congêneres. Um exemplo de falha é o desgaste físico de um relé devido à perda de magnetismo no núcleo magnético causado por excessivo calor. A Figura I.3 mostra as curvas de taxa de falhas instantânea do hardware e de taxa de detecção de erros de projeto de software, ao longo do tempo (cronológico para o hardware e de execução para o software).

Este gráfico (no caso do hardware) é comumente conhecido como "curva da banheira" ("bathtube"). Sua aplicação tem se mostrado válida para a maioria dos sistemas complexos de longa vida. Distinguem-se três regiões básicas:

Período de falhas prematuras

Intervalo inicial de duração relativamente curta durante o qual falha uma pequena porcentagem da população devido a defeitos de projeto, de material, erros de montagem e danos devido ao transporte.

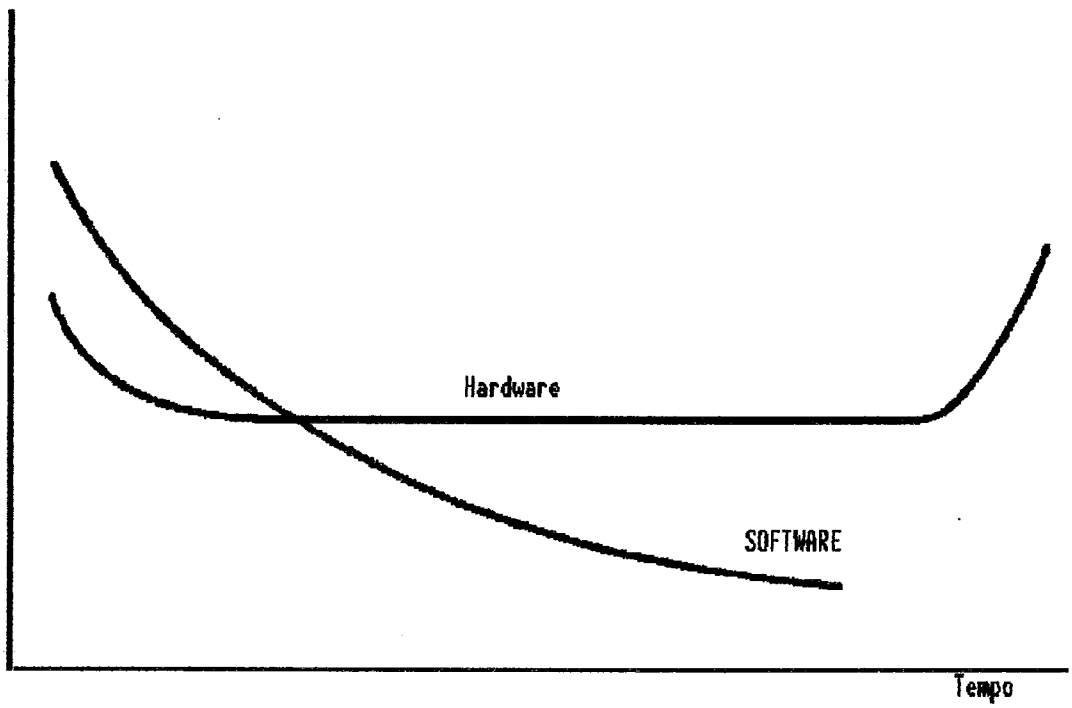


Fig. I.3

Taxa de falhas instantâneas do hardware e taxa de detecção de erros de projeto de software ao longo do tempo

Fonte [43]

Período de taxa de falhas constante

Neste período as falhas ocorrem aleatoriamente, decorrentes de solicitações normais de uso, debilidades inerentes ao projeto, diferentes combinações de condições de uso e, também, devido a acidentes causados por uso incorreto e manutenção inadequada.

Período de falhas por deterioração

Intervalo de tempo durante o qual a taxa de falhas cresce rapidamente em comparação com a taxa de falhas do período anterior, devido a processo de envelhecimento natural.

Com relação à confiabilidade de software, as diferenças são consideráveis. A priori, o software não deteriora. Além do mais, erros de fabricação, tais como um erro durante a cópia de um programa para um disquete de distribuição, não são considerados, já que são relativamente raros e facilmente detectados. Falhas de software são então, fundamentalmente, devido a erros de projeto, isto é, erros que foram inicialmente introduzidos durante o processo de produção.

A Figura I.3 mostra a mudança da taxa de detecção de erros do software ao longo do tempo se admitirmos que todos os erros que aparecem são corrigidos (significa que eles nunca aparecerão de novo) e que estas correções nunca introduzirão novos erros.

Na realidade, ocorre que o software sofrerá alterações durante a fase de manutenção (correções e/ou melhorias), quando provavelmente novos defeitos serão introduzidos, ocasionando o aparecimento do fenômeno mostrado na Figura I.4. Antes que a curva da taxa de detecção possa retornar ao estado estacionário original, outro conjunto de alterações é realizado causando bruscos incrementos nesta taxa. Lentamente, o nível mínimo da curva começa a aumentar, mostrando a deterioração do software causada pelas diversas alterações efetuadas [38].

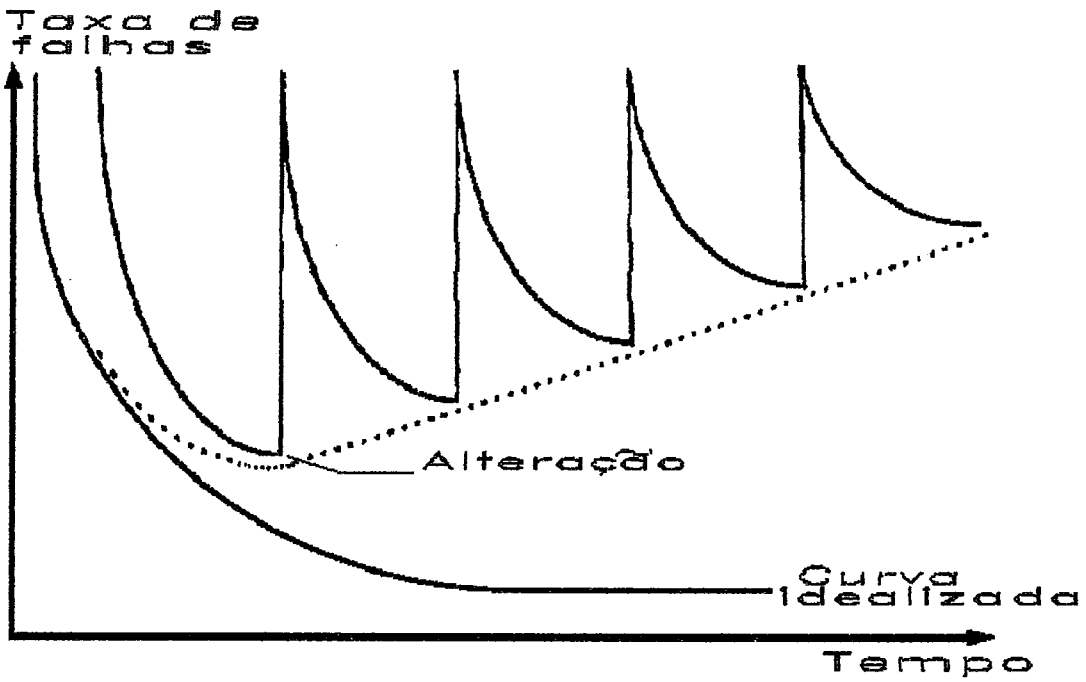


Fig. I.4

Curva real de falhas do software

Fonte: [38]

Como nos mostra a Figura I.3, o comportamento da confiabilidade de software e de hardware são inerentemente diferentes. Enquanto que no software os erros são pré-determinados (ainda que não conhecidos), no hardware as falhas são aleatórias. Entretanto, matematicamente, a confiabilidade de software e de hardware podem ser manipuladas da mesma forma [35].

Falhas de hardware são fundamentalmente dependentes do tempo. Todos os dispositivos físicos tem uma determinada vida útil, depois da qual a taxa de falhas cresce rapidamente. A detecção de erros de software, embora não aparentem ser função do tempo e sim apenas função das entradas e do estado atual do sistema, podem ser modeladas estatisticamente, a partir da fase de testes, como função do tempo de execução do programa.

Durante a maior parte da vida de um componente de hardware, as falhas são aleatórias e seguem uma distribuição de Poisson [72]. A detecção das erros de software, a partir da fase de testes, segue uma distribuição de Poisson não homogênea [44].

1.4- Objetivo e organização deste trabalho

O objetivo deste trabalho é propor um método e uma ferramenta automatizada para a avaliação da qualidade de software através de um acompanhamento sistemático do seu comportamento ao longo de todo o ciclo de vida, desde as fases iniciais de concepção até sua operação final.

O capítulo I contém a introdução geral e uma discussão sobre os aspectos básicos da confiabilidade de programas no contexto da Engenharia de Software e suas diferenças com relação a confiabilidade de hardware. É também apresentado um modelo de ciclo de vida para o software e abordada a importância da avaliação da qualidade de modo a auxiliar o gerenciamento de projetos.

No capítulo II é proposto um método para a avaliação da confiabilidade de software ao longo do processo de desenvolvimento, descrevendo-se as etapas de sua especificação, predição, estimação e medição. É realizado um estudo sobre os esforços de padronização das métricas de avaliação, tendo em vista a utilização das mesmas na atividade de predição da confiabilidade. É proposto um método para a avaliação da qualidade de software baseado na aplicação de medidas preditivas, cuja finalidade é fornecer indicações quantitativas da qualidade alcançada nas etapas de especificação, projeto e codificação. Um conjunto de fatores, subfatores e critérios é apresentado visando determinar o grau em que os objetivos de qualidade são satisfeitos, ou seja, se as propriedades gerais que o produto deve possuir estão presentes. A quantificação deste grau é obtida através da predição da confiabilidade do software, que irá avaliar se os objetivos de qualidade foram satisfeitos aos níveis conceitual, da representação e da utilização. Também é apresentado um conjunto de modelos matemáticos para a estimativa da confiabilidade de software, cuja finalidade é avaliar a qualidade durante a fase de testes do programa. Tais

modelos são apresentados por categoria. Finalmente, é abordada a questão da medição da confiabilidade durante a fase operacional do produto, como forma de corrigir e aperfeiçoar o método utilizado, principalmente em relação as atividades de predição e estimação. Um sistema de gerenciamento de falhas é proposto, a fim de oferecer informações básicas a respeito da descoberta e correção de erros, classificando-os quanto a seus tipos e distribuição.

No capítulo III é proposto um ambiente automatizado para a predição da confiabilidade nas etapas de especificação, projeto e codificação, estimação na fase de testes e medição na fase operacional. É, ainda, descrito o projeto e realizada a implementação de um protótipo de uma das ferramentas deste ambiente automatizado.

O capítulo IV apresenta as conclusões deste trabalho e as perspectivas futuras.

CAPÍTULO II

MÉTODO PARA AVALIAÇÃO DA QUALIDADE COM BASE NA PREDIÇÃO, ESTIMAÇÃO E MEDIÇÃO DA CONFIABILIDADE DE SOFTWARE

2.1- Introdução

Cavano [3] propõe um método para avaliação da confiabilidade durante o ciclo de vida do software com base na realização, além das três atividades mencionadas no item 1.2 (predição, estimação e medição), de uma quarta atividade: a especificação da meta de confiabilidade, atividade que antecede às outras três. As quatro atividades que compõem o método são, portanto:

1- Especificação da meta de confiabilidade e os processos necessários a fim de alcançar tal meta.

2- Predição e acompanhamento do progresso durante o desenvolvimento (tomando ações corretivas, quando necessário, para garantir a meta especificada).

3- Estimação da confiabilidade com base na aplicação de testes.

4- Medição da confiabilidade alcançada durante a fase operacional, através de procedimentos de medida no ambiente de operação.

A Tabela II.1 define as quatro atividades e a Figura II.1 apresenta o modelo para a avaliação da confiabilidade.

Meta especificada de confiabilidade	valor quantitativo da meta de confiabilidade.
Confiabilidade predita	valor quantitativo da futura confiabilidade como função da aplicação das métricas.
Confiabilidade estimada	valor quantitativo derivado dos dados obtidos a partir da aplicação dos testes.
Confiabilidade medida	valor quantitativo derivado dos dados coletados no ambiente operacional.

Tab. II.1

Relação entre atividades da confiabilidade e seu valor

Fonte: [3]

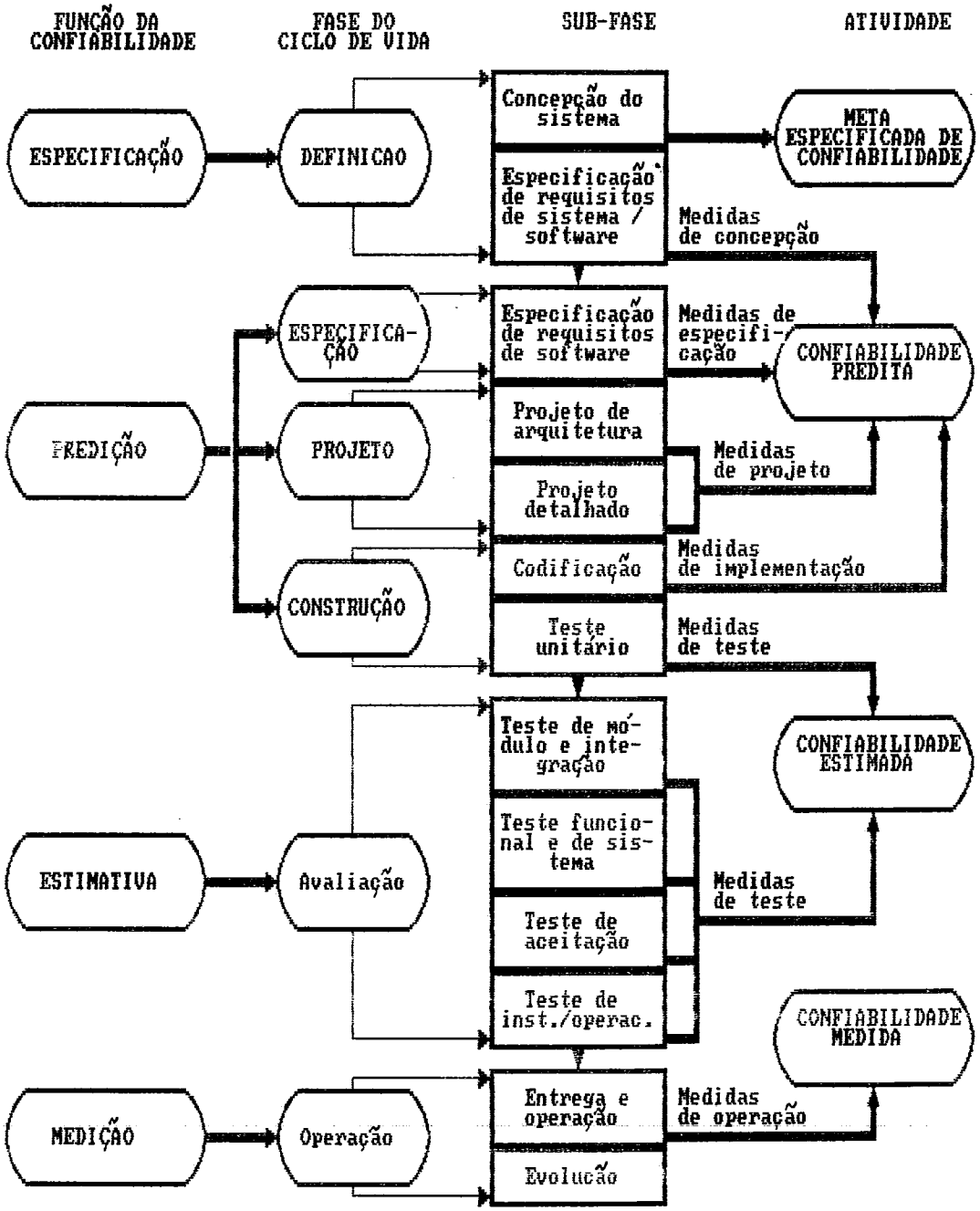


Fig. II.1

Modelo para avaliação da confiabilidade

Fonte: [3]

2.2- Especificação da confiabilidade

Determinar a meta de confiabilidade durante a fase de definição, envolve estabelecer os atributos (objetivos, fatores, subfatores e critérios de qualidade) que a aplicação pretende alcançar, assim como o peso relativo de cada um deles, pois dependendo do tipo de aplicação que se está desenvolvendo, a importância relativa entre os diferentes atributos pode variar. Ambos, usuário e desenvolvedor, devem interagir e se envolver na especificação da meta, que deve ser expressa em termos mensuráveis, com base, por exemplo, no que foi alcançado em projetos anteriores. Também deve ser especificado o ambiente (métodos e ferramentas) de desenvolvimento mais apropriado para atender a meta estabelecida.

Uma pesquisa realizada por Ferreira [12] em sessenta sistemas de natureza administrativa e militar permitiu que fossem levantados os pesos (importância relativa) correspondentes aos atributos de qualidade que mais influenciam as diversas aplicações (valores entre 0 e 10). A descrição dos atributos de qualidade apresentados nas tabelas a seguir podem ser encontrados em [12]. As tabelas a seguir mostram o resultado da pesquisa.

FATORES OU SUBFATORES	PESOS
Estilo	7,79 - 8,55
Modularidade	8,61 - 9,43
Fidelidade	8,20 - 9,34
Precisão	8,08 - 9,16
Segurança	7,71 - 8,41
Manutenibilidade	9,15 -10,87
Operacionalidade	8,78 - 9,87
Eficiência	8,24 - 9,33

Tab. II.2

Fatores e subfatores de qualidade de maior importância em sistemas militares em tempo real

Fonte: [12]

FATORES OU SUBFATORES	PESOS
Estilo	7,94 - 9,06
Modularidade	8,23 - 9,79
Fidelidade	8,02 - 8,72
Completude	7,82 - 8,42
Robustez	8,63 -10,37
Manutenibilidade	8,46 -10,19
Operacionalidade	7,96 - 9,95
Eficiência	8,84 -10,21
Rentabilidade	7,94 - 9,02

Tab. II.3

Fatores e subfatores de qualidade de maior importância em sistemas administrativos em tempo real

Fonte: [12]

FATORES E SUBFATORES	PESOS
Simplicidade	7,72- 8,48
Estilo	7,82- 8,82
Modularidade	7,89- 8,75
Completude	8,15- 9,13
Manutenibilidade	10,45- 12,52
Rentabilidade	7,95- 9,15

Tab. II.4

Fatores e subfatores de qualidade de maior importância em sistemas administrativos em "batch"

Fonte: [12]

Os valores apresentados levam em conta que o somatório dos pesos é igual a cem. Em alguns casos são mostrados valores acima de dez (faixa superior de pesos). Embora o valor médio seja menor ou igual a dez, por questões estatisticas (intervalo de confiança) este valor ultrapassa o limite máximo permitido para o valor do peso (dez). É importante ressaltar que, embora as faixas apresentem valores com duas casas decimais, isto não siginifica que as amostras obtidas também possuissem tal precisão. As faixas de valores de pesos definem apenas o intervalo representado pela média, mais ou menos o desvio-padrão da amostra.

Estão disponíveis na literatura algumas fontes onde podem ser encontrados critérios para a especificação da meta de confiabilidade como também técnicas de Engenharia de Software, que visam tornar possível o alcance de tal meta [4] [5] [6]. Os valores mostrados na Tabela II.5 permitem-nos situar a meta de confiabilidade dentro do nível de qualidade desejado. Estes

valores estão baseados em estudos desenvolvidos em 1983 pelo "Military Management Comand, Systems Management Division" [7].

VALOR DA META	INTERPRETAÇÃO
0.95 - 1.00	Alta probabilidade dos requisitos e objetivos serem atingidos.
0.90 - 0.94	Prosseguir com o cronograma, resolvendo em paralelo os problemas detectados.
0.60 - 0.89	Problemas existentes, historicamente resultam em um produto final pobre e com custos elevados. Ações imediatas são necessárias.
0.00 - 0.59	Problemas substanciais podem fazer com que o projeto seja significativamente alterado e/ou seus objetivos revistos.

Tab. II.5

Interpretação dos níveis de confiabilidade

Fonte:[7]

Como podemos notar na tabela acima, o valor 0.95 representa o nível a partir do qual considera-se alta a probabilidade do projeto atingir seus objetivos. Entretanto, para algumas áreas de aplicação, este valor pode ser ainda insuficiente, uma vez que necessitam de um alto grau de confiabilidade, como no caso de sistemas de controle de tráfego aéreo, monitoramento de pacientes em hospitais, controle de centrais nucleares, etc. Por outro lado, altos níveis de confiabilidade resultam, como

conseqüência, custos elevados e prazos mais longos. Para determinar o esforço necessário ao desenvolvimento de um programa com um determinado nível de confiabilidade, devemos levar em consideração a sua respectiva criticalidade, ou seja, as restrições que uma determinada aplicação possui com relação a ocorrência de falhas. A seguir é mostrada a relação entre o nível de confiabilidade, efeitos da falha e aplicações típicas.

Confiabilidade muito baixa- o efeito da falha gera uma leve inconveniência. Exemplos típicos são programas demonstrativos, versões protótipo de sistemas de pequeno porte e jogos.

Confiabilidade baixa- o dano causado pela falha é baixo e as perdas facilmente recuperadas. Exemplos típicos são programas de simulação e de apoio ao planejamento.

Confiabilidade normal- o efeito de uma falha gera uma perda moderada aos usuários. Exemplos típicos são os sistemas de controle gerencial e controle de estoques.

Confiabilidade alta- o efeito da falha pode provocar uma elevada perda financeira ou uma grande inconveniência para os usuários. Exemplos típicos são sistemas bancários e de distribuição de energia.

Confiabilidade muito alta- o efeito da falha coloca em risco a vida humana. Exemplos típicos são sistemas de defesa e controle de reatores nucleares.

2.3- Predição da confiabilidade

Estamos interessados em saber reconhecer uma boa especificação, um bom projeto e uma boa implementação. Como é muito difícil, nestas etapas, avaliar se a qualidade foi ou não atingida, torna-se necessário a existência de medidas quantitativas que avaliem estas etapas.

Uma vez que a meta foi estabelecida, o projeto do software deve ser avaliado em etapas pré-determinadas a fim de garantir que esta meta continue sendo mantida. Durante as etapas de especificação, projeto e codificação, a aplicação de um conjunto de métricas que caracterizem o processo empregado e seus produtos associados, podem ser usadas como indicadores para predizer a confiabilidade futura e para acompanhar o progresso do projeto.

Três atividades fundamentais são necessárias para que se determine o grau com que qualquer uma das etapas acima mencionadas cumpre as especificações ou requisitos pré-estabelecidos. Estas atividades são conhecidas como verificação, validação e aceitação. Verificação é um processo que determina o grau no qual um produto intermediário (um documemto), de uma determinada fase no processo de desenvolvimento cumpre com as especificações estabelecidas nas fases anteriores. Validação é o processo de avaliar o software e o conjunto de documentos associados ao final do processo de desenvolvimento para determinar se cumprem com os requisitos

[32]. Aceitação é processo de avaliar o produto final (software e manuais) sob a ótica do usuário. Boehm [27] comenta que verificação pode ser entendida como: "estamos construindo certo o produto?" e validação como: "estamos construindo o produto certo?"

2.3.1- Esforços de padronização de métricas

Segundo Dobbins [1], a literatura está repleta de métricas de todos os tipos, cada uma com seu próprio processo de avaliação. Antes que qualquer processo de seleção de métricas tome lugar, é necessário que exista um quadro de referência para que a discussão das métricas que serão utilizadas para avaliação do software possa ser feita.

O IEEE ("Institute of Eletrical and Eletronic Engineers"), através da "Computer Society" reconheceu esta necessidade. Em 1982, iniciou o projeto 982 [8] cuja finalidade é fornecer à comunidade um conjunto de medidas que possam ser usadas no gerenciamento da confiabilidade de software. Este projeto foi apresentado em 1985, e fornece um conjunto de aproximadamente 40 métricas.

Outro trabalho do IEEE na área de qualidade de software é o projeto de norma P.1061 [9] ("Standard for a Software Quality Metrics Methodology"), cujos objetivos são: estabelecer requisitos de qualidade, critérios de aceitação e padrões; calcular o nível de qualidade alcançada; desenvolver um programa de utilização das métricas com base em requisitos

pré-estabelecidos; detectar anomalias ou problemas potenciais no sistema; predizer o nível de qualidade que será alcançado no futuro; comparar os atributos de qualidade de um sistema com os de outros e monitorar a degradação da qualidade durante a fase de manutenção.

Duas normas foram preparadas pelo Departamento de Defesa norte-americano para operarem em conjunto: as normas DOD-STD-2168 [10] e DOD-STD-2167 [11]. Estas normas obrigam o desenvolvedor do software a analisar o projeto em desenvolvimento e selecionar um conjunto de métricas que permitam o cálculo da qualidade com base na sua confiabilidade, permitindo que o comprador e/ou usuário possa ter visibilidade sobre este processo.

O RADC ("Rome Air Development Center") no relatório RADC-TR-85-37 [4] produzido pela BAC ("Boeing Aerospace Company") propõe um conjunto de métricas para a avaliação da qualidade de software. Os objetivos deste relatório são: consolidar os resultados anteriormente obtidos na área de avaliação da qualidade, melhorar o sistema de garantia da qualidade e desenvolver uma metodologia capaz de determinar e especificar requisitos de qualidade para o software.

2.3.1.1- IEEE P.982 ("Measures to Produce Reliable Software") [8]

Este projeto de norma do IEEE fornece um conjunto de medidas indicativas da confiabilidade do software que pode ser aplicado ao produto e ao processo de desenvolvimento e de suporte. Este projeto foi motivado pela necessidade dos desenvolvedores e consumidores (que se defrontavam com uma grande variedade de modelos, técnicas e medidas disponíveis na literatura, mas que careciam de suficiente orientação para utilização efetiva) em compartilharem uma interpretação uniforme dos indicadores de confiabilidade.

Este projeto pretende chamar a atenção para a necessidade de medidas que possam ser aplicadas antecipadamente no processo de desenvolvimento e que podem ser os indicadores de confiabilidade para o produto a ser entregue.

Neste documento existem indicadores tanto de processo quanto de produto. A finalidade dos indicadores de processo é fornecer medidas que possam ser aplicáveis através do ciclo de vida de forma a oferecer um meio para uma contínua garantia e melhoria da confiabilidade. A finalidade dos indicadores de produto é aumentar a confiabilidade do software no seu ambiente real de utilização, isto é, durante as fases de operação e suporte.

Esta norma pretende ser de interesse para o projeto, desenvolvimento e avaliação (auditores, etc.) de software,

pessoal de manutenção, de confiabilidade e de qualidade, assim como os gerentes de operação e suporte.

Classificação funcional das medidas

As medidas podem ser divididas em duas categorias funcionais: medidas de produto e medidas de processo. Medidas de produto são aplicadas aos objetivos do software produzido e são divididas em seis subcategorias. Medidas de processo são aplicadas às atividades de desenvolvimento, teste e manutenção. Medidas de processo são divididas em três subcategorias.

a) **Medidas de produto**

As medidas de produto tratam da causa e efeito dos aspectos estáticos e dinâmicos tanto para a confiabilidade previamente especificada para a operação, como para a confiabilidade operacional. Estas medidas de produto cobrem, além da confiabilidade a nível conceitual e da representação, também o aspecto de utilizabilidade do software (confiabilidade da utilização). As seguintes seis medidas de produto (subcategorias) tratam destes enfoques da confiabilidade.

Erros, defeitos e falhas- contagem dos problemas observados devido as causas humanas e erros de programa, assim como mau funcionamento do sistema.

Tempo médio até falhar, taxa de falhas- medidas derivadas das ocorrências de falhas ao longo do tempo.

Crescimento da confiabilidade e projeção- determinação da taxa de diminuição de falhas do produto sob teste e em operação.

Falhas remanescentes do produto- determinação da taxa de diminuição de falhas do produto no desenvolvimento, teste e manutenção.

Completude e consistência- determinação da presença e concordância de todas as partes necessárias do software.

Complexidade- determinação dos fatores de complicação no sistema.

b) **Medidas de processo**

Três subcategorias para medidas de processo estão diretamente relacionadas com o gerenciamento do processo de desenvolvimento

Controle gerencial- garantia da orientação a ser dada ao processo de desenvolvimento e de manutenção.

Suporte- determinação da presença de todas as atividades necessárias para desenvolver e manter o produto de software.

Risco, Benefício e Cálculo de Custos- determinação do relacionamento entre custos, cronograma e desempenho do processo.

Um conjunto de métricas é então proposto, organizadas segundo os seguintes itens:

Aplicação- propósito e identificação da área onde a métrica pode ser aplicada.

Primitivas- informações relativas ao desenvolvimento ou utilização do software. São usadas para medir ou descrevê-lo quantitativamente. Primitivas são diretamente medidas ou contabilizadas. Podem ser valores constantes ou condições para medidas específicas.

Implementação- utilização prática da métrica e descrição detalhada da implementação da mesma.

A seguir, seguem as métricas propostas com suas respectivas aplicações. Maiores detalhes sobre as primitivas, assim como sua implementação poderão ser encontradas na própria norma [8].

Densidade de falha- tem a função de predizer o número de falhas remanescentes no programa, determinar se testes suficientes foram realizados com base em metas pré-determinadas de severidade.

Densidade de defeitos- pode ser usada após as atividades de inspeção de projeto e de código. Esta métrica é uma variação da anterior, entretanto focaliza especificamente o processo de inspeção.

Falhas acumuladas- método gráfico utilizado para predizer a confiabilidade, estimar o tempo adicional para teste e identificar módulos e subsistemas que necessitem de teste adicional.

Número de dias/falhas em reparo- representa o número decorridos desde o aparecimento de uma falha de software até sua remoção.

Garantia de teste funcional- utilizada para quantificar um índice de garantia para o teste de software. O usuário operacional tem mais familiaridade com os requisitos funcionais do sistema e reportará os problemas do sistema com base nestes requisitos do que com base nos requisitos de teste de módulo.

Gráfico de causa e efeito- tem a função de identificar requisitos que estão incompletos e ambíguos. Esta medida explora as entradas e as saídas esperadas de um programa a fim de identificar as ambigüidades.

Localização de requisitos- utilizada para identificar requisitos que estão perdidos, ou erradamente inseridos, nos requisitos originais.

Índice de defeitos- tem a função de fornecer um contínuo índice relativo de quão correto o software está progredindo através do ciclo de vida.

Distribuição de erros- a procura pela causa dos erros de software envolve uma análise dos dados coletados durante cada fase do desenvolvimento. Os resultados desta análise podem ser melhor retratados através de um gráfico de distribuição de erros de acordo com cada critério de distribuição de erros por fase, categoria e causa para não detecção do erro.

Índice de maturação do software- utilizada para quantificar a taxa de modificações do produto.

Homens-hora por defeito detectado- fornece uma medida quantitativa que pode ser usada para calcular a eficiência dos processos de inspeção de projeto e de código.

Número de requisitos conflitantes- utilizada para determinar a confiabilidade de um programa, considerando uma dada arquitetura, como por exemplo, a especificada no modelo de Entidades, Relacionamentos e Atributos (ERA).

Número de entradas e saídas por módulo- utilizada para determinar a complexidade da estrutura do programa representada pelos pontos de entrada e saída identificados em uma especificação modular ou linguagem de projeto.

Medidas da "Ciência de Software"- utilizada para determinar a complexidade de um programa com base na "Ciência de Software" de Halstead [88].

Complexidade "Graph-Theoretic" para a arquitetura- muitos defeitos podem ser introduzidos na fase operacional por modificações feitas no sistema que seja confiável mas difícil de ser entendido. Existem três tipos de medidas de complexidade de arquitetura: complexidade estática, estática generalizada e dinâmica.

Complexidade ciclomática- utilizada para determinar a complexidade estrutural do código do módulo.

Determinação mínima de casos de teste- esta métrica determina o número de distintos caminhos através de um módulo. Deste modo um número mínimo de casos de teste pode ser gerado.

Confiabilidade de execução- é a probabilidade de que k execuções selecionadas aleatoriamente produzam resultados corretos.

Estrutura do projeto- utilizada para determinar a simplicidade do projeto detalhado de um programa.

Tempo médio para remover a próxima k falha- utilizada para estimar o espaço médio de tempo até que a próxima k falha seja descoberta e removida.

Nível de pureza do software- fornece uma estimativa relativa à inexistência de falha de um programa em qualquer ponto específico de tempo durante a fase operacional.

Número estimado de falhas remanescentes por disseminação- o número de falhas remanescentes está relacionado com a confiabilidade do programa. Esta métrica utiliza a técnica da disseminação de falhas como forma de determinar o número de falhas remanescentes.

Conformidade dos requisitos- esta análise é utilizada para verificar a conformidade dos requisitos através de diagramas de verificação de sistemas (SVDs), que detectam inconsistências, incompletudes e mal-entendidos.

Garantia de teste- tem a função de determinar a eficácia do processo de teste sob a perspectiva tanto do desenvolvedor quanto do usuário.

Complexidade de dados ou fluxo de informações- é uma medida da complexidade estrutural ou procedural utilizada para avaliar a estrutura do fluxo de informações de sistemas de grande porte, a estrutura do fluxo de informações dos procedimentos e módulos e a complexidade das interconexões entre módulos.

Função de crescimento da confiabilidade- fornece uma estimativa ou predição do tempo e do estágio em que o teste se encontra quando uma desejada taxa de falhas ou de defeitos é alcançada.

Contagem de falhas residuais- fornece uma indicação da integridade do software com relação a existência de falhas remanescentes.

Análise de falhas usando o tempo total decorrido- utilizada para determinar o aumento da confiabilidade com base no número estimado de falhas remanescentes.

Suficiência de testes- esta métrica tem a função de avaliar a suficiência dos testes de integração software-software a partir da comparação entre falhas reais e preditas.

Tempo médio até falhar- esta métrica é usada como hipótese de teste para garantir um especificado requisito de MTTF ("Mean Time To Failure").

Taxa de falhas- pode ser usada como indicador do aumemto da confiabilidade como função do tempo gasto com os testes.

Disponibilidade de código fonte e documentação de software- o objetivo desta métrica é obter informações que permitam identificar, dentre os recursos que compõem o material necessário para a atividade de manutenção, aqueles que possam ser inadequados para uso no ambiente de manutenção.

Confiabilidade necessária de software (RELY)- tem a função de fornecer antecipadamente na fase de planejamento do projeto

uma medida que torne clara a relação entre o custo e o grau de confiabilidade.

Facilidade de versão do software- utilizada para quantificar o risco de propriedade do software, tanto do usuário quanto do suporte.

Completude- tem a função de determinar a completude da especificação de software. O valor determinado pelas primitivas associadas com a medida de completude, pode ser usado para identificar problemas na especificação.

Acurácia de teste- utilizada para determinar a acurácia do programa de teste em detectar falhas. Com a métrica garantia de teste, pode-se estimar a porcentagem de falhas remanescentes.

Confiabilidade de desempenho do sistema- esta métrica avalia a confiabilidade de desempenho do software e a probabilidade que o valor de cada requisito de desempenho seja menor ou igual a um pré-definido valor de limiar.

Confiabilidade de independência de processo- fornece um método direto para computar a confiabilidade de certos tipos de programas, tais como controle de tráfego telefônico e de transações comerciais em que não há "loop" nem dependência local nos programas. Esta métrica é melhor denominada de "confiabilidade do serviço que o software oferece".

Disponibilidade operacional combinada de software e hardware- esta métrica fornece uma indicação direta da disponibilidade do software, isto é, se ele está operando ou está operável quando o usuário dele precisar.

2.3.1.2- IEEE P.1061 ("Standard for Software Quality Metrics Methodology") [9]

Esta norma é dirigida àquelas pessoas envolvidas com a aquisição, desenvolvimento, suporte ou auditoria de software, que precisem medir ou avaliar a qualidade do mesmo, podendo ser utilizada para identificar e priorizar os requisitos de qualidade de um sistema, identificar as peculiaridades específicas que devem ser levadas em consideração a fim de alcançar os requisitos de qualidade pré-estabelecidos e avaliar se os mesmos, no decorrer do processo de desenvolvimento, estão sendo alcançados. A metodologia proposta pela norma para o cálculo da qualidade de software se baseia nos seguintes passos:

1- Estabelecimento dos requisitos de qualidade de software- seleção e priorização de uma lista de atributos de qualidade a serem usados como guia e controle do desenvolvimento, assim como um instrumento para avaliar, na fase de operação, se o sistema alcançou os requisitos estabelecidos em contrato.

2- Identificação das métricas de qualidade- seleção das métricas relevantes a partir de uma estrutura de métricas existente.

3- Implementação das métricas- desenvolvimento de ferramentas, obtenção de dados e aplicação de métricas para cada fase do ciclo de vida.

4- Análise dos resultados obtidos a partir da aplicação das métricas- os resultados são analisados e reportados a fim de auxiliar o controle da qualidade do desenvolvimento e avaliar o produto final.

5- Validação das métricas de qualidade- o resultado das métricas preditivas são comparados com o resultado das métricas diretas a fim de determinar se as métricas preditivas medem acuradamente os seus fatores associados.

2.3.1.3- DOD-STD-2167/68 - ("Defense System Software Development" e "Software Quality Evaluation Program") [10] [11]

Esta duas normas foram escritas e projetadas para operarem em conjunto. A primeira, 2167, contém os requisitos necessários para o desenvolvimento de software crítico e estabelece um processo uniforme para este desenvolvimento, aplicável a todo o ciclo de vida. O processo de desenvolvimento de software define atividades de desenvolvimento que resultam em:

1- Geração de diferentes tipos e níveis de software e documentação.

2- Aplicação no desenvolvimento de ferramentas, técnicas e métodos.

3- Controle e planejamento de projeto.

Esta norma incorpora práticas que têm se mostrado eficientes em termos de custos sob a perspectiva do ciclo de vida, com base em informações coletadas pelo Departamento de Defesa e pela indústria.

O desenvolvimento de software é, geralmente, um processo iterativo, onde uma iteração no seu ciclo de desenvolvimento ocorre uma ou mais vezes para cada uma das fases do ciclo de vida. Cada iteração é iniciada pela especificação de requisitos de sistema para o software ou uma subseqüente revisão destes requisitos. O relacionamento entre as fases do ciclo de vida com seus respectivos produtos finais, revisões e auditorias é mostrado na Fig. II.2, refletindo a seqüencialidade do ciclo de desenvolvimento do software, assim como os documentos que tipicamente existem antes de iniciar uma iteração.

Durante o desenvolvimento do software mais de uma iteração no ciclo de vida pode estar em progresso ao mesmo tempo. Cada iteração representa uma diferente versão do software. Este processo pode ser denominado enfoque de aquisição evolutiva ou de construção incremental. A Figura II.2, conhecida como diagrama "V", ilustra os mais importantes aspectos do ciclo de vida. O lado esquerdo do diagrama representa o lado dos

requisitos e está associado com o processo de definição. O lado direito do "V" representa o lado da validação e está associado com o processo de composição.

Fases	Análise dos requisitos	Projeto	Código	Integração e teste	aceitação
Níveis	Sistema Subsistema Software CSCI	Preliminar Detalhado	Teste unitário	Teste CSC Integração CSC Teste CSCI Integração CSCI	Sistema Subsistema

Revisão: SRR SDR SSR PDR CDR TRR FCA PCA

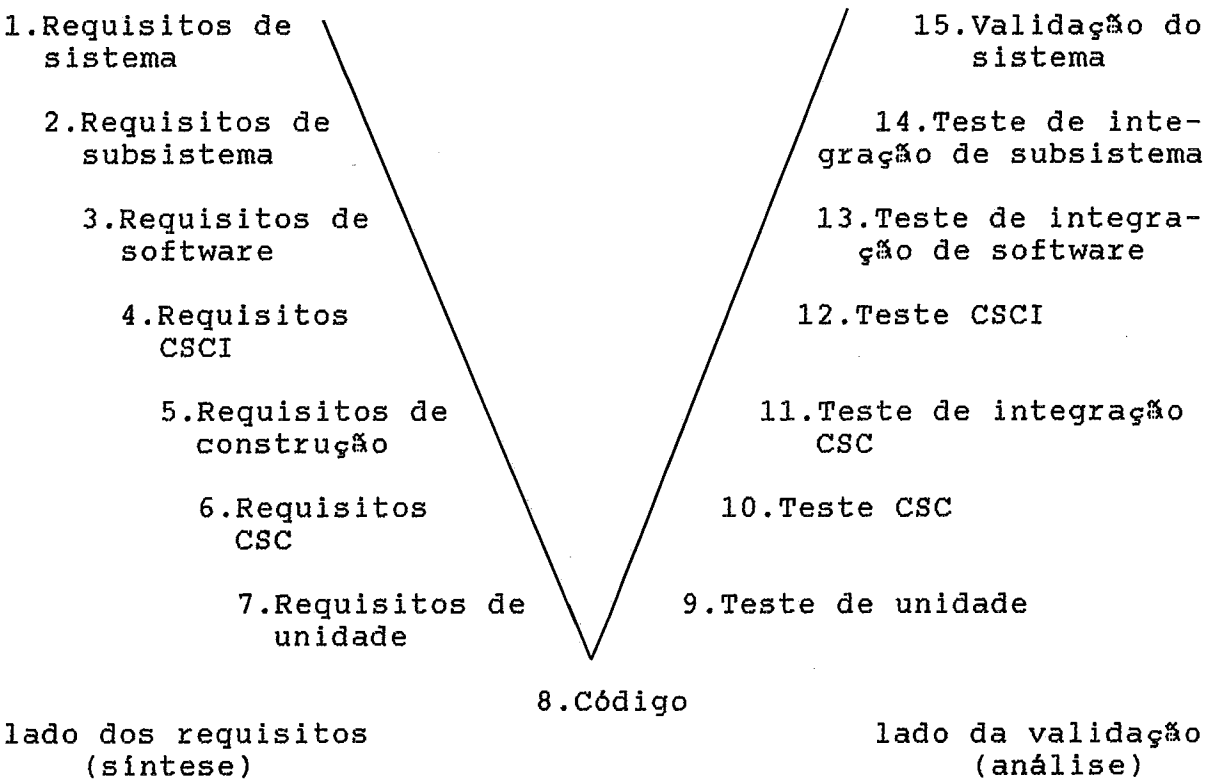


Fig. II.2

Diagrama "V" do ciclo de desenvolvimento de software

Fonte: [11]

A seguir, temos a definição dos termos usados no modelo mostrado anteriormente:

CDR- "Critical Design Review"- revisão do projeto crítico.

CSC- "Computer Software Component"- parte funcional ou lógica de um item de configuração CSCI. Pode ser de alto-nível ou de baixo-nível.

CSCI- "Computer Software Configuration on Item"- hardware ou software, ou uma agregação dos dois.

FCA- "Functional Configuration on Audit"- auditoria de configuração de funcionalidade.

PDR- "Preliminary Design Review"- revisão do projeto preliminar.

SDR- "Systems Design Review"- revisão do projeto dos sistemas.

SRR- "Systems Requirements Review"- revisão dos requisitos dos sistemas.

SSR- "Systems Specifications Review"- revisão das especificações dos sistemas.

TRR- "Test Readiness Review"- revisão da prontidão dos testes.

A mais recente revisão desta norma (2167A), apresentada em 1988, evita sugerir o modelo do ciclo de desenvovimento. O usuário da norma tem a possibilidade de selecionar seu próprio modelo - "V" ou então prototipagem rápida, construção múltipla, espiral ou outro híbrido dos anteriores.

A norma DOD-STD-2168 obriga o desenvolvedor a inspecionar o projeto em desenvolvimento e selecionar um conjunto de medidas que permitirá calcular a qualidade do software e conseqüentemente a sua confiabilidade [1], oferecendo assim ao consumidor final (usuário) a visibilidade sobre este processo. Duas partes desta norma são especialmente benéficas às entidades envolvidas com a qualidade de software. Primeiro, uma planilha do ciclo de desenvolvimento de software com os produtos intermediários contidos em cada fase, permitindo a identificação das linhas básicas e a configuração do processo de desenvolvimento. A outra contribuição importante se refere às explicações fornecidas para o cálculo dos critérios de avaliação. Dentre os critérios propostos pela norma, podemos citar: atendimento ao padrão de documentação, atendimento dos requisitos contratuais, consistência interna, compreensão, adequação técnica, grau apropriado de completude, rastreabilidade dos documentos indicados, consistência com os documentos indicados, implementabilidade, verificabilidade dos requisitos, utilização de técnicas apropriadas para especificação, projeto e codificação, nível adequado de detalhamento, alocação adequada de recursos (quantidade e tempo), adequada cobertura dos testes de requisitos, adequação

das ferramentas, facilidades, procedimentos e recursos propostos e conteúdo apropriado para a audiência esperada.

2.3.1.4- RADC-TR-85-37 ("Specification of Software Quality Attributes") [4]

Este trabalho foi preparado pela Boeing Aerospace Company (BAC) para o "Rome Air Development Center" (RADC) e teve como objetivos desenvolver uma metodologia capaz de dar ao comprador do software a oportunidade de determinar e especificar os seus requisitos da qualidade, assim como possibilitar ao desenvolvedor melhorar o seu sistema de garantia de qualidade. Este sistema consiste basicamente de fatores, atributos, critérios, métricas, além dos mecanismos, formulários, tabelas e planilhas que permitem a especificação dos requisitos de qualidade e o cálculo do nível de qualidade alcançado.

Uma vez estabelecidos os requisitos de qualidade é necessário monitorar, passo a passo, o desenvolvimento do software. Este monitoramento continua através do projeto de arquitetura, projeto detalhado, codificação e teste unitário, teste de integração e teste de sistema. A primeira preocupação, e mais importante que as questões de custos e cronograma, é avaliar se o software satisfaz seus requisitos ao longo do processo de desenvolvimento com base no progresso técnico e no nível de qualidade alcançado em cada fase. Isto é possível a partir de duas atividades fundamentais "V&V" (Verificação e Validação) e "GQ" (Garantia de Qualidade).

- Verificação e validação

O propósito da "V&V" é garantir que o software irá realizar suas funções de acordo com seus requisitos. No ciclo de desenvolvimento do software cada uma destas atividades possui conceitos distintos, a saber:

Verificação- É um processo iterativo, cujo objetivo é determinar se o produto de cada fase do desenvolvimento cumpre os requisitos estabelecidos na fase anterior. Este processo não considera se o software, a nível de sistema, cumpre seus requisitos ou se realmente eles satisfazem às necessidades do usuário.

Validação- Algumas vezes, a validação é considerada como uma atividade de teste de sistema. Na realidade, ela é muito mais do que isso. Validação, assim como verificação, é um processo iterativo ao longo do ciclo de vida. Por exemplo, quando um requisito é considerado, durante a fase de projeto, não implementável ou ambíguo, isto quer dizer que um requisito, a nível de sistema, pode ser vago ou incorreto. Este tipo de realimentação permite que ações corretivas possam ser tomadas logo no início do desenvolvimento, reduzindo riscos e custos.

- Garantia da qualidade

O propósito da atividade de Garantia da Qualidade é garantir que o software a ser comercializado atende os requisitos especificados. Colocada desta forma, esta atividade não

garante o desenvolvimento de produtos de alta qualidade, a menos que atributos de qualidade de software sejam especificados em termos mensuráveis. O objetivo dos programas de garantia da qualidade atuais é permitir a realimentação ao responsável pelo projeto, sobre vários aspectos do processo de desenvolvimento. "GQ" é similar a "V&V": a maior diferença está em que "V&V" fornece uma realimentação técnica do produto em intervalos específicos de tempo, enquanto "GQ" fornece uma realimentação sobre uma ampla faixa de atividades do desenvolvimento. Os programas de garantia de qualidade possuem um enfoque eminentemente mais administrativo do que técnico.

2.3.1.4.1- Métricas de qualidade

O propósito das métricas de qualidade é permitir que se especifique o nível desejado de qualidade para cada um dos fatores de qualidade, assim como medir quantitativamente os níveis de qualidade alcançados em pontos específicos do ciclo de vida.

O relatório RADC-TR-85-37 identifica cerca de 30 fatores de qualidade e 70 critérios, agrupados sob as seguintes áreas de interesse: Desempenho, Projeto e Adaptação. A Figura II.3 relaciona as áreas de interesse e respectivos fatores e critérios.

Critérios são aquelas características que definem os fatores de qualidade. Os fatores são divididos em critérios independentes, de tal forma que estes possam ser facilmente

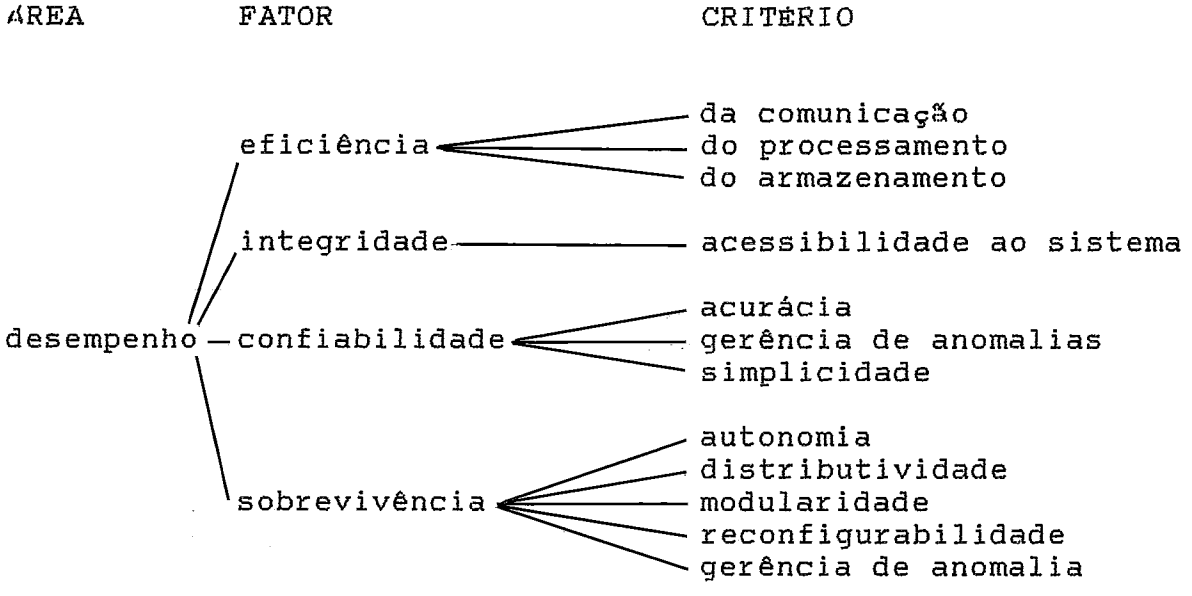
mensurados. A pontuação total alcançada pelos critérios que compõem um determinado fator irão estabelecer o grau em que este fator está presente no produto. Segundo Vincent [7], existe uma série de razões para desenvolver uma lista de critérios para cada fator:

1- Critérios oferecem uma definição mais completa e concreta dos fatores.

2- Critérios comuns entre fatores ajudam a ilustrar a interrelação entre os fatores.

3- Critérios permitem a auditoria e revisão das métricas (medidas numéricas da qualidade) com maior facilidade.

4- Critérios permitem-nos definir os fatores de qualidade que melhor se prestam a torna-se padrões de aceitação.



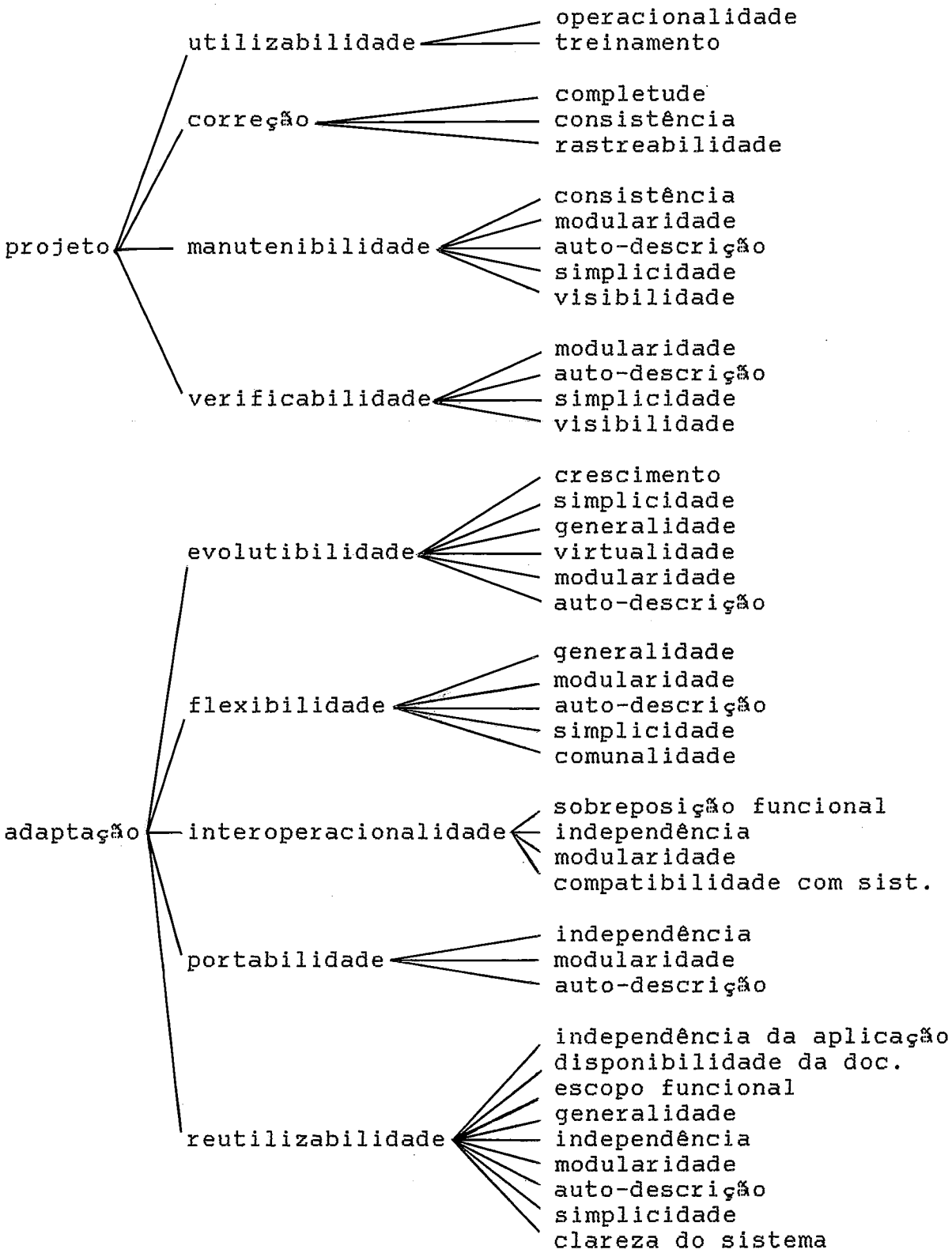


Fig. II.3

Relacionamento entre áreas de interesse, fatores

e critérios de qualidade

Fonte: [4]

Cada critério é composto por uma ou mais métricas, como mostrado a seguir:

acurácia- lista de verificação da acurácia.

gerenciamento de anomalias- controle/tolerância a erros, entrada de dados imprópria, falhas computacionais, falhas de hardware, erros em dispositivos, erros de comunicação, falhas em nós de comunicação.

independência da aplicação- independência de implementação do banco de dados, estrutura de dados, padronização da arquitetura, independência do microcódigo, independência funcional.

crescimento- expansibilidade do armazenamento de dados, extensibilidade computacional, extensibilidade de canais, extensibilidade do projeto.

autonomia- complexidade da interface, auto-suficiência.

comunalidade- comunalidade das comunicações, comunalidade de dados, vocabulário comum.

completude- lista de verificação da completude.

consistência- consistência de procedimentos, consistência de dados, distributividade, estrutura do projeto.

disponibilidade da documentação- acesso a documentação, estrutura da documentação.

eficiência da comunicação- medida da eficiência da comunicação.

eficiência do processamento- medida de eficiência do processamento, medida da eficiência da utilização de dados.

eficiência do armazenamento- medida da eficiência do armazenamento.

sobreposição funcional- lista de verificação de sobreposições funcionais.

escopo funcional- especificidade de funções, comunalidade de funções, utilizabilidade seletiva de funções.

generalidade- referência unitária, implementação unitária.

independência- independência software X sistema, independência de máquina.

modularidade- implementação modular, projeto modular.

operacionalidade- lista de verificação operacional, comunicabilidade de entradas do usuário, comunicabilidade de saídas para o usuário.

reconfigurabilidade- lista de verificação de reestruturação.

auto-descrição- quantidade de comentários, eficiência dos comentários, linguagem de descrição.

simplicidade- estrutura do projeto, estrutura da linguagem/pré-processador, complexidade dos dados e do fluxo de controle, simplicidade do código, especificidade, medida do nível de dificuldade ("Halstead" [88]).

acessibilidade ao sistema- controle de acesso, auditoria de acesso.

clareza do sistema- complexidade da interface, complexidade do fluxo do programa, complexidade da aplicação funcional, complexidade da comunicação, clareza da estrutura.

compatibilidade- compatibilidade de comunicações, compatibilidade de dados, compatibilidade de hardware com o sistema, compatibilidade de software, documentação para outros sistemas.

rastreabilidade- referência cruzada.

treinamento- lista de verificação de treinamento.

virtualidade- independência de dados do sistema.

visibilidade- teste unitário, teste de integração, teste de sistema.

Descrições detalhadas das métricas anteriormente mencionadas podem ser encontradas em [4]. Estas métricas são organizadas em forma de planilhas compostas por um conjunto de questões. Os produtos intermediários do desenvolvimento (especificações, documentos e listagens) são usados como fonte de informação para as respostas às questões. Estas respostas são traduzidas para uma tabela de pontuação a fim de quantificá-las (sim = 1, não = 0), e uma fórmula com o resultado na faixa <0-1>.

Sete planilhas diferentes são aplicadas em diferentes fases do desenvolvimento [4]. A primeira é aplicada sobre a análise de requisitos de sistema, pois para sistemas grandes o software pode não ser um componente discernível no projeto e não ter requisitos separados a nível de sistema. A segunda é aplicada sobre a análise de requisitos de software. A terceira sobre o projeto preliminar. A quarta e a quinta sobre o projeto detalhado. A sexta e a sétima planilhas são aplicadas sobre o código e o teste unitário.

As respostas, em forma de pontos, são então organizadas em treze planilhas de pontuação, cada uma delas relacionadas com o seu respectivo fator de qualidade. Estas planilhas têm o objetivo de contabilizar a pontuação resultante de cada métrica aplicada em um valor quantitativo que represente o nível de qualidade alcançado de um determinado fator.

Exemplos completos de um sistema de controle de tráfego aéreo e de um sistema comercial podem ser encontrado em [9].

2.3.2- Método para a predição da confiabilidade de software

Rocha [14] propôs um método que permite a predição da confiabilidade de especificações, projetos e programas, sendo tal método influenciado pelos trabalhos de Boehm [16] e McCall [17]. O método baseia-se nos seguintes conceitos:

Objetivos de qualidade- propriedades gerais que o produto deve possuir.

Fatores de qualidade- características através das quais se atingem os objetivos.

Critérios- atributos primitivos possíveis de serem avaliados.

Processo de avaliação- processo e instrumento para medir o grau de presença de um determinado atributo.

Medidas- indicam o grau de presença de um determinado critério.

Medidas agregadas- resultado da agregação das medidas obtidas ao avaliar segundo os critérios e quantificam os fatores.

A Figura II.4 apresenta a estrutura do método.

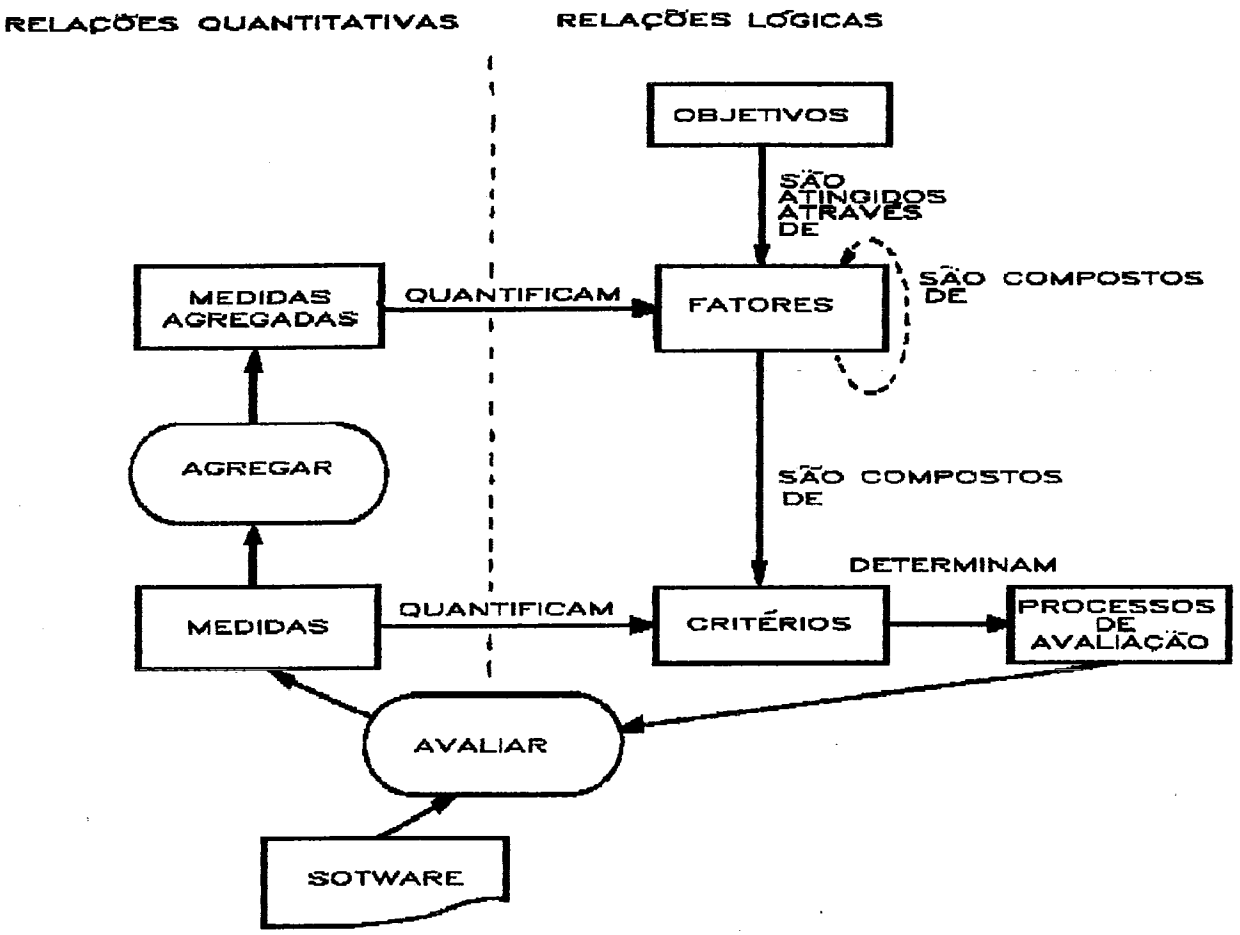


Fig. II.4

Método para Avaliação da Qualidade de Software

Fonte: [14]

Os objetivos de qualidade são atingidos através dos fatores que podem ser compostos de outros subfatores e são avaliados através de critérios. Os critérios definem atributos de qualidade para os fatores. Medidas são valores resultantes da avaliação de um produto segundo um critério especifico. Objetivos e fatores não são diretamente mensuráveis e só podem ser avaliados através de critérios. Um critério é um atributo primitivo, isto é, um atributo independente de todos os outros

atributos. Nenhum critério isolado é uma descrição completa de um determinado fator ou subfator. Da mesma forma, nenhum fator define completamente um objetivo. A fim de organizar a aplicação das medidas utilizaremos o sistema de pontuação proposto em [4]. Segundo as premissas apresentadas neste trabalho, pode-se realizar a avaliação através de duas formas básicas:

1- Utilizando uma faixa de pontuação normalizada para alguns elementos, por exemplo, o número de módulos que violam uma dada regra dividido pelo número total de módulos e subtraído de 1;

2- Uma simples métrica binária, 1 para sim e 0 para não.

Uma forma de melhorar este sistema é assinalar pesos diferenciados para cada um dos elementos, com base na importância de cada um deles. Este peso pode variar tipicamente na faixa de 1 para o elemento menos importante até 5 para o mais importante. As avaliações devem ser realizadas com base em padrões de qualidade pré-estabelecidos, de tal forma que se possa determinar se um documento possui um dado critério em um nivel adequado, sem que se utilize julgamento subjetivo. A seguir são apresentados três exemplos de tabelas de pontuação, utilizadas nas seguintes atividades:

1 - Avaliação de uma métrica especifica;

2 - Avaliação de um fator;

3 - Totalização dos valores encontrados, cujo resultado final represente a confiabilidade predita. Este valor deve então ser confrontado com aquele anteriormente estabelecido para a meta, permitindo assim, caso seja igual ou maior, que o projeto prossiga no seu ciclo de desenvolvimento, caso contrário, que seja revisto nos pontos que se fizerem necessários.

PRESTIME - Predição da Confiabilidade de Especificações de Requisitos de Software		
OBJETIVO: Confiabilidade da Representação	FATOR: Comunicabilidade SUBFATOR: Modularidade	CRITÉRIO DE AVALIAÇÃO: Organização Global (MO.3)

PROCESSOS DE AVALIAÇÃO:	%	0/1	PESO
1- O documento tem os capítulos e as seções organizadas numa seqüência lógica estruturada? $(1 - \frac{\text{no. de capítulos que violam a regra}}{\text{total de capítulos}})$ $(1 - \frac{\text{no. de seções que violam a regra}}{\text{total de seções}})$	☐ ☐	☐ ☐	☐ ☐
2- A documentação mostra de forma clara o controle do fluxo de dados do sistema (através de gráficos e explicações)?		☐	☐
3- A documentação contém algum esquema que facilite a rápida localização e acesso às informações ali contidas (índices de tabelas, índices remissivos, divisórias para os capítulos, etc.)?		☐	☐
4- A documentação possui capítulos (volumes) separados de acordo com as funções a serem oferecidas pelo sistema?		☐	☐
$\text{Valor da Métrica} = \frac{\sum_{i=1}^{4} \text{item}_i \times \text{peso}_i}{\sum_{i=1}^{4} \text{peso}_i}$		Total de pesos ☐	☐

Definição: A especificação possui o critério Organização Global na medida em que o seu conteúdo esteja organizado numa seqüência lógica e estruturada.

Discussão: Para que a organização do documento seja adequada é necessário que ela ofereça uma visão clara do mesmo, de forma a tornar possível o rápido acesso às informações.

Tab. II.6

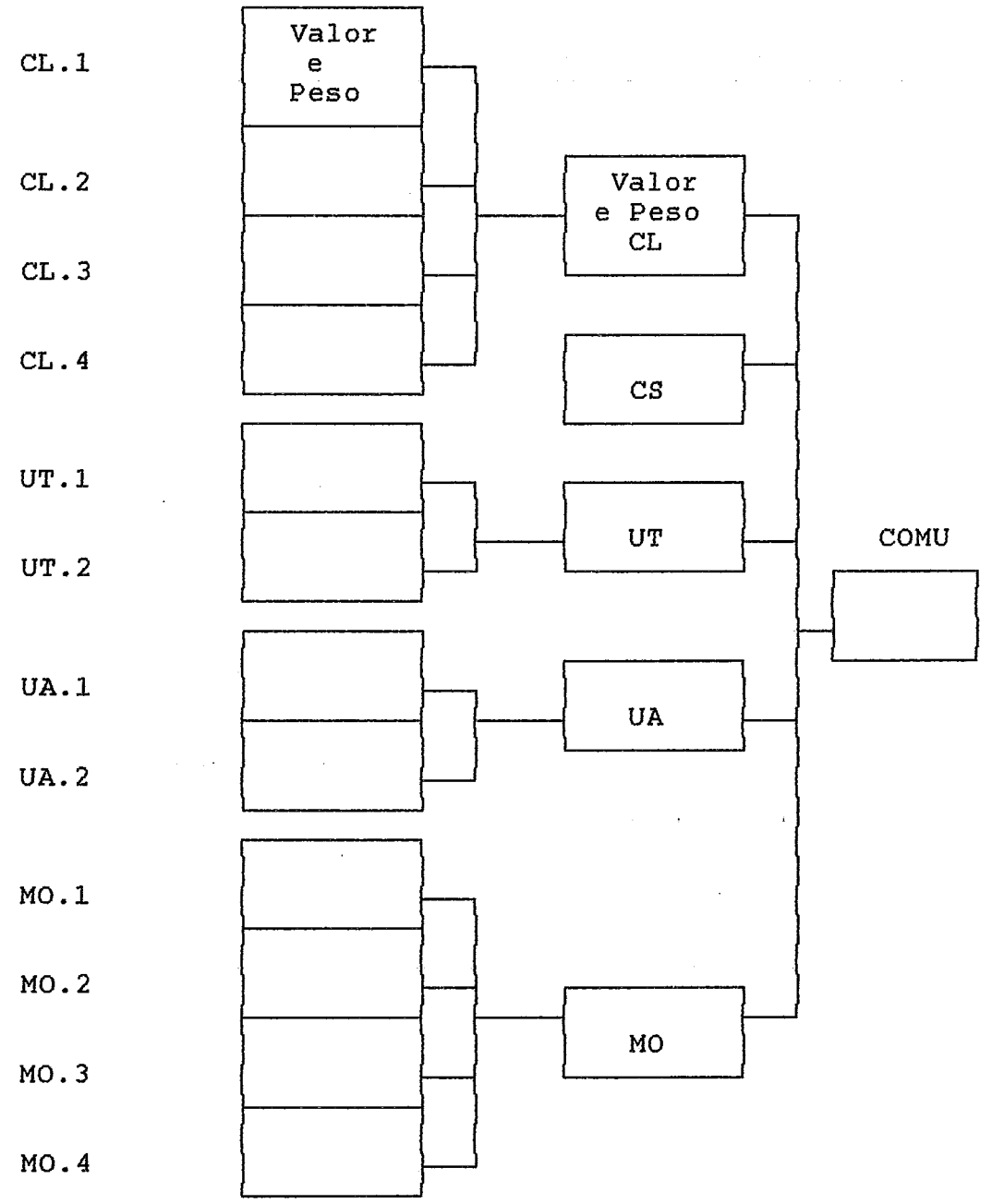
Formulário para avaliação de uma métrica específica

PRESTIME - Predição da Confiabilidade de Especificações de Requisitos de Software

Avaliação da Confiabilidade do Fator: Comunicabilidade

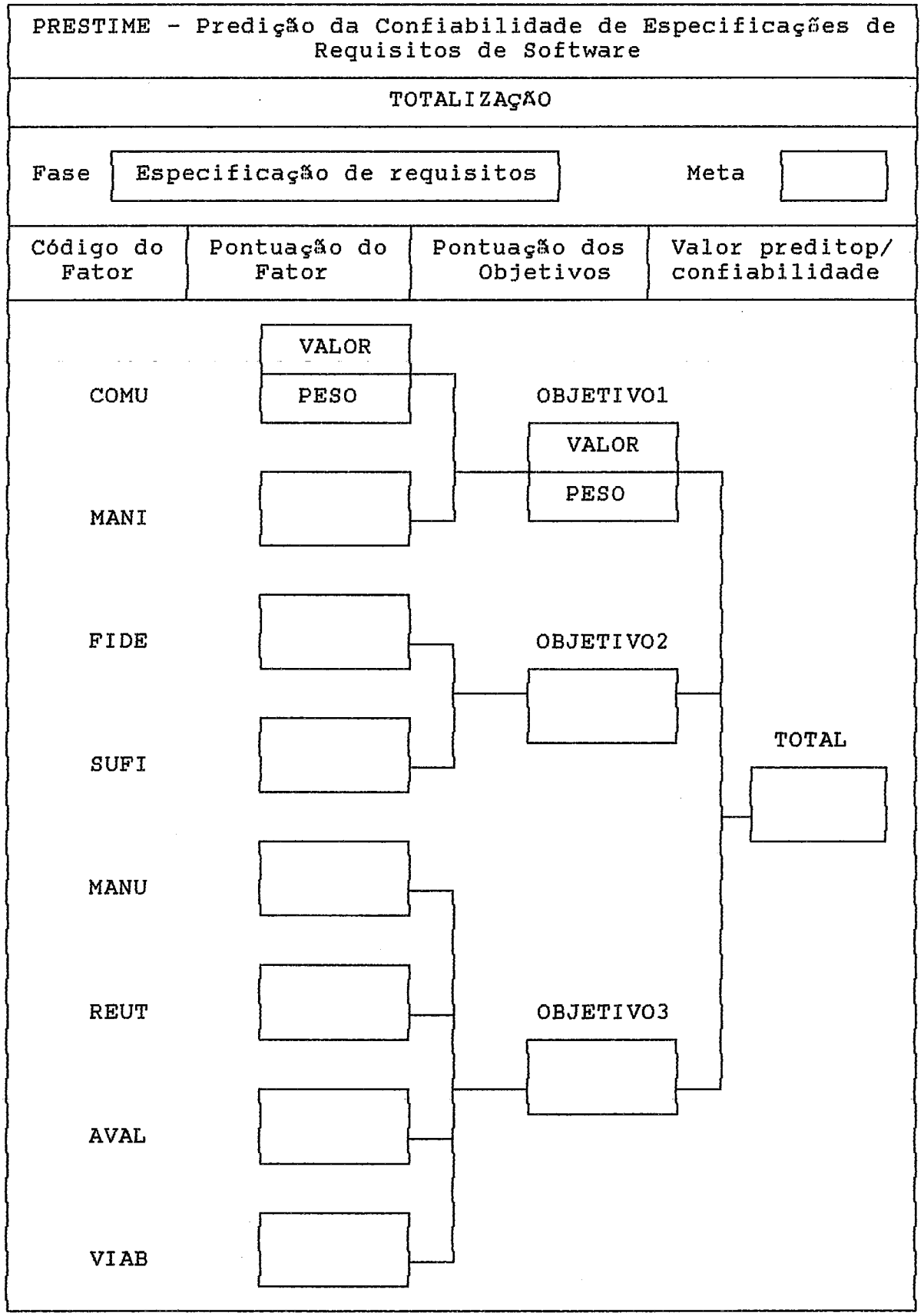
Código: COMU

Código do Critério	Pontuação dos Critérios	Pontuação dos SubFatores	Pontuação do Fator



Tab. II.7

Formulário para avaliação de um fator de qualidade



Tab. II.8

Formulário para a totalização da confiabilidade de software

2.3.3- Predição da confiabilidade de especificações

A geração de produtos que satisfaçam a expectativa dos seus usuários depende fundamentalmente da qualidade das especificações, base para o projeto e para sua posterior implementação. A completa especificação dos requisitos do programa é essencial para o sucesso do esforço de desenvolvimento e conseqüente atendimento às características de qualidade desejadas.

Especificações são um meio de comunicação entre o pessoal envolvido no projeto. Deste modo, especificações devem assegurar que a intenção e o entendimento que o especificador tem do problema a ser resolvido está documentado de forma confiável. Além disso, devem assegurar que esta intenção e o entendimento podem ser perfeitamente compreendidos por qualquer leitor autorizado a isso, mesmo que especificadores e leitores não se conheçam e não tenham possibilidade de interagir [14].

Determinar se a especificação atingiu um grau satisfatório de qualidade envolve a aplicação de medidas quantitativas. No entanto, segundo Beaufond [18], medidas para o software não podem ser, como nas demais disciplinas da engenharia, baseadas em ciências físicas. Isto torna necessário que a abordagem dada às medidas de software seja marcada pelo paradigma científico tradicional de hipótese, avaliação, crítica e revisão.

Uma série de técnicas manuais e automatizadas para a verificação de especificações podem ser aplicadas [26]. Não há uma técnica universalmente aceita, devendo-se estabelecer a melhor combinação delas para os propósitos especificos, já que todas elas apresentam pontos positivos e negativos. Das técnicas manuais, a mais utilizada é a da leitura da especificação software por um indivíduo que não participou da sua elaboração e que procura descobrir problemas potenciais. A técnica de referências cruzadas é utilizada em conjunto com a leitura do documento, e se baseia na construção de tabelas de referências cruzadas e diagramas que mostram as interações entre as distintas entidades especificadas. Outras técnicas manuais existentes são Listas de Verificações, Modelos Manuais e Cenários. Também é possível a aplicação de técnicas automatizadas, tais como Referências Cruzadas automatizadas, Modelos automatizados simples, Cenários Detalhados, Provas Matemáticas, etc.

Baseado no modelo descrito por Rocha [14], Beaufond [18] examinou o problema da qualidade de especificações, identificando caracteristicas primitivas e definindo processos de avaliação. No que se refere a especificações foram identificados três objetivos de qualidade, denominados confiabilidade da representação, confiabilidade conceitual e confiabilidade de utilização. Estes mesmos objetivos são utilizados para avaliar a qualidade de projetos e de programas, o que é realizado com base em processos de avaliação especificos para cada uma destas atividades.

Para que seja possível avaliar se a especificação reflete o que o usuário deseja, é necessário que ele seja capaz de ler e compreender a especificação. Além disso, as especificações são documentos lidos e manipulados por diversas pessoas não necessariamente conhecidas entre si. Isto evidencia a importância da confiabilidade da representação, isto é, a necessidade de que a especificação seja confiável de forma que as informações nela contidas comuniquem satisfatoriamente o seu conteúdo ao pessoal envolvido no projeto.

É fundamental construir um produto de acordo com as necessidades e expectativas do usuário, portanto é necessário que o conteúdo da especificação esteja conceitualmente livre de erros, para que possa atingir o objetivo "confiabilidade conceitual". Segundo Fairley [21], uma especificação que possui um conjunto de requisitos incorreto ou incompleto pode gerar um produto que satisfaça a estes requisitos, mas não satisfaça as necessidades e expectativas do usuário. Uma especificação inconsistente gera requisitos contraditórios em diferentes partes do documento, enquanto requisitos ambíguos permitem diferentes interpretações por diferentes indivíduos.

Uma especificação existe para ser utilizada. Assim sendo, uma especificação deve atingir o objetivo "confiabilidade da utilização". A seguir, temos o modelo proposto que caracteriza os objetivos da confiabilidade de especificações em termos de seus fatores e subfatores.

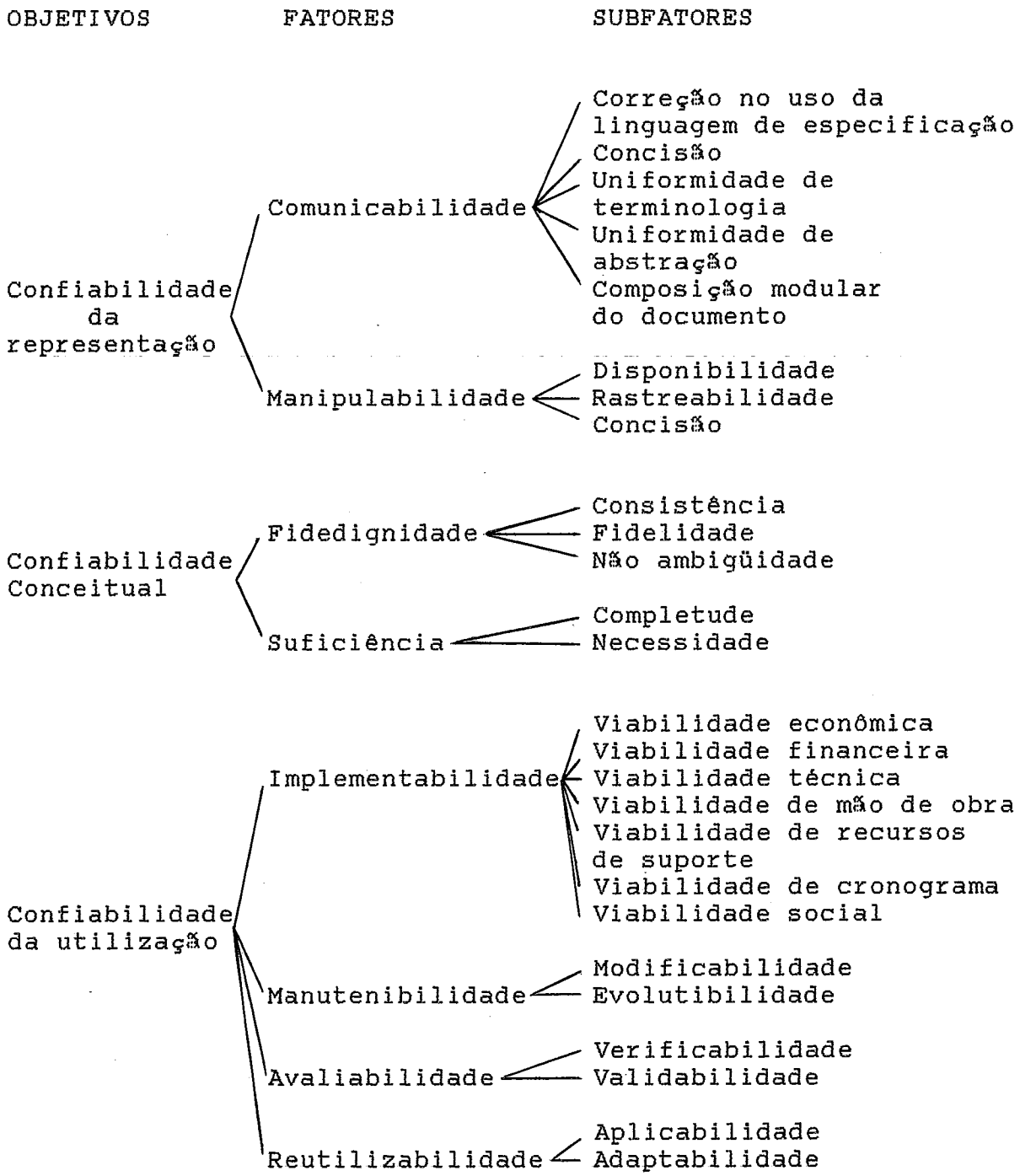


Fig. II.5

Objetivos, fatores e subfatores da qualidade de especificações

Fonte: [14]

2.3.3.1- Confiabilidade da representação de especificações

É atingido através dos fatores comunicabilidade e manipulabilidade.

Comunicabilidade- Refere-se à capacidade de comunicação da especificação. É atingida através dos subfatores correção no uso da linguagem de especificação, concisão, uniformidade de terminologia, uniformidade de abstração e modularidade.

Correção no uso da linguagem de especificação- é a característica de uma especificação de requisitos estar correta do ponto de vista do uso da linguagem de especificação. É avaliada segundo os critérios: correção no uso do formato de documentação, correção da notação, correção sintática e correção semântica.

Correção no uso do formato de documentação- a especificação é escrita utilizando, de forma correta, o formato padrão definido para a especificação de requisitos.

Correção da notação- utilização correta da notação pré-definida para a linguagem de especificação.

Correção sintática- utilização correta do conjunto de regras sintáticas pré-definidas na linguagem de especificação.

Correção semântica- utilização correta do conjunto de regras semânticas pré-definidas na linguagem de especificação.

Concisão- é a característica de uma especificação de requisitos estar escrita com um mínimo de texto, isto é, com uma maximização da quantidade de informações por unidade de texto. É avaliada através dos critérios: não redundância, complementabilidade, formalidade e tamanho.

Não redundância- um mesmo aspecto não aparece desnecessariamente em mais de um lugar na especificação.

Complementabilidade- existência de referências a documentos auxiliares que facilitem o entendimento da especificação.

Formalidade- os aspectos especificados são descritos de forma rigorosa e precisa.

Tamanho- os capítulos e seções possuam uma extensão adequada.

Uniformidade de terminologia- é a característica de uma especificação de requisitos estar escrita com uniformidade de terminologia e simbologia. É avaliada através dos critérios: padronização dos termos técnicos e uniformidade de notação.

Padronização de termos técnicos- utilização de termos técnicos, ao longo do texto, de acordo com as definições pré-estabelecidas.

Uniformidade de notação- utilização, ao longo de uma especificação escrita em linguagem natural, de uma notação gráfico-lingüistica uniforme e precisa.

Uniformidade de abstração- é a característica de uma especificação de requisitos estar escrita com o mesmo grau de detalhamento nos diversos níveis de abstração. É avaliada através dos critérios: uniformidade de detalhe e independência de detalhes de projeto.

Uniformidade de detalhe- para um determinado nível de abstração, todos os aspectos são tratados com o mesmo nível de detalhamento.

Independência de detalhes de projeto- inexistência de restrições referentes às características próprias da fase de projeto.

Modularidade- é a característica de uma especificação de requisitos estar dividida em módulos, permitindo um perfeito entendimento de seus diversos aspectos de forma exclusiva. É avaliada através dos critérios: coesão, acoplamento, organização global e tamanho.

Coesão- existência de uma forte associação das informações dentro dos capítulos e de suas seções.

Acoplamento- o grau de independência entre os capítulos e seções é adequado, segundo os padrões de qualidade definidos pela organização.

Organização Global- condição em que os capítulos e seções de uma especificação estejam organizados segundo uma seqüência lógica.

Tamanho- ver subfator concisão.

Manipulabilidade- refere-se a facilidade de manipulação, utilização e modificação da especificação. É atingido através dos subfatores: concisão, disponibilidade e rastreabilidade.

Concisão- ver fator comunicabilidade.

Disponibilidade- é a característica de uma especificação de requisitos estar disponível para o acesso de seus usuários, na versão mais atualizada. É avaliada através dos critérios: acessibilidade e estar atualizada.

Acessibilidade- qualquer dos usuários autorizados pode facilmente consultar a especificação.

Estar atualizada- o conteúdo da especificação reflete as informações mais recentes.

Rastreabilidade- é a caracteristica de uma especificação de requisitos permitir o acompanhamento de uma seqüência de agregação de detalhes relacionados a um determinado aspecto ou assunto. É avaliada através dos critérios: localização interna, localização externa e organização da documentação.

Localização interna- existência de facilidades para a localização de todos os elementos dentro da especificação.

Localização externa- existência de facilidades para a localização de todas as informações relacionadas com a especificação, mas que ali não estejam.

Organização da documentação- condição em que a especificação esteja, ela própria e junto a outras especificações, organizada de tal forma que permita uma fácil manipulação.

2.3.3.2- Confiabilidade conceitual de especificações

É atingido através dos fatores fidedignidade e suficiência

Fidedignidade- refere-se a confiança de que a especificação representa o que pode ser entendido como sendo as necessidades e expectativas de seus usuários. É avaliada através dos subfatores fidelidade, consistência e não ambigüidade.

Fidelidade- é a caracteristica de uma especificação de requisitos estar escrita coerentemente com as necessidades do usuário.

Consistência- é a caracteristica de uma especificação de requisitos estar escrita isenta de contradições. É avaliada através dos critérios: consistência interna e consistência externa.

Consistência interna- inexistência de conflitos entre os aspectos de uma mesma especificação.

Consistência externa- condição em que, aspectos contidos na especificação não conflitam com outras especificações ou entidades externas.

Não ambigüidade- é a caracteristica de uma especificação de requisitos estar escrita isenta de possibilidade de diferentes interpretações para qualquer requisito. É avaliada através dos critérios: ser explicita e exatidão.

Ser explicita- inexistência de condições, hipóteses ou restrições definidas por contexto e isenta de definições implicitas.

Exatidão- aspectos especificados são descritos de forma quantificável.

Suficiência- refere-se a completude, necessidade e maneira implícita com que os requisitos são especificados. É atingido através dos subfatores: completude e necessidade.

Completude- é a característica de uma especificação de requisitos conter todos os requisitos determinantes do software em desenvolvimento completamente definidos. É avaliada através dos critérios: completude com relação ao roteiro definido pela organização e completude da documentação prevista pelo método.

Completude com relação ao roteiro definido pela organização- condição em que os aspectos a serem especificados são baseados num roteiro pré-definido.

Completude da documentação prevista pelo método- utilização de todos os instrumentos que auxiliem na produção da documentação e que estejam previstas no método.

Necessidade- é a característica de uma especificação de requisitos estar escrita isenta de descrições inúteis. É avaliada através do critério necessidade dos requisitos.

Necessidade dos requisitos- consideração somente dos requisitos considerados obrigatórios ou desejáveis.

2.3.3.3- Confiabilidade da utilização de especificações

É atingido através dos fatores implementabilidade, manutenibilidade, reutilizabilidade e avaliabilidade.

Implementabilidade- Não tem sentido continuar-se o desenvolvimento de um produto baseando-se em especificações fora da realidade ou inviáveis. Este fator é avaliado através dos subfatores: viabilidade econômica, financeira, de mão de obra, de cronograma e social.

Viabilidade econômica- é a característica de se poder construir o software tendo como critério de avaliação a relação entre os custos e os benefícios. É avaliada através dos critérios: aceitabilidade de custos, relevância de benefícios e compatibilidade custos/benefício.

Aceitabilidade de custos- as estimativas de custos são aceitas por usuários e desenvolvedores.

Relevância de benefícios- as estimativas de benefícios tangíveis e intangíveis são aceitos por usuários e desenvolvedores.

Compatibilidade custos/benefício- os custos estimados para o desenvolvimento e manutenção são compatíveis com os benefícios econômicos esperados.

Viabilidade financeira- é a característica de se poder construir o software tendo como critério a avaliação da disponibilidade orçamentária. É avaliada através dos critérios: existência de capital e disponibilidade de capital.

Existência de capital- situação em que a organização possua capital suficiente para custear o desenvolvimento.

Disponibilidade de capital- situação em que a organização seja capaz de tornar disponível o capital necessário para o desenvolvimento.

Viabilidade técnica- é a característica de se poder construir o software tendo como critério de avaliação, a posse e o domínio da tecnologia necessária para conduzir o desenvolvimento. É avaliada através dos critérios: existência de tecnologia e disponibilidade de tecnologia.

Existência de tecnologia- existência do nível de tecnologia necessária para conduzir o desenvolvimento.

Disponibilidade de tecnologia- disponibilidade da tecnologia necessária aos desenvolvedores.

Viabilidade de mão de obra- é a característica de se poder construir o software tendo como critério de avaliação a disponibilidade de mão de obra necessária para desenvolver

o software. É avaliada através dos critérios: existência de mão de obra e disponibilidade de mão de obra.

Existência de mão de obra- existência, na organização, da mão de obra necessária para o desenvolvimento.

Disponibilidade de mão de obra- disponibilidade dos recursos humanos com o conhecimento e experiência necessários para realizar o desenvolvimento e operação do software.

Viabilidade de recursos de suporte- é a característica de se poder construir o software tendo como critério de avaliação a disponibilidade de recursos de suporte necessários para desenvolver o sistema. É avaliada através dos critérios: existência de recursos de suporte e disponibilidade de recursos de suporte.

Existência de recurso de suporte- existência, na organização, de recursos de hardware e de software necessários à atividade de suporte.

Disponibilidade de recursos de suporte- disponibilidade dos recursos de hardware e de software necessários à atividade de suporte.

Viabilidade de cronograma- é a característica de se poder construir o software tendo como critério de avaliação as

restrições de cronograma. É avaliada através dos critérios: adequabilidade de cronograma e flexibilidade de cronograma.

Adequabilidade do cronograma- condição em que o software possa ser construído no tempo previsto, levando em consideração a ocorrência de possíveis imprevistos.

Flexibilidade do cronograma- o cronograma aceito atende a introdução de fatores imprevistos, tais como, atividades não projetadas e contingências.

Viabilidade social- é a característica de se poder construir o software tendo como critério de avaliação as implicações deste software na sociedade. É avaliada através dos critérios: aceitabilidade dos impactos sociais e aceitabilidade da engenharia humana.

Aceitabilidade dos impactos sociais- a especificação possui este critério na medida em que o software que for construído leve em consideração seus impactos sobre o sistema social ao qual deverá servir.

Aceitabilidade da engenharia humana- a especificação possui este critério na medida em que o software que for construído leve em consideração o grau de satisfação e o desenvolvimento do potencial humano previsto para os usuários.

Manutenibilidade- é o atributo que uma especificação deve ter de forma a poder ser facilmente modificada e detalhada. É atingido através dos subfatores: modificabilidade e evolutibilidade.

Modificabilidade- é a característica de uma especificação de requisitos ser facilmente alterável. É avaliada através dos critérios definidos nos subfatores: correção no uso da linguagem de especificação, modularidade, rastreabilidade, consistência e dos critérios não redundância e acessibilidade.

Evolutibilidade- é a característica de uma especificação de requisitos ser de fácil detalhamento a medida que se for crescendo no conhecimento do sistema. É avaliado através dos subfatores: correção no uso da linguagem de especificação, uniformidade de terminologia, uniformidade de abstração, modularidade, rastreabilidade e dos critérios não redundância, acessibilidade e ser explícita.

Reutilizabilidade- é o atributo que a especificação deve ter de forma a poder ser reutilizada, totalmente ou em parte, ao se realizar o desenvolvimento de outro software com área de aplicação semelhante. É atingido através dos subfatores: aplicabilidade e adaptabilidade.

Aplicabilidade- é a característica de uma especificação de requisitos poder ser reutilizada ao ser especificada outra aplicação na mesma área de domínio do problema. É avaliada

através dos critérios definidos no subfator modularidade e do critério padronização dos termos técnicos.

Adaptabilidade- é a característica de uma especificação de requisitos poder ser adaptada para corresponder perfeitamente a outra aplicação. É avaliada através do subfator modularidade.

Avaliabilidade- especificações devem ser avaliadas de forma a se poder realizar um controle de sua qualidade. Esta avaliação deve ser realizada tendo-se em conta a forma e o conteúdo da especificação. Este fator é atingido através dos subfatores: verificabilidade e validabilidade.

Verificabilidade- é a característica de uma especificação de requisitos poder ser analisada com relação a forma de sua apresentação. É avaliada através dos critérios: acessibilidade e localização interna.

Validabilidade- é a característica de uma especificação de requisitos poder ser analisada quanto ao seu conteúdo. É avaliada através dos critérios definidos nos subfatores: correção no uso da linguagem de especificação, concisão, uniformidade de terminologia, modularidade, disponibilidade, rastreabilidade e do critério uniformidade de detalhe.

2.3.4- Predição da confiabilidade de projeto

A fase de desenvolvimento absorve 75% ou mais do custo de software excluindo a manutenção [38]. É na etapa de projeto que se tomam as decisões que irão influir decisivamente no sucesso da implementação do software e na facilidade com que a manutenção executará suas atividades.

Existe uma tendência em se querer iniciar a fase de construção antes mesmo de se ter chegado aos níveis finais de detalhamento para cada função a ser implementada.

Freqüentemente, a pressa em terminar a construção do programa induz a resultados que não representam os objetivos reais do produto anteriormente definidos [38].

Segundo Rocha [14], ao se desenvolver um projeto de software, deve-se ter em mente o detalhamento, a nível de projeto, de todas as funções que foram identificadas na fase de definição.

A transformação dos requisitos especificados em funções implementáveis deve ser cuidadosa, já que pode, algumas vezes, incluir funções que não são relevantes.

O projeto é o único caminho por onde se pode, precisamente, traduzir os desejos do usuário para um produto que atenda aos requisitos especificados e sirva como suporte para toda a fase de desenvolvimento e manutenção. A Figura II.6 ilustra esta idéia.

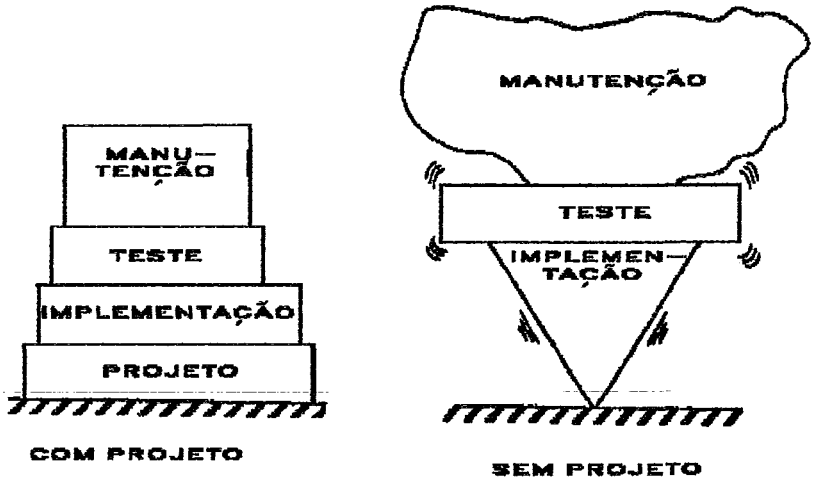


Fig. II.6

Ciclo de desenvolvimento de software com e sem projeto

Fonte: [38]

A fase de projeto tem como base especificações geradas na fase de definição. Deve-se, portanto, manter as características de qualidade da especificação. Uma das maiores dificuldades encontradas pelos projetistas de software é determinar quando se deve parar de projetar, a nível de detalhamento, e iniciar a codificação do programa. Uma das responsabilidades da gerência é determinar se os atributos de qualidade, próprios do projeto em questão, e os objetivos priorizados e explicitados antes de iniciar o projeto foram alcançados. Desta forma, o projeto termina quando estes atributos e objetivos forem atingidos.

Nesta fase, são gerados três produtos intermediários: Projeto Externo, Projeto de Arquitetura e Projeto Detalhado. O Projeto Externo envolve a concepção, planejamento e especificação das

características observáveis do produto (telas para o usuário, formato de relatórios, fontes de dados externos, etc.) [21]. O Projeto de Arquitetura é a especificação modular lógica, capaz de tornar real o programa em algum ambiente de programação. O Projeto Detalhado é a especificação dos algoritmos que implementam as funções identificadas na organização modular [14].

Rocha [14] identificou os objetivos, fatores e critérios desejáveis na fase de projeto. Baseando-nos no método descrito anteriormente, podemos manter as características de qualidade já alcançadas na fase de especificação. Segundo o modelo proposto, o produto deverá ser desenvolvido de forma a atingir os mesmos objetivos de qualidade: Confiabilidade de Representação, Confiabilidade Conceitual e Confiabilidade de Utilização.

Confiabilidade da Representação de projetos deve ser atendida a fim de garantir que a especificação de projeto contenha um correto detalhamento das necessidades e expectativas dos usuários, já que é necessário que este detalhamento seja lido, compreendido e manipulado por diversas pessoas que, na maioria das vezes, não se conhecem. Em última análise, o documento em questão deve comunicar satisfatoriamente seu conteúdo.

A Figura II.7 apresenta o esquema proposto contendo os objetivos de qualidade, fatores e subfatores que permitem atingir a qualidade de projeto.

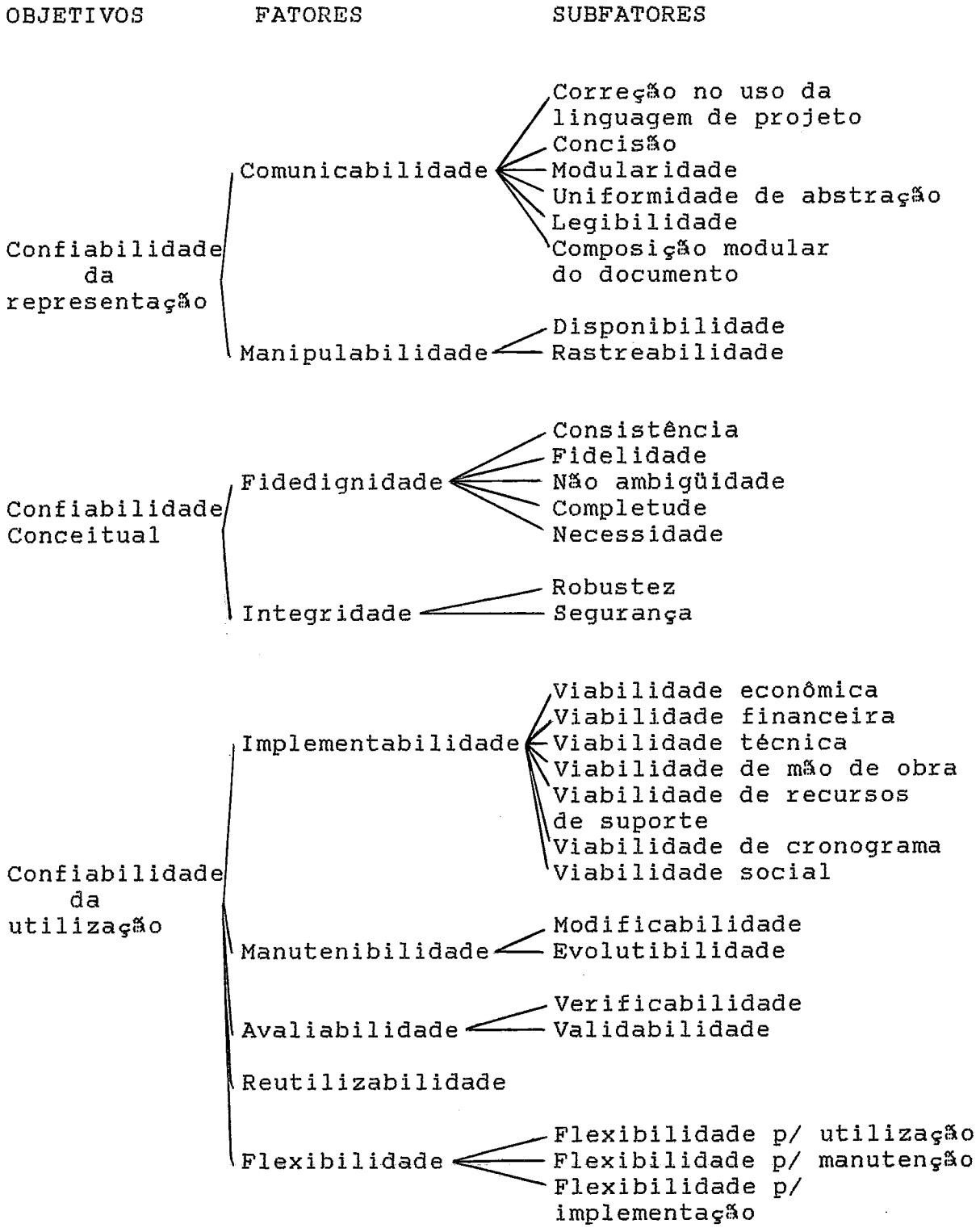


Fig. II.7

Objetivos, fatores e subfatores da qualidade de projetos

Fonte: [14]

A Confiabilidade Conceitual de projetos se refere à correção dos documentos gerados na fase de projeto com relação ao seu conteúdo, isto é, tais documentos devem atender aos objetivos que serviram de base para sua elaboração e detalhar claramente as funções que serão implementadas.

Os projetos são desenvolvidos com o intuito de serem implementados. Não há sentido em se elaborar um projeto que não seja utilizado. Logo, há necessidade em se atender ao objetivo Confiabilidade de Utilização de Projetos, ou seja, o projeto deve permitir sua fácil modificação, reutilização, avaliação e implementação.

2.3.4.1- Confiabilidade da representação de projetos

É atingida através dos fatores comunicabilidade e manipulabilidade.

Comunicabilidade- refere-se à capacidade que o documento tem em transmitir aos leitores todas as informações necessárias, exclusivamente através delas próprias. É atingido através dos subfatores: correção no uso da linguagem de projeto, composição modular do documento, concisão, uniformidade no nível de abstração e modularidade.

Correção no uso da linguagem de projeto- é a característica dos Projetos de Arquitetura e Detalhado em estarem corretos no que tange às regras estabelecidas pela linguagem utilizada. É avaliada através dos critérios: correção da

notação, correção sintática, correção semântica, correção no uso do formato de documentação, padronização dos termos técnicos, uniformidade de notação e adesão às normas da organização.

Correção da notação- utilização correta da notação definida na linguagem de projeto.

Correção sintática- utilização correta do conjunto de regras sintáticas definidas pela linguagem de projeto.

Correção semântica- utilização correta e de maneira uniforme dos símbolos gráficos, matemáticos ou fonemas definidos na linguagem.

Correção no uso do formato de documentação- observada a obediência às regras de utilização do formato de documentação definido pela linguagem de projeto.

Padronização dos termos técnicos- utilização de uma terminologia consistente e uniforme ao longo do documento.

Uniformidade de notação- utilização da simbologia de maneira uniforme ao longo do documento.

Adesão às normas da organização- condição em que são obedecidos normas e padrões técnicos estabelecidos pela organização.

Composição modular do documento- é a característica do documento gerado na fase de projeto poder ser dividido em partes logicamente encadeadas, facilitando o entendimento de seus vários tópicos de maneira exclusiva e sendo avaliado através dos critérios: coesão do documento, acoplamento do documento, organização global e tamanho.

Coesão do documento- existência de uma alta associação das informações dentro de um capítulo e de uma seção.

Acoplamento do documento- existência de uma alta independência entre os capítulos ou seções.

Organização global- o conteúdo do documento deve estar organizado segundo uma seqüência lógica.

Tamanho- condição em que os capítulos e seções estejam elaborados com uma extensão dentro da faixa definida pelos padrões de qualidade adotados e adequada para o seu entendimento.

Concisão- é a característica que um projeto deve ter de forma a maximizar o número de informações por unidade de texto. É avaliada através dos critérios: não redundância, complementabilidade, formalidade e tamanho.

Não redundância- uma mesma definição ou assunto aparece em apenas um local no documento gerado.

Complementabilidade- existência de indicações a documentos auxiliares que facilitem o entendimento do projeto.

Formalidade- utilização de uma abordagem rigorosa, precisa e metódica do projeto.

Tamanho- ver subfator composição modular do documento.

Uniformidade no nível de abstração- é a característica que um projeto deve ter para que seja mantido o nível de refinamento adequado, sem a introdução de detalhes irrelevantes ao nível da abstração em questão, garantindo que este nível será desenvolvido dentro dos limites de cada fase de refinamento. É avaliado através dos critérios: nível de refinamento inicial adequado, uniformidade de detalhes e independência dos detalhes de implementação.

Nível de refinamento inicial adequado- condição em que o nível de abstração final da especificação de requisitos esteja apropriado a transformação em uma modelagem de projeto de arquitetura.

Uniformidade de detalhes- condição em que o nível de detalhamento é uniforme ao longo de todo o documento.

Independência dos detalhes de implementação- não existência de restrições a detalhes cuja abordagem seja própria da codificação.

Modularidade- É atingida através dos critérios: modularidade estrutural, modularidade lógica e modularidade psicológica.

Modularidade estrutural- é a característica de um Projeto de Arquitetura estar elaborado como uma estrutura de módulos altamente independentes, isto é, módulos com alta coesão e baixo acoplamento. É avaliado através dos subcritérios coesão, acoplamento, número de módulos subordinados, número de módulos superiores e balanceamento.

Coesão- alto nível de união entre os elementos de um mesmo módulo.

Acoplamento- baixa possibilidade de que uma modificação feita num módulo provoque efeitos colaterais em outros módulos.

Número de módulos subordinados ("fan-out")- existência de um número adequado (menor do que 7 [33]) de módulos imediatamente subordinados a um determinado módulo.

Número de módulos superiores ("fan-in")- existência de um número elevado de módulos imediatamente superiores a um determinado módulo, de maneira a evitar a codificação da mesma função em vários lugares.

Balanceamento- situação em que os módulos de nível mais alto tratam os dados de forma lógica de modo que estes não sejam dependentes das características físicas que serão tratadas nos níveis mais baixos.

Modularidade lógica- refere-se a preocupação com as características intrínsecas de um projeto detalhado, sendo avaliada através dos subcritérios: complexidade, abrangência, isolamento e memória de estado.

Complexidade- condição em que o número de caminhos básicos dentro de um módulo seja menor do que 10 [45].

Abrangência- elevado nível de flexibilidade das informações de forma a facilitar a reutilização do módulo.

Isolamento- possibilidade de implementação do módulo utilizando apenas a sua definição.

Memória de estado- os módulos de um programa não devem guardar a memória da sua execução anterior.

Modularidade psicológica- refere-se a preocupação com as caracteristicas dos módulos que afetam sua compreensão pelas pessoas. É avaliada através dos subcritérios: tamanho do módulo, simplicidade, relato de erros e documentação.

Tamanho do módulo- ver subfator composição modular do documento.

Simplicidade- alcance dos objetivos da forma mais simples possível, satisfazendo ainda os demais requisitos de qualidade.

Relato de erros- situação em que o próprio módulo que detecta uma falha, apresenta a mensagem de erro correspondente.

Documentação- a especificação do módulo encontra-se disponível e atualizada.

Legibilidade- refere-se a capacidade de discernir facilmente as funções do software através da especificação do projeto detalhado [16]. É avaliada através dos critérios estilo, auto-descrição, tamanho do módulo, simplicidade e relato de erros.

Estilo- o projeto possui este critério na medida em que é elaborado de maneira clara.

Auto-descrição- a especificação do módulo contém comentários que visam auxiliar a compreensão de sua lógica.

Tamanho do módulo- ver subfator composição modular do documento.

Simplicidade- ver critério modularidade psicológica.

Relato de erros- ver critério modularidade psicológica

Manipulabilidade- refere-se à facilidade de acesso e manipulação das diferentes versões do projeto e as suas partes, durante o seu uso, modificação ou teste. É atingida através dos subfatores: disponibilidade e rastreabilidade.

Disponibilidade- refere-se à facilidade de consulta do projeto por todos os seus usuários, em sua versão mais atualizada. É avaliada através dos critérios: acessibilidade, atualização e documentação.

Acessibilidade- facilidade de consulta e obtenção de cópias da especificação do projeto por qualquer usuário autorizado.

Atualização- o conteúdo da documentação reflete as informações mais recentes.

Documentação- ver subfator modularidade psicológica.

Rastreabilidade- é possivel o acompanhamento de um determinado aspecto ao longo dos documentos produzidos. É avaliado através dos critérios: organização da documentação, hierarquização, localização interna e localização externa.

Organização da documentação- existência de facilidade de percorrer, manipular e entender toda a documentação do projeto.

Hierarquização- situação em que as estruturas que compõem a documentação do projeto estejam organizadas hierarquicamente.

Localização interna- facilidade de localizar e percorrer itens dentro de cada documento que compõe a especificação.

Localização externa- facilidade de localizar todos os documentos que compõem a especificação de projeto e que tratem de um determinado aspecto.

2.3.4.2- Confiabilidade da utilização de projetos

É atingida através dos fatores: manutenibilidade, reutilizabilidade, avaliabilidade, flexibilidade e implementabilidade.

Manutenibilidade- refere-se à capacidade do projeto em ser facilmente atualizado e detalhado ao longo do processo de desenvolvimento. É avaliado através dos subfatores: modificabilidade e evolutibilidade.

Modificabilidade- é a característica do projeto poder ser elaborado de forma a ser facilmente atualizável.

Evolutibilidade- é a característica do projeto ser elaborado de forma a ser facilmente detalhado a medida que avança no desenvolvimento.

Reutilizabilidade- refere-se à capacidade do projeto ter suas partes organizadas e completamente desenvolvidas de maneira a permitir sua utilização parcial ou total em outras aplicações. É avaliada através dos subfatores: aplicabilidade e adaptabilidade.

Aplicabilidade- é a característica de um projeto poder ser reutilizado ao se realizar o desenvolvimento de outro software com área de aplicação semelhante. É atingida através dos critérios: padronização de termos técnicos, coesão, acoplamento e organização global.

Padronização de termos técnicos- ver subfator correção no uso da linguagem de projeto.

Coesão- ver critério modularidade estrutural.

Acoplamento- ver critério modularidade estrutural.

Organização global- ver subfator composição modular do documento.

Adaptabilidade- é a característica de um projeto poder ser adaptado de maneira a corresponder perfeitamente a outra aplicação. É avaliado através dos critérios: coesão, acoplamento e organização global.

Coesão- ver critério modularidade estrutural.

Acoplamento- ver critério modularidade estrutural.

Organização global- ver subfator composição modular do documento.

Avaliabilidade- refere-se à capacidade da especificação de projeto poder ser avaliada. É atingido através dos subfatores: validabilidade e verificabilidade.

Validabilidade- é a característica de um projeto poder ser facilmente avaliado, dentro de um processo finito e economicamente viável, com relação ao atendimento das especificações. É atingida através dos critérios: correção da notação, correção sintática, correção semântica, correção no uso do formato de documentação, uniformidade de documentação, uniformidade de notação e padronização de termos técnicos.

Correção da notação- ver subfator correção no uso da linguagem de projeto.

Correção sintática- ver subfator correção no uso da linguagem de projeto.

Correção no uso do formato da documentação- ver subfator correção no uso da linguagem de projeto.

Uniformidade de notação- ver subfator correção no uso da linguagem de projeto.

Padronização de termos técnicos- ver subfator correção no uso da linguagem de projeto.

Uniformidade de detalhes- ver subfator uniformidade no nível da abstração.

Complementabilidade- ver subfator concisão.

Formalidade- ver subfator concisão.

Coesão- ver critério modularidade estrutural.

Acoplamento- ver critério modularidade estrutural.

Organização global- ver subfator aplicabilidade.

Acessibilidade- ver subfator manipulabilidade.

Atualização- ver subfator manipulabilidade.

Localizabilidade interna- ver subfator rastreabilidade.

Localizabilidade externa- ver subfator rastreabilidade.

Organização da documentação- ver subfator rastreabilidade.

Verificabilidade- é a característica de um projeto poder ser avaliado com relação à forma de sua apresentação. É atingida através dos critérios: acessibilidade e organização da documentação.

Acessibilidade- ver subfator manipulabilidade.

Organização da documentação- ver subfator rastreabilidade.

Flexibilidade- refere-se à característica de um projeto ser elaborado de forma a não conter detalhes de implementação, isto é, sem definir como o software deverá ser construído. É atingida através dos subfatores: flexibilidade para reutilização, flexibilidade para manutenção e flexibilidade para implementação.

Flexibilidade para reutilização- é a característica de um projeto não conter detalhes de implementação e incluir, apenas as características gerais de abordagem do problema. É avaliada através dos critérios: coesão para reutilização, acoplamento para reutilização, abrangência e isolamento.

Coesão para reutilização- existência de uma forte ligação entre as informações contidas num mesmo módulo.

Acoplamento para reutilização- existência de um alto grau de interdependência entre os módulos.

Abrangência- existência de um nível de generalização adequado.

Isolamento- condição em que o entendimento de um módulo não implica na necessidade de conhecimento sobre seus detalhes técnicos.

Flexibilidade para manutenção- é a característica de um projeto poder ser elaborado de forma a permitir facilmente a inclusão ou alteração de requisitos, que podem ter finalidades corretivas, adaptativas ou evolutivas. É avaliada através dos critérios: flexibilidade corretiva, flexibilidade adaptativa e flexibilidade evolutiva.

Flexibilidade corretiva- facilidade de alteração num projeto no que se refere a modificações corretivas.

Implementabilidade- refere-se à capacidade de um projeto permitir a construção do software levando em conta aspectos econômicos, técnicos, financeiros, de mão de obra, de recursos de suporte, de cronograma e sociais. É atingido através dos subfatores: viabilidade econômica, técnica, financeira, de mão de obra, de recursos de suporte, de cronograma e social.

Viabilidade econômica- é a caracteristica de um projeto poder ser implementado levando em conta uma análise de custos e benefícios, ou seja, a comparação entre os custos envolvidos no desenvolvimento e operação do software e os benefícios advindos de seu uso. É avaliada através dos critérios: aceitabilidade de custos, Relevância de benefícios e compatibilidade custos/benefícios.

Aceitabilidade de custos- existência de concordância no que se refere aos custos estimados para o desenvolvimento e operação do software projetado.

Relevância de benefícios- situação em que são aceitáveis as estimativas de benefícios tangíveis e intangíveis advindos do uso do software.

Compatibilidade custos/benefícios- existência de compatibilidade entre os custos estimados e os benefícios esperados com a construção do software.

Viabilidade técnica- é a caracteristica de um projeto poder ser implementado levando em conta a existência, posse

Flexibilidade adaptativa- facilidade de alteração num projeto no que se refere a modificações adaptativas.

Flexibilidade evolutiva- facilidade de alteração num projeto no que se refere a modificações evolutivas.

Flexibilidade para implementação- é a característica de um projeto possuir alternativas de implementação de acordo com as necessidades encontradas no estudo de viabilidade. É possível a redefinição de módulos de acordo com os recursos disponíveis. Este subfator é avaliado através dos critérios: flexibilidade tecnológica, de recursos de suporte, de custos e de objetivos.

Flexibilidade tecnológica- inexistência de vinculações a tecnologias específicas.

Flexibilidade de recursos de suporte- independência dos recursos de suporte a serem adotados.

Flexibilidade de custos- existência de alternativas para soluções que envolvam altos custos.

Flexibilidade de objetivos- existência de capacidade de subdividir o projeto em partes, de tal forma que seja possível dar prioridades diferentes para a implementação de suas diversas partes.

e dominio da tecnologia necessária para a construção do software. É avaliada através dos critérios: existência de tecnologia e disponibilidade de tecnologia.

Existência de tecnologia- existência da tecnologia necessária para construção do software.

Disponibilidade de tecnologia- disponibilidade da tecnologia necessária para a implementação, de forma que a equipe de projeto encarregada da construção do software possa utilizar-se dela.

Viabilidade financeira- é a caracteristica de um projeto poder ser implementado levando em conta a disponibilidade de capital necessário para a construção do software, mesmo que este seja viável econômica e tecnicamente. É avaliada através dos critérios: existência de capital e disponibilidade de capital.

Existência de capital- existência de capital suficiente para custear a construção do software.

Disponibilidade de capital (fluxo de caixa)- condição em que a organização desenvolvedora seja capaz de tornar disponivel o capital necessário no momento adequado.

Viabilidade de mão de obra- é a característica de um projeto poder ser implementado levando em conta a

disponibilidade de uma equipe de desenvolvimento com conhecimento e experiência profissional adequada para construção do software. É avaliada através dos critérios: existência de mão de obra e disponibilidade de mão de obra.

Existência de mão de obra- existência de mão de obra qualificada para a construção do software.

Disponibilidade de mão de obra- disponibilidade de mão de obra necessária para a construção do software.

Viabilidade de recursos de suporte- é a característica de um projeto poder ser implementado levando em conta a necessidade de utilização de determinados recursos de hardware e software. É avaliado através dos critérios: existência de recursos de suporte, disponibilidade de recursos de suporte e compatibilidade custos/benefícios de recursos de suporte.

Existência de recursos de suporte- situação em que hajam recursos de suporte para a construção do software.

Disponibilidade de recursos de suporte- situação em que acham-se disponíveis os recursos de suporte necessários para a construção do software.

Compatibilidade custos/benefícios de recursos de suporte- existência de um grau adequado de

compatibilidade entre custos estimados e benefícios esperados com a utilização dos recursos de suporte.

Viabilidade de cronograma- é a característica de um projeto poder ser implementado levando em conta os prazos necessários para o desenvolvimento do software e tendo margem para imprevistos que porventura venham a ocorrer. É avaliada através dos critérios: adequabilidade do cronograma e flexibilidade do cronograma.

Adequabilidade do cronograma- situação na qual o software possa ser construído no tempo previsto pelo cronograma.

Flexibilidade do cronograma- condição na qual o cronograma possa ser cumprido mesmo com a ocorrência de imprevistos ou com a inclusão de atividades não previstas.

Viabilidade social- é a característica de um projeto poder ser implementado levando em conta a análise dos impactos sociais (desemprego, mudanças na rotina de trabalho, influência na privacidade dos indivíduos, etc.) e a garantia que o software resultante executará as funções adequadas e úteis à sociedade. É avaliada através dos critérios: aceitabilidade de impactos sociais e aceitabilidade da engenharia humana.

Aceitabilidade de impactos sociais- situação em que os impactos sociais advindos da construção do software são aceitáveis no sistema social ao qual deverá servir.

Aceitabilidade da engenharia humana- situação em que os fatores psicológicos e físicos que determinam o conforto, tolerância e amenidade ao uso do software são aceitáveis.

2.3.4.3- Confiabilidade conceitual de projeto

É atingido através dos fatores fidedignidade e integridade.

Fidedignidade- refere-se ao atributo que um projeto deve ter de forma a garantir que ele realmente retrata, a nível de projeto, o que foi especificado. É avaliado através dos subfatores: fidelidade, consistência, não ambigüidade, completude e necessidade.

Fidelidade- é a característica existente no projeto de corresponder a realidade do problema que se pretende resolver, de forma a garantir que será elaborado coerentemente com a especificação.

Consistência- é a característica existente no projeto que assegura que as suas diferentes partes estejam elaboradas de modo a se completarem sem provocar contradições. É avaliada através dos critérios: consistência interna e consistência externa.

Consistência interna- situação na qual o projeto da arquitetura e o projeto detalhado não conflitem entre si.

Consistência externa- situação na qual os aspectos contidos no projeto de arquitetura e no projeto detalhado (e que se relacionem) não conflitem entre si.

Não ambigüidade- é a característica existente no projeto que assegura que o mesmo seja elaborado de forma a estar isento de possibilidade de falsas interpretações. É avaliada através dos critérios ser explícito e exatidão.

Ser explícito- não existência de aspectos definidos implicitamente.

Exatidão- condição em que os aspectos especificados estejam descritos de forma quantificável.

Completude- é a característica existente no projeto que assegura que o mesmo esteja elaborado de forma a conter todas as funções necessárias à resolução do problema em questão. É avaliada através dos critérios: completude da especificação de projeto com relação ao roteiro elaborado pela organização, completude do projeto de arquitetura com relação à especificação de requisitos, completude do projeto detalhado com relação ao projeto de arquitetura e

completude do projeto de arquivos com relação à especificação de requisitos.

Completude da especificação do projeto com relação ao roteiro elaborado pela organização- situação em que a especificação de projeto é construída tendo por base um roteiro pré-definido.

Completude do projeto de arquitetura com relação a especificação de requisitos- situação em que o projeto de arquitetura contenha todas as funções definidas na especificação de requisitos.

Completude do projeto detalhado com relação ao projeto de arquitetura- situação em que são construídos todos os algoritmos que implementam as funções identificadas no projeto de arquitetura.

Completude do projeto de arquivos com relação a especificação de requisitos- situação em que a especificação de projeto contenha a descrição lógica e física de todos os arquivos identificados na especificação de requisitos.

Necessidade- é a característica existente no projeto de forma a assegurar que o mesmo esteja elaborado sem conter funções inúteis, que podem causar ineficiência e baixa rentabilidade do software. É avaliada através do critério necessidade das funções especificadas.

Necessidade das funções especificadas- situação em que a especificação de projeto contenha somente funções imprescindíveis ao alcance dos objetivos definidos na especificação de requisitos.

Integridade- refere-se ao atributo que um projeto deve ter de forma a permitir que se tenha confiança em sua resposta a falhas e a situações hostis. É avaliado através dos subfatores: robustez e segurança.

Robustez- é a característica existente no projeto de forma a prever soluções para situações hostis ao software, prevendo o adequado tratamento das mesmas (a previsão de possíveis situações excepcionais causadas por um mau uso do software propiciará, futuramente, a continuidade na execução do programa, que só será interrompido quando houver risco para a segurança). É avaliada através dos critérios: tolerância à entrada de dados incorretos, ratificabilidade, detectabilidade da falha, recuperabilidade, redundância, distintividade, reparabilidade e confinamento da falha.

Tolerância a entrada de dados incorretos- situação em que é prevista a capacidade do software tolerar um grau de variações na entrada de dados sem mau funcionamento ou rejeição.

Ratificabilidade- situação em que é prevista a realimentação para o o indivíduo que gerou os dados de entrada, de forma a permitir que estes possam ser posteriormente confirmados.

Detectabilidade da falha- existência de mecanismos que sejam capazes de detectar desvios do comportamento esperado para as operações de software, hardware e operador.

Recuperabilidade- existência de mecanismos que possibilitem a correção de erros pelo programa, associado a sua habilidade de reconstruir dados na forma e conteúdo desejados.

Redundância- previsão no projeto, para os módulos mais críticos, de cópias funcionalmente equivalentes, de tal forma que, caso ocorra um distúrbio no módulo em operação, este possa ser substituído pelo equivalente.

Distintividade- existência de mecanismos que ofereçam independência temporal do ponto onde a falha poderia ocorrer em programas que estejam executando funções redundantes.

Reparabilidade- existência de mecanismos que permitam ao software ser reparado e retorne às condições de operação dentro de um tempo de reparo especificado.

Confinamento da falha- os erros que porventura ocorram numa determinada região do programa não devem ser repassados a outras partes deste mesmo programa.

Acurácia- adoção de algoritmos que forneçam resultados dentro de uma tolerância especificada.

Segurança- é a característica existente no projeto de forma a prever soluções que evitem falhas passíveis de provocar um alto risco (a análise do risco deve ser criteriosa e resultante de um estudo da possibilidade de ocorrência de desastre em termos de perda de vidas humanas ou grandes prejuízos financeiros decorrentes de mau funcionamento do software). É avaliada através dos critérios: detectabilidade de violação, privacidade, sensibilidade, vulnerabilidade, imputabilidade, confidencialidade e criticalidade.

Detectabilidade de violação- utilização de mecanismos cuja finalidade seja evitar que o dano objetivado pelo acesso indevido venha acontecer.

Privacidade- utilização de mecanismos de proteção e controle das operações realizadas por um determinado indivíduo.

Sensibilidade- o projeto possui este critério na medida em que seja possível determinar o grau de importância das informações para a organização que as utiliza.

Vulnerabilidade- o projeto possui este critério na medida em que possui habilidade de tornar-se seguro a determinados tipos de ataque.

Imputabilidade- utilização de mecanismos que garantam que uma entidade ou pessoa possa ser responsabilizada pelas suas ações, de modo que as violações ou tentativas de violação do software possam ser imputadas inequivocamente ao violador.

Confidencialidade- utilização de mecanismos que garantam que a informação será protegida contra a revelação ou colocada a disposição de indivíduos ou órgãos não autorizados.

Criticalidade- o projeto possui este critério na medida em que determina as conseqüências das falhas que afetem o funcionamento do programa e os procedimentos a serem tomados em tais casos.

2.3.5- Predição da confiabilidade de programas

Todos os passos da engenharia de software até aqui descritos têm como objetivo final a transformação de algumas das muitas formas de sua representação num formato que possa ser compreendido pelo computador [38], ou seja, o processo de transformação do projeto numa linguagem de programação, ou simplesmente codificação.

Certamente, durante a fase operacional do produto, este será em maior ou menor grau, dependendo da qualidade com que foi produzido, alvo de alterações, sejam elas corretivas ou evolutivas.

Durante a codificação, os projetistas freqüentemente produzem um trabalho baseado em suas experiências e no estilo próprio de programação. Isto seria aceitável se, durante a fase de manutenção, o projetista estivesse sempre disponível para fazer as alterações necessárias no programa, lembrando-se do que fez e porque fez. Muitas vezes, fatores externos impedem que o projetista continue acompanhando o produto durante sua fase operacional.

Em geral, um novo projeto já foi iniciado antes mesmo que tenha terminado o anterior, o que implica num gasto de grande parte do tempo de manutenção para que outro indivíduo adquira um conhecimento satisfatório do programa e possa realizar as alterações necessárias.

Outros fatores, tais como a insatisfação profissional, seja ela técnica ou financeira, acidentes, doenças, mudanças frequentes nas atividades das pessoas, etc, podem da mesma forma afetar seriamente a qualidade do programa e conseqüentemente os custos e a complexidade da atividade de manutenção.

A qualidade do código gerado é fundamental, não apenas para manter a complexidade e a eficiência do programa em um nível adequado mas, também, para facilitar o seu entendimento por parte de outros projetistas.

A fim de evitar, ou pelo menos minorar os problemas acima mencionados, novos métodos e técnicas vêm sendo utilizados. Vantagens significativas em termos de cronograma e custos vêm sendo conseguidos com a elaboração de módulos que desempenham funções específicas e bem definidas, permitindo assim, que os mesmos sejam reutilizados quando necessário, conseqüentemente reduzindo a atividade de testes de módulo, necessária apenas na primeira utilização, e aumentando o nível de confiança do programa.

As características da linguagem da programação escolhida e o estilo podem afetar profundamente a qualidade do software e sua manutenibilidade. Uma melhor clareza do código fonte é conseguida com técnicas de programação estruturada, por um bom estilo de programação, pelos documentos de suporte apropriados, pelos bons comentários internos e pelas características oferecidas pelas modernas linguagens de programação, facilitando, assim, as atividades de depuração, teste e modificação do código fonte.

Mesmo com a adoção destes conceitos, não podemos garantir, a priores, que estamos construindo um "bom programa". São necessárias informações quantitativas que nos mostrem, quão bom ou ruim ele está. Além do mais, as propriedades exigidas

variam de acordo com as necessidades e prioridades dos usuários e das características das diversas aplicações, logo não se pode estabelecer uma medida única de qualidade de programas.

Várias tentativas têm sido feitas com o objetivo de determinar características ou atributos de qualidade de programas, já que é necessário que estes possam ser avaliados de forma prática na determinação da sua qualidade.

Para que os programas desenvolvidos atendam às necessidades dos usuários e tenham uma vida útil longa e produtiva, após terem sido colocados em operação, é necessário que satisfaçam os objetivos da confiabilidade de programas conceitual, de utilização e de representação [14].

O objetivo da confiabilidade da representação de programas deve ser atingido a fim de permitir que diferentes pessoas, que necessitem ler ou manipular um dado programa, possam fazê-lo com facilidade durante todo o tempo em que o mesmo estiver operacional.

O objetivo da confiabilidade da utilização de programas é garantir que possa ser usado pelos seus usuários, já que de nada adianta o esforço gasto em seu desenvolvimento se o mesmo não puder ser utilizável.

Utilizaremos, novamente, o modelo Rocha [14] a fim de avaliar a confiabilidade de programas. A Figura II.8 mostra o esquema

proposto pelo modelo contendo os objetivos, fatores e subfatores de qualidade.

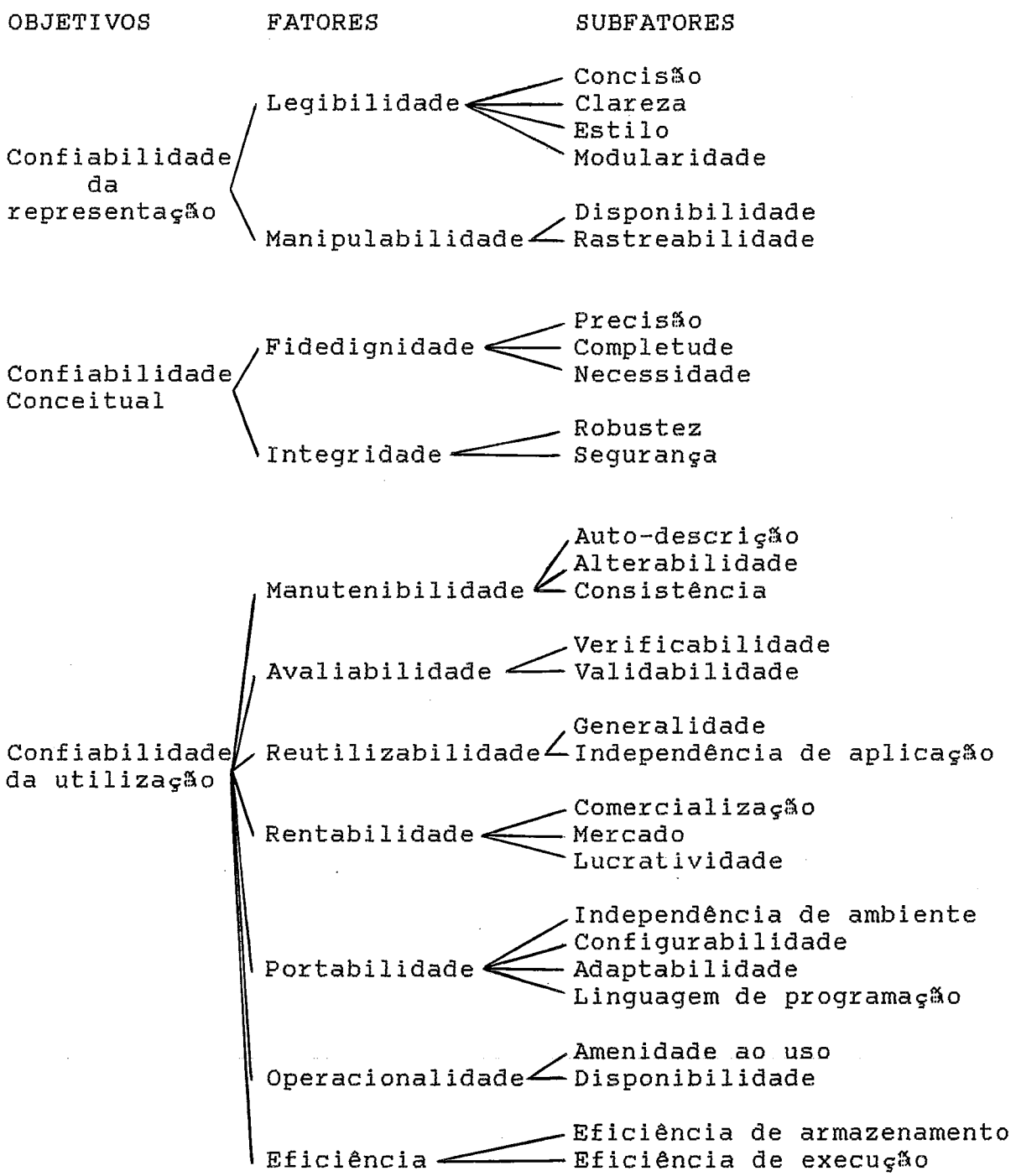


Fig. II.8

Objetivos, fatores e subfatores da qualidade de programas

Fonte: [14]

O objetivo da confiabilidade conceitual de programas é atingido se as necessidades e requisitos que motivaram a sua construção são satisfeitos.

2.3.5.1- Confiabilidade da representação de programas

Refere-se às características de representação do programa que afetam sua compreensão e manipulação pelas pessoas [14]. É atingida através dos fatores legibilidade e manipulabilidade.

Legibilidade- refere-se à capacidade que um programa deve ter de tal forma que possa ser lido sem dificuldade por diferentes pessoas, muitas das quais não participaram do seu desenvolvimento. É atingida através dos subfatores: clareza, concisão, estilo e modularidade.

Clareza- é a característica de um programa ter as funções que lhe cabem codificadas da forma mais clara possível, isento de práticas que o tornem complexo e de difícil entendimento. É avaliado através dos critérios: complexidade das interfaces, complexidade do fluxo do programa, complexidade funcional da aplicação, complexidade da comunicação e complexidade da execução [4].

Complexidade das interfaces- existência de número adequado de parâmetros passados para as funções internas ou externas do módulo.

Complexidade do fluxo do programa- existência de um número adequado (menor do que 10 [33]) de caminhos básicos. Também chamada de complexidade ciclomática ou lógica [89]. Esta medida avalia o número de caminhos básicos de um programa através de um grafo de controle e é expressa pela seguinte equação:

V(g) = e - n + 2p, onde:
e = número de ligações no grafo.
n = número de nós no grafo.
r = número de componentes conexo no grafo.

Complexidade funcional da aplicação- a área de dados utilizada para guardar os resultados intermediários gerados pelas operações realizadas durante a execução do módulo não deve ser compartilhada por outros módulos.

Complexidade da comunicação- o programa possui este critério na medida em que os módulos utilizam um número adequado de referências globais, possuem apenas um ponto de entrada e não há passagem de elementos de controle entre eles.

Complexidade da execução- não necessidade de recalcular valores já calculados, não possuir declarações que nunca serão executadas e não utilizar dados intermediários desnecessários.

Concisão- é a característica de um programa ter as funções que lhe cabem implementadas com a quantidade mínima de código. É atingida através dos critérios: complexidade do código, simplicidade do código e especificidade.

Complexidade do código- Esta medida baseia-se na análise estática do texto do programa, isto é, no número de operadores e operandos [88]. Um vocabulário extenso significa que o programador precisará aprender mais palavras para poder entender, codificar e manter o sistema [45]. O maior problema com os programas de computador é que o vocabulário muda de programa para programa. Os operadores básicos permanecem os mesmos, mas os nomes dos dados mudam de acordo com a vontade do programador. Por isso os dicionário de dados são tão úteis. Eles restringem o vocabulário dos operandos a serem usados pelo programador.

A avaliação é feita através das seguintes equações:

Vocabulário = n1 + n2.

Dificuldade = n2 N2 / 2 N2.

onde:

n1 = número de operadores únicos.

n2 = número de operandos.

N1 = número total de operadores.

N2 = número total de operandos.

Também fazem parte deste método outras equações que podem fornecer informações importantes a cerca das características do programa, tais como:

Tamanho = N1 + N2.

Estimativa do tamanho = $n1 \log_2(n1) + n2 \log_2(n2)$.

Volume = tamanho X $\log_2(n1+n2)$.

Nível de linguagem = volume / dificuldade^2.

Esforço = dificuldade X volume.

Simplicidade do código- utilização de poucos comandos e poucas variáveis para realizar uma determinada função.

Especificidade- a escolha de determinadas opções do programa não deve conduzir a estados indeterminados e que não tenham sido previstos. Este critério pode ser avaliado utilizando-se, em parte, das equações descritas no critério "complexidade do fluxo do programa".

Estilo- é a característica de um programa estar codificado com endentação, comentários adequados e padronizações de identificações. É avaliado através dos critérios: linguagem estruturada, quantidade de comentários, efetividade dos comentários, legibilidade da linguagem, endentação e significância.

Linguagem estruturada- implementação utilizando linguagens estruturadas e que possuam pré-processador [4].

Quantidade de comentários- existência de um número adequado de comentários.

Efetividade dos comentários- condição em que os comentários utilizados descrevam com precisão e concisamente o elemento em questão.

Legibilidade da linguagem- não utilização de identificadores que dificultem o entendimento do programa.

Endentação- situação em que as estruturas do programa estão corretamente endentadas.

Significância das identificações- os nomes atribuídos as variáveis, constantes, funções, subrotinas, subprogramas e procedimentos traduzem com facilidade o seu significado.

Modularidade- é a característica de um programa ao implementar as funções que lhe cabem com uma estrutura de módulos altamente independente de acordo com as definições de projeto. É avaliada através dos critérios: modularidade estrutural, modularidade lógica e modularidade psicológica.

Modularidade estrutural- refere-se a preocupação com a garantia da estrutura modular definida no projeto. É avaliada através dos subcritérios: coesão, acoplamento, no. de módulos subordinados, no. de módulos superiores e balanceamento.

Coesão- garantia da existência de um alto nivel de coesão.

Acoplamento- garantia da existência de um baixo grau de interdependência entre os módulos.

No. de módulos subordinados- garantia da existência de um número adequado de módulos subordinados (não mais que 7) [33].

No. de módulos superiores- garantia que os módulos de niveis inferiores, sempre que possível, sejam chamados por vários outros módulos superiores, de forma a evitar duplicação de código.

Balanceamento- garantia de que os módulos de nível mais alto tratem os dados de uma forma lógica a fim de que estes não sejam dependentes das características fisicas.

Modularidade lógica- refere-se as características intrínsecas de cada módulo. É avaliada através dos

subcritérios: complexidade, abrangência, isolamento e memória de estado.

Complexidade- existência de um número adequado (menos que 10) de caminhos básicos [45].

Abrangência- existência de um nível adequado de generalização de forma a permitir uma maior flexibilidade do dados.

Isolamento- situação em que os módulos tenham sido implementados de forma que não seja necessário conhecer suas características internas.

Memória de estado- situação em que os módulos, sempre que invocados, executem suas funções como se fosse a 1ª vez, mesmo que já tenham sido invocados anteriormente.

Modularidade psicológica- refere-se as características do programa que o tornam ou não inteligível às pessoas. É avaliada através dos critérios: tamanho, simplicidade, relato de erro e documentação.

Tamanho- existência de um número adequado de comandos por módulo (entre 30 e 60 comandos) [33].

Simplicidade- os módulos atingem seus objetivos da forma mais simples, satisfazendo ainda todos os requisitos de qualidade.

Relato de erro- os erros são relatados pelo módulo que os detecta e identifica.

Documentação- a documentação do módulo (código fonte) deve estar disponível, assim como os roteiros, dados de teste, laudos de avaliação e informação de alterações efetuadas [14].

Manipulabilidade- refere-se a capacidade que um programa deve ter a fim de que possa ser facilmente manipulado por diferentes pessoas. É atingido através dos subfatores: disponibilidade e rastreabilidade.

Disponibilidade- é a característica de um programa e sua documentação estarem atualizados e prontos para uso quando necessário. É avaliado através dos critérios: acessibilidade e estar atualizada.

Acessibilidade- qualquer pessoa autorizada deve poder consultar os programas fontes e a documentação associada em sua forma mais atualizada.

Estar atualizada- o conteúdo do programa fonte deve refletir as características mais recentes.

Rastreabilidade- é a característica de um programa e sua documentação permitirem a procura de uma informação através da seqüência de agregação de detalhes de um determinado aspecto. É avaliada através dos critérios: localização interna, localização externa e organização global.

Localização interna- existência de facilidades dentro do programa fonte, de maneira a permitir a rápida localização de seus elementos (procedimentos, funções, declarações, definições, etc.).

Localização externa- situação em que os módulos estejam implementados de forma a que se possa localizar com facilidade os elementos que não estejam ali contidos (por ex. funções externas).

Organização de documentação- os fontes do programa e a documentação associada devem estar organizados de tal forma que permitam uma fácil manipulação.

2.3.5.2- Confiabilidade conceitual de programas

É atingido através dos fatores fidedignidade e integridade.

Fidedignidade- refere-se a característica que um programa deve ter de tal forma que este corresponda ao que foi especificado e projetado. É avaliado através dos subfatores: precisão, completude e necessidade.

Precisão- é a característica de um programa proporcionar a suficiente exatidão nos cálculos e resultados de modo a satisfazer a utilização pretendida pelo usuário e descrita nos requisitos de desempenho. É atingida através dos critérios: acurácia e correção.

Acurácia- o programa possui este critério na medida em que os métodos numéricos usados são consistentes com os requisitos de acurácia da aplicação.

Correção- os resultados obtidos são coerentes com aqueles esperados pelo usuário.

Completude- é a característica de um programa ter implementadas todas as funções específicas. É avaliada através dos critérios: completude do programa com relação aos algoritmos especificados no projeto detalhado e completude do programa com relação a norma de elaboração de programas fonte de organização [4].

Completude do programa com relação aos algoritmos especificados no projeto detalhado- condição em que todos os algoritmos descritos no projeto detalhado devem estar implementados no programa.

Completude do programa com relação a norma de elaboração de programas fonte da organização- o programa deve estar implementado tendo por base a norma para elaboração de programas fonte utilizado pela

organização (folha de rosto, códigos de identificação, estrutura do controle de alterações, critérios de endentação, etc.).

Necessidade- é a característica de um programa ter implementada somente as funções que foram especificadas. É avaliado através do critério necessidades das funções implementadas.

Necessidade das funções implementadas- somente as funções imprescindíveis ao alcance dos objetivos definidos no projeto de arquitetura devem ser implementadas.

Integridade- refere-se à característica que um programa deve possuir para enfrentar situações hostis, dados errados ou agressões. É atingido através dos fatores: robustez e segurança.

Robustez- refere-se à característica de um programa possuir meios apropriados para enfrentar situações hostis, reagindo a elas sem perda do controle. É atingida através dos critérios: tolerância a entrada de dados incorreta, ratificabilidade, detectabilidade da falha, recuperabilidade, redundância, distintividade, reparabilidade e confinamento da falha.

Tolerância a entrada de dados incorreta- situação em que o programa é capaz de tolerar um grau de variação

na entrada de dados sem mau funcionamento ou rejeição [90].

Ratificabilidade- existência de realimentação para o operador dos dados de entrada do programa de forma a permitir que estes possam ser posteriormente validados [91].

Detectabilidade da falha- o programa é capaz de detectar desvios de comportamento esperado para o software, hardware e operador [90].

Recuperabilidade- existência de mecanismos que possibilitem a correção de erros pelo próprio programa. É o mesmo capaz de reconstruir as informações na forma e no conteúdo desejados [90].

Redundância- situação em que estejam implementadas cópias funcionalmente equivalentes dos módulos considerados mais críticos, de tal forma que caso ocorra um distúrbio no módulo em operação, este possa ser substituído por seu equivalente [92].

Distintividade- situação em que há independência temporal de ocorrência de uma falha em módulos redundantes [90].

Reparabilidade- existência de mecanismos que permitam ao software ser reparado e retorne as condições de

operação dentro de um especificado tempo de reparo [90].

Confinamento de falha- condição em que os erros que por ventura ocorram numa determinada região do programa não sejam repassados para outras partes do mesmo programa [43].

Segurança- refere-se à característica que um programa deve ter para evitar falhas que possam provocar conseqüências desastrosas em termos de custo econômico ou humano. É atingida através dos critérios: detectabilidade de violação, privacidade, sensibilidade, vulnerabilidade, imputabilidade, confidenciabilidade e criticalidade.

Detectabilidade de violação- existência de mecanismos cuja finalidade seja evitar que o dano objetivado pelo acesso indevido venha a ocorrer [90].

Privacidade- existência de mecanismos de proteção e controle das operações realizadas por um determinado indivíduo [91].

Sensibilidade- existência de indicações da importância das informações processadas [91].

Vulnerabilidade- condição em que o programa possua a habilidade de tornar-se seguro a determinados tipos de ataque [91].

Imputabilidade- existência de mecanismos que garantam que uma entidade ou pessoa possa ser responsabilizada pelas suas ações [91].

Confidenciabilidade- existência de mecanismos que garantam que a informação será protegida contra revelação ou colocada a disposição de indivíduos ou órgãos não autorizados [91].

Criticalidade- possibilidade de poder determinar as conseqüências de determinadas falhas que por ventura ocorram e que afetem o funcionamento do programa [48].

2.3.5.3- Confiabilidade da utilização de programas

É atingido através dos fatores manutenibilidade, operacionalidade, portabilidade, avaliabilidade, reutilizabilidade, eficiência e rentabilidade.

Manutenibilidade- é a característica de um programa permitir a introdução de alterações, após ter sido inicialmente definido, desenvolvido e aceito como operacional. É avaliado através dos subfatores: auto-descrição, alterabilidade e consistência.

Auto-descrição- refere-se à característica que um programa deve ter de forma que o leitor do código fonte possa determinar seus objetivos, restrições, entradas, saídas, componentes e o estado da revisão. É avaliado através dos

critérios: objetivos do módulo, E/S do módulo, procedimentos do módulo e tabela de revisão [4].

Objetivos do módulo- existência de identificação dos objetivos dos módulos, submódulos, subrotinas e procedimentos no início de cada um deles.

E/S do módulo- situação em que sejam identificadas e comentadas todas as entradas e saídas do módulo.

Estrutura do módulo- condição em que é descrita suscintamente no início do programa a estrutura (em pseudocódigo) correspondente às funções a serem desempenhadas pelo mesmo.

Tabela de revisão- existência de tabela de controle de alterações para cada módulo.

Alterabilidade- refere-se à característica que um programa deve ter de forma a facilitar a incorporação de modificações. É avaliado através dos critérios: alteração para evolução e alteração para correção.

Alteração para evolução- o programa deve ser implementado levando em conta possíveis melhorias e evoluções de tal forma que estas sejam mencionadas na documentação e no programa fonte.

Alteração para correção- o programa deve ser implementado utilizando técnicas de instrumentação.

Consistência- refere-se à característica que um programa deve ter de forma a apresentar uma notação uniforme para os símbolos utilizados. É avaliado através dos critérios: consistência interna e consistência externa.

Consistência interna- o programa possui este critério na medida em que não existam conflitos entre os nomes das referências, representações de dados, variáveis, etc. dentro de um mesmo módulo [4].

Consistência externa- o programa possui este critério na medida em que não existam conflitos entre a forma em que são implementados os módulos, os protocolos utilizados para as chamadas de função e os procedimentos padronizados para estes aspectos [4].

Operacionalidade- é a característica de um programa ser oportuno e ameno ao uso, facilitando a comunicação com o usuário durante todo o tempo que este precisar. É avaliado através do subfator amenidade ao uso.

Amenidade ao uso- refere-se à capacidade do programa apresentar os resultados esperados oferecendo sentimentos de satisfação e confiança ao usuário que o utiliza. É avaliado através dos critérios: facilidade de aprendizado,

facilidade de utilização, apresentação, verbosidade e seleção de auxílio.

Facilidade de aprendizado- existência de funções de exemplo e de demonstração.

Facilidade de utilização- o programa é implementado levando em conta aspectos que facilitem a manipulação do mesmo pelo usuário através da utilização de recursos tais como, "mouse", caneta ótica, mesa digitalizadora ou dispositivos semelhantes.

Apresentação- utilização de interfaces gráficas e outras características que facilitem a apresentação das informações.

Verbosidade- disponibilidade para que o usuário possa estabelecer o nível de detalhamento das informações que lhe serão apresentadas (erros, advertências, intervenções, etc).

Seleção de auxílio- disponibilidade para o usuário, no momento em que ele desejar (via tecla de função, por exemplo), da descrição dos procedimentos, características e informações que possam servir para seu auxílio.

Portabilidade- é a característica de um programa poder ser operado de maneira fácil e adequada em configurações de

equipamentos diferentes da original. É avaliado através dos subfatores: independência de ambiente, configurabilidade, adaptabilidade e linguagem de implementação.

Independência de ambiente- refere-se a característica de um programa não conter restrições quanto a aspectos de software e de hardware. É avaliado através dos critérios: independência de software e independência de hardware.

Independência de software- possibilidade de execução do programa em diferentes ambientes de software.

Independência de hardware- possibilidade de execução do programa em diferentes ambientes de hardware.

Configurabilidade- refere-se à característica que um programa deve ter de forma a facilitar a alteração de suas características dentro de um mesmo ambiente.

Adaptabilidade- refere-se à capacidade que um programa deve ter de forma a facilitar a alteração de suas características entre vários ambientes. É avaliada através dos critérios: coesão, acoplamento e organização da documentação.

Coesão- ver subfator modularidade.

Acoplamento- ver subfator modularidade.

Organização global- ver subfator rastreabilidade.

Linguagem de implementação- refere-se à característica de um programa ser implementado tendo por base uma linguagem padronizada e que facilite o transporte para outros ambientes. É avaliado através dos critérios: comunalidade da linguagem de programação, existência de biblioteca padrão de E/S e linguagem de programação padronizada [4].

Comunalidade da linguagem de programação- condição onde a linguagem utilizada na implementação do programa está disponível em outros ambientes de operação.

Existência de biblioteca padrão de E/S- situação em que seja utilizada uma linguagem que possua uma biblioteca de funções para manipulação de E/S, com característica de implementação interna, transparentes para o usuário.

Linguagem de programação padronizada- existência de uma linguagem com padrão definido em norma.

Avaliabilidade- é a característica de um programa que retrata a facilidade em verificá-lo de modo a assegurar a execução da função que lhe cabe. É avaliado através dos subfatores: verificabilidade e validabilidade.

Verificabilidade- é a característica de um programa poder ser facilmente avaliado com relação a forma de sua

apresentação. É atingido através dos critérios: documentação e organização da documentação.

Documentação- ver subfator modularidade.

Organização da documentação- ver subfator rastreabilidade.

Validabilidade- é a característica de um programa poder ser facilmente avaliado dentro de um processo finito e economicamente viável, com relação ao que foi projetado. É atingido através dos critérios: complexidade do fluxo do programa, simplicidade do código, efetividade dos comentários, legibilidade da linguagem, localização interna e localização externa [4].

Complexidade do fluxo do programa- ver subfator clareza.

Simplicidade do código- ver subfator concisão.

Efetividade dos comentários- ver subfator estilo.

Legibilidade da linguagem- ver subfator estilo.

Localização interna- ver subfator rastreabilidade.

Localização externa- ver subfator rastreabilidade.

Reutilizabilidade- é a característica de um programa em ter suas funções desenvolvidas de maneira a permitir sua reutilização parcial ou total em outras aplicações. É avaliada através dos subfatores: modularidade, generalidade e independência da aplicação.

Modularidade- ver fator legibilidade.

Generalidade- refere-se à característica que um programa deve ter de forma que seus módulos tenham o mínimo possível de restrições no que concerne a tipos e quantidade de dados. É avaliada através dos critérios: independência de volume de dados e independência de tipos de dados.

Independência de volume de dados- inexistência de limitações por parte do programa no que concerne ao tratamento de quantidades maiores ou menores que o previsto.

Independência de tipos de dados- capacidade do programa de operar com os diferentes tipos de dados disponíveis na linguagem utilizada.

Independência de aplicação- refere-se à característica que um programa deve ter para que os algoritmos utilizados na implementação de uma determinada função permitam que possa ser reaproveitado em outras aplicações. É avaliado através dos critérios: independência do esquema de gerenciamento do

banco de dados e existência de padronização para a descrição dos dados utilizados [4].

Independência do esquema de gerenciamento do banco de dados- inexistência de restrições a referências específicas feitas ao banco de dados utilizado.

Existência de padronização para a descrição dos dados utilizados- existência de descrições de como os dados são produzidos, compostos e utilizados.

Eficiência- é a característica de um programa realizar suas funções sem desperdício de recursos. É avaliada através dos subfatores: eficiência de armazenamento e eficiência de execução [4].

Eficiência de armazenamento- refere-se à característica de um programa utilizar adequadamente a memória disponível. É avaliado através dos critérios: otimização do armazenamento e duplicidade de dados.

Otimização do armazenamento- condição em que sejam previstas e utilizadas as características de otimização de armazenamento disponíveis na linguagem de programação utilizada.

Duplicidade de dados- inexistência de redundância de informações.

Eficiência de execução- refere-se à característica de um programa executar suas esperadas funções dentro do menor tempo possível. É avaliado através dos critérios: sobrecarga do sistema, sobrecarga de entrada e saída.

Sobrecarga do sistema- inexistência de perda de tempo no processamento de funções que não estejam diretamente relacionadas com a obtenção do resultado desejado.

Sobrecarga de entrada e saída- condição em que é baixo o tempo gasto pelo programa na espera do término de funções que dependem de operações de entrada e saída.

Rentabilidade- é a característica de um programa possuir relação custo/benefício aceitável. É avaliada através dos subfatores: comercialização, mercado e lucratividade.

Comercialização- refere-se às características e recursos necessários pela organização para comercializar e distribuir o programa. É avaliado através dos critérios: recursos financeiros, recursos técnicos e recursos de mão de obra.

Recursos financeiros- existência de disponibilidade financeira para comercializar o programa.

Recursos técnicos- a mão de obra disponível está tecnicamente capacitada para comercializar o programa.

Recursos de mão de obra- existência de mão de obra suficiente para comercializar o programa.

Mercado- refere-se às caracteristicas existentes no mercado que afetem as vendas do programa. É avaliado através dos critérios: concorrência, usuários alvo, tempo de obsolescência e aderência aos padrões.

Concorrência- viabilidade de comercialização do programa tendo em vista a existência de produtos funcionalmente equivalentes na mesma faixa de preços.

Usuários alvo- existência de usuários em número adequado e em condições de adquirir o produto.

Tempo de obsolescência- situação em que o tempo de obsolescência do produto é suficiente para remunerar o capital investido.

Aderência aos padrões- inexistência de caracteristicas que possam inviabilizar a compra do programa por determinados usuários, devido ao não atendimento a padrões estabelecidos.

Lucratividade- refere-se as caracteristicas do programa com relação ao retorno financeiro, vantagens e benefícios advindos de sua comercialização. É avaliado através do critério taxa de retorno.

Taxa de retorno- situação em que a taxa de retorno do investimento feito no produto seja superior a taxa mínima de atratividade.

2.4- Estimação da confiabilidade

A confiabilidade de software é definida em termos estatísticos como "a probabilidade de falha de operação de um programa em um determinado ambiente durante um especificado período de tempo" [23]. Como exemplo, tomemos um programa cuja estimativa de confiabilidade é igual 0.95 em 8 horas de processamento real. Se tal programa for executado 100 vezes e necessitar de 8 horas para a sua execução, o mesmo estaria pronto para operar corretamente (sem falhas) em 95 vezes do total de 100.

Uma estimativa precisa da confiabilidade do software é necessária durante a fase de testes e validação para avaliar se o produto é aceitável e comporta-se como esperado no seu ambiente operacional.

Segundo Silva [22] a estimação da confiabilidade só é viável através da realização de testes, muito embora a prática atual mostre insuficiências devido a uma falta de base para gerar casos de testes aleatórios. Naturalmente, a atividade de estimação será fortemente influenciada pelos tipos de teste e tempo gasto para realizá-los. Esta atividade depende fundamentalmente que o ambiente de testes represente fielmente o ambiente de operação, o que implica num cuidadoso planejamento, envolvendo a estimativa dos recursos necessários

para a realização dos testes e o custo associado a este planejamento [15].

Modelos estatísticos vêm sendo utilizados largamente em indústrias fabris há muitos anos como uma poderosa ferramenta para o controle da qualidade. Ainda não têm sido usados na indústria do software por que sua utilidade não tem sido bem compreendida.

Estatisticamente, testar um programa é equivalente a determinar a proporção de saídas defeituosas no total de saídas que o programa pode gerar. O número total de saídas, de qualquer programa não trivial, vai de um número muito grande até infinito. Tais resultados formam uma população infinita, que é a base do estudo estatístico. As maiores vantagens em usar o método estatístico são as seguintes:

a) Níveis de confiabilidade numéricos, tais como 95%, podem ser impostos a fim de que, quando tal nível for atingido, a atividade de teste possa ser dada como concluída. Este nível de confiabilidade depende da disponibilidade de recursos. Quanto maiores, mais alto o nível.

b) É eficiente e poderoso. 5% dos esforços em se testar podem remover 95% dos erros de um programa (pode-se necessitar 95% de esforços em teste para remover os restantes 5%).

c) É aplicável a programas de diferentes tipos, sejam eles científicos, de tempo real, comerciais, grandes ou pequenos.

d) Em muitas aplicações, o teste pode ser completamente automatizado, da geração dos casos de teste até a análise dos resultados.

Segundo Myers [43], de todos os problemas que envolvem o gerenciamento de um projeto, talvez o mais significativo deles seja o que se refere com o controle do número de erros remanescentes em um programa. Este dado não só seria uma razoável ajuda para se determinar quando devemos parar a execução dos testes, mas também uma facilidade para se estimar os custos de manutenção depois que o programa for colocado em operação, estimar a sua confiabilidade e o tempo médio até falhar.

Devido a quase inexistência de formas de medidas que possam nos informar acerca da evolução do projeto, torna-se difícil estimar seu estado e quanto tempo mais será preciso para terminar o trabalho. Entretanto, alguns métodos podem ser utilizados para resolver este problema: uma forma é recorrer ao número de linhas de código escritas e estimar o número total de linhas de código no final do projeto;.outra forma é utilizar o número de linhas de código corrigidas e por último, utilizar o número total de erros removidos do programa [46].

A seguir, temos um exemplo dos três métodos.

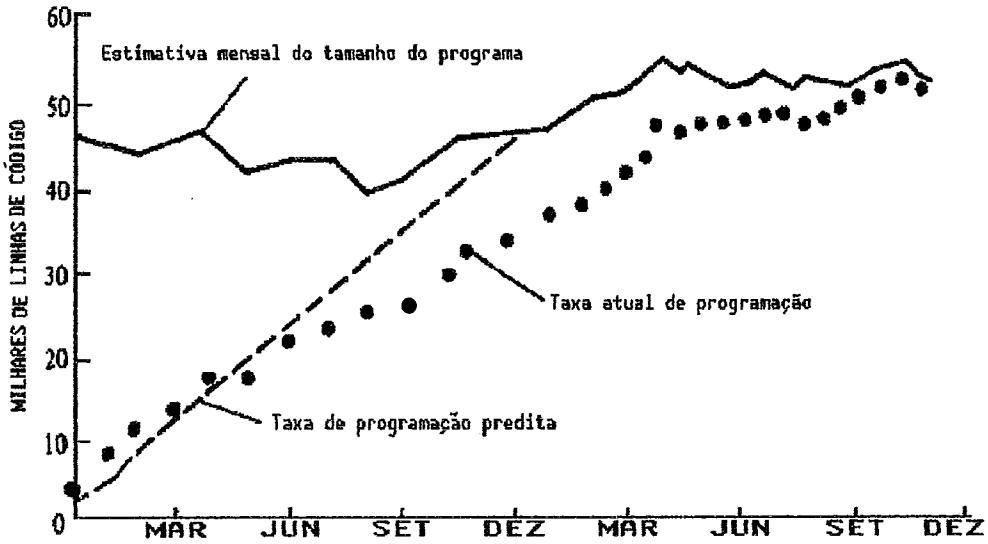


Fig. II.9

Estimativa do número total de linhas de código no final do projeto ao longo do tempo

Fonte: [46]

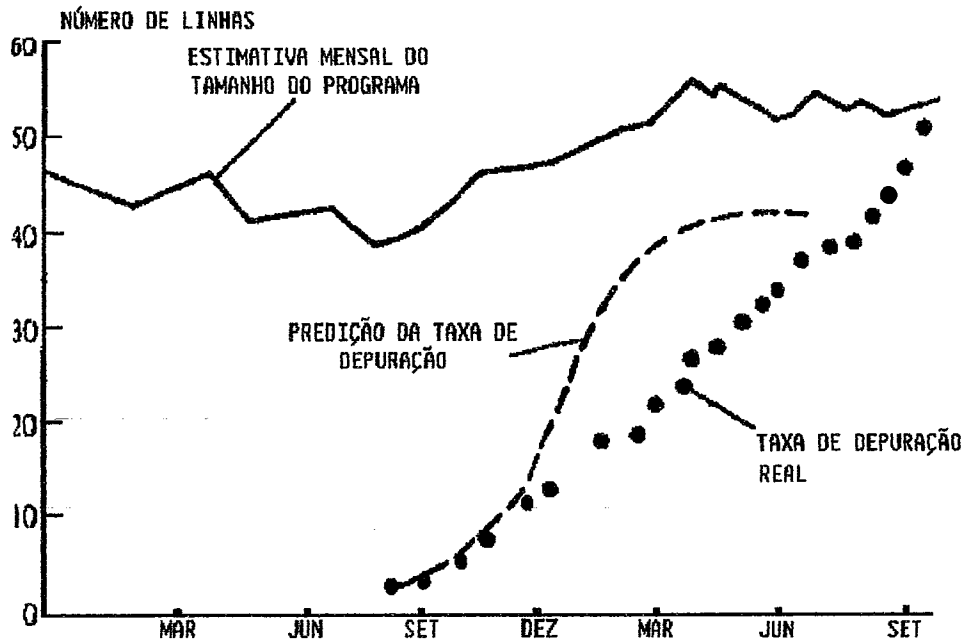


Fig. II.10

Estimativa do número acumulado de linhas de código corrigidas ao longo do tempo

Fonte: [46]

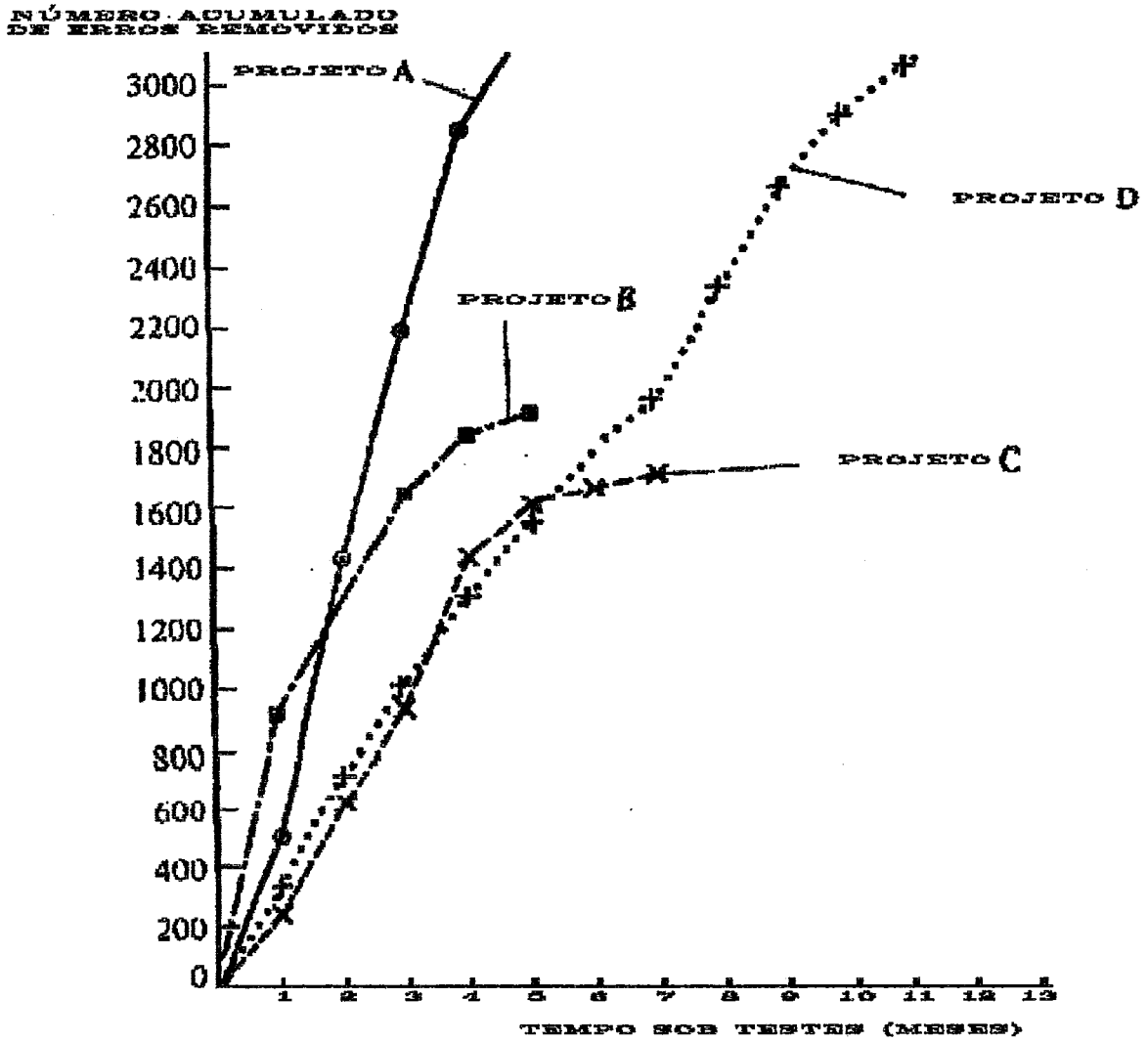


Fig. II.11

Número acumulado de erros removidos ao longo do tempo

Fonte: [46]

Uma vantagem clara do terceiro método é que, a medida em que os erros são removidos, a curva de estimativa começa a se aproximar da assíntota horizontal (estimativa do número total de erros do programa). Evidentemente tal assíntota não representa a remoção de todos os erros e sim dos erros que podem ser detectados com os métodos de teste utilizados.

Para estimar o número de erros no programa pode-se, primeiramente, compará-lo com programas similares já desenvolvidos e supor que o número de erros é aproximadamente igual. A forma da curva de detecção de erros pode, também, mostrar se o projeto será ou não bem sucedido (Figura II.12).

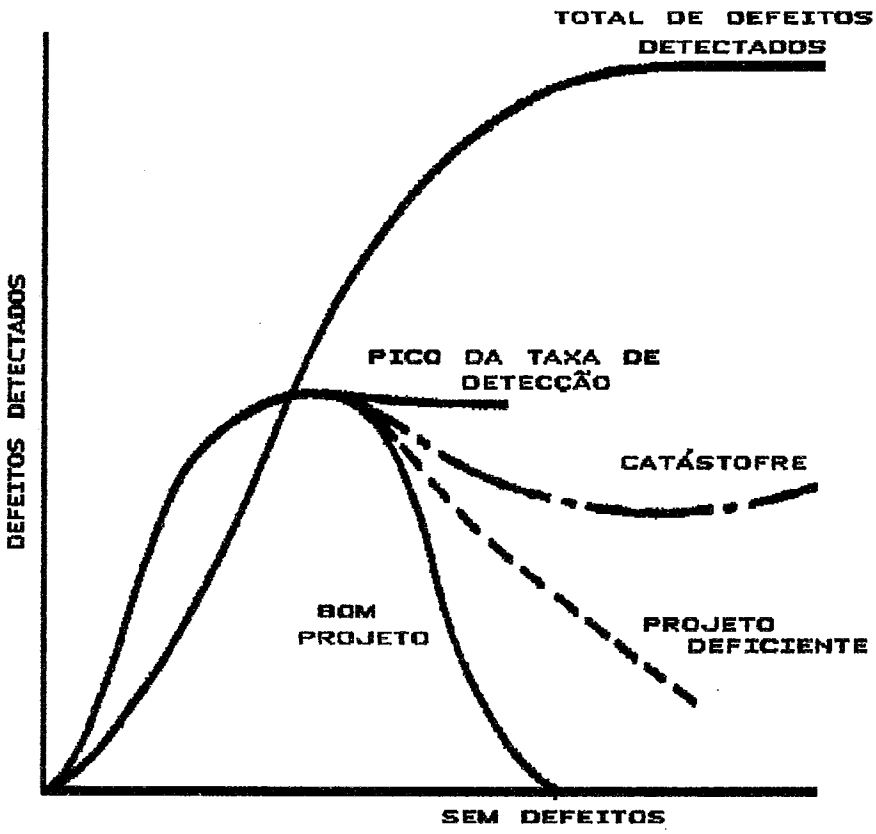


Fig. II.12

Taxa de detecção de erros ao longo do tempo

Fonte: [48]

Outra forma de avaliar se o projeto será bem sucedido é através do estudo da curva de crescimento da confiabilidade. A Figura II.13 apresenta três exemplos, caracterizando diferentes comportamentos em termos de tendência a maturação.

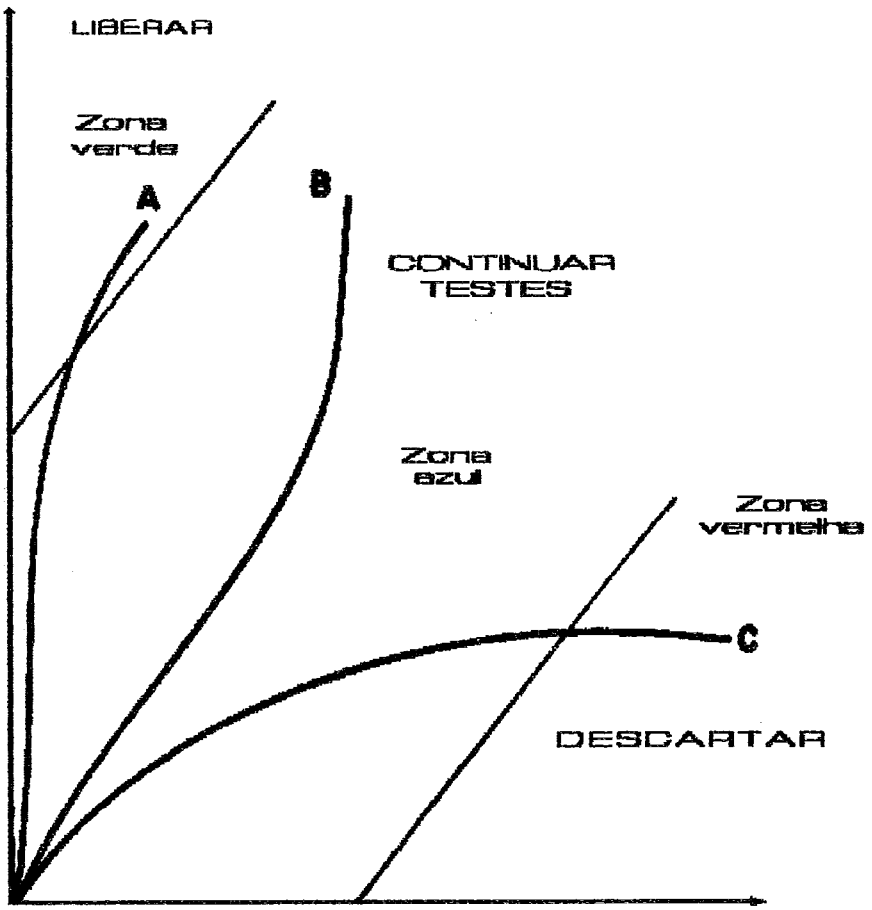


Fig. II.13

Curva de crescimento da confiabilidade

Fonte: [47]

Desde o início dos anos 70 vários modelos matemáticos para o cálculo da confiabilidade de software vêm sendo propostos. Musa [15] descreve tais modelos da seguinte maneira: "Modelos de confiabilidade de software são utilizados para caracterizar e predizer o comportamento futuro do produto para gerentes e engenheiros". Portanto, a estimação de confiabilidade tende a se tornar um poderoso instrumento tanto para a gerência quanto para os projetistas de software.

Antes que um dado modelo seja escolhido, deve haver uma perfeita compreensão dos fundamentos básicos, termos e conceitos utilizados em cada um dos modelos de confiabilidade propostos na literatura. Esta experiência é fundamental para que decisões inteligentes possam ser tomadas em aplicações práticas. Por exemplo, qual o modelo a ser utilizado num sistema distribuído ou baseado em multiprocessadores? Um estudo detalhado sobre os modelos de confiabilidade de software e suas aplicações pode ser encontrado em [15].

O modelo de estimação da confiabilidade escolhido tem como função, a partir das informações de falhas, ajudar a responder a três questões gerenciais relacionadas com o "status" do projeto e seu cronograma:

a) O programa está pronto para ser liberado?

b) Quando o programa estará pronto para ser liberado?

c) Devemos regressar a uma versão anterior (se a atual não corresponder às expectativas)?

A escolha do modelo depende de vários fatores. Musa recomenda que sejam utilizados os seguintes critérios de seleção:

Validade preditiva- refere-se à capacidade do modelo em predizer o comportamento futuro das falhas, a partir dos comportamentos presente e passado.

Capacidade- refere-se à habilidade do modelo em estimar com acurácia satisfatória a confiabilidade atual, a data em que a confiabilidade especificada será alcançada e os requisitos de custos, recursos humanos e materiais necessários a fim de que o objetivo seja alcançado.

Qualidade das hipóteses- refere-se à capacidade em que as hipóteses sejam confirmadas através dos dados obtidos, caso as mesmas possam ser testadas. Caso contrário, as hipóteses devem ser dadas como plausíveis ou não sob o ponto de vista de sua consistência com relação aos conceitos da Engenharia de Software.

Aplicabilidade- refere-se à avaliação do modelo segundo o grau de aplicabilidade em relação a diferentes produtos de software, que variam em tamanho, estrutura e funcionalidade. É desejável que o modelo possa ser aplicado em diferentes ambientes de desenvolvimento e operação, além de poder ser utilizado em diferentes fases do ciclo de vida.

Simplicidade- o modelo deve ser simples com relação a três aspectos:

a) de baixo custo para coleta de dados necessários a atender as particularidades do modelo;

b) a nível conceitual, de forma que os projetistas de software não necessitem de um profundo conhecimento matemático para compreender o modelo e suas hipóteses;

c) fácil no sentido de poder ser implementado como um programa automatizado de maneira a se tornar uma ferramenta gerencial e de engenharia.

A fim de modelar a estimação da confiabilidade de software, primeiro devemos considerar os principais fatores que a afetam: geração da falha, remoção da falha e o ambiente. A geração da falha depende fundamentalmente das características de codificação do programa (por exemplo o tamanho) e das características do processo de desenvolvimento tais como, tecnologia, ferramentas utilizadas, nível de experiência da equipe de projeto, etc.

É importante observar que o código pode ser desenvolvido para adicionar novas facilidades ou remover falhas. A remoção de falhas depende do tempo, perfil operacional e qualidade da atividade de reparo. O ambiente depende do perfil operacional. Desde que alguns dos fatores são probabilísticos por natureza e dependem do tempo, os modelos de confiabilidade podem ser caracterizados como processos aleatórios. Tais modelos podem ser divididos em duas grandes categorias:

a) Modelos que são função do tempo cronológico;

b) Modelos que são função do tempo de processamento real (tempo de execução da UCP).

Musa advoga que os modelos baseados no tempo de processamento mostram melhores resultados. Além das duas categorias mencionadas, podemos, também, encontrar na literatura modelos derivados da confiabilidade de hardware e modelos baseados na disseminação de erros. O primeiro trabalha sobre as seguintes hipóteses:

a) O tempo de depuração entre a ocorrência de erros tem uma distribuição exponencial com uma taxa de ocorrência de erros que é proporcional ao número de erros remanescentes.

b) Cada erro descoberto é imediatamente removido, diminuindo o número do total de erros de 1.

c) A taxa de falhas entre erros é constante.

Segundo Pressman [38] a validade destas hipóteses pode ser questionada. Correções realizadas podem, inadvertidamente, introduzir novos erros, invalidando a segunda hipótese.

A segunda categoria (disseminação) pode ser utilizada na prática como indicador do poder de detecção de erros de um conjunto de casos de testes. O programa sofre uma contaminação por um conhecido e calibrado conjunto de erros. Após a realização de testes, pode-se estimar a quantidade total de erros originais a partir da relação obtida entre o total de erros disseminados e os erros disseminados encontrados.

Segundo Musa [15] a maioria dos modelos para estimação da confiabilidade propostos na literatura se divide em sete grandes grupos - classe exponencial, classe de Weibull, classe Pareto, família geométrica, família linear inversa, família polinomial inversa e família potência. Os modelos pertencentes a estes grupos podem ser classificados em termos de cinco diferentes atributos:

a) Domínio do tempo- tempo de calendário ou tempo de execução (CPU).

b) Categoria (finita ou infinita)- O número de falhas que podem ser encontradas em um tempo infinito pode ser finito ou infinito.

c) Tipo- distribuição do número de falhas encontradas em um tempo t.

d) Classe (somente categoria finita de falhas)- forma funcional da intensidade de falhas em termos do tempo.

e) Família (somente categoria infinita de falhas)- forma funcional da intensidade de falhas em termos do número esperado de falhas encontradas.

O esquema de classificação adotado e apresentado a seguir foi desenvolvido por Musa e Okumoto [93] para os modelos de confiabilidade de software. A adoção deste esquema para a classificação dos modelos foi escolhido por poder diferenciar

duas categorias diferentes e pela melhor simplicidade analítica e significado físico.

As tabelas II.9 e II.10 apresentam o esquema de classificação proposto.

Classe	Tipo		
	Poisson	Binomial	Outros tipos
Exponencial	Musa [34] Moranda [52] Schneidewind [77] Goel-Okumoto [78]	Jelinsk-Moranda [76] Shooman [54]	Goel-Okumoto [59] Musa [79] Keiller-Littlewood [80]
Weibull		Schick-Wolverton [81] Wagoner [73]	
C1		Schick-Wolverton [49]	
Pareto		Littlewood [82]	
Gamma	Yamada-Ohba-Osaki [83]		

Tab. II.9

Modelos de categoria finita de falhas

Fonte: [15]

Como pode ser visto, a maioria dos modelos propostos na literatura se baseiam na categoria de falhas finita. Poisson e Binomial são os mais importantes tipos. As referências apresentadas junto aos modelos fornecem uma descrição detalhada dos mesmos. Os vazios encontrados nas Tabelas II.9 e II.10 não implicam em que no futuro tais modelos possam ser desenvolvidos, já que algumas combinações de família e tipo ou classe e tipo podem ser impossíveis.

Classe	Tipo			
	T1	T2	T3	Poisson
Geométrico	Moranda [52]			Musa-Okumoto [84]
Linear inversa		Littlewood-Verrall [61]		
Polinomial inversa (2º grau)			Littlewood-Verrall [61]	
Potência				Crow [85]

Tab. II.10

Modelos de categoria infinita de falhas

Fonte: [15]

A seguir segue uma descrição resumida dos principais modelos de estimação da confiabilidade propostos na literatura.

2.4.1- Modelo de disseminação [46] [72]

Este modelo é um melhoramento da técnica usada para estimar o número de indivíduos em uma população. Dada uma população N com características definidas e outra N_t diferente, se adicionarmos N_t em N e assumirmos que não existe diferença na dispersão, uma amostra retirada da população N_t+N, deverá conter, na mesma proporção, n_t+n. Logo, podemos concluir que:

$$\frac{N_t}{N+N_t} = \frac{n_t}{n+n_t}$$

A equação acima pode ser resolvida para uma estimativa de N em termos de quantidades conhecidas de N_t, n e n_t.

$$\hat{N} = \frac{n}{n_t} N_t$$

Por analogia podemos considerar:

N= número de erros no início do processo de correção (desconhecido)

$N_S = N_t$ = número de erros disseminados (desconhecido para o depurador)

n = número de erros não disseminados encontrados após um período de correção τ

$n_t = n_s$ = número de erros disseminados encontrados após um período de correção τ

Logo:

$$\hat{N} = \frac{n}{n_s} N_s$$

A maior dificuldade deste método e a criação de erros reais a serem disseminados no programa. Uma nova técnica baseada na anterior foi proposta por Hyman [72], e baseia-se no trabalho de dois programadores trabalhando independentemente na correção de erros e sendo avaliados depois de algumas semanas por um terceiro programador.

Por analogia com a equação $\hat{N} = \frac{n}{n_s} N_s$, temos:

$$\hat{B}_o = \frac{B_2}{b_c} B_1$$

onde,

τ = tempo de desenvolvimento em meses

B_o = número de erros no programa ($\tau = 0$)

B_1 = número de erros achados no programa pelo programador 1 no intervalo $0 < \tau < \tau_1$

b_c = número de erros que o programador 2 achou no mesmo intervalo anterior e comuns aos achados pelo programador 1

b_1 = número de erros independentes achados pelo programador 2 em relação ao programador 1

$B_2 = b_c + b_1$ = número de erros achados pelo programador 2

A variância de B_o é:

$$\hat{\text{Var}}\ B_o = \frac{B_o^2}{b_c}$$

Apos a realização de transformações baseadas na teoria estatistica, obtemos:

$$\hat{\text{Var}}\ B_o = \frac{B_1^2 B_2^2}{b_c^3}$$

Como exemplo, suponhamos que existam 100 erros em um programa, com o programador 1 achando 55 e o programador 2 achando 25, sendo que 10 erros são comuns.

Então, aplicando as fórmulas anteriores obtemos uma estimativa de um total de 139 erros no programa, o que caracteriza uma acurácia de 39%.

2.4.2- Modelo de Schick-Wolverton [49] [50] [51]

Este modelo calcula a estimativa de confiabilidade e a estimativa do tempo total necessário para achar todos os erros restantes. Baseia-se na seguinte função Hazard:

$$z(t_i) = \phi\ [N-(i-1)]t_i$$

onde,

ϕ = uma constante de proporcionalidade que representa a área sobre a curva de probabilidade.

N = número de erros iniciais presentes no programa.

t_i = i-ésimo intervalo de tempo de correção, ou seja, o intervalo entre o i-ésimo e o i-ésimo - 1 erro descoberto.

As hipóteses do modelo são:

a) O tempo total gasto durante a depuração do programa tem uma distribuição de Rayleigh;

b) A taxa de falhas é proporcional ao número de defeitos remanescentes e ao tempo gasto na depuração do programa;

c) Cada defeito descoberto é imediatamente corrigido, portanto reduzindo o número total de defeitos de um.

Confiabilidade:

$$R(t_i) = \exp\{-\phi\ [N-(i-1)]\ \frac{t_i^2}{2}\}$$

Tempo total necessário para achar todos os defeitos restantes:

$$\hat{T} = \frac{\left[\sum_{i=1}^{N-n} \frac{1}{i}\right]}{\phi}$$

onde n é o número acumulado de erros achados até esta data.

Desvio padrão:

$$\hat{\delta} = \frac{\left[\sum_{i=1}^{N-n} \frac{1}{i^2} \right]^{1/2}}{\phi}$$

MTTF ("Meam time to failure" - tempo médio até falhar):

Obs. Ver item 2.4.21)

$$\left[\pi / 2\phi \, (N-n) \right]^{1/2}$$

2.4.3- Modelo modificado de Schick-Wolverton [50] [51]

Segue o mesmo princípio do modelo anterior, exceto que a taxa de defeitos descobertos é constante durante o intervalo de tempo e é proporcional ao número de erros remanescentes e posteriores ao i-ésimo intervalo de tempo e ao total de tempo previamente gasto em testes.

Função Hazard:

$$z(t_i) = \phi \, (N-n_{i-1}) \, (T_{i-1} + t_i/2)$$

onde,

n_{i-1} = número acumulado de erros observados durante o i-ésimo - 1 intervalo de tempo.

T_{i-1} = tempo acumulado gasto para a correção de erros durante o i- ésimo intervalo.

Os restantes parâmetros seguem a mesma definição do modelo anterior.

Confiabilidade:

$$R(t_i) = \exp\ [-\phi\ (N-n)\ (T_{i-1}\ t_i + t_i^2/4)]$$

2.4.4- Modelo de Poisson geométrico Jelinski-Moranda [52] [53]

Este modelo utiliza as informações relativas às falhas de software de uma forma sumária (número de erros detectados em um período fixo de tempo), sendo que a taxa de detecção é assumida como constante durante um período fixo de tempo.

Função Hazard:

$$z(t_i) = \lambda k^{i-1}$$

2.4.5- Modelo De-Eutroficação Jelinski-Moranda [50] [51] [52] [53]

Este modelo calcula uma estimativa de confiabilidade e do menor tempo até falhar. Ele assume que existem N defeitos no programa no início dos testes, cada um deles independente dos outros e com a mesma probabilidade de causar uma falha durante os testes.

Função Hazard:

$$Z(t_i) = \phi\ [N-(i-1)]$$

onde,

N = número total de erros iniciais

ϕ = constante de proporcionalidade

t_i = i-ésimo intervalo de tempo de depuração

Confiabilidade:

$$R(t_i) = \exp\ [-\phi\ (N-n)t_i]$$

onde,

n = número de erros achados até o presente momento

MTTF:

$$1\ /\ \phi\ (N-n)$$

Este modelo fornece também, uma estimativa para o tempo total necessário para achar todos os erros remanescentes (TDRE) e o desvio padrão desta estimativa (STD).

$$TDRE = \left[\sum_{i=1}^{N-n} \frac{1}{i} \right] / \phi$$

$$STD = \left[\sum_{i=1}^{N-n} \frac{1}{i^2} \right]^{½} / \phi$$

2.4.6- Modelo Geométrico De-Eutroficação Jelinski-Moranda [50] [51] [52] [53]

Similar ao modelo descrito anteriormente, entretanto, assume que a taxa de falhas entre erros sucessivos forma uma progressão geométrica e que o número inicial de erros é infinito.

Função Hazard:

$$Z(t_i) = DK^{i-1}$$

onde, D e K são constantes calculadas com relação à taxa de detecção de erros.

Confiabilidade:

$$R(ti) = \exp\ (-DK^n\ t_i)$$

MTTF:

$$1\ /\ DK^n$$

2.4.7- Modelo Geométrico modificado De-Eutroficação [52] [53]

Este modelo é uma modificação do modelo anterior e assume que mais do que um erro pode ser achado durante um determinado período de correção.

Função Hazard:

$$Z(ti) = DK^{n_i-1}$$

onde,

n_i = número acumulado de erros achados no i-ésimo intervalo de tempo

Confiabilidade:

$$R(ti) = \exp\ [-DK\ m_{ti}^{n}]$$

2.4.8- Modelo de crescimento da confiabilidade [50] [51]

Estima a confiabilidade baseando-se no sucesso/falha dos dados coletados em uma seqüência de estágios de teste.

Confiabilidade:

$$R(k) = R(u) - A/k$$

onde,

R(k) = confiabilidade durante k-ésimo estágio
R(u) = R(k) quando k tende ao infinito
A = parâmetro de crescimento

R(u) e A são calculados usando a estimativa de mínimos quadrados.

$$\sum_{k=1}^{N} \left[\frac{S(k)}{n(k)} - R(u) + \frac{A}{R} \right] = 0$$

$$\sum_{k=1}^{N} \left[\frac{S(k)}{n(k)} - R(u) + \frac{A}{R} \right] \frac{1}{k} = 0$$

onde,

S(k) = número de sucessos para o k-ésimo grupo

n(k) = número de testes para o k-ésimo grupo

N = número total de estágios.

2.4.9- Modelo exponencial de Shooman [50] [51] [54] [55] [56] [57]

Este modelo assume que o tempo até falhar tem uma distribuição de probabilidade da forma:

$$f(t) = \lambda \exp [-\lambda t]$$

onde t é o tempo de operação do sistema, medido desde a ativação inicial e λ (uma constante) é a forma assumida para a função Hazard z(t). O número total de erros remanescentes no programa depois de um tempo de depuração "τ" e normalizados com relação ao número total de instruções em linguagem de máquina é:

$$e_r(\tau) = (E/I) - e_c(\tau)$$

Erros acumulados e taxa de erros:

Se E_r (t) representa o número de erros removidos no intervalo 0 a τ, temos:

Obs. $r(\tau)$ = taxa de erros removidos

$p(\tau)$ = taxa de erros normalizada

I_T = número total de instruções em linguagem de máquina

τ = meses gastos na correção de erros

$$r(\tau) = \frac{dE_r(\tau)}{d\tau}$$

$$p(\tau) = \frac{r(\tau)}{I_T}$$

Na Figura II.14 podemos ver quatro exemplos em termos de taxa de erros normalizada.

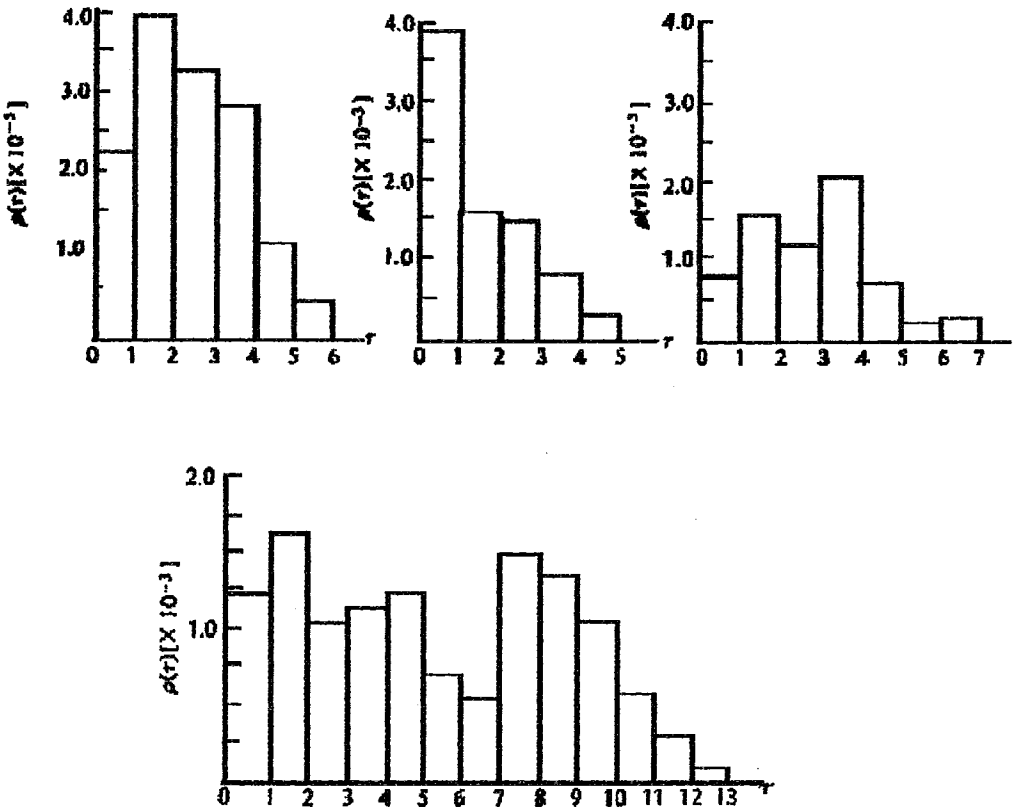


Fig. II.14

Exemplos de taxa de erros normalizada

Fonte [46]

Desde que estamos mais interessados num total normalizado de erros removidos do que no número atual $E_T(\tau)$, definimos então:

e (τ) = curva normalizada de erros acumulados, que é a área sobre a curva p (τ). Logo:

$$\frac{E\ (\tau)}{I_T} = e(\tau) = \int_0^{\tau} p\ (x)\ dx$$

Abaixo temos um exemplo da curva e(τ):

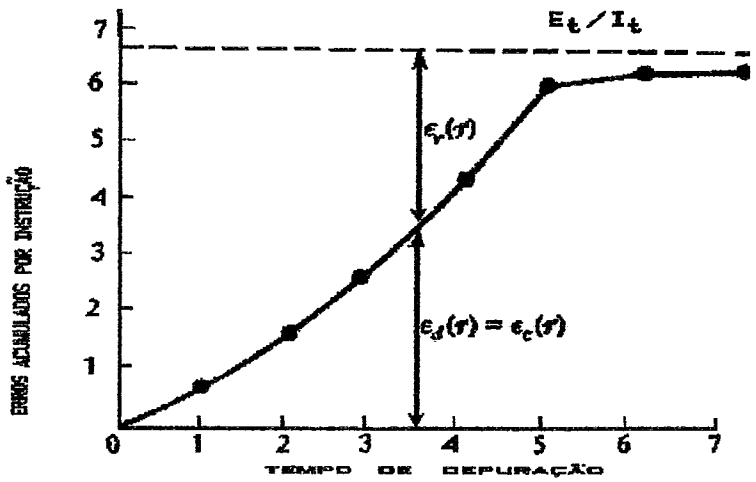


Fig. II.15

Curva e(τ)

Fonte: [46]

Modelo exponencial de erros removidos:

Assumindo que a taxa de detecção de erros mostrada a seguir:

$$\frac{dE_d(\tau)}{d(\tau)}$$

é em qualquer período, proporcional ao número de erros restantes no programa ($E_r(\tau)$), e que todos os erros detectados são imediatamente corrigidos, então:

$$E_c(\tau) = E_d(\tau)$$

Se o número de erros gerados ($E_g(\tau)$) durante o processo de correção for zero, o que resta é a quantidade de erros no início menos a quantidade que foi removida. Logo:

$$E_r(\tau) = E_T - E_c(\tau)$$

Dividindo todos os termos por I_T, temos;

$$\frac{E_r(\tau)}{I_T} = \frac{E_T}{I_T} - \frac{E_c(\tau)}{I_T}$$

$$e_r(\tau) = \frac{E_T}{I_T} - e_c(\tau)$$

$$e_d(\tau) = e_c(\tau)$$

A interpretação desta equação e mostrada na Figura II.14. Mas:

$$e_c(\tau) < \frac{E_T}{I_T}$$

$$ec(\tau) > 0$$

Relacionando a taxa de detecção de erros com o número de erros residuais, obtemos a taxa de erro exponencial:

$$\frac{de_d(\tau)}{d\tau} = \frac{de_c(\tau)}{d\tau} = \frac{k_1 E_T}{I_T} \exp(-k_1\tau)$$

Como podemos ver na Figura II.14, muitas curvas de taxa de correção mostram um inicial incremento nos primeiros meses e, em alguns casos, um "plateau" no meio do tempo de correção.

2.4.10- Modelo de Shooman e Natarajan [58]

Neste caso os modelos de confiabilidade de Shooman e de Jelinski-Moranda são estendidos para considerar a possibilidade de geração de erros durante o processo de correção. A equação básica é:

$$\frac{dn}{dt} = r_g(t) - r_c(t)$$

onde n(t) é o número de erros, $r_g(t)$ a taxa de geração de erros, e $r_c(t)$ a taxa de correção, como função do tempo de depuração t.

2.4.11- Modelo de Weibull [50] [51] [73]

Estima a confiabilidade e o menor tempo até falhar utilizando a função Hazard abaixo:

$$Z(t) = (n/\delta)\ (t/\delta)^{n-1}$$

onde,

n = parâmetro de forma

δ = parâmetro de escala

t = tempo

Confiabilidade:

$$R(t) = \exp\ [-(t/\delta)^{n}]$$

MTTF:

$$= [\delta\Gamma(1/n)]n$$

onde Γ é a função gamma

2.4.12- Modelo Bayesiano de Goel e Okumoto [59]

Um modelo estocástico é desenvolvido para falhas de software onde a remoção de erros não pode ser feita com eficácia. Fórmulas para o cálculo da confiabilidade e diversas outras métricas são apresentadas na referência [59].

Função Hazard:

$$Z(t_1) = [N-p(i-1)]\lambda$$

pdf (função de distribuição da probabilidade) do tempo até a próxima falha:

$$f_i(t) = i\lambda e^{-i\lambda t}$$

cdf (função de distribuição cumulativa):

$$F_i(t) = 1-e^{-i\lambda t}$$

Taxa de falhas:

$$r_k(x) = \phi_k(x) \ / \ R_k(x)$$

2.4.13- Modelo Bayesiano de Littlewood e Verrall [60] [61]

Este modelo é aplicável durante as fases de teste e operação. Uma regra de reparo é apresentada de tal forma que as medidas de confiabilidade podem ser desenvolvidas levando em conta as incertezas inerentes ao processo de reparo. Segundo este modelo os tempos entre falhas seguem uma distribuição exponencial, mas o parâmetro de sua distribuição é tratado como variável aleatória com uma distribuição gamma.

Função de Distribuição da Probabilidade:

$$\frac{n\, y^n}{t+\Psi(n)} \left[y + \log\left(\frac{t + \Psi(n)}{\Psi(n)} \right) \right]^{-(n+1)}$$

2.4.14- Modelo de Shooman e Trivedi-Markov [62] [63]

Um modelo markoviano é proposto com a função de estimar e predizer o número mais provável de erros que irão ser corrigidos como função do tempo.

Disponibilidade:

$$A(t) = \sum_{k=0}^{\infty} P_k(t)$$

onde,

$P_k(t)$ = probabilidade de que exatamente k erros tenham sido descobertos e corrigidos num periodo t.

Confiabilidade:

$$R_k(\tau) = e^{-\lambda(k)\tau} \quad 0 \leq \tau \leq T_{k+1}$$

onde,

$\lambda(k)$ = taxa de falhas (constante) depois da correção do k-ésimo erro mas antes da ocorrência do (k+1)-ésimo erro no tempo T_{k+1}.

2.4.15- Modelo Markoviano de Littlewood [64]

Neste modelo considera-se que o funcionamento do sistema se realiza através de transições que tomam lugar entre os subsistemas de acordo com um homogêneo e contínuo (no tempo) processo markoviano de estados finitos. Nele é assumido que as falhas ocorrem nas subunidades de acordo com um processo de Poisson em que as falhas podem ocorrer nas transições entre as subunidades de acordo com uma distribuição discreta e independente no tempo.

Valor estimado do tempo de espera para o próximo evento:

$$E(t) = 1 \; / \; \Sigma \|_i \; (\; vi + \Sigma_{j<>i} \; q_{ij}\lambda_{ij})$$

2.4.16- Modelo semi-Markoviano de Littlewood [65]

Este modelo assume que o programa pode ser dividido em R subprogramas e que o controle dinâmico é exercido pela transição entre eles de acordo com um processo semi-markoviano no tempo (contínuo) real. A descrição dos parâmetros e das fórmulas pode ser encontrada na referência [65].

2.4.17- Modelo de Moranda [66]

No artigo que descreve este modelo o autor propõe uma metodologia para a determinação de requisitos de teste e procedimentos para realizá-los. Também é descrito como efetuar a "instrumentação" do programa a fim de obter dados para análise.

Erro médio operacional:

$$M = 1 / 50 \left[\sum_{i=1}^{m} N_i / m \right]$$

onde,

M = Número de execuções

1 / 50 = Parâmetro de Poisson dos programadores - uma constante universal (um erro por 50 linhas de código)

N_i = Número de instruções executadas na i-ésima execução

2.4.18- Micromodelo de Shooman [67]

Um modelo é desenvolvido a partir de medidas feitas em projetos de software, onde a taxa de falhas de um grande sistema pode ser relacionada com o tempo de execução dos caminhos, frequência relativa da execução dos caminhos e probabilidade de falhas ao longo de cada caminho.

Estimativa da taxa de falhas:

$$Z_o = \frac{\sum_{j=1}^{i} f_j \, q_j}{\sum_{j=1}^{i} f_j \, (1 - q_j/2) \, t_j}$$

onde,

f_j = frequência com que o j-ésimo caminho é executado

t_j = tempo de execução de cada caminho

q_j = probabilidade de erro ao longo de cada caminho.

2.4.19- Modelo de Nelson [68] [69] [70]

Este modelo utiliza inferências referentes a confiabilidade baseada na execução correta das entradas escolhidas da totalidade de todas as entradas por uma função de distribuição chamada "perfil operacional".

Confiabilidade:

$$R = 1 - N_e / N$$

onde,

N_e = número de falhas em N tentativas.

2.4.20- Modelo de Hecht [71]

Este modelo é aplicável às atividades de estimação e medição da confiabilidade.

a) Medição:

R = S / N -> confiabilidade

R' = S / (N.L) -> confiabilidade normalizada pelo tamanho do programa

R" = S / (N.L.W) -> confiabilidade normalizada pelo tamanho do programa e tamanho da palavra

onde,

N = número total de percursos

S = número total de percursos bem-sucedidos

L = Tamanho do programa medido em instruções de máquina

W = Tamanho médio de bites por instrução

MTBF ("Mean time between failure"):

t / F (F=T-S)

b) Estimação

estimativa da não confiabilidade:

$U = \sum_k (F_k / N_k) P(Z_k)$ (modo discreto)

$U = \sum_k (F_k / t_k) P(Z_k)$ (modo continuo)

onde,

N_k = número de execuções utilizando o conjunto de dados Z_k.

F_k = número de falhas utilizando o conjunto de dados Z_k.

$P(Z_k)$ = probabilidade que o conjunto de dados Z_k seja executado no ambiente operacional.

t_k = tempo de processamento de Z_k.

Crescimento da confiabilidade:

$\hat{E}_o = U_o C / (U_o - U_1)$ -> estimativa inicial do número de erros do programa

$\hat{k}$ = estimativa da constante de proporcionalidade k

$U(t) = k E_o \exp(-kt)$

onde,

U_o = taxa de falhas no início dos testes.

U_1 = taxa de falhas depois de C erros terem sido removidos.

t = tempo total de operação do programa durante os testes.

2.4.21- Modelo Básico de Musa [34]

A confiabilidade do software é caracterizada pelo seu "mean-time-to-failure" (MTTF). O principal resultado é que o atual MTTF é uma função exponencial crescente do tempo de execução. Este modelo utiliza parâmetros derivados dos objetivos do projeto e dos recursos disponíveis.

MTTF:

$$T_0 \exp (C\tau / M_0 T_0)$$

onde,

T_0 = inicial MTTF antes dos testes. Estimado a partir da relação $T_0 = 1/ fKN_0$.

C = taxa de detecção de falhas durante os testes dividida pela taxa de detecção de falhas durante a execução.

M_0 = número de falhas necessárias para expor e remover N_0 erros inerentes. Estimada a partir da relação: $M_0 = N_0 / B$.

N_0 = número de erros inerentes no software, inicialmente o número de erros por instrução.

B = taxa de correção de erros dividido pela taxa de ocorrência de falhas.

f = taxa de execução de instruções dividida pelo número de instruções em linguagem assembly.

k = frequência dos erros dividido pela frequência da execução linear.

A estimativa de máxima verossimilhança é utilizada para re-estimar os valores dos parâmetros do programa, tendo por base os intervalos de tempo de execução entre falhas ocorridas durante os testes. A Figura II.16 ilustra conceitualmente este processo.

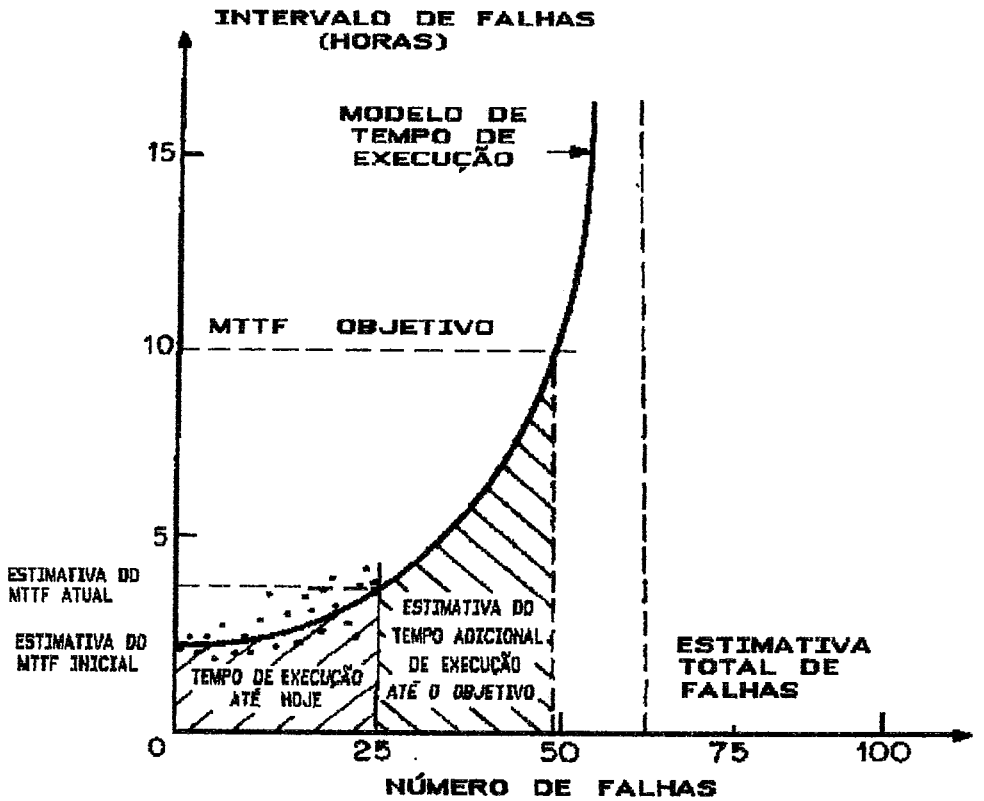


Fig. II.16

Reestimativa dos parâmetros do programa

Fonte: [34]

Confiabilidade:

$$R = \exp (-\tau'/T)$$

onde τ' é o tempo referente ao período de operação.

2.4.22- Modelo de Poisson Logaritimico de Musa-Okumoto [84]

Este modelo se baseia no tempo de execução do programa, tendo porém a capacidade de converter este tempo para o tempo cronológico. Quantidades expressas segundo um critério cronológico são mais significativas para engenheiros e gerentes. A conversão é realizada através de um segundo componente do modelo, que caracteriza de que forma os recursos humanos e computacionais são aplicados no projeto.

Intensidade de Falhas:

$$\lambda(\mu) = \lambda_0 \exp(-\theta\mu)$$

onde,

μ = número de falhas ocorridas.
λ_0 = intensidade de falhas inicial.
θ = parâmetro de decaimento.

Número de falhas após um periodo τ:

$$\mu(\tau) = (1/\theta) \ln(\lambda_0 \theta_\tau + 1)$$

Confiabilidade:

$$R(\tau) = \exp(-\lambda\tau)$$

Adotaremos os dois últimos modelos (itens 2.4.21 e 2.4.22) para a estimativa da confiabilidade, seguindo a proposta de

Musa [14]. O modelo básico será utilizado até a fase de testes de sistema (perfil operacional uniforme) e a partir da etapa de testes de sistema utilizaremos o modelo de Poisson Logarítimico (perfil operacional não uniforme). Este último modelo também será utilizado durante a atividade de medição da confiabilidade.

2.5- Medição da confiabilidade

Esta atividade tem a função de garantir que a confiabilidade alcançada nas fases anteriores seja mantida na fase operacional. Isto é realizado a partir da coleta de dados reais e não dados de teste. O intuito é mostrar objetivamente como o software desempenha suas funções no ambiente real de operação através da observação direta das falhas do produto. Os resultados encontrados a partir desta observação devem ser comparados com os resultados obtidos pela predição e estimação da confiabilidade, a fim de validar as medidas relacionadas com estas atividades. Portanto, a atividade de medição da confiabilidade tem a finalidade de responder a quatro questões fundamentais:

1- O valor predito e estimado para a confiabilidade está sendo verificado na fase operacional?

2- Que métodos e técnicas de desenvolvimento produzem melhores resultados em termos de incremento no nível de confiabilidade?

3- Quando uma nova versão do programa pode ser liberada para os usuários?

4- Com que velocidade as falhas que ocorrem no ambiente operacional são corrigidas e qual o custo associado à atividade de manutenção?

Garantir que as atividades de predição e estimação da confiabilidade antecipem com precisão o comportamento real do programa em seu ambiente de operação é fundamental para que ações corretivas possam ser tomadas com rapidez caso sejam necessárias, reduzindo assim, custos e aumentando a qualidade do programa.

Durante a operação do programa, falhas irão ocorrer e novas características (evoluções) serão adicionadas. Isto ocorre na maioria dos sistemas, independente de seu tamanho. Em geral, a introdução de melhoramentos e correção de falhas obedece a um cronograma pré-estabelecido para a liberação de novas versões. Como cada versão possui uma determinada intensidade de falhas, a intensidade de falhas do programa muda com a instalação de uma nova versão, o que em alguns casos pode gerar sérios conflitos. Um bom exemplo é a introdução de uma nova versão (com melhoramentos) de um sistema operacional, o que fatalmente diminuirá a confiabilidade, colocando em risco a atividade de produção. Para que conflitos deste tipo possam ser solucionados, torna-se necessário que a nova versão só seja introduzida quando a intensidade de falhas tenha atingido

um nível pré-determinado, mostrando assim a maturação do produto.

Nenhum método, seja ele qual for, pode ser considerado definitivo, ainda mais em se tratando de avaliar quantitativamente a qualidade do software durante todo o seu ciclo de vida, pois são inúmeras e de todos os tipos as variáveis que afetam esta qualidade, principalmente se levarmos em conta os inter-relacionamentos entre elas.

Informações detalhadas sobre a atividade do software durante sua fase operacional devem ser obtidas a fim de permitir que o método em uso possa ser corrigido ou aperfeiçoado no ponto em que se fizer necessário.

A abordagem dada à atividade de predição e estimação é marcada, como já vimos anteriormente, pelo paradigma científico de hipótese, avaliação, crítica e revisão. Sendo assim, é necessário validar os resultados destas atividades com base na comparação com medidas obtidas através da observação direta das falhas do programa. Caso tal comparação revele informações conflitantes, os processos de avaliação e o modelo de estimação utilizados deverão ser revistos e ajustados até que tais conflitos não mais ocorram.

Mendis [13] afirma que o principal requisito para operacionalizar a atividade de correção ou aperfeiçoamento do método é a implementação de um sistema de gerenciamento de falhas e um bem planejado mecanismo de suporte. As

ferramentas, técnicas e metodologias utilizadas pela organização devem ser capazes de implementar os requisitos especificados sem dificuldade. Opcionalmente pode-se utilizar um sistema automatizado que a organização já possua para relato de falhas ou de alterações de programa e que seja facilmente modificável.

2.5.1- Sistema de gerenciamento de falhas

Este sistema deve conter todas as informações básicas a respeito da descoberta e correção de erros. Também deve ser suficientemente completo para obter todos os dados pertinentes e necessários a uma posterior análise de erros e capaz de obter tais informações tão cedo quanto possível.

As Tabelas II.12 e II.13 apresentam um modelo de formulário para o gerenciamento de falhas de operação (relatório de investigação). Tal formulário tem o objetivo de reunir todas as informações pertinentes à atividade de manutenção. Estas atividades englobam o cadastramento da falha (incluindo as informações referentes a configuração de SW e HW), diagnóstico por parte do órgão encarregado da análise das falhas, correção do problema, validação da alteração efetuada e liberação da nova versão (corrigida) do programa.

```
PRESTIME-Predição, Estimação e Medição da Confiabilidade
                Sistema de Gerenciamento de Falhas
Emissão: 00001     Estado < VALIDAÇÃO >     Data:01/01/1990

Cadastramento | órgão: ...  | Razão:  [EVOLUÇÃO]  [FALHA]
                             | Prioridade:  [A]  [M]  [B]
Data de Entrada : .......    | Palavra-chave: ..............
Data de Detecção: .......    | Palavra-chave: ..............
Projetista: .............    | Palavra-chave: ..............
Cargo: .. Coordenador: ...   | Cliente: ....................
Módulo: .. Submódulo: ....   | Tempo gasto: ...dias  ...horas
Versão: .................    | Destinatário: ...............
Resumo do problema: ..........................................
.....................................................................
Configuração de HW ———————————— Configuração de SW
Fabricante: ..... Ano: ...   | Sistema operac.: .. Versão: ..
Modelo: ..... Revisão: ...   | Ling.  de prog.: .. Versão: ..
Processador: ...Clock: ...   | Drivers instalados: .........
Memoria: .. Terminais: ...   | ............................
Disp. de Armazenamento ...   | Programas residentes: ........
Potência de fonte: .......   | ............................
Versão de firmware: ......   | Obs.: .......................

Diagnóstico | órgão: .....  | Parecer: ....................
                             | ............................
Data de recebimento: .....   | ............................
Projetista: .............    | Tempo gasto:... dias ... horas
Cargo: .. Coordenador: ...   | Destinatário:................
Razão:[S] [H] [D] [O] [I]    | Altera testes:  [SIM]  [NÃO]
Impos.: [REPRO] [RECURSO]    | Programas: ..................

Solução | órgão: ........    | Módulo: ..... Submódulo: .....
                             | Fontes alterados: ...........
Término Previsto: ........   | ............................
Data de inicio: ..........   | Linhas alteradas: ...........
Data de término: .........   | Linhas removidas: ...........
Projetista: .............    | Linhas adicionadas: .........
Cargo: ... Coordenador: ..   | Versão implementada: ........
Destinatário: ............   | Altera documentação:[SIM][NÃO]
Observações: .............   | Manual alterado: ............
.........................    | Páginas alteradas: ..........

Validação | órgão: ...       | Tempo gasto: ...horas ...dias
                             | Data de término: ............
Data de recebimento: .....   | Previsão de liberação: .......
Projetista: .............    | Restrição:  [SIM]  [NÃO]
Cargo: ... Coordenador: ..   | Cliente: ....................
Resultado: [SIM][NÃO][INL]   | Observação: .................

Liberação | Gerência: ...    | Parecer: ....................
                             | ............................
Data de recebimento: .....   | Responsável:.................
```

Fig. II.17

Formulário de acompanhamento de falhas (1ª parte)

PRESTIME-Predição, Estimação e Medição da Confiabilidade
Sistema de Gerenciamento de Falhas de Operação
Emissão: 00001 Estado < VALIDAÇÃO > Data: 01/01/1990

Origem do Erro

AC - Alteração de Concepção
ARS- Alteração de Req. de Software
APS- Alteração de Proj. de Software
APP- Alteração de Plano de Proj.
EDC- Erro de Codificação/Digitação
FS - Falha secundária
FMO- Falha de Operação/Manutenção
EDC- Alteração de Documentação

[ ]-SUPORTE
[ ]-VALIDAÇÃO
[ ]-DESENVOLVIMENTO
[ ]-GERÊNCIA

Atributo de Qualidade Afetado

1 - Correção no uso da ling.
2 - Concisão
3 - Uniformidade
4 - Modularidade
5 - Disponibilidade
6 - Estrutura
7 - Rastreabilidade
8 - Modificabilidade
9 - Evolutibilidade
10 - Verificabilidade
11 - Validabilidade
12 - Viabilidade Econômica
13 - Viabilidade Financeira
14 - Viabilidade Técnica
15 - Viabilidade de Pessoal
16 - Viabilidade de Suporte
17 - Viabilidade de Cronograma
18 - Viabilidade Social
19 - Fidelidade
20 - Consistência
21 - Não ambigüidade
22 - Completude
23 - Necessidade
24 - Ser explícita
25 - Robustez
26 - Segurança
27 - Estilo
28 - Oportunidade
29 - Amenidade ao uso
30 - Portabilidade
31 - Eficiência
32 - Rentabilidade
33 - Reutilizabilidade
34 - Configurabilidade
35 - Treinamento
36 - Independência
37 - Compatibilidade

[ ]-SUPORTE [ ]-VALIDAÇÃO [ ]-DESENVOLVIMENTO [ ]-GERÊNCIA

Tipo do Erro

AA - Erro Computacional
BB - Erro Lógico
CC - Erro de Entrada de Dados
DD - Erro no Trat. dos Dados
EE - Erro de Saída de Dados
FF - Erro de Suporte do S.O.
GG - Erro de Configuração
HH - Erro na Def. de Variáveis
II - Erro Colateral
JJ - Erro de Documentação
KK - Alteração Sol. p/ usuário
LL - Erro na Interf. entre mod.
MM -Erro de Int. Mod./Sis.
NN -Erro de Int. c/usuário
OO -Erro de Int. c/ B.D.
PP -Erro no Banco de Dados
QQ -Erro de Aten. aos Req.
RR -Erro do Operador
SS -Erro de Hardware
TT -Erro de Interf. SW/HW
UU -Erro Não Identificado
VV -Erro Não Reproduzível
XX -Erro de Conformidade
ZZ -Outros: .....

[ ]-SUPORTE [ ]-VALIDAÇÃO [ ]-DESENVOLVIMENTO [ ]-GERÊNCIA

Fig. II.18

Formulário de acompanhamento de falhas (2ª parte)

Como podemos observar nas duas figuras anteriores o campo de "estado" do relatório de investigação mostra que tal relatório se encontra na fase de validação (uma das seis possíveis - cadastramento, diagnóstico, correção, validação, liberação e liberada). O relatório se compõe de nove blocos de informações, sendo que os três últimos blocos se referem a dados de controle (Figura II.16) e são preenchidos pelo suporte, pela validação, pelo desenvolvimento (manutenção) e, pela gerência de projeto caso haja conflito no preenchimento pelas três primeiras áreas.

As informações levantadas irão permitir que uma análise criteriosa dos erros possa ser realizada com vistas a corrigir ou aperfeiçoar o sistema de avaliação da confiabilidade no que concerne às atividades de predição e estimação.

A primeira tarefa para a quantificação dos erros é classificá-los com relação aos seus tipos e sua distribuição. Tal classificação e feita conforme as seguintes categorias:

Alteração de Concepção (AC): Uma modificação em algum dos objetivos do projeto causada por uma incorreta caracterização do problema a ser resolvido ou devido a novas necessidades dos usuários.

Alteração de Requisitos de Software (ARS): Uma modificação do programa causada por uma alteração no hardware ou num

requisito de software ou ainda se este opera de acordo com a documentação, embora exista uma deficiência no projeto.

Alteração de Projeto de Software (APS): Uma correção no programa causada por uma implementação incorreta da especificação de projeto.

Alteração de Plano do Projeto: Uma alteração no planejamento do projeto causada por problemas com o cronograma ou devido a inviabilidade de determinados recursos necessários à conclusão do projeto.

Erro de Digitação/Codificação (EDC): Um erro de programação introduzido como resultado da atividade de digitação, codificação ou manipulação do programa.

Falha Secundária (FS): Um efeito colateral causado por uma alteração feita no programa.

Falha de Manutenção/Operação (FMO): Um erro introduzido durante a atividade de manutenção ou causado por uma má utilização do programa pelos operadores.

Erro de Documentação (EDC): O software não opera de acordo com a documentação, mas funciona corretamente.

Outros (OTR): Novos requisitos ou aumento do escopo do problema que o software inicialmente pretendia resolver.

A Tabela II.11 mostra o resultado da análise de um programa real [13]. Os dados nas colunas representam o número de erros identificados através dos vários níveis de revisão do programa. Uma comparação dos resultados, mostra que existe um decréscimo substancial no número de erros ao longo das diversas revisões. Cabe observar que neste exemplo (descrito por Mendis em [13]) não são utilizadas as categorias "AC" (Alterações de Concepção) e "APP" (Alteração do Plano do Projeto).

CATEGORIA	REV 1	REV 2	REV 3	REV 4	REV 5	TOTAL
Alteração de Requisito de SW	26	12	6	5	7	56
Alteração de Projeto de SW	12	11	11	9	7	50
Erro de Digitação/Codificação	4	3	-	-	2	9
Falha de Manutenção/Operação	-	-	-	-	-	-
Erro de Documentação	5	-	-	-	2	7
Outros	2	1	-	-	3	6
Total	49	27	17	14	21	128

Tab. II.11

Erros de software por categoria

Fonte: [13]

Uma análise destes mesmos dados, em termos de percentual do total de erros para cada nível de revisão é mostrado na Tabela II.12:

CATEGORIA	REV 1	REV 2	REV 3	REV 4	REV 5	TOTAL
Alteração de Requisito de SW	53%	44%	35%	36%	33%	44%
Alteração de Projeto de SW	25%	41%	65%	64%	33%	39%
Erro de Digitação/Codificação	8%	11%	-	-	10%	9%
Falha de Manutenção/Operação	-	-	-	-	-	-
Erro de documentação	10%	-	-	-	10%	6%
Outros	4%	4%	-	-	14%	5%

Tab. II.12

Percentual de erros de software por categoria

Fonte: [13]

Como podemos notar existem duas categorias que se destacam: "Alteração de Requisitos de Software" e "Alteração de Projeto de Software", que representam juntas aproximadamente 83% dos erros identificados. A Figura II.19 mostra a relação entre os erros acumulados por período mensal e o tempo em meses a partir do início da operação do programa. A forma do gráfico mostra um gradual e constante decréscimo da descoberta de erros.

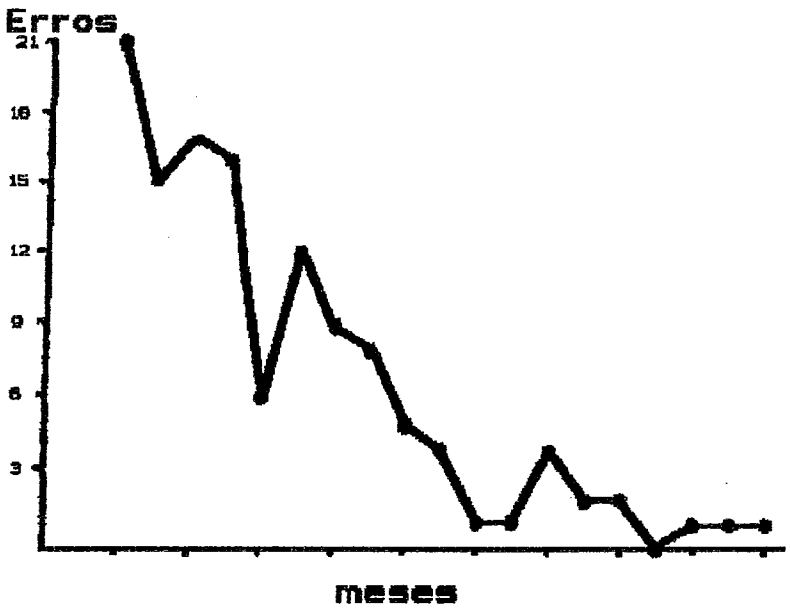


Fig. II.19

Erros acumulados por período mensal ao longo do tempo

Fonte: [13]

Os dados de erros por categoria mostrados na Tabela II.11 e na Figura II.20 permitem observar um gradual e constante decréscimo em todas as categorias identificadas.

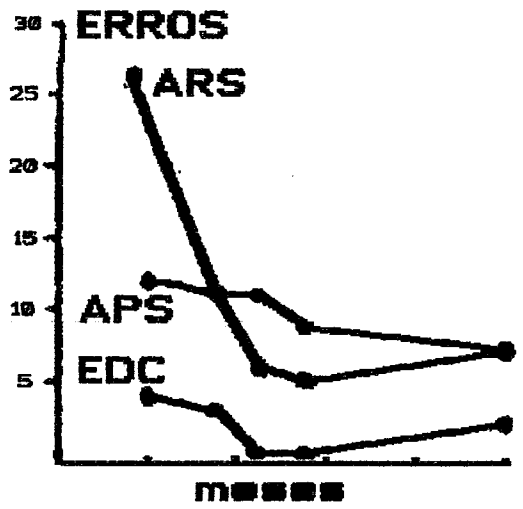


Fig. II.20

Número de erros ao longo do tempo

Fonte: [13]

A Figura II.21 mostra a relação entre a percentagem de erros e o tempo decorrido.

Vista sob esta perspectiva, a categoria "APS" apresenta um gradual incremento na taxa de erro até o 12º mês, ao mesmo tempo que os erros de digitação e codificação "EDC" se mantém variáveis embora reduzidos. Como era de se esperar a curva de erros "ARS" decresce e então se estabiliza com a maturação do projeto. O comportamento do programa confirma o que dele era esperado, ou seja, com a maturação do projeto os tipos de erros encontrados se deslocam da categoria "ARS" para "APS" e desta para a "EDC".

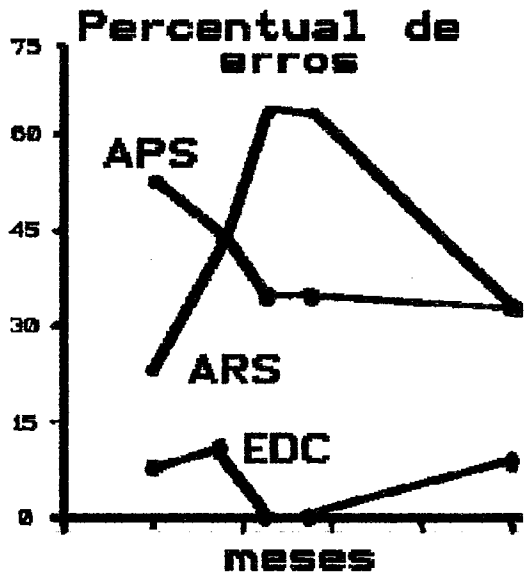


Fig. II.21

Percentual de erros ao longo do tempo

Fonte: [13]

Outro fator relevante é a determinação da densidade de erros, isto é, o número de erros de software por tamanho de programa. No caso do exemplo aqui estudado foram reportados 128 erros que deram origem ao mesmo número de alterações no programa. Dividindo estas alterações pelo tamanho do programa (28k) em linhas de instrução, obtemos o valor para a densidade de erro igual à 0.0046. Valores entre 0.005 e 0.02 são mencionados na literatura [34]. Torna-se necessário, ainda, que os tipos de erros também sejam analisados conforme a identificação mostrada na Tabela II.13:

TIPO DE ERRO	CATEGORIA
Erro computacional	AA
Erro lógico	BB
Erro de entrada de dados	CC
Erro no tratamento de dados	DD
Erro de saída de dados	EE
Erro de suporte do sistema operacional	FF
Erro de configuração	GG
Erro na definição de variáveis	HH
Erro recorrente (efeito colateral)	II
Erro de documentação	JJ
Alteração solicitada pelo usuário	KK
Erro de interface entre módulos	LL
Erro de interface módulo/sistema	MM
Erro de interface com o usuário	NN
Erro de interface como banco de dados	OO
Erro presente no banco de dados	PP
Erro de atendimento aos requisitos	QQ
Erro do operador	RR
Erro de hardware	SS
Erro de interface SW/HW	TT
Erro não identificado	UU
Erro não reproduzivel	VV
Erro de conformidade com o padrão	XX
Outros	ZZ

Tab. II.13

Identificação dos tipos de erro de software

Fonte: [13]

No caso do mesmo exemplo aqui estudado o erro mais frequente foi o erro lógico com cerca de 31% do total, vindo a seguir os erros computacionais com 17%, erros de documentação com 13% e alterações pedidas pelo usuário com aproximadamente 10%. Estas quatro categorias de erro respondem por 70% do total de erros reportados. Estas percentagens foram obtidas a partir da análise de todos os relatórios de falha. Os problemas encontrados podem, também, ser agrupados em três categorias de severidade, como é mostrado na Tabela II.14.

SEVERIDADE	No.	%
Baixa	67	52
Média	58	45
Alta	3	3
Total	128	100

Tab. II.14

Severidade dos erros de software

Fonte: [13]

Um problema considerado de baixa severidade não afeta o desempenho funcional do programa. Um problema considerado de média severidade afeta o desempenho funcional, embora não seja suficiente para requerer uma imediata correção. Um problema considerado de alta severidade interrompe a operação do programa provocando uma imediata ação corretiva. Esta e outras

informações, quando devidamente classificadas, permitem não somente avaliar a qualidade do desenvolvimento, como também identificar áreas potencialmente problemáticas.

Uma das medidas mais importantes é a estimativa do total de erros ainda restantes no programa. Segundo Mendis o problema pode ter dois enfoques: o primeiro é desenvolver um modelo de regressão linear do MTTF baseado nos primeiros 11 meses de falhas encontradas no programa.

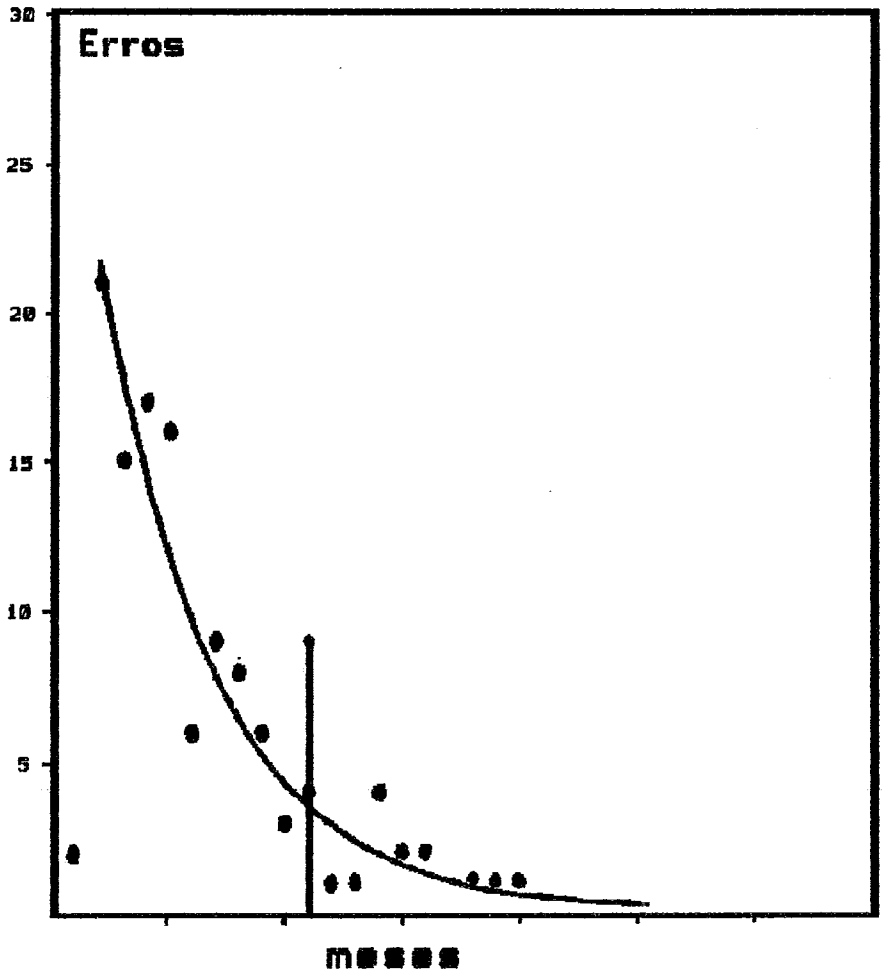


Fig. II.22

Taxa de falhas ao longo do tempo

Fonte: [13]

A Figura II.22 mostra o decréscimo da taxa de falhas ao longo do tempo. Assumindo que os erros remanescentes decrescem monotonicamente e que a taxa de descoberta de erros é proporcional a quantidade de tais erros, podemos então esperar que os erros remanescentes decairão exponencialmente. O gráfico permite observar que os erros ainda existentes no programa em qualquer mês devem seguir uma curva exponencial negativa da forma $E(t) = A\ e^{-tB}$, onde:

E(t) = Total de erros num dado período de tempo

A = Constante

B = Parâmetro de modelagem

t = Tempo.

Utilizando a técnica de regressão linear obtemos a seguinte equação $E(t) = 32.14\ e^{-0.2t}$. Este enfoque está condicionado a que todos os erros são únicos, iguais no tempo de detecção e corrigidos sem causar outros erros. O esforço de descoberta de erros é constante ao longo do tempo. Um método alternativo para determinar a confiabilidade é considerar o MTTF ou o inverso de E(t), e se baseia no fato de que o log de uma função exponencial é uma reta.

O primeiro passo na aplicação do método e completar o ln MTTF para todos os dados coletados. Aplicando o método dos mínimos quadrados e ajustando os dados numa reta da forma Y= a+by, obtemos as seguintes retas limites para um nível de confiança de 95%.

$$t1 = \frac{\ln MTTF + 3.00}{0.24}$$

$$t2 = \frac{\ln MTTF + 3.74}{0.61}$$

Desta forma, é possível avaliar o comportamento do programa durante sua fase operacional e estimar quantos defeitos ainda existem. Estas e outras informações oriundas de um sistema de gerenciamento de falhas eficiente permitirão que o processo de desenvolvimento seja realimentado de modo a que se possa solucionar os problemas existentes mais rapidamente, assim como permitir um contínuo aprimoramento do mesmo.

CAPíTULO III

PRESTIME- "Um ambiente para a PRedição, EStimação e MEdição da Confiabilidade de Software"

3.1- Introdução

O propósito de PRESTIME é automatizar as atividades descritas anteriormente utilizando para isto um conjunto de mecanismos integrados em um mesmo ambiente e que permitam garantir a qualidade de software através da observação de seu comportamento ao longo do ciclo de vida. Este ambiente se compõe de um conjunto de ferramentas de avaliação, cada uma delas aplicável a uma etapa especifica do ciclo de vida.

3.2- Descrição geral

A Figura III.1 mostra o diagrama de fluxo de dados do sistema de avaliação da confiabilidade e sua relação com o ciclo de vida do software. O processo A.0 apresenta as etapas do ciclo de vida e os aspectos importantes para avaliação da qualidade do produto.

A finalidade do processo A.1 é especificar o nivel de confiabilidade desejado para o projeto a ser desenvolvido. O valor obtido é fruto da experiência adquirida em projetos passados, dos requisitos de confiabilidade tidos como importantes pelo usuário, da disponibilidade de recursos e dos requisitos de confiabilidade tidos como necessários pelos projetistas para o sucesso do projeto. Além da especificação

da meta de confiabilidade, este processo determina o ambiente, técnicas e métodos de desenvolvimento adequados para que tal meta seja alcançada.

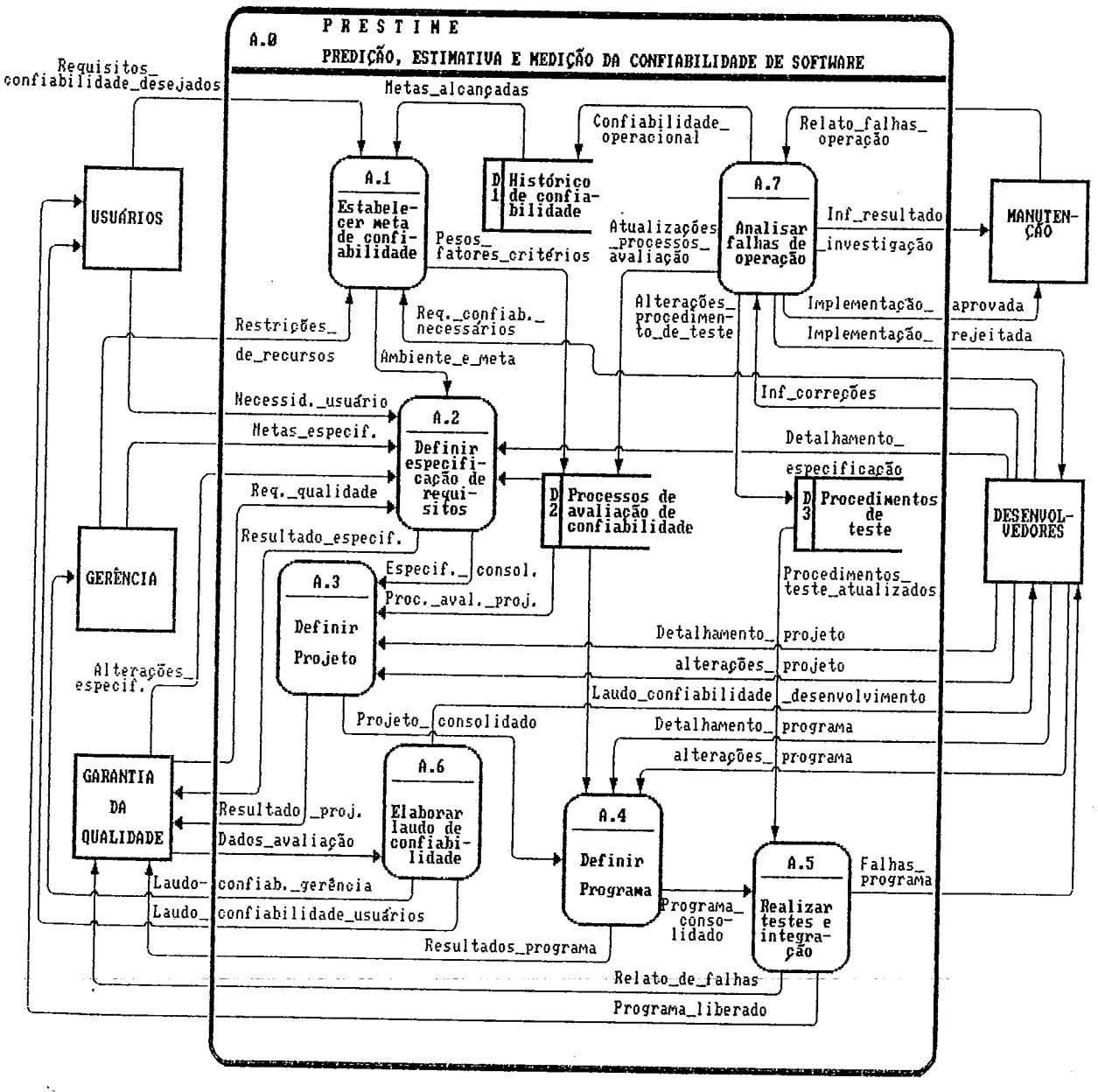


Fig. III.1

Diagrama de fluxo de dados do ambiente para avaliação da confiabilidade de software (PRESTIME)

O processo A.2 tem como objetivo gerar a especificação de requisitos para os responsáveis pelo projeto com base nos seguintes aspectos: técnicas e métodos definidos anteriormente pelo processo A.1, meta de confiabilidade especificada, necessidades dos usuários, metas e restrições definidas pela gerência e requisitos de qualidade.

A elaboração da especificação de requisitos é um processo iterativo. É realizado através de detalhamentos efetuados pelos projetistas, alterações solicitadas pela garantia da qualidade e avaliações da especificação com base em critérios de qualidade pré-definidos.

A especificação de requisitos consolidada satisfazendo os critérios de qualidade e por conseguinte a meta de confiabilidade serve então como subsídio para a elaboração do projeto (processo A.3). A sua elaboração deve levar em consideração os requisitos de projeto definidos anteriormente. É realizado também através de um processo iterativo. Após o projeto ter sido detalhado, o mesmo deverá sofrer avaliações, que informarão se a confiabilidade desejada foi ou não alcançada. Se tal fato não ocorrer, alterações são recomendadas aos projetistas a fim de que os critérios de qualidade não satisfeitos anteriormente possam assim o ser.

Quando este processo de ajuste tiver terminado, teremos um projeto cujo nível de confiabilidade será igual ou superior

desejado, podendo então ser liberado para que seja implementado.

O mesmo procedimento anterior é utilizado para a geração do projeto-consolidado (processo A.3) e programa-consolidado (processo A.4). Este último será submetido a testes a fim de identificar os erros do programa e reportá-los aos projetistas e aos responsáveis pela garantia de qualidade.

Quando o programa tiver atingido o nível de confiabilidade especificado para a fase de testes (número de falhas por hora de execução) o mesmo poderá ser liberado para os usuários.

Durante a utilização do programa no seu ambiente real de operação, falhas poderão ocorrer devido a erros residuais que não puderam ser detectados, sejam eles conseqüentes de condições específicas não testadas ou restrições de tempo de modo a atender ao cronograma pré-estabelecido para a fase de testes. Tais falhas serão objeto de atenção por parte dos encarregados da manutenção do programa, que as relatarão aos responsáveis pela sua correção (processo A.7). A análise destas falhas é também realizada com vistas o obter o valor da confiabilidade operacional do produto. Este valor deve se apresentar no mesmo nível daquele já anteriormente predito e estimado. Se tal fato não ocorrer, deverão então ser efetuadas alterações nos processos de avaliação e no modelo de estimação empregados, a fim de que a confiabilidade operacional medida confirme os valores preditos e estimados.

Com o objetivo de balizar futuros projetos, quanto a especificação da meta de confiabilidade, os valores medidos da confiabilidade operacional são armazenados e periodicamente atualizados.

Esta é a descrição geral do sistema PRESTIME. Vários de seus procedimentos poderão ser automatizados, PRESTIME é constituído pelos sistemas de especificação, predição, estimação e medição da confiabilidade, que serão descritos a seguir. Tais sistemas estão inspirados nos trabalhos realizados por Rocha [14], Musa [15], RADC [4] [5] e Mendis [13], descritos nos capítulos anteriores.

3.3- Sistema de especificação da confiabilidade

Para automatizar esta atividade, duas ferramentas são necessárias: o Especificador de Confiabilidade e o Especificador de Ambientes de Desenvolvimento.

3.3.1- Descrição geral

A Figura III.2 mostra o diagrama de fluxo de dados do processo "estabelecer meta de confiabilidade". Com o objetivo de avaliar as necessidades de confiabilidade (processo A.1.1) para o produto a ser desenvolvido, usuários, gerência e projetistas interagem de forma a determinar corretamente a natureza da aplicação (comercial em batch, militar de tempo real, software básico, administrativo on-line, etc.) e a estabelecer um primeiro valor para a meta de confiabilidade.

Esta primeira proposta é avaliada (processo A.1.3) com base no histórico de confiabilidade de projetos passados (processo A.1.2) e ajustada levando em conta o intervalo de confiança da confiabilidade típica da natureza da aplicação, gerando, assim, o resultado final para a meta de confiabilidade. Esta meta, junto com as características do sistema, serve como guia para a determinação do ambiente de desenvolvimento necessário para que a mesma possa ser atingida (processo A.1.5).

Para que a predição da confiabilidade possa ser realizada, com base nas características do software antes da fase de teste, pesos devem ser estabelecidos (processo A.1.4) para os fatores, subfatores e critérios de qualidade, que serão avaliados nas etapas de especificação, projeto e codificação. A escolha destes pesos se baseia em informações de projetos passados, pesos utilizados em tais projetos e nas propostas feitas pelos usuários, gerência e projetistas.

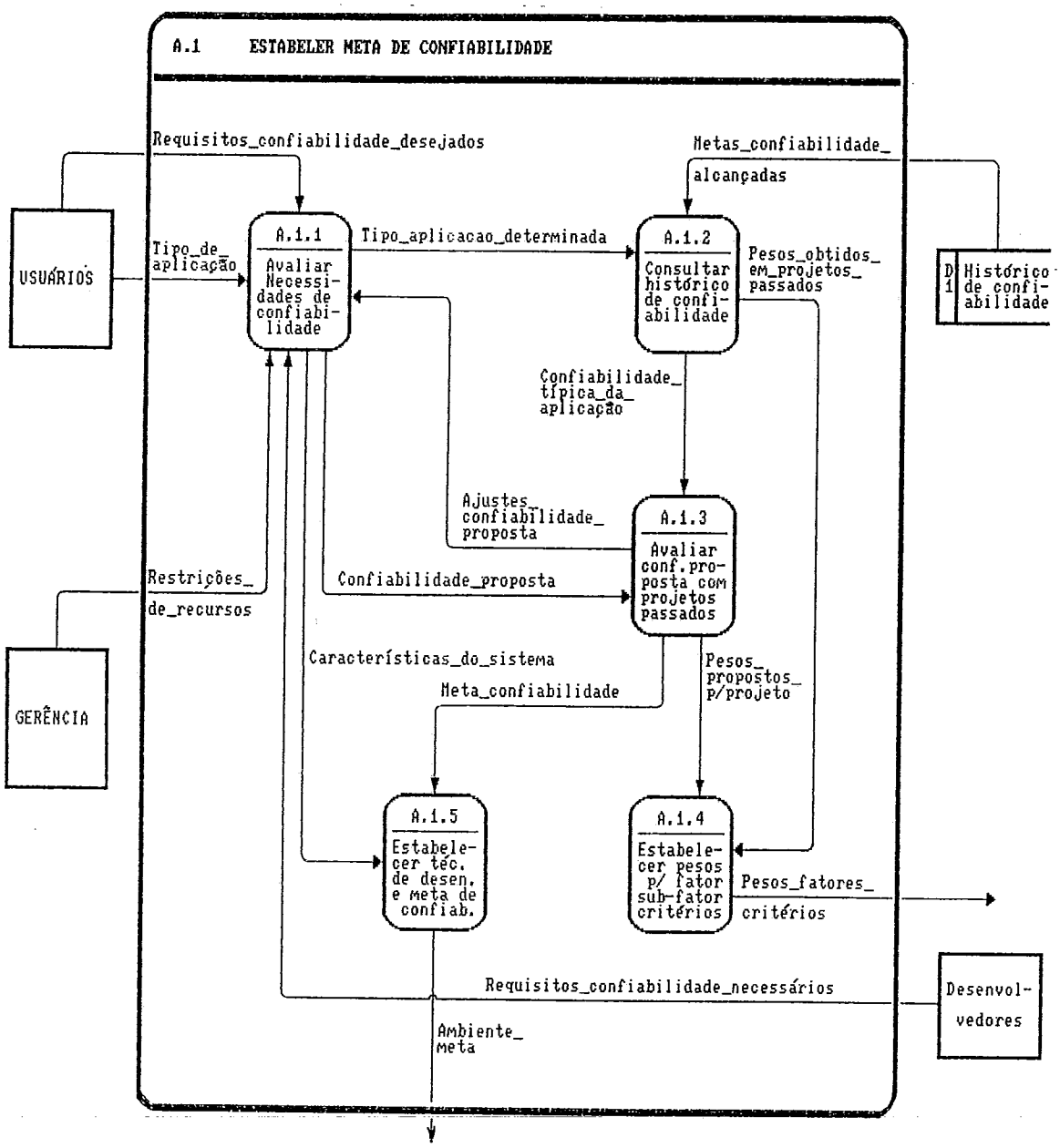


Fig. III.2

Diagrama de fluxo de dados do processo estabelecer meta de confiabilidade

3.3.2- Ferramentas de apoio a especificação da meta de confiabilidade

Duas ferramentas automatizadas de apoio podem ser visualizadas na Figura III.2. A primeira, englobando os processos A.1.1, A.1.2, A.1.3, A.1.4 e o depósito D1, caracteriza a ferramenta que denominamos Especificador de Confiabilidade. A segunda, representada pelo processo A.1.5 é denominada Especificador de Ambientes de Desenvolvimento.

3.3.2.1- Especificador de confiabilidade

O Especificador de Confiabilidade tem a finalidade de auxiliar a determinação da meta de confiabilidade. Tal meta pode ser determinada a partir do que foi alcançado em projetos passados. O objetivo é estabelecer os fatores, subfatores, critérios e processos de avaliação que mais se adequem ao tipo de aplicação que irá ser desenvolvida, assim como o peso e o intervalo de confiança associado a cada um destes atributos. Este intervalo tenderá a ser grande no início, devido a existência de poucos projetos analisados, tendendo a diminuir com o decorrer do tempo e com o acúmulo de projetos encerrados.

3.3.2.2- Especificador de ambientes de desenvolvimento

A finalidade do Especificador de Ambientes de Desenvolvimento é apresentar métodos e ferramentas mais apropriados à

construção do sistema em questão. Seu objetivo é especificar e gerar ambientes de desenvolvimento de software adequados a diferentes domínios de aplicação. Tal ferramenta vem sendo desenvolvida na COPPE/UFRJ e faz parte do "Projeto TABA" [24]. Esta ferramenta é representada pelo processo A.1.5.

3.4- Sistema de predição da confiabilidade

A atividade de elaboração da especificação é um processo baseado em refinamentos sucessivos e tem por base as características do sistema tidas como desejáveis pelos usuários. É realizado através do detalhamento destas características pelos projetistas. É levado em consideração o atendimento dos requisitos de qualidade específicos e das metas estabelecidas pela gerência.

A revisão da especificação é essencial para garantir que os desenvolvedores e os usuários tenham a mesma percepção do problema. A revisão deve ser realizada a partir dos requisitos de qualidade especificados. Caso tais requisitos não tenham sido atendidos, alterações devem ser realizadas até que a especificação se consolide.

3.4.1- Descrição geral

A Figura III.3 mostra o diagrama de fluxo de dados do processo "definir especificação de requisitos". A especificação é elaborada (processo A.2.1) a partir dos seguintes elementos: meta de confiabilidade especificada, necessidades dos

usuários, metas de projeto definidas pela gerência e requisitos de qualidade especificados pelos responsáveis pela garantia da qualidade. O resultado final é uma proposta de especificação de requisitos. Esta será submetida a uma posterior avaliação, através de mecanismos de predição da confiabilidade (processo A.2.2). Se o valor predito para a confiabilidade não tiver alcançado a meta especificada, após ter sido realizado a análise dos resultados da avaliação (processo A.2.4), medidas corretivas deverão ser propostas a fim de que os requisitos não atendidos venham a ser.

O plano do projeto é o documento resultante da atividade de planejamento. É extremamente importante sua elaboração formal (processo A.2.3). O plano é um contrato onde estão envolvidos vários elementos: estimativas, metas, objetivos, eventos, prazos, pessoal e custos. Ele é a ferramenta básica que deverá ser usada para guiar o projeto em direção a seus objetivos.

3.4.2- Ferramentas de apoio a predição da confiabilidade

Santos [20], propõe três ferramentas para avaliação da qualidade de especificações, projetos e programas, denominadas "Avaliador de Especificações de Requisitos de Software", "Avaliador de Projetos de Software" e "Avaliador de Programas Sensível a Linguagem". Tais ferramentas se baseiam no conceito de sistemas especialistas e no método descrito por Rocha [14].

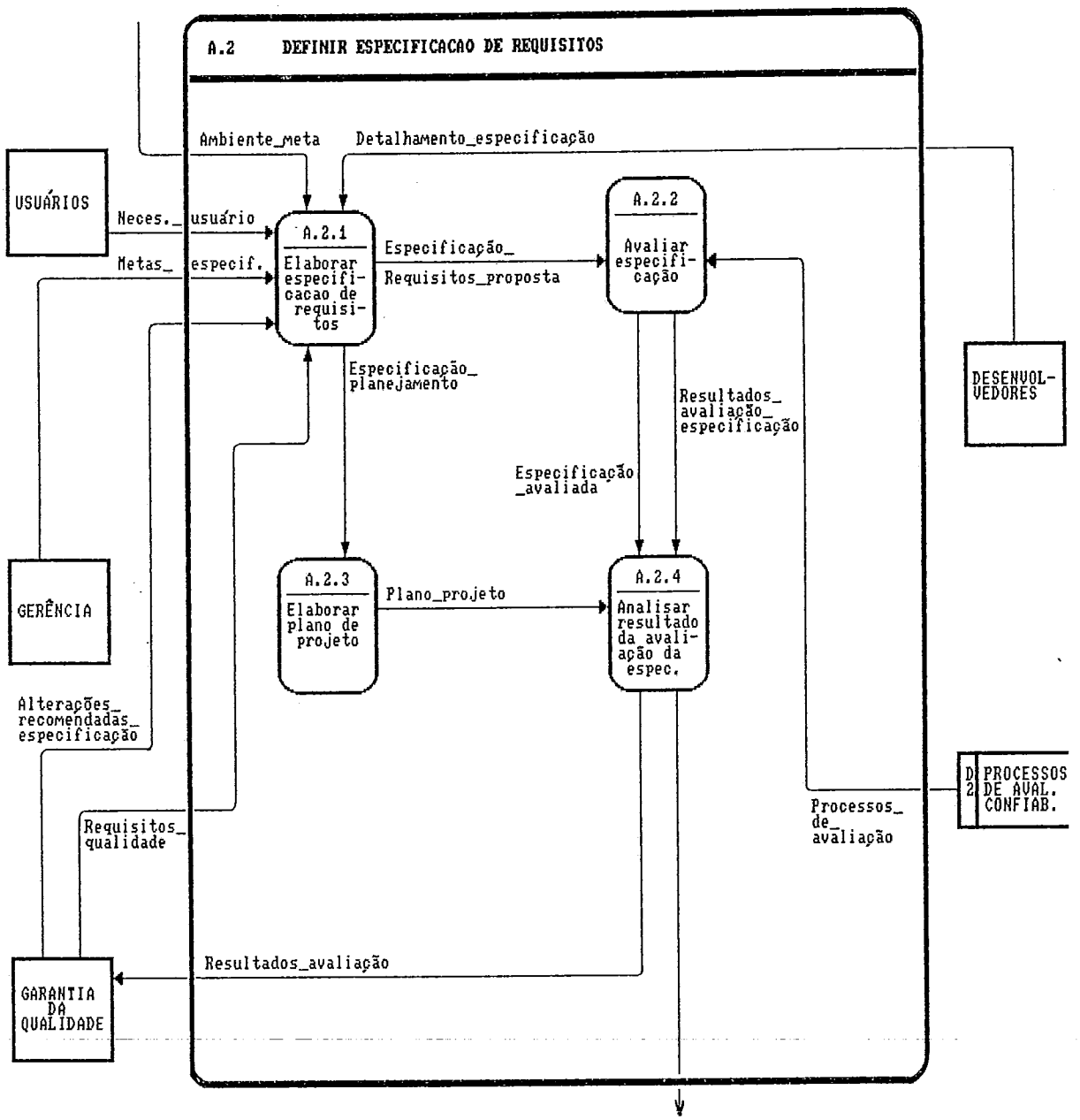


Fig. III.3

Diagrama de fluxo de dados do processo definir especificação de requisitos

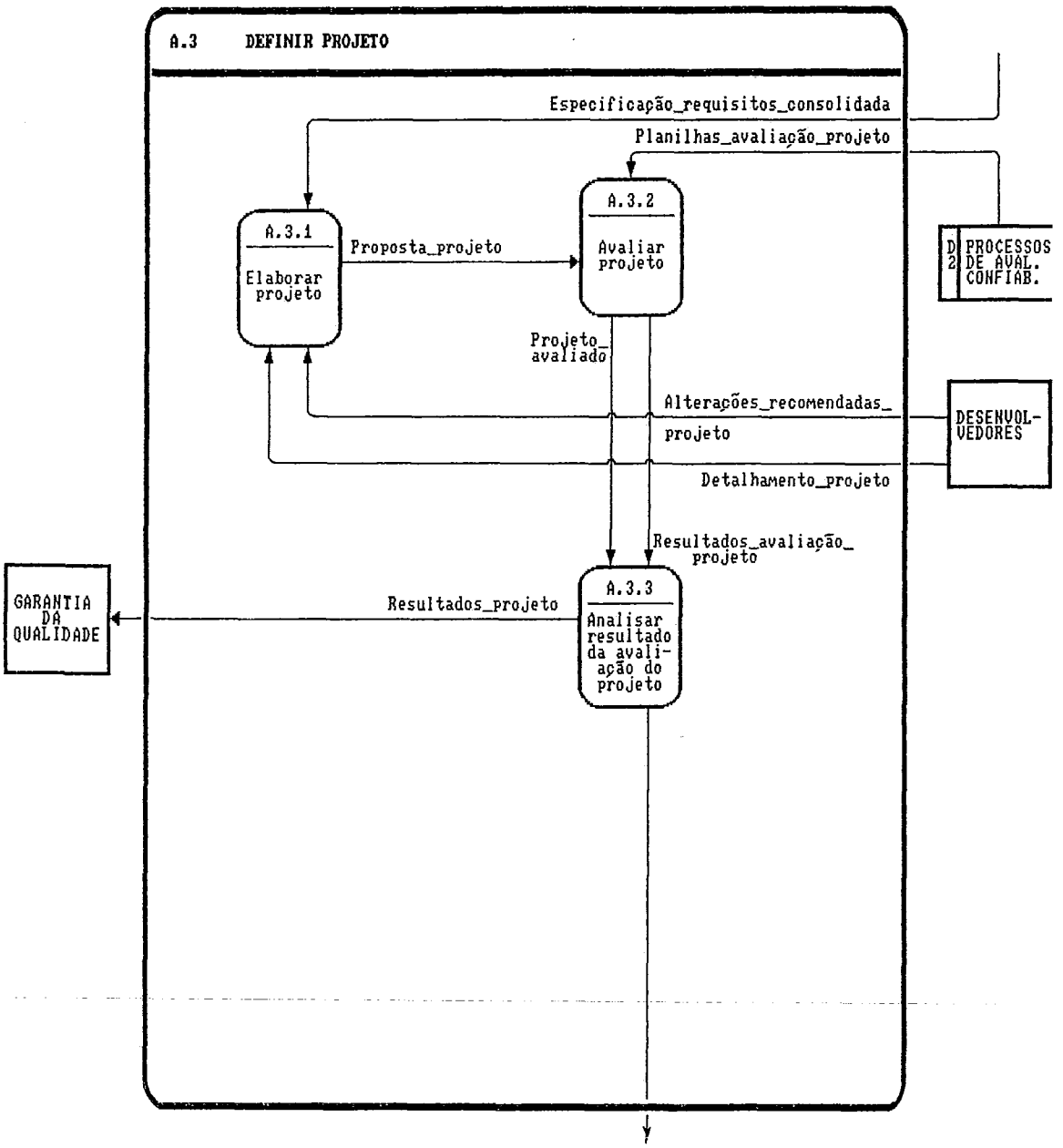


Fig. III.4

Diagrama de fluxo de dados do processo definir projeto

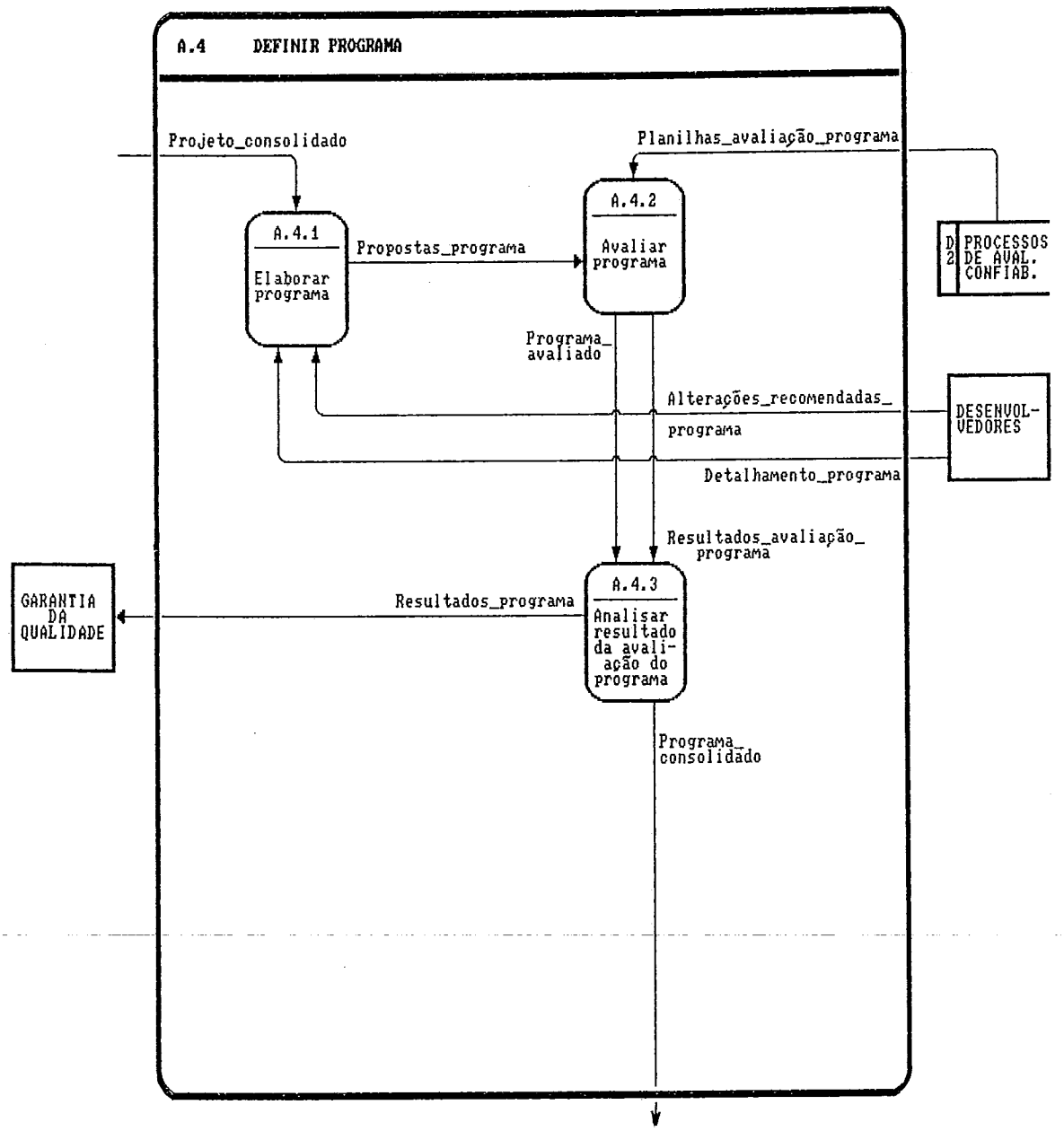


Fig. III.5

Diagrama de fluxo de dados do processo definir programa

3.4.2.1- Avaliador de especificações de requisitos de software

Trata-se de um sistema especialista e baseia-se na avaliação de fatores e critérios de qualidade estabelecidos para o sistema. Os processos de avaliação tem como base o manual de avaliação de qualidade de especificações de requisitos de software, onde são descritos os objetivos da confiabilidade de especificações em termos de seus fatores, subfatores e critérios de qualidade.

3.4.2.2- Avaliador de projetos

Rocha [14] identificou os objetivos, fatores e subfatores de qualidade desejáveis na fase de projeto. A avaliação de tais atributos é o objetivo do "Avaliador de Projetos de Software".

3.4.2.3- Avaliador de programas sensível a linguagem

O "Avaliador de Programas Sensível a Linguagem" tem a finalidade de avaliar a qualidade de programas-fonte, segundo o padrão da linguagem utilizada e dos atributos de qualidade estabelecidos e priorizados. Tais atributos podem ser encontrados em [14].

3.5- Sistema de estimação da confiabilidade

3.5.1- Descrição geral

A Figura III.6 mostra o diagrama de fluxo de dados da etapa de testes e integração. Terminada a codificação do módulo programa consolidado, o mesmo é submetido a realização de testes de módulo (processo A.7.2). É somente enviado para testes de integração quando for liberado pelo processo A.7.8 (aplicar modelos de estimação da confiabilidade). Este evento só é realizado quando o modelo escolhido sinalizar que o nível de confiabilidade especificado foi atingido. As falhas de módulo detectadas são encaminhadas aos projetistas através de relatórios de falha e armazenadas no depósito de dados D6, a fim de gerar periódicos relatórios de acompanhamento para os responsáveis pela garantia da qualidade de produto. O mesmo procedimento de liberação do módulo para testes de integração (processo A.7.2) é repetido nos processos A.7.3, A.7.4, A.7.5, A.7.6, A.7.7, respectivamente realizar testes de integração, funcionais, de sistema, de aceitação e de instalação. As falhas encontradas, além de encaminhadas aos projetistas, são submetidas ao modelo de estimação de confiabilidade escolhido.

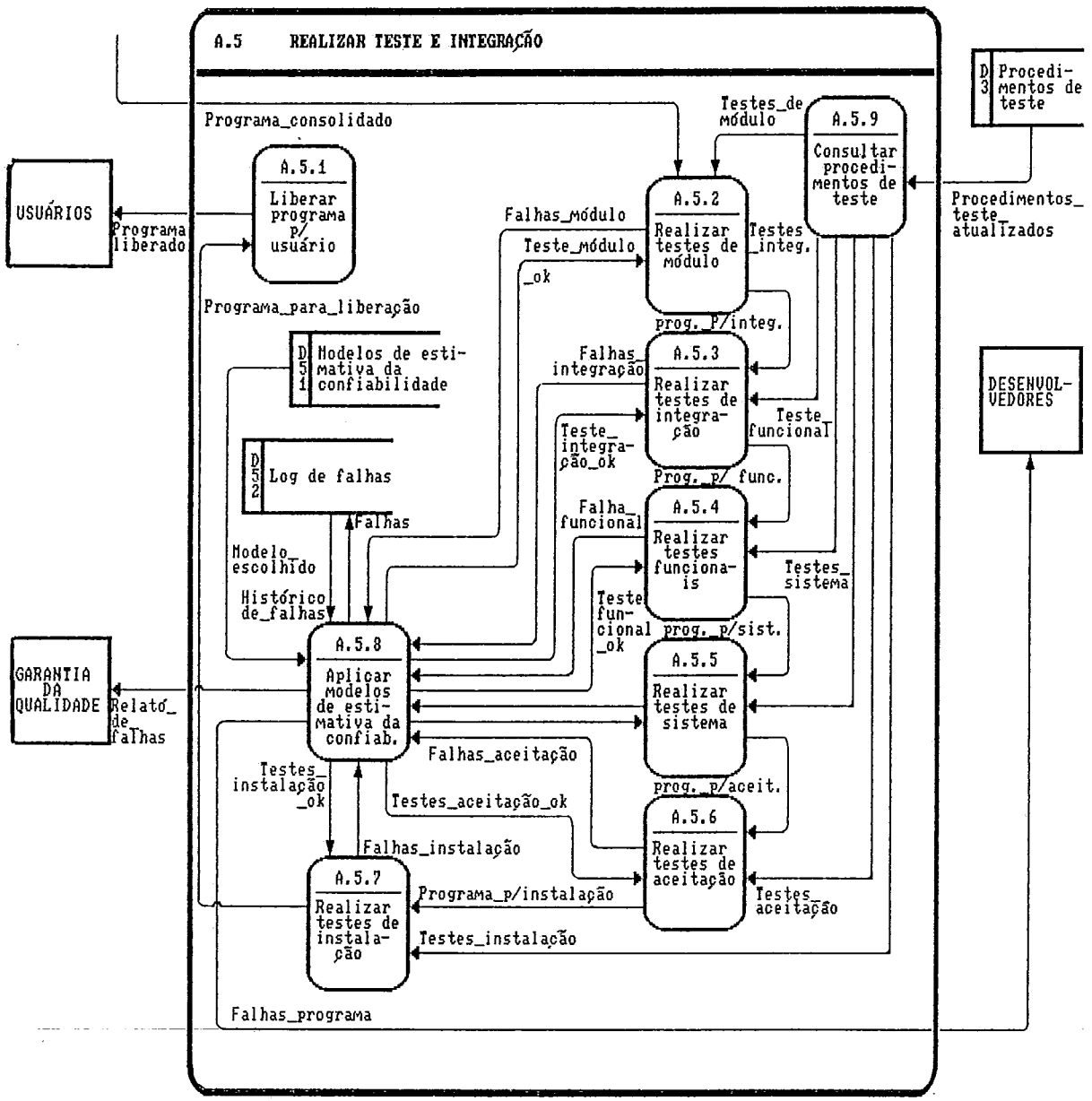


Fig. III.6

Diagrama de fluxo de dados do sistema de estimação da confiabilidade

3.5.2- Ferramentas de apoio ao sistema de estimação da confiabilidade

Musa [15] advoga a necessidade de quatro ferramentas de apoio, úteis na aplicação de um programa de estimação da confiabilidade e que serão adotadas em PRESTIME:

1- Analisador de Falhas
2- Gerador de Gráficos de Confiabilidade
3- Simulador de Custos e Cronograma
4- Gerenciador de Falhas de Teste

Além destas quatro ferramentas (propostas por Musa), é também necessário um mecanismo que permita a geração de casos de teste aleatórios, o que configura a necessidade de uma quinta ferramenta (Gerador de Casos de Teste Aleatórios).

3.5.2.1- Analisador de falhas

O Analisador de Falhas permite determinar o "status" do sistema num dado ponto do processo de testes, apresentando como saída as seguintes informações:

- lista de parâmetros utilizados e seus valores associados;
- lista das falhas ocorridas;
- lista dos dados ajustados;
- tempo decorrido desde o início dos testes;
- número de falhas associadas ao cronograma.

A Figura III.7 mostra uma saída típica do programa.

Relatório baseado na amostra de: 54 Falhas de teste
Tempo de execução: 32.78 Horas
Objetivo da intensidade de falhas: 0.36e-01 Falhas/hora CPU
Tempo cronológico até hoje 74 Dias
Data atual: 31/05/74

	LIMITES DE CONFIANÇA				MAIS PROV.	LIMITES DE CONFIANÇA			
	95%	90%	75%	50%		50%	75%	90%	95%
DECAIMENTO DA INTENSIDADE DE FALHAS	.043	.046	.050	.055	.061	.068	.072	.077	.080
INTENSIDADE DE FALHAS (FALHAS / 1000 HORAS-CPU)									
FI INIC.	6440	7185	8563	10172	13054	16856	20260	2460027873	
FI ATUAL	378	392	415	440	481	528	567	612	644
** REQUISITOS ADICIONAIS PARA ALCANÇAR O OBJETIVO DA INTENSIDADE DE FALHAS **									
FALHAS	30	31	34	37	42	49	55	62	68
TEMPO DE EXECUÇAO	316	329	352	378	420	473	517	573	615
TEMPO CRONOLdG.	142	149	159	171	191	215	236	262	282
DATA FINAL	1812	2612	1001	2801	2302	3103	2904	0306	3006

Fig. III.7

Relatório de falhas

Fonte: [15]

3.5.2.2- Gerador de gráficos de confiabilidade

O Gerador de Gráficos de Confiabilidade tem a finalidade de apresentar, de forma gráfica, o histórico da intensidade de falhas, assim como o seu intervalo de confiança (50, 75, 90 e 95%). A ferramenta apresenta graficamente as informações mostradas no analisador de falhas (tempo de execução, tempo

cronológico, datas ou número de falhas). A fim de evitar que o gráfico torne-se confuso, o usuário tem a possibilidade de determinar se os intervalos de confiança devem ser ou não apresentados, assim como, determinar o início e o fim da escala horizontal, já que pode ser necessário examinar com mais detalhes uma seção do gráfico. A Figura III.8 apresenta o exemplo de um gráfico.

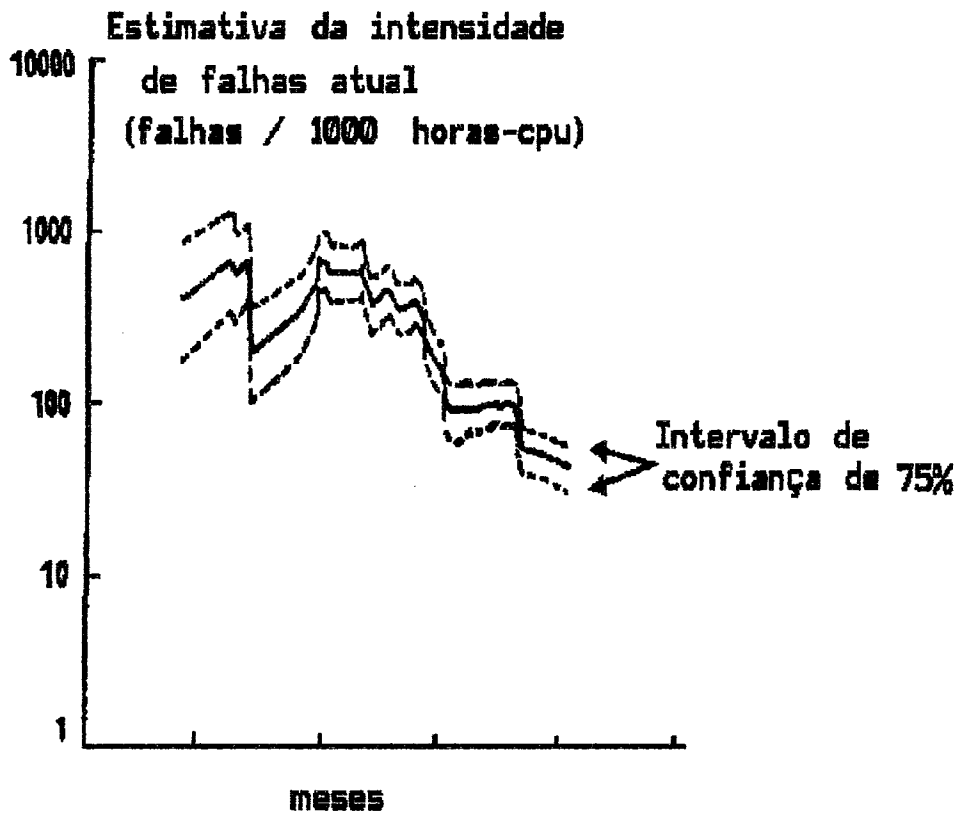


Fig. III.8

Gráfico de intensidade de falhas

Fonte: [15]

3.5.2.3- Simulador de custos e cronograma

O Simulador de Custos e Cronograma permite que os parâmetros do modelo sejam variados de modo a tornar possível a análise do comportamento do sistema sob diferentes influências. A ferramenta realiza a função equivalente a do analisador de falhas, e permite diferentes simulações com valores distintos para os diversos parâmetros. Por exemplo, pode-se obter o efeito da mudança no objetivo a atingir para o valor da intensidade de falhas na época prevista para a liberação do produto.

3.5.2.4- Gerenciador de falhas

O Gerenciador de Falhas de Teste tem a função de controlar o armazenamento das informações sobre as falhas ocorridas durante a realização dos testes e processá-las a fim de convertê-las em intervalos de falhas. Esta ferramenta manipula incertezas de tempo e múltiplas cópias de programas, incluindo aquelas que rodam com diferentes tempos de execução, assim como ordena as informações de diferentes subsistemas que tenham sido guardadas juntas, ponderando cada intervalo de tempo pela sua utilização. Ela funciona independentemente do modelo de estimação escolhido.

3.5.2.5- Gerador de casos de teste aleatórios

Esta ferramenta tem a finalidade de realizar uma seleção aleatória das possíveis entradas de um programa através da

utilização de um mecanismo automático. Tal ferramenta deve levar em consideração a probabilidade de ocorrências entre as diferentes entradas e o custo associado a uma determinada falha decorrente de uma entrada especifica (em termos de perdas financeiras ou até mesmo humanas).

3.6- Sistema de medição da confiabilidade

Um dos maiores problemas da atividade de manutenção é determinar que falhas devem ser prioritariamente reparadas e liberadas dentro dos menores prazos e custos. Cada falha resulta, inexoravelmente, em um custo financeiro, assim como a preparação, disseminação e instalação de reparos. Não se pode deixar de levar em consideração que tais reparos podem causar outras falhas. Um enfoque razoável é decidir em que falhas os esforços serão concentrados, de forma a conduzir em uma rápida análise e em um reduzido impacto financeiro.

Para realizar esta tarefa, um Sistema de Gerenciamento de Falhas de Operação deve ser implementado, a fim de quantificar falhas e classificá-las quanto aos seus tipos e distribuições. Estas informações devem ser separadas levando em consideração os danos causados. Este sistema deve, enfim, conter as informações básicas a respeito da descoberta e correção de falhas, permitindo assim, medir o tempo de maturação do projeto, avaliar a qualidade do esforço de desenvolvimento e identificar possíveis problemas em áreas específicas.

3.6.1- Descrição geral

A Figura III.9 mostra o diagrama de fluxo de dados do sistema de medição da confiabilidade. Inicialmente, quando uma falha é detectada pela manutenção, um relatório de investigação é aberto (processo A.7.1). A função deste relatório é avaliar se o problema relatado realmente se deve a um erro de software e não há um problema de hardware ou utilização indevida do programa por parte do usuário. O resultado do relatório de investigação é comunicado a manutenção e, se for confirmada a falha de software, a mesma é cadastrada para posterior correção (processo A.7.2). Também, neste caso, os procedimentos de teste devem ser corrigidos (processo A.7.3), já que houve transparência nos testes.

Após terem sido efetuadas as correções, a implementação proposta é encaminhada para certificação (processo A.7), que então irá liberá-la ou não para manutenção, dependendo dos resultados obtidos. A confiabilidade operacional é medida (processo A.7.4) a partir das informações recolhidas pela manutenção. Os resultados são armazenados com vistas a produzir um histórico da confiabilidade. Conflitos entre o valor medido da confiabilidade e os valores predito e estimado são analisados (processo A.7.5), visando corrigir os processos de avaliação.

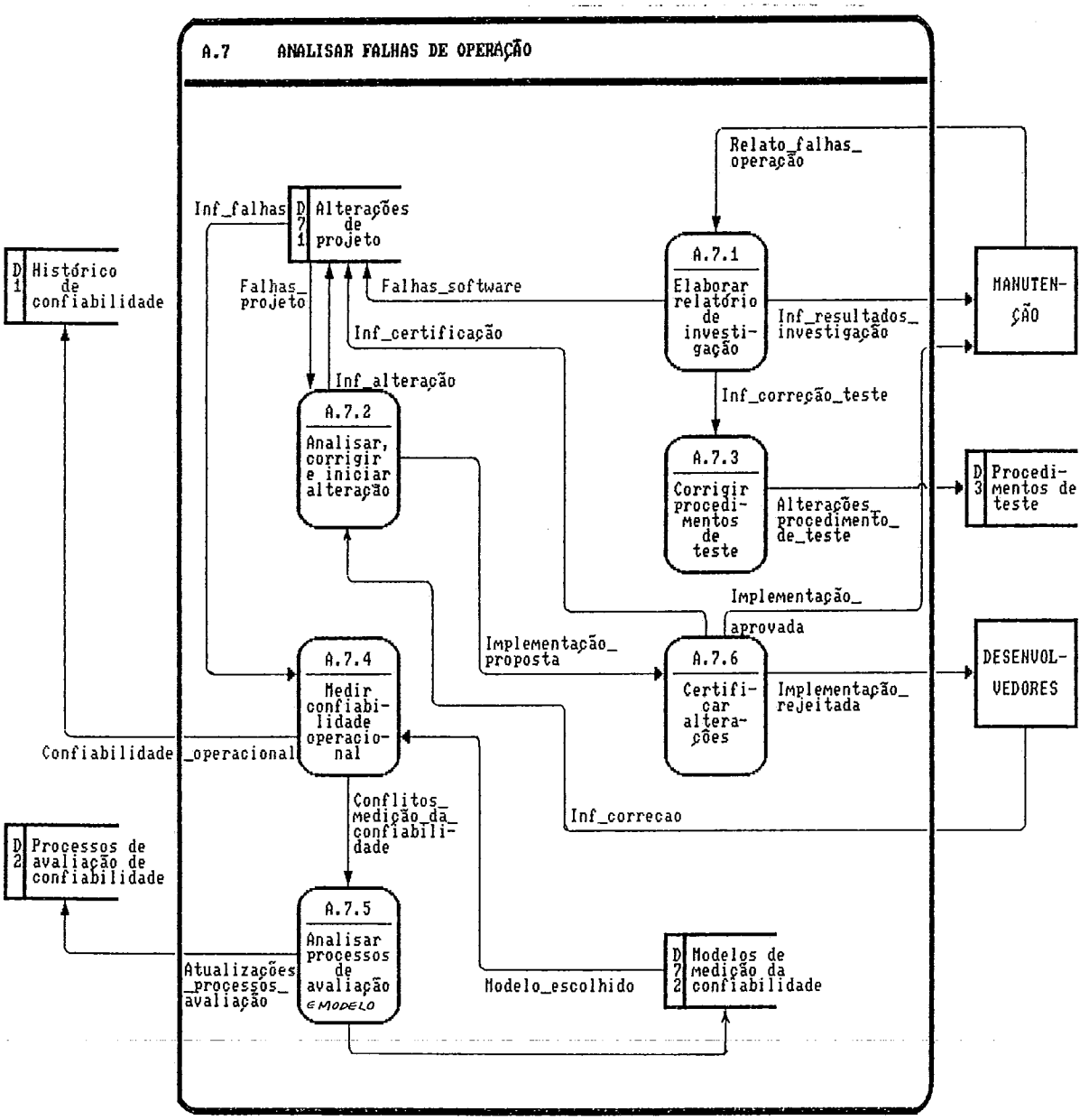


Fig. III.9

Diagrama de fluxo de dados do sistema de medição da confiabilidade

3.6.2- Ferramentas de apoio ao sistema de medição da confiabilidade

De maneira a facilitar a atividade de manutenção da confiabilidade, duas ferramentas automatizadas são necessárias: um gerenciador de falhas de operação, representado pelos processos A.7.1, A.7.2, A.7.3, A.7.6 e o depósito de dados D.7.1, e um avaliador de comportamento operacional, representado pelo processo A.7.4 e depósito de dados D1.

3.6.2.1- Gerenciador de falhas de operação

O principal objetivo desta ferramenta é armazenar informações detalhadas sobre a frequência e tipo das falhas que ocorrem durante a operação do programa. Também permite determinar o esforço necessário para detectá-las e corrigi-las. Tal esforço é composto pelo tempo gasto em horas de programadores e de máquinas. O sistema permite ainda, o acompanhamento da falha do seu diagnóstico até a liberação do reparo realizado, passando pelas etapas de análise/correção e certificação.

3.6.2.2- Avaliador de comportamento operacional

Compreender o significado da ocorrência de falhas ao longo do tempo pode não ser uma tarefa fácil, principalmente em se tratando de sistemas complexos onde várias áreas interagem na atividade de manutenção. Portanto, o Avaliador de

Comportamento Operacional tem a finalidade de, a partir de um modelo de medição da confiabilidade, determinar a intensidade de falhas ao longo do tempo, oferecendo, assim, informações fundamentais para que sejam feitas análises acerca da eficácia das medidas de predição e de estimação de confiabilidade, medidas estas realizadas antes da fase de operação.

3.7- Gerador de laudos de confiabilidade

O gerador de laudos de confiabilidade tem o objetivo de produzir relatórios técnicos e/ou gerenciais padronizados. Tais relatórios devem descrever o comportamento do software ao longo do ciclo de vida (análises técnicas que avaliem se a evolução do produto transcorre conforme o esperado).

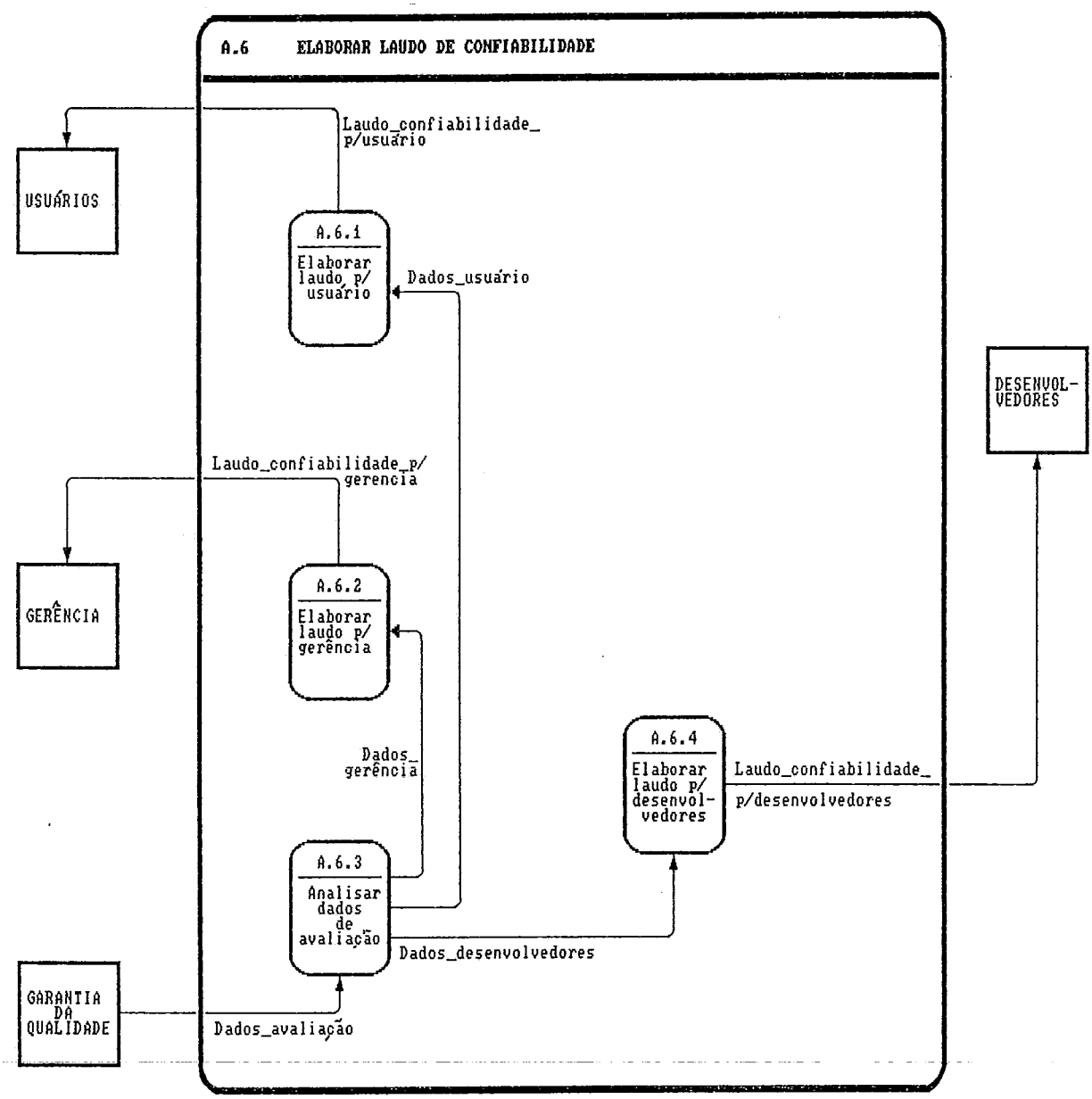


Fig. III.10

Diagrama de fluxo de dados do processo "Elaborar laudo de confiabilidade"

CAPÍTULO IV

ESPECIFICAÇÃO DE REQUISITOS DO PROTÓTIPO DO SISTEMA DE MEDIÇÃO DA CONFIABILIDADE

4.1- Introdução

4.1.1- Propósito

O propósito do Sistema de Medição da Confiabilidade (SMC) é automatizar a atividade de acompanhamento de falhas de operação desde a sua identificação no campo, passando pelo seu diagnóstico e sua solução, até a liberação da correção para os usuários. Também faz parte deste sistema uma ferramenta para análise do seu comportamento durante a fase operacional do mesmo. Interagem com o sistema os órgãos encarregados pela garantia da qualidade, desenvolvimento, teste e manutenção de software.

4.1.2- Escopo

O SMC se compõe de duas ferramentas, o Gerenciador de Falhas de Operação (GFO) e o Analisador de Comportamento Operacional (ACO). A primeira ferramenta visa criar uma ambiente integrado, onde todos os órgãos que participem do processo de solução de falhas possam facilmente interagir.

O ACO opera sobre a base de dados onde as informações sobre as falhas são guardadas, produzindo como resultado, gráficos que mostrem se o produto tende à maturar conforme as expectativas.

Desta forma é possível avaliar se as medidas efetuadas durante o desenvolvimento predizem e estimam com precisão o comportamento real do produto.

Um protótipo do SMC será implementado, o que significa que esta especificação se restringe apenas aos aspectos básicos e mais importantes do sistema. O GFO é resultado de melhorias e de uma evolução do atual sistema de controle de alterações da COBRA Computadores (SCAP), principalmente no que se refere à interface com o usuário e com os aspectos relacionados com o processo de gerenciamento. O ACO é fruto do trabalho de K. S. MENDIS [13].

4.1.3- Referências

K. S. MENDIS, "Quantifying Software Quality", Quality Progress, 18-22, maio 1983.

The Whitewater Group, "ACTOR Language Manual", 1989.

The Whitewater Group, "ACTOR 2.0 Addendum Manual", 1990.

The Whitewater Group, "ACTOR Add-On Product Series: Language Extensions I", 1990.

M. FRANZ, "Object-Oriented Programming - Featuring ACTOR", Scott, Foresman and Company, Illinois, 370pp., 1990.

4.1.4- Glossário

PRESTIME- PRedição, ESTImação e MEdição da Confiabilidade

SMC- Sistema de Medição da Confiabilidade

STC- Sistema de Estimação da Confiabilidade

SPC- Sistema de Predição da Confiabilidade

SEC- Sistema de Especificação da Confiabilidade

GFO- Gerenciador de Falhas de Operação

ACO- Analisador de Comportamento Operacional

DFD- Diagrama de Fluxo de Dados

SCAP- Sistema de Controle de Alterações de Projeto

4.1.5- Visão geral da especificação

Para a elaboração deste documento seguimos o roteiro sugerido por Rocha [14], assim como itens especificados por Fairlay [21].

Este documento está organizado em três partes: Introdução, Requisitos específicos e Aspectos de implementação. A primeira parte define o propósito da especificação assim como seus

usuários, descreve os objetivos e benefícios do produto a ser construído, identifica todos os documentos referenciados na especificação, define todos os termos necessários a perfeita compreensão da especificação, aborda o conteúdo e organização da mesma, relaciona as interfaces com outros produtos e descreve as funções do sistema.

A segunda parte (Requisitos específicos), contém a especificação dos requisitos de informação, de interface, de desempenho e funcionais, este último abordado sob a ótica de modelagem por processos (a la "GANE") visando sua posterior transformação em um modelo de objetos [94].

A terceira parte trata de aspectos relacionados com a implementação tais como, prioridades, modificações e melhoramentos previstos.

4.1.6- Interface do produto com outros produtos ou projetos

O SMC faz parte do ambiente para avaliação da confiabilidade "PRESTIME". Também fazem parte deste ambiente os sistemas de especificação, predição e estimação da confiabilidade (SEC, SPC e STC). A integração entre os sistemas deve ser feita através de uma base de dados comum.

4.1.7- Funções do produto

- Oferecer um método automatizado para o cadastramento das falhas ocorridas no campo, visando o posterior diagnóstico, correção, validação e liberação da solução;

- fornecer aos órgãos que interagem com o sistema relatórios periódicos sobre o comportamento do produto;

- prever, com base nas informações obtidas até então, quanto tempo será necessário para o produto estabilizar;

- realimentar o processo de avaliação da confiabilidade no que concerne às métricas utilizadas para a sua predição e aos modelos utilizados na sua estimação.

4.2- Requisitos específicos

4.2.1- Requisitos de informação

Somente os órgãos envolvidos com o processo de acompanhamento de falhas (desenvolvimento, teste, manutenção e garantia da qualidade) podem alterar as informações do sistema. Outros órgãos podem apenas consultar.

O SMC deve ser implementado de modo a permitir com facilidade a inclusão ou exclusão de itens do relatório de acompanhamento de falhas.

O modelo matemático do ACO, utilizado para a análise do programa durante a fase operacional, não poderá ser alterado no caso do protótipo, embora no sistema definitivo isto seja permitido.

4.2.2- Requisitos de Interface

4.2.2.1- Interface do ambiente PRESTIME com o usuário

A comunicação com o usuário do SMC é realizada através de janelas na sua estação de trabalho. Na tela ele visualiza em primeiro lugar o janela básica do ambiente PRESTIME, conforme é mostrado na Figura IV.1.

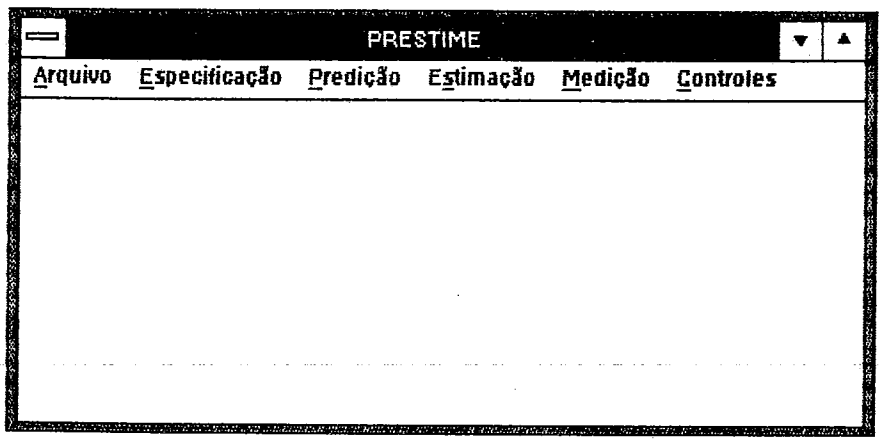


Fig. IV.1

O ambiente PRESTIME

Seis opções são mostradas no menu do PRESTIME (Arquivo, Especificação, Predição, Estimação, Medição e Controles). Com exceção das opções Arquivo e Controles as demais se referem a ativação de uma ou mais ferramentas.

a) Opção Arquivo

Esta opção esta relacionada com o controle dos projetos de avaliação da confiabilidade do ambiente. A Figura IV.2 mostra o menu correspondente a opção Arquivo.

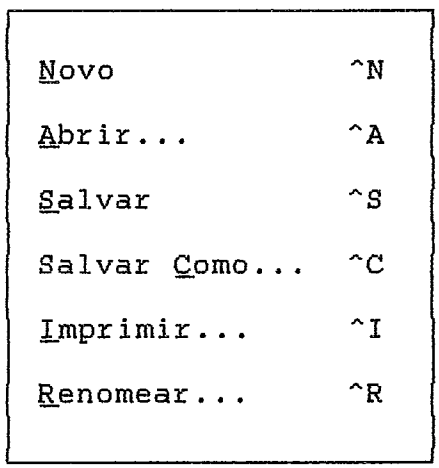


Fig. IV.2

Menu da opção Arquivo

. item Novo- inicia um novo projeto de acompanhamento;
. item Abrir- carrega um projeto já iniciado;
. item Salvar- salva as informações relativas ao projeto, até então existentes, com o nome em que o mesmo foi aberto;
. item Salvar Como- salva as informações relativas ao projeto, até então existentes, como um novo nome;

. item Imprimir- imprime um relatório com as informações do projeto;

. item Renomear- troca o nome de um projeto.

b) Opção Especificação

Esta opção está relacionada com a determinação da meta de confiabilidade. É especificado o tipo de aplicação a ser desenvolvida, os atributos de qualidade adequados, seus pesos relativos, as intensidades de falha esperadas para as fases de teste e operação e o ambiente de desenvolvimento mais indicado para o tipo de aplicação escolhida. A Figura IV.3 apresenta o menu correspondente a esta opção.

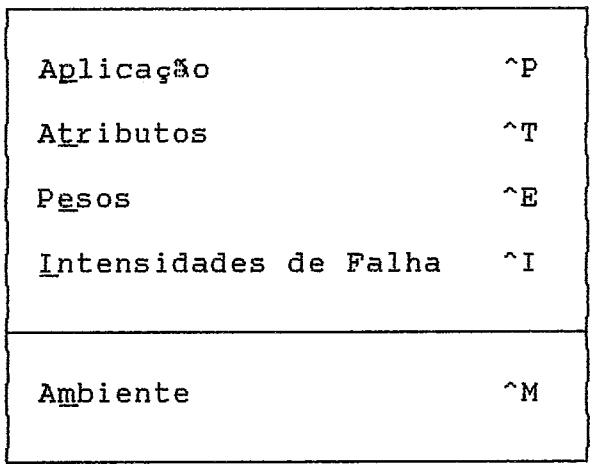


Fig IV.3

Menu da opção Especificação

. item Aplicação- define o tipo de aplicação que será desenvolvida, de modo a sugerir os atributos de qualidade, pesos relativos e intensidades de falha mais adequados;

. item Atributos- define os atributos de qualidade desejáveis;

. item Pesos- define os pesos relativos desejáveis;
. item Intensidades de Falha- define as intensidades de falha desejáveis;
. item Ambiente- ativa a ferramenta que define o ciclo de vida, técnicas e ferramentas de desenvolvimento mais adequadas.

c) Opção Predição

A escolha desta opção apresenta um menu com os itens Especificação, Projeto e Programa, correspondendo respectivamente às avaliações da especificação de requisitos, especificação de projeto e código fonte, com base nos atributos de qualidade definidos na opção Especificação. A Figura IV.4 mostra o menu desta opção.

Especificação de Requisitos	F1
Especificação de Projeto	F2
Código Fonte	F3

Fig. IV.4

Menu da opção Predição

. item Especificação de Requisitos- avalia a especificação com base nos atributos pré-definidos;
. item Especificação de Projeto- avalia o projeto com base nos atributos pré-definidos;
. item Código Fonte- avalia o programa com base em atributos

de qualidade pré-definidos.

d) Opção Estimação

Esta opção permite o acesso à cinco ferramentas, o gerenciador de falhas de teste, o analisador de falhas de teste, o simulador de custos e cronograma, o gerador de gráficos de confiabilidade e o gerador de casos de teste aleatórios. Os itens de menu desta opção são apresentados na Figura IV.5.

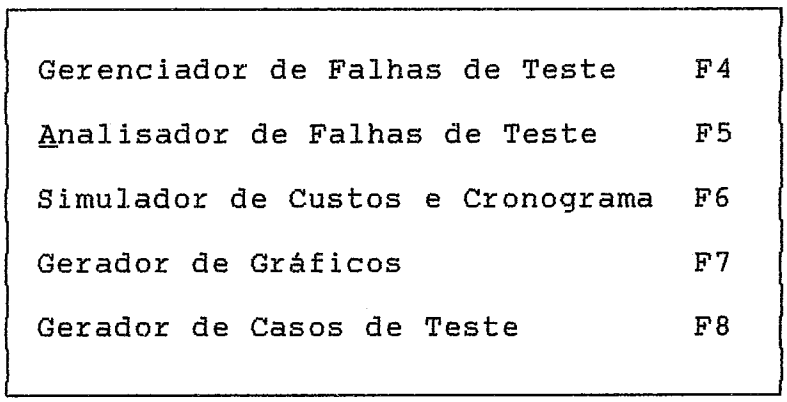


Fig. IV.5

Menu da Opção Estimação

. item Gerenciador de Falhas de Teste- ativa a ferramenta responsável pelo controle das informações referentes à ocorrência de falhas durante a realização dos testes;
. item Analisador de Falhas de Teste- ativa a ferramenta que calcula a intensidade de falhas instantânea a as estimativas de recursos (tempo e quantidades) durante a fase de testes;
. item Simulador de Custos e Cronograma- ativa a ferramenta que possibilitará simular o comportamento do processo de testes sob a influência de diversas condições;

. item Gerador de Gráficos- ativa a ferramenta que, a partir de análises já realizadas, apresentará graficamente estes resultados;
. item Gerador de Casos de Teste- ativa a ferramenta que irá produzir diferentes combinações (aleatórias) de entrada de dados para o programa em teste.

e) Opção Medição

Esta opção permite que sejam ativadas duas ferramentas de auxílio a atividade de avaliação do comportamento do programa durante a sua fase operacional. O Gerenciador de Falhas de Operação e o Analisador de Comportamento Operacional. A Figura IV.6 apresenta o menu da opção Medição.

Gerenciador de Falhas de Operação	F9
Analisador de Comportamento	F10

Fig. IV.6

Menu da opção Medição

. item Gerenciador de Falhas de Operação- ativa a ferramenta que controla as informações relativas a ocorrência de falhas de operação;
. item Analisador de Comportamento- ativa a ferramenta que permitirá realizar o estudo do comportamento do programa durante a sua fase operacional.

f) Opção Controles

Esta opção permite que sejam realizadas operações de manutenção do ambiente, tais como: criação e destruição de senhas de acesso, controle de prazos limites para a execução de determinadas atividades, estatísticas sobre os projetos encerrados, atualizações nas ferramentas (novos formulários, modelos de estimação, métricas, etc.), emissão de laudos técnicos e pesquisas diversas. A Figura IV.7 mostra o menu da opção Controles.

Senhas	ALT+S
Prazos	ALT+P
Estatísticas	ALT+E
Atualizações	ALT+A
Laudos	ALT+L
Pesquisas	ALT+Q

Fig. IV.7

Menu da opção Controles

. item Senhas- permite o controle de acesso ao ambiente através da utilização de senhas;
. item Prazos- identifica os atrasos que ocorram durante o processo de correção de falhas;
. item Estatísticas- fornece informações sobre o estado dos diversos projetos encerrados e em andamento;
. item Atualizações- permite que novos atributos de qualidade

sejam especificados, assim como novos formulários de predição, novos modelos estimativos, etc.;

. item Laudos- fornece relatórios técnicos sobre a confiabilidade do projeto durante todo o seu ciclo de vida;

. item Pesquisas- permite que determinadas informações sobre o processo de avaliação da confiabilidade sejam encontradas e listadas com facilidade.

4.2.2.2- Interface do SMC com o usuário

A escolha da opção Medição do menu do ambiente prestime permite que sejam ativadas duas ferramentas, a saber:

a) Gerenciador de Falhas de Operação

Esta ferramenta tem o objetivo de gerenciar as informações relativa à descoberta e correção de falhas, controlando a interação entre as diversas áreas envolvidas com este processo. A Figura IV.8 apresenta a ferramenta.

A janela do Gerente de Falhas apresenta sete opções (Operações, Abrir, Diagnóstico, Solução, Validação, Liberação e Erros). Com exceção da opção Operações todas as outras opções quando ativadas apresentam um quadro de diálogo que relaciona cada uma delas com uma determinada seção do registro de investigação de falha. A seguir temos a descrição de cada uma das opções.

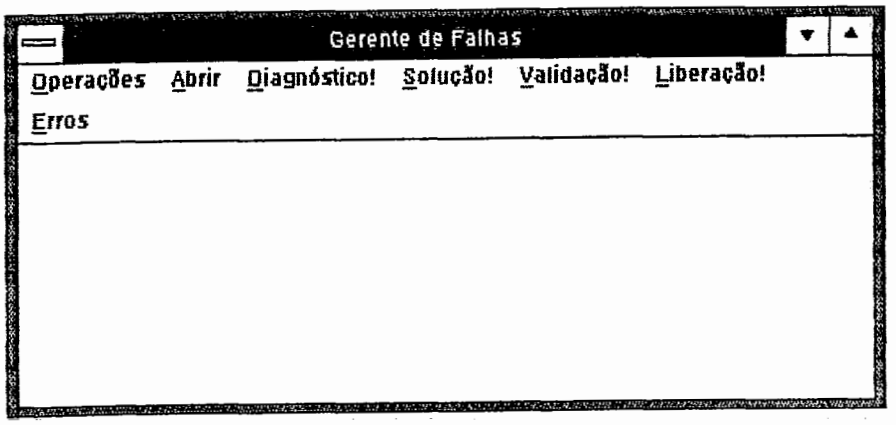


Fig. IV.8

Gerenciador de Falhas de Operação

a.1) Opção Operações

Esta opção permite que os registros de investigação possam ser criados, removidos, editados, localizados e listados. A figura IV.9 mostra o menu da opção Operações.

Novo	^N
Abrir	^A
Editar	^E
Localizar	^L
Remover	^R
Listar	^I

Fig. IV.9

Menu da opção Operações

. item Novo- permite a criação de um novo registro de investigação;
. item Abrir- permite que um registro existente seja carregado;
. item Editar- permite que o registro carregado possa ser editado;
. item Localizar- permite que um registro seja localizado a partir de determinadas informações;
. item Remover- permite que um determinado registro seja removido;
. item Listar- permite que um ou mais registros sejam listados integralmente ou em parte.

a.2) Opção Abrir

É apresentada nesta opção um menu com dois itens, cadastramento e configuração. A ativação destes itens produz como resultado os quadros de diálogo mostrados nas Figuras IV.10 e IV.11 respectivamente.

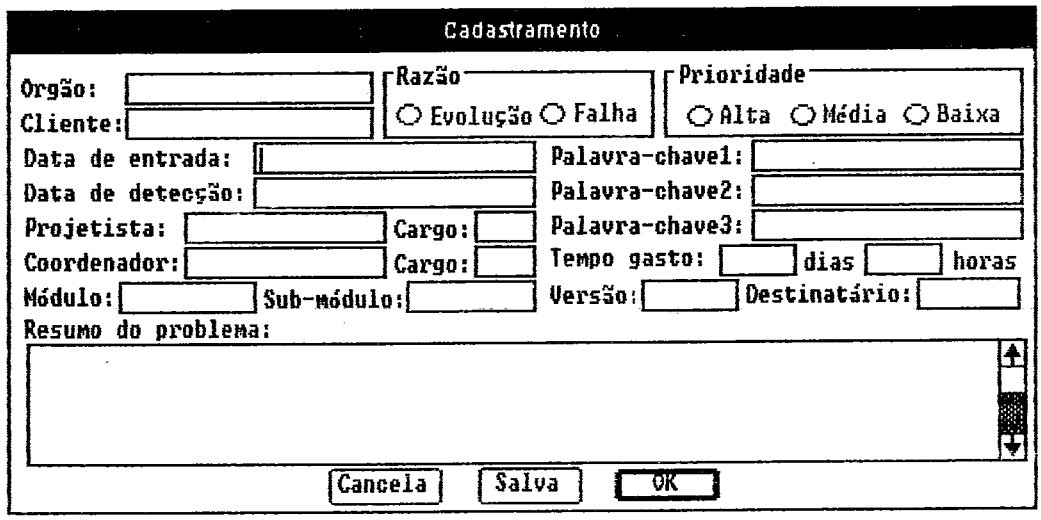


Fig. IV.10

Quadro de diálogo da opção cadastramento

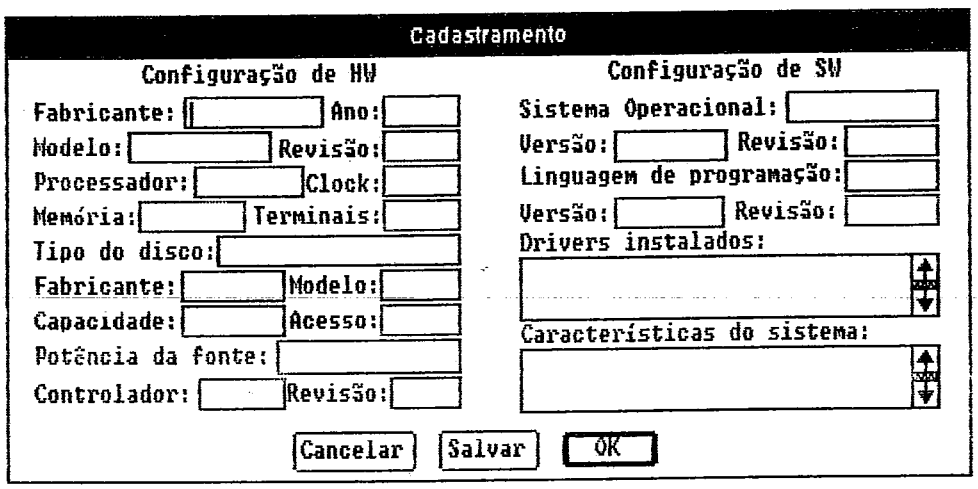


Fig. IV.11

Quadro de diálogo da opção configuração

a.3) Opção Diagnóstico

Esta opção apresenta um quadro de diálogo que contém as informações pertinentes a atividade de diagnóstico da falha. A Figura IV.12 apresenta tal quadro.

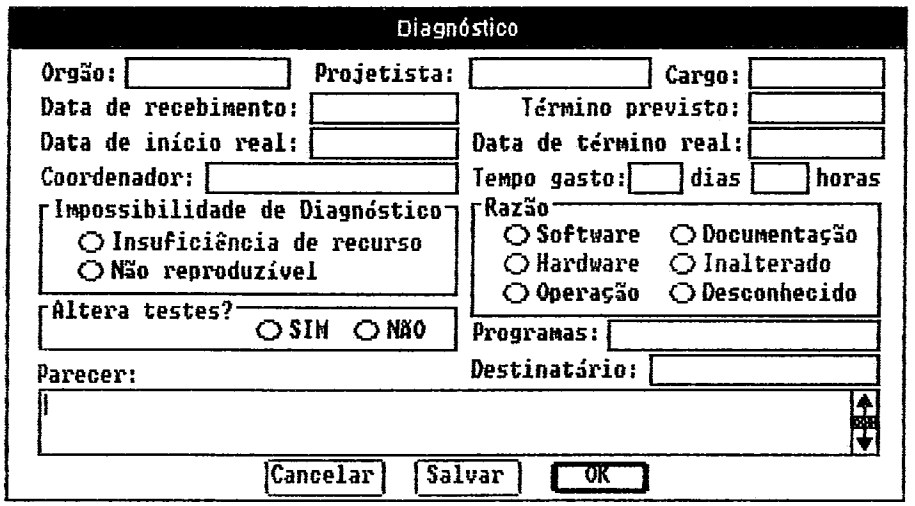


Fig. IV.12

Quadro de diálogo da opção Diagnóstico

a.4) Opção Solução

Esta opção apresenta um quadro de diálogo que contém as informações pertinentes a atividade de correção da falha. A Figura IV.13 apresenta tal quadro.

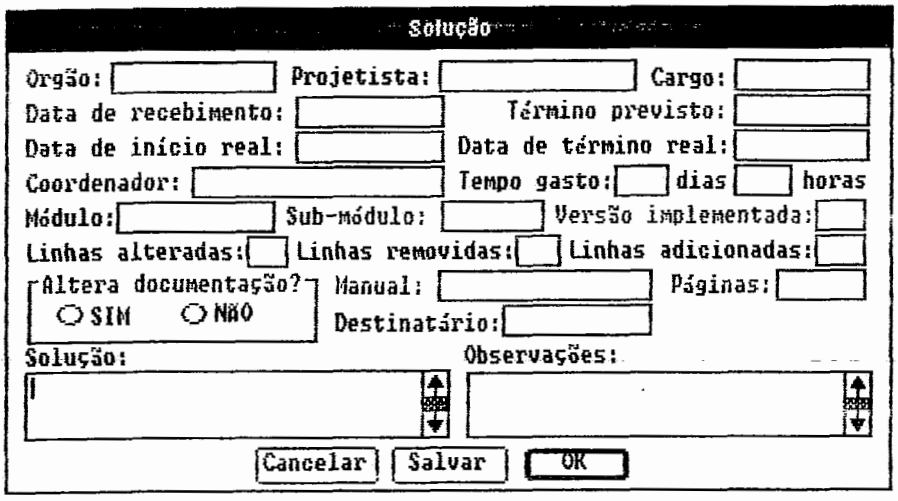


Fig. IV.13

Quadro de diálogo da opção Solução

a.5) Opção Validação

Esta opção apresenta um quadro de diálogo que contém as informações pertinentes a atividade de validação da correção. A Figura IV.14 apresenta tal quadro.

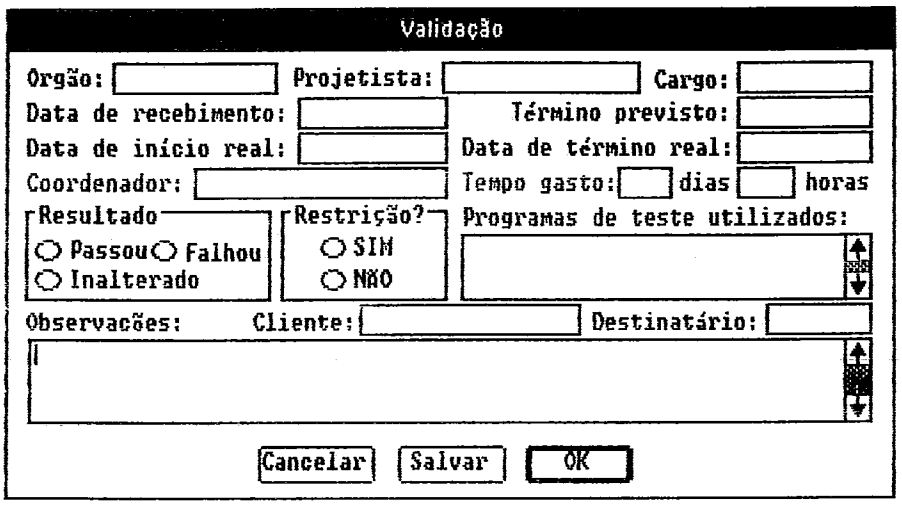


Fig. IV.14

Quadro de diálogo da opção Validação

a.6) Opção Liberação

Esta opção apresenta um quadro de diálogo que contém as informações pertinentes a atividade de liberação da correção. A Figura IV.15 apresenta tal quadro

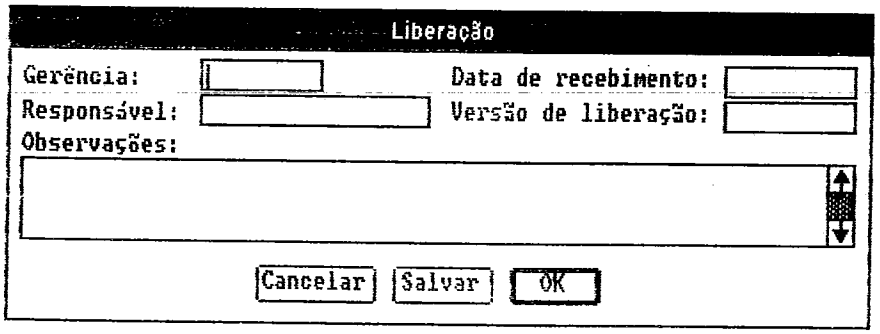


Fig. IV.15

Quadro de diálogo da opção Liberação

a.7) Opção Erros

A ativação desta opção faz com que seja apresentado um menu com três itens (origem, atributo e tipo). A Figura IV.16 apresenta o menu desta opção.

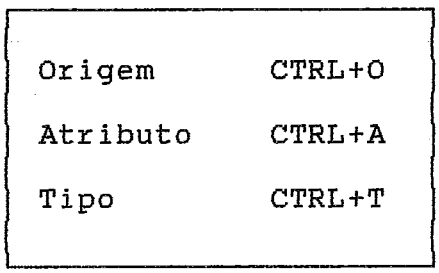


Fig. IV.16

Menu da opção Erros

. item Origem- apresenta um quadro de diálogo onde os órgãos responsáveis pelo suporte, validação e desenvolvimento, junto à gerência de projeto identificarão a fase do ciclo de vida na qual o erro se originou. A Figura IV.17 mostra o quadro de diálogo mencionado;
. item Atributo- semelhante ao item anterior com a diferença que neste caso é apontado o atributo de qualidade que foi afetado devido a ocorrência da falha;
. item Tipo- semelhante ao item anterior com a diferença que neste caso é apontado o tipo de erro ocorrido.

O quadro de diálogo é igual para os três itens, com exceção do conteúdo do "list box", que conterá respectivamente para o 1o.

2o. e 3o. itens as fases do ciclo de vida, os atributos de qualidade e os tipos de erros.

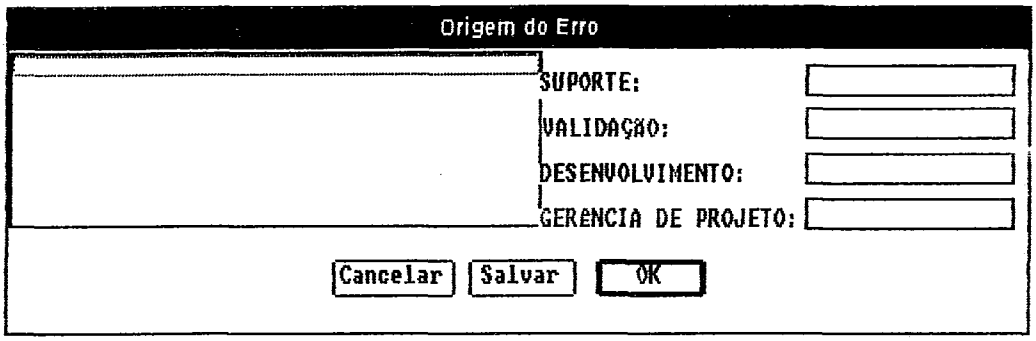


Fig. IV.17

Quadro de diálogo da opção Origem

b) Analisador de Comportamento Operacional

Esta ferramenta apresenta de forma gráfica o resultado da análise das falhas até então ocorridas. Estas falhas permitirão que a curva da intensidade de falhas instantânea seja traçada segundo um modelo matemático pré-definido. A Figura IV.18 mostra um exemplo de uma curva traçada com base na falhas ocorridas no início da operação do programa.

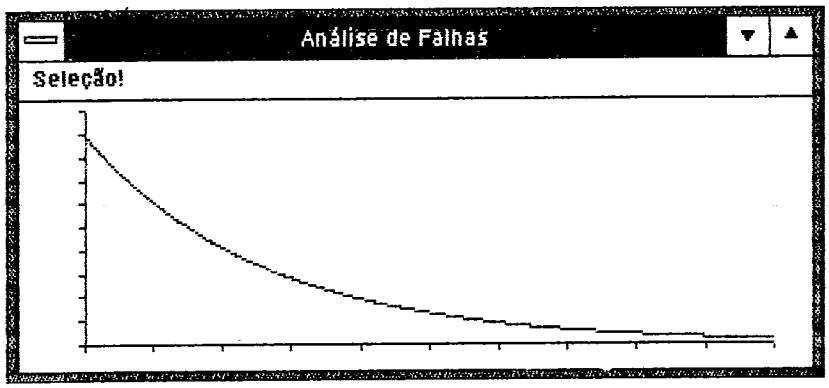


Fig. IV.18

Analisador de Comportamento Operacional

4.2.2.3- Interface com o software

O ambiente PRESTIME e conseqüentemente o SMC devem se executados em um ambiente de software que ofereça uma interface com o usuário eficiente e poderosa, isto é, devem estar disponíveis neste ambiente recursos que facilitem a manipulação de janelas, a apresentação de gráficos e a operação com dispositivos do tipo "mouse". O ambiente deve também oferecer recursos que provavelmente serão utilizados por evoluções de PRESTIME, tais como operação em rede local e a longa distância, já que a atividade de suporte poderá ser realizada em diferentes cidades (filiais da organização) e a realização dos testes envolvendo a utilização de diversos equipamentos interligados.

4.2.2.4- Interface com o hardware

Os equipamentos a serem utilizados pelo PRESTIME devem ser do tipo IBM-AT, bastante utilizado pelo mercado e de fácil operação.

A interface gráfica deve possuir uma resolução igual ou superior a oferecida pelo padrão EGA. Deve também estar disponivel uma unidade de disco rígido com capacidade igual ou superior a 20Moctetos. Estas são necessidades impostas pelos ambientes WINDOWS e ACTOR (ver maiores detalhes no item 4.3 deste documento).

4.2.3- Requisitos de desempenho

A existência de diversas camadas de software (sistema operacional, sistema de manipulação de janelas e um sistema de suporte a programação orientada a objetos), necessárias a operação do ambiente PRESTIME, torna necessário que o equipamento seja dotado de um poder de processamento não inferior a 2 MIPS e compatível com as necessidades de memória do sistema (não inferior a 2Moctetos), o que configura a necessidade de computadores do tipo IBM-AT com relógio igual ou superior a 12Mhz.

4.2.4- Requisitos de qualidade

a- Evolutibilidade

O SMC deve ser implementado de modo a permitir a inclusão de novas características (deve ser construído utilizando programação orientada a objetos), principalmente a utilização de outros tratamentos matemáticos para a análise de falhas.

b- Modificabilidade

O SMC deve ser implementado de modo a permitir a alteração de suas características com facilidade, especialmente se forem necessárias modificações nos quadros de diálogo do Gerenciador de Falhas de Operação.

c- Segurança

O SMC deve ser implementado de modo a não permitir que pessoas ou órgãos não autorizados tenham acesso e possibilidade de incluir, alterar, remover ou consultar informações não devem estar disponíveis a eles.

d- Interface Amigável

O SMC deve ser implementado levando em consideração os aspectos que facilitem a interação com usuário tais como, a utilização de um ambiente gráfico e a disponibilidade de telas de auxílio.

4.2.5- Requisitos funcionais

4.2.5.1- Diagrama de Fluxo de Dados

A Figura III.9 apresenta o diagrama de fluxo de dados do sistema de medição da confiabilidade, que é composto por seis processos e dois depósitos de dados. A descrição deste fluxo é feita no item 3.6.1. A seguir temos a descrição dos processos e depósitos de dados.

4.2.5.2- Descrição dos Processos

4.2.5.2.1- Elaborar Relatório de Investigação

Realiza o cadastramento de novos problemas a fim de que os mesmos sejam armazenados para a futura análise e correção por parte dos responsáveis pelo desenvolvimento. Informa à manutenção a abertura do relatório e inicia o processo de alteração dos casos de teste a fim de eliminar a transparência existente.

4.2.5.2.2- Analisar, Corrigir e Iniciar Alteração

Diagnostica o problema e encaminha ao órgão competente para que a correção seja efetuada. Responde ao relatório de investigação e encaminha a proposta de solução para os responsáveis pela atividade de teste, a fim de que tal solução seja certificada.

4.2.5.2.3- Corrigir Procedimentos de Teste

Analisa os efeitos da falha e elabora casos de teste de tal forma que quando uma nova versão do programa for liberada para testes, os mesmos estarão preparados para identificar falhas semelhantes àquelas que os casos de teste utilizados até então não foram capazes de detectar.

4.2.5.2.4- Medir a Confiabilidade Operacional

Calcula a melhor curva (segundo um modelo matemático) que represente o comportamento da intensidade de falhas instantânea ao longo da fase operacional do programa. Compara os resultados com as expectativas e, caso hajam conflitos, encaminha-os para análise com vistas a realimentar o processo de avaliação da confiabilidade.

4.2.5.2.5- Analisar Processos de Avaliação e Modelos

Identifica inconsistências existentes entre o comportamento real do programa no seu ambiente de operação e o comportamento esperado para o mesmo com base nos processos de avaliação utilizados para predição da confiabilidade durante as etapas de especificação de requisitos, especificação de projeto e codificação, alterando-os de forma a "sintoniza-los" com os resultados reais. Da mesma forma, as informações geradas pelos modelos estatisticos utilizados na estimação da confiabilidade são confrontadas com os dados da fase operacional do programa.

4.2.5.2.6- Certificar Alterações

Avalia se as correções efetuadas realmente resolvem o problema detectado e, em caso positivo, encaminha a versão do programa corrigida para a manutenção que se encarregará de distribuí-la aos usuários. Caso a correção não tenha sido bem sucedida este fato é relatado ao responsáveis através do relatório de investigação

4.2.5.3- Descrição dos Depósitos de Dados

4.2.5.3.1- Alterações de Projeto

São as informações referentes a todo o processo de alteração, englobando as falhas ocorridas, diagnósticos, correções efetuadas, testes e liberações.

4.2.5.3.2- Modelos de Medição da Confiabilidade

São as equações referentes aos modelos matemáticos utilizados para aproximar a melhor curva que represente o comportamento do programa durante a sua fase operacional.

4.2.5.4- Modelo de Objetos

Seguindo o modelo proposto por COAD e YOURDON [87] apresentaremos a seguir os diagramas referentes as duas ferramentas do SMC.

4.2.5.4.1- Gerenciador de Falhas de Operação

a) Camada de assunto

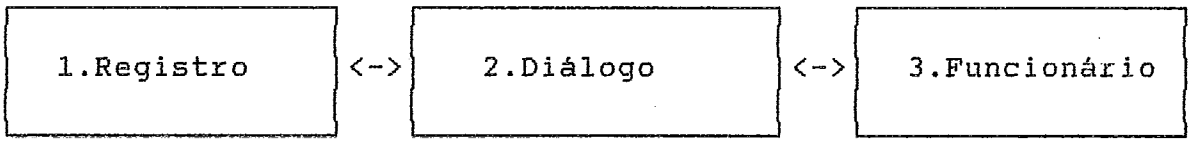


Fig. IV.19

Camada de assunto do gerenciador de falhas de operação

b) Camada de atributos

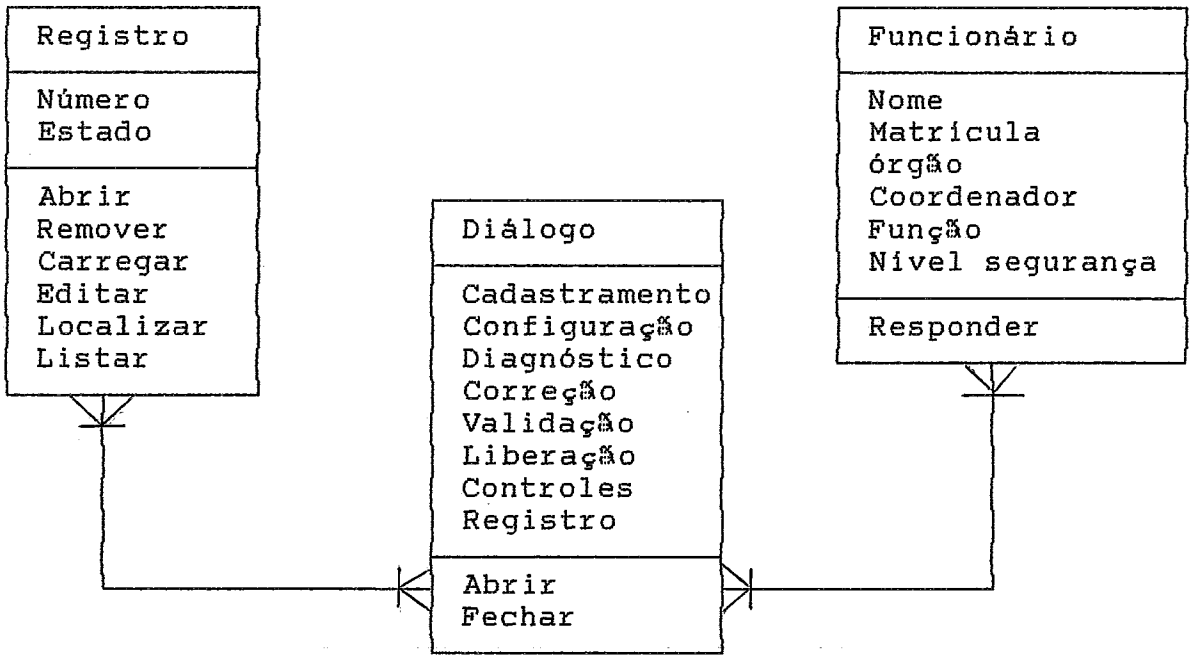


Fig. IV.20

Camada de atributos do gerenciador de falhas de operação

4.2.5.4.2- Analisador de Comportamento Operacional

a) Camada de assunto

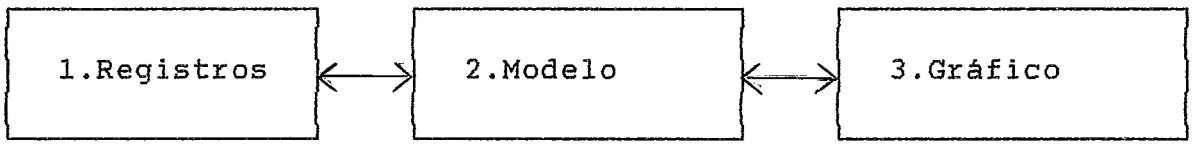


Fig. IV.21

Camada de Assunto do Analisador de Falhas de Operação

b) Camada de Atributos

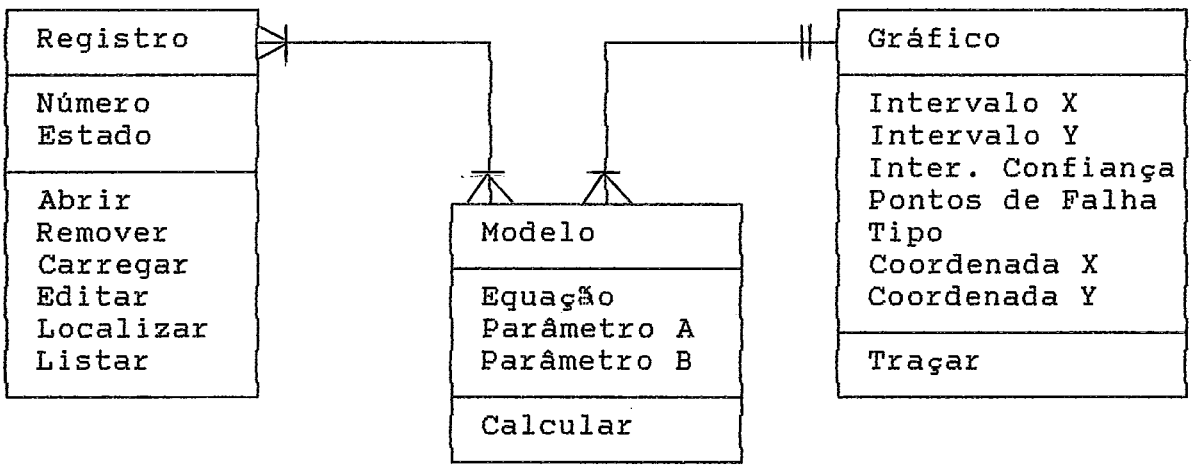


Fig. IV.22

Camada de atributos do Analisador de Falhas de Operação

4.3- Aspectos de implementação

Deve ser seguida a primeira alternativa do modelo proposto por Rotenberg [94], que sugere duas alternativas para a fase de especificação. A primeira é utilizar DFD'S e transformá-los, em seguida, para uma notação que se expressa por objetos. A segunda alternativa se refere a utilização direta de alguma

técnica que permita expressar diretamente descrições e relações entre objetos. A Figura IV.23 mostra o esquema do modelo.

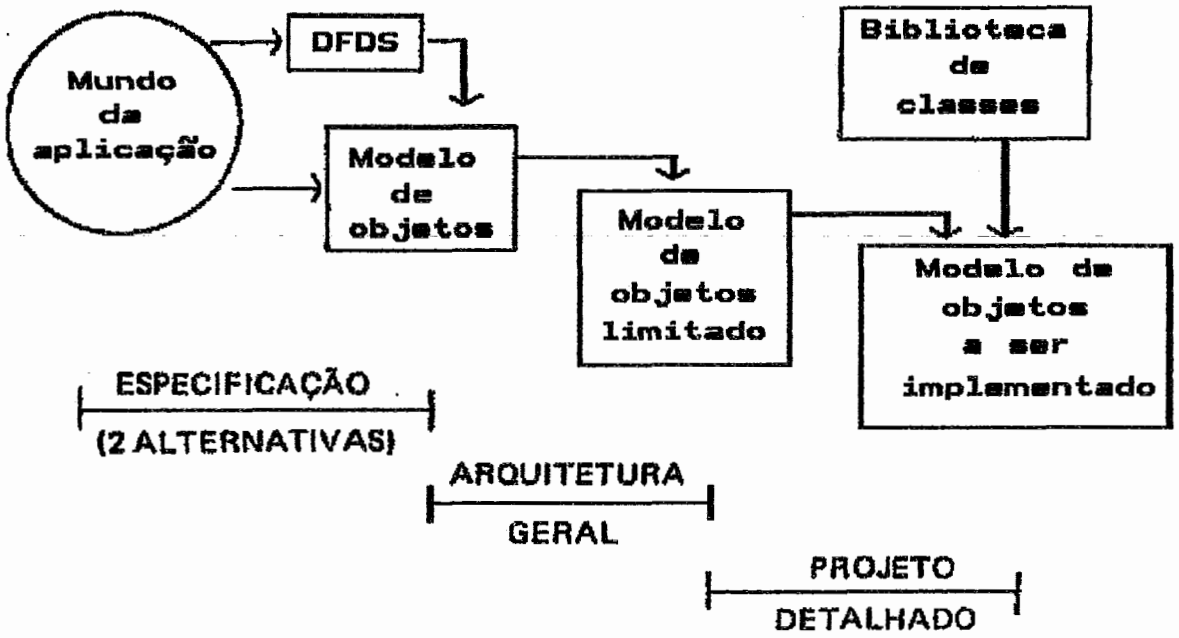


Fig. IV.23

Modelo de Desenvolvimento de Rotenberg

Seguindo a primeira alternativa deve-se iniciar a especificação com a elaboração de DFD'S até o nível de detalhe que o projetista julgar adequado, aplicando técnicas e notações como os propostos por GANE e SARSON [95]. Em seguida deve-se efetuar uma transformação, baseada na idéia de identificar objetos como originando de depósitos de dados em conjunto com o depósito correspondente, no nível mais detalhado da hierarquia de DFD'S. Resumindo [96], a transformação deve seguir os seguintes passos:

. Identificar Candidatos a Objetos (CO'S)
  (Analisar depósitos de dados);
. Identificar operações associadas a cada CO
  (Analisar processos e identificar a quais objetos potenciais a associação será mais natural.

Quanto às questões relacionadas diretamente com a fase de codificação, a seguinte consideração deve ser feita: o SMC será implementado utilizando-se o ambiente operacional WINDOWS-MICROSOFT e o ambiente de desenvolvimento orientado a objetos ACTOR.

CAPÍTULO V

CONCLUSÃO

Este trabalho teve o objetivo de oferecer mecanismos que garantam a qualidade de programas através de um acompanhamento sistemático do seu comportamento ao longo de todo o ciclo de vida, desde as fases iniciais de concepção até sua operação final. Foi realizado um estudo da literatura no que se refere às proposições de métricas para a predição da confiabilidade de software, aos modelos estatísticos utilizados para a sua estimação e aos procedimentos necessários para a sua medição na fase operacional. Um ambiente automatizado foi proposto com o objetivo de integrar em um único ambiente um conjunto de ferramentas úteis para avaliação da confiabilidade, ou seja, predizer a confiabilidade nas etapas de especificação, projeto e codificação, estimá-la na fase de testes e medi-la na fase operacional.

Atualmente, pouca atenção é dirigida no sentido de especificar quantitativamente a qualidade desejada. Isto é explicado, em parte, pelo fato da produção de software ainda ser com frequência encarada de forma artesanal [17].

Sem a disponibilidade de medidas que permitam avaliar a qualidade do software, as decisões são tomadas sem a certeza de sua eficácia e são, em grande parte, resultado do sentimento ou intuição baseada em algum nível de experiência do responsável pelo projeto. Portanto, indústrias que estejam preocupadas com a produção de produtos de alta qualidade devem

estabelecer um nível aceitável de qualidade e, através de mecanismos específicos, como os propostos nesta tese, assegurar que este nível de qualidade seja mantido.

Para que seja possível produzir-se programas confiáveis, dentro de prazos e custos pré-determinados e a níveis aceitáveis de preço e desempenho, é necessário que exista um acompanhamento da qualidade durante todas as etapas do projeto em conjunto com um planejamento coerente com os recursos tecnológicos e financeiros disponíveis.

O método apresentado para avaliar a confiabilidade de software ao longo de seu processo de desenvolvimento pretende, não só permitir medir o tempo de maturação de um produto, como também ajudar a aliviar os problemas e preocupações decorrentes do esforço de desenvolvimento de programas cada vez mais complexos. Como o esforço em garantir a confiabilidade alcançada, e portanto a qualidade, deve ser preventivo e não corretivo, identificando problemas antecipadamente teremos um produto mais confiável, dentro das estimativas de custo e sem atrasos no cronograma.

Um artigo baseado nesta tese foi aceito e apresentado no XXIII Congresso Nacional de Informática (SUCESU'90) em agosto de 1990, no Rio de Janeiro.

Dentre as contribuições deste trabalho podemos citar: a extensa revisão da literatura no que refere a proposição de métricas preditivas e de modelos estatísticos para estimação

da confiabilidade, a proposta de uma documentação específica para acompanhamento de falhas operacionais no campo e um conjunto de ferramentas automatizadas úteis às atividades de predição, estimação e medição da confiabilidade. Entre as ferramentas aqui propostas, já se encontra implementado uma versão protótipo do Especificador de Ambientes de Desenvolvimento e um protótipo do Sistema de Medição da Confiabilidade. O Sistema de Estimação da Confiabilidade vem sendo desenvolvido no Instituto Militar de Engenharia (IME) como tese de mestrado.

Como perspetivas futuras para evoluções ou novos trabalhos, citamos a elaboração definitiva dos dois protótipos mencionados no parágrafo anterior e a construção das ferramentas para predição da confiabilidade (sistemas especialistas para avaliação de especificações de requisitos, especificações de projeto e código fonte).

REFERÊNCIAS BIBLIOGRÁFICAS

[1] J. H. DOBBINS, Software Reliability Management, Handbook of Software Quality Assurance, New York, Van Nostrand, 385-407, 1987.

[2] H. HECHT, Measurement, estimation, and prediction of software reliability, Software Engineering Technology, Vol.2: 209-224, 1977.

[3] J. P. CAVANO, Toward High Confidence Software, IEEE Trans. on Software Engineering, SE-11 (12), 1449-1455, Dez. 1985.

[4] T. BOWEN, G. WIGLE e J. TSAI, Specification of Software Quality Attributes, RADC-TR-85-37, Out. 1984.

[5] E. PRESSON, Software Test Guidebook, RADC-TR-84-53, Mar. 1984.

[6] E. SOISTMAN e K. RAGSDALE, Impact of Hardware / Software Faults on System Reliability, RADC-TR-228-85, Abr. 1985.

[7] J. VINCENT, A. WATERS e J. SINCLAIR, Software Quality Assurance- Practice and Implementation, New Jersey, Prentice Hall, 1988, 409pp.

[8] IEEE, IEEE/P982- Standard for Measures to Produce Reliable Software (draft), Software Reliability Measurement Working Group of the Software Engineering Standards Sub-commitee, IEEE Computer Society, Mar. 1987.

[9] IEEE, IEEE/P1061- Standard for Software Quality Metrics Methodology- draft, IEEE Quality Metrics Standard Commitee, Jul. 1987.

[10] U. S. Dept. of Defense, DOD-STD-2168- Military Standard Software Quality Evaluation Program (draft), Washington D. C., 1985.

[11] U. S. Joint Logistics Command, DOD-STD-2167- Military Standard- Defense System Software Development, Washington D. C., 1985.

[12] R. L. FERREIRA, A Relação entre a Natureza da Aplicação e os Fatores de Qualidade de Software, Tese Msc., IME, Rio de Janeiro, 1987.

[13] K. S. MENDIS, Quantifying Software Quality, Quality Progress, 18-22, Mai. 1982.

[14] A. R. C. da ROCHA, Análise e Projeto Estruturado de Sistemas, Rio de Janeiro, Campus, 1987, 141pp.

[15] J. D. MUSA, A. IANNINO, K. OKUMOTO, Software Reliability Measurement, Prediction, Application, Singapore, MacGraw-Hill, 1987, 621pp.

[16] B. W. BOEHM, J. R. BROWN, H. KASPAR, M. LIPOW, G. J. MacLEOD and M. J. MERRITT, Characteristics of Software Quality, Amsterdam, North-Holland, 1978.

[17] J. McCALL, An Introduction of Software Quality Metrics, in Software Quality Management, Petrocelli, 1979.

[18] C. E. C. BEAUFOND, Verificação e validação de Software na fase de especificação de requisitos, Tese Msc., Rio de Janeiro, COPPE/UFRJ, 1987, 244pp.

[19] J. M. NEIGHBORS, Software Development Using Components, Phd Thesis, Departament of Information and Computer Science, Unviersity of California, Irivne, 1981.

[20] C. J. dos SANTOS, Um Ambiente de Apoio Automatizado para Desenvolvimento de Software Básico, Tese Msc., Rio de Janeiro, COPPE/UFRJ, 1987, 194pp.

[21] R. E. FAIRLEY, Software Engineering Concepts, Singapore, MacGraw-Hill, 1985, 364pp.

[22] P. A. L. da SILVA, Controle da Qualidade de Software- Conceitos Básicos de Probabilidade e Estatistica, GKL Treinamento Empresarial Avançado.

[23] J. M. JURAN, "Basic Concepts" em Quality Control Handbook, New York, MacGraw-Hill, 1979.

[24] A. R. C. da ROCHA, T. C. de AGUIAR, J. M. de SOUZA e C. D'IPOLITTO, O Meta-Ambiente da Estação TABA, Rel.Técnico, Rio de Janeiro, COPPE/UFRJ, março 1989, 18pp.

[25] J. R. ADAMS, Measurement of Software Reliability (Panel), Proceedings Annual Reliability and Maintainability Symposium, IEEE, 201-203, 1983.

[26] B. W. BOEHM, Software Engineering Economics, Prenctice Hall, , 1981.

[27] B. W. BOEHM, Verifying and Validation Software Requirements and Design Specifications, IEEE Software, Vol.1, No 1, jan. 1984.

[28] S. H. GLOSS-SOLER, The DACS Glossary, Rome Air Development Center, IIT Research Institute, out. 1979, 147pp..

[29] A. C. C. T. de FREITAS, M. F. BAGUT, A. R. C. da ROCHA, Caracteristicas de Qualidade de Programas, ES-83/85, Rio de Janeiro, COPPE/UFRJ, 1985

[30] R. L. GLASS, Software Reliability Guidebook-Prenctice Hall, , 1979.

[31] IEEE, ANSI/IEEE Std 729-1983, Glossary of Software Engineering Terminology, Software Engineering Technical Commitee of the IEEE Computer Society, ago. 1983, 38pp.

[32] IEEE, ANSI/IEEE Std 830-1984, Guide to Software Requirements Especifications, IEEE Computer Society, New York, 1984, 24pp.

[33] M. PAGE-JONES, The practical guide to structured systems design, Yourdon Inc., New York, 1980, 396pp.

[34] J. D. MUSA, A Theory of Software Reliability and its Application, IEEE Trans. on Software Engineering, set. 1975, pp. 312-327.

[35] J. D. MUSA, Software reliability measures applied to system engineering, AFIPS Proceedings, National Computer Conference, 1979, pp. 941-946.

[36] G. J. MYERS, Composite/Structured Design, Van Nostrand Reinhold, New York, 1978.

[37] E. M. NASCIMENTO, A. R. C. da ROCHA, Manual para avaliação da Qualidade de Programas, Subfator modularidade, ES-112/86, Rio de Janeiro, COPPE/UFRJ, jun. 1986, 15pp.

[38] R. S. PRESSMAN, Software Engineering- A Practioner's Approach, 2a. Ed., Mac-Graw Hill, Singapore, 1987, 567pp.

[39] RADC, Quantitative Software Models, Data Analysis Center for Software, Rome Air Development Center, mar. 1979.

[40] N. F. SCHNEIDEWIND, The Applicability of Hardware Reliability Principles to Computer Software, Software Quality Management, New York, Petrocelli Books, 1979.

[41] R. N. CHARETTE, Software Engineering Environments - Concepts and Technology, New York, McGraw-Hill, 1987, 407pp.

[42] J. VINCENT, A. WATERS, J. SINCLAIR, Software Quality Assurance, Practice and Implementation, Volume 1, New Jersey, Prenctice Hall, 1988, 409pp.

[43] G. J. MYERS, Software Reliability - Principles and Practices, New York, John Wiley & Sons, 1976, 360pp.

[44] P. N. MISRA, Software Reliability Analysis, IBM Systems Journal, Vol. 22 NO.3, pp. 262-270, 1983

[45] L. J. ARTHUR, Measuring Programmer Productivity and Software Quality, New York, John Wiley & Sons, 1985, 292pp.

[46] M. L. SHOOMAN, Software Engineering - Design, Reliability and Management, New York, MacGraw Hill, 1983, 683pp.

[47] J. R. DUNHAM, Experiments in Software Reliability: Life Critical Applications, IEEE Trans. os Software Engineering, Vol. SE-12 NO. 1, pp. 110-123, 1986.

[48] B. BEIZER, Software System Testing and Quality Assurance, New York, Van Nostrand, 1984, 358pp.

[49] G. J. SCHICK e R. W. WOLVERTON, "An analysis of Competing Software Reliability Models", IEEE Transactions on Software Engineering, Volume SE-4, Number 2, Mar 1978.

[50] A. N. SUKERT, "A Software Reliability Modeling Study", RADC-TR-76-247, ago 1976. RADC-TR-76-24, ago 1977.

[51] A. N. SUKERT, "An Investigation of Software Reliability Models", Proceedings 77 Annual Reliability

and Maintainability Symposium, Philadelphia, pp. 78-84, Jan 1977.

[52] P. B. MORANDA, "Prediction of Software Reliability During Debugging", Proceedings 1975 Annual Reliability and Maintainability Symposium, pp. 327-332, Jan 1975.

[53] P. B. MORANDA, Z. JELINSKI, "Software Reliability Predictions", 6th Trennial World Congress IFAC, em Proceedings IFAC/75, ago. 1975.

[54] M. L. SHOOMAN, "Probabilistic Models for Software Reliability Prediction", 1972 International Symposium on Fault-Tolerant Computing, IEEE, Jun 1972.

[55] M. L. SHOOMAN, "Software Reliability: Measurement and Models", Proceedings 1975 Annual Reliability and Maintainability Synposium, pp. 485-491, Jan. 1975

[56] M. L. SHOOMAN, "Software Reliability: Analysis and Prediction", em Integrity in Electronic Flight Control Systems, pp. 7/1-17. Report AGARD AG224, AGARD, Neuilly sur Seine, France, Abr 1977.

[57] J. C. DICKSON, J. L. Hesse, A. C. Kientz, M. L. Shooman, "Quantitative Analysis of Software Reliability", proceedings 1972 Annual Reliability and Maintainability Synposium, IEEE, pp. 148-157, Jan 1972.

[58] M. L. Shooman, S. Natarajan, "Effect of Manpower Development and Bug Generation on Software Error Models", RADC-TR-76-400, Jan 1977.

[59] A. L. GOEL, K. OKUMOTO, "Bayersian Software Prediction Models. An Imperfect Debugging Model for Reliability and Other Quantitative Measures for Software Systems", RADC-TR-78-155, jul 1978.

[60] B. LITTLEWOOD, J. L. VERRAL, "A Bayersian Reliability Model with Stochastically Monotone Failure Rate", IEEE Transactions on Reliability, Volume R-23, No. 2, pp. 108-114, Jun 1974.

[61] B. LITTLEWOOD, J. L. VERRAL, "A Bayersian Reliability Growth Model for Computer Software", Applied Statistics, Volume 12, No. 3, pp. 12-27, 1973

[62] M. L. SHOOMAN, A. K. TRIVERDI, "Computer Software Reliability, Many State Markov Modeling Techniques", Polytechnic Institute, New York, Interim Report, RADC-TR-75-169, Jul 1975.

[63] M. L. SHOOMAN, A. K. TRIVERDI, "A Many-State Markov Model for Computer Software Performance Parameters", IEEE Transactions Reliability, Vol. R25, No. 2, pp. 66-68, Jun 1976.

[64] B. LITTLEWOOD, "A Reliability Model for Markov Structured Software", Applied Statistics, Volume 24, No. 2, pp. 171-177, 1975.

[65] B. LITTLEWOOD, "A Semi-Markov Model for Software Reliability with Failure Costs", Proceedings of Symposium on Computer Software Engineering, pp. 281-300, Abr 1976.

[66] P. B. MORANDA, "Quantitative Methods for Software Reliability Measurements", Defense Documentation Center, Alexandria, 188pp, Dez 1976.

[67] M. L. SHOOMAN, "Structural Models for Software Reliability Prediction", Second International Conference on Software Engineering, IEEE, pp. 268-280, Out 1976.

[68] E. C. NELSON, "Estimating Software Reliability from Test Data", Microeletronics and Reliability, Volume 17, No.1, pp. 67-74, Jan 1978.

[69] T. A. THAYER, "Software Reliability Study", RADC-TR-76-238, Final Technical Report, pp. 5-13/6-49, Ago 1976

[70] D. K. LLOYD, M. LIPOW, "Reliability: Management Methods and Mathematics", Second Edition, 1977.

[71] H. HECHT, "Measurement, Estimation and Prediction of Software Reliability", NASA-CR-145135, Aerospace Corp., 43pp., Jan 1977.

[72] P. PARDAL, "Introdução a Estatistica com aplicações à indústria", Escola de Engenharia da UFRJ, D.A.P., 205pp., Jan 1980.

[73] W. L. WAGONER, "The Final Report on a Software Reliability Measurement Study", Aerospace Corporation, Report TOR-0074 (4112)-1, 1973.

[74] A. L. GOEL, "Software Reliability Models: Assumptions, Limitations, and Applicability", IEEE, IEEE Transactions on Software Engineering, Vol. SE-11, NO. 12, pp. 1411-1423, Dez 1985.

[75] C. V. RAMAMOORTHY, F. B. BASTANI, "Software Reliability - Status and Perspectives", IEEE, IEEE Transactions on Software Engineering, Vol. SE-8, NO. 4, pp. 354-371, Jul 1982.

[76] Z. JELINSKI, P. B. MORANDA, "Software Reliability Research", Statistical Computer Performance Evaluation, W. Freiberger Editors, New York, pp. 465-484, 1972.

[77] N. F. SCHNEIDEWIND, "Analysis of Error Process in Computer Software", Proceedings 1975 International Conference on Reliable Software, Los Angeles, 1975.

[78] A. L. GOEL, K. OKUMOTO, "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures, IEEE, IEEE Transactions on Reliability, Vol. R-28 (3), pp. 206-211, 1979.

[79] J. D. MUSA, Private Communication to B. Littlewood, 1979.

[80] P. A. KEILLER, B. LITTLEWOOD, D. R. MILLER, A. SOFER, "On The Quality of Software Reliability Prediction", Eletronic Systems Effectiveness and Life-Cycle Costing, NATO ASI series, F3, Springer-Verlag, Heidelberg, pp. 441-460, 1983.

[81] G. J. SCHICK, R. W. WOLVERTON, "Assessment of Software Reliability", Proceedings Operations Research, Physica-Verlag, Wurzburg-Wien, pp. 395-422, 1973.

[82] B. LITTLEWOOD, "Stochastic Reliability-Growth: A Model for Fault-Removal in Computer-Programs and Hardware Design", IEEE, IEEE Transactions on Reliability, Vol. R-30(4), pp. 313-320, 1981.

[83] S. YAMADA, M. OHBA, S. OSAKI, "S-Shaped Reliability Growth Modeling for Software Error Detection", IEEE, IEEE Transactions on Reliability, Vol. R-32(5), pp.475-478, 1983.

[84] J. D. MUSA, K. OKUMOTO, "A Logarithmic Execution Time Model for Software Reliability Measurement", Proceedings Seventh International Conference on Software Engineering, Orlando, pp. 230-288, 1984.

[85] L. H. CROW, "Reliability Analysis for Complex, Repairable Systems", Reliability And Biometry, Philadelphia, pp. 379-410, 1974.

[86] T. de MARCO, "Controlling Software Projects", Yourdon Press, 1982.

[87] P. COAD, E. YOURDON, "Object-Oriented Analisys", Yourdon Press, New Jersey, Prentice Hall, 220 pp., 1990.

[88] M. H. HALSTEAD, "Elements of Software Science", North-Holland, New-York, 1077.

[89] M. T. J. McCABE, "A Complexity Measure", IEEE Trans. of Software Engineering, 2(4), pp. 308-320, dez. 1976.

[90] T. GILB, "Software Metrics", Winthrop Publishers, Massachusetts, 281pp., 1977.

[91] SEI - Secretaria Especial de Informática, "Relatório da Comissão Especial no. 21 / Proteção de dados", Brasilia, 343pp., 1986.

[92] A. MEREY, "Automated Software Quality Assurance", IEEE Trans. on Software Engineering, IEEE, Vol SE-11 no.9, 1985.

[93] J. D. MUSA, K. OKUMOTO, "Software Reliability Models: Concepts, Classification, Comparisons, and Practice", J. K. SKWIRZYNSKI editor, Eletronic Systems Effectiveness and Life Cycle Costing, NATO ASI series, F3, Springer-Verlang, Heidelberg, pp. 395-424, 1983.

[94] H. B. ROTEMBERG, "Programação Orientada a Objetos - Um Enfoque de Engenharia de Software, Tese de Mestrado, DI/PUC, Rio de Janeiro, abril 1987.

[95] C. GANE e T. SARSON, "Análise Estruturada de Sistemas", Rio de Janeiro, LTC-Livros Técnicos e Cientificos Editora S.A., 257pp., 1983.

[96] T. Takahashi, "Introdução a Programação Orientada a Objetos", III EBAI, Curitiba, 148pp., jan. 1988.