AVALIAÇÃO DA QUALIDADE DE PROGRAMAS

Cacilda Jorge de Andrade

TESE SUBMETIDA AO CORPO DOCENTE EA COORDENAÇÃO DQS PROGRAMAS BE POS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIBABE FEDERAL ao RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSARIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIENCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Ana Regina Cavalcanti da Rocha

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

Paulo Afonso Lopes da Silva

Prof. Paulo Afonso Lopes da Silva, PhD.

Regina Celia de Souza Pereira

Prof. Regina Celia de Souza Pereira, D.Sc.

RIO DE JANEIRO - RJ - BRASIL

ABRIL DE 1.991

ANDRADE, CACILDA **JORGE** DE

Avaliação **da Qualidade de Programas** [Rio **de Janeiro3** 1991.

x, ***257*** p. 29,7 cm (COPPE/UFRJ, **M. Sc., Engenharia de Sistemas e Computação, 1991**)

Tese – **Universidade Federal do Rio de Janeiro**, COPPE

I. Engenharia de Software/Qualidade **de** programas **I.** COPPE/UFRJ II. **Titulo (série)**

Resumo da Tese Apresentada à COPPE/UFRJ coma parte dos requisitos necessárias para a obtenção da grau de Mestre em Ciências (M. Sc.).

AVALIAÇÃO DA QUALIDADE DE PROGRAMAS

Cacilda Jorge de Andrade

Abril de 1991

Orientadora: Ana Regina Cavalcanti da Rocha
Programa: Engenharia de Sistemas e Computação

O nível de qualidade de um software depende do nível de qualidade de seus produtos de desenvolvimento. Diante disto, este trabalha atua dentro da área de Controle da Qualidade de Programas, mais especificamente sobre a avaliação da qualidade de programas no que diz respeito à sua descrição, organização e representação.

A partir de um método e de uma classificação para as características de qualidade, são apresentados um Manual para avaliação e a ferramenta APFOR - Analisador Estático para Apoio à Avaliação da Qualidade de Programas FORTRAN.

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

PROGRAM QUALITY EUALUATION

Cacilda Jorge de Andrade

April, 1991

Thesis Supervisor: Ana Regina Cavalcanti da Rocha

Department: Computing and Systems Engineering

The quality of the final software depends on the quality of its products of development. Therefore, this work is focused in tke Program Quality Control area, specifically in program quality evaluation issues as deãcription, organization and representation.

Beginning with a method and classification, for the quality attributes, a Manual for evaluation and a tool called APFOR (Static Analyser for Evaluation Support of the Quality of FORTRAN Programs) are presented.

A comunidade de Informática,

Deus,

Obrigada por tudo que me das,

Obrigada por mais esta jornada,

Obrigada pela minha energia e disposição,

Obrigada pelos meus pais e meu irmão,

Obrigada pelos meus amigos pessoais,

Obrigada peles meus superiores e amigos do Bamerindus,

Obrigada pelos atuantes profissionais de outras áreas,

Obrigada pela minha orientadora de tese,

por tanto amor, atenção, força, carinho e compreensão dedicados à mim ao longo destes cinco anos de muito trabalho para elaboração de mais uma obra da Informática que é minha tese.

Muito obrigada meu Deus...

INDICE

ÍNDICE

ÍNDICE DE FIGURAS

INDICE DE FIGURAS

INDICE DE TABELAS

CAPÍTULO I

INTRODUÇÃO

1.1 - Definição e Importância

O software, ainda hoje, envolve um alto custo de produção ao longo do seu ciclo de vida, principalmente porque os erros e inadequações geralmente são propagados pelas fases de processo de desenvolvimento e detectados mais tarde, quando já se está em operação ou manutenção. Logo, e custo da correção de erros é muito maior na operação ou manutenção do que seria se estes mesmos erros fossem detectados e corrigidos desde e inicio do desenvolvimento. Para diminuir esses custos, é preciso que se desenvolva software com alto nível de qualidade.

Entre as várias definições de **Qualidade**, PERRY (1983) define-a como um conjunto estabelecide de atributos e características referentes a um item em questão, sendo um conceito multidimensional e variável do ponto de vista de cada instituição.

Desta forma, para a realização da avaliação da qualidade de um software é necessário identificar e avaliar as características que determinam a qualidade dos produtos resultantes do desenvolvimento do mesmo. Isto pode ser feito através de dois caminhos, efetuando-se um processo de Garantia de Qualidade sobre o software em processo de desenvolvimento, ou um processo de Controle da Qualidade sobre e produto pronto do software.

O processo de **Controle da Qualidade** de **Seftware** avalia cada produto pronto, já desenvolvido, de software, com o

objetivo de verificar se o mesmo está satisfazendo aos seus requisitos de qualidade pré-determinados. Este processo é feito através da aplicação das métricas de qualidade, que são medidas ou avaliações quantitativas das características de qualidade, sobre o produto que estiver em questão. Logo, ele possui a tarefa de verificar o nível de qualidade existente num produto especifico, resultante do processo de desenvolvimento do software, através da comparação das características do produto com um padrão definido por algum método de avaliação da qualidade de software, não se preocupando com a avaliação de nível de desempenho técnico (STAA, 1987).

Para que o processo de Controle da Qualidade seja realmente vantajoso, viável e utilizável , é necessário que se usem ferramentas automatizadas, auxiliadoras da aplicação do processa adotado. Isto porque a coleta manual dos dados durante uma avaliação é um processo tedioso, propensa a erro e consumidor de tempo. Logo, ferramentas automatizadas implicam na melhoria da coleta e análise dos dados oriundos da aplicação das métricas, na redução dos custos desse processo e na disponibilidade mais rápida dos resultados sempre que necessário.

O processo de **Garantia** da **Qualidade** de **Software** controla o processo de desenvolvimento de cada produto do software, de modo que o mesmo, quando pronta, satisfaça aos requisitos de qualidade pré-determinados. A Garantia de Qualidade, que depende do decorrer do processo de desenvolvimento do software, envolve procedimentos de Controle da Qualidade, um Plano de Garantia de Qualidade,

um método de desenvolvimento de software e a definição rigorosa do problema a ser resolvido (STAA, 1987).

Quanto ao conjunto de características de qualidade envolvidas num processo de avaliação da qualidade de software, este deve ser adequado ao tipo da área de aplicação, como por exemplo, áreas comercial, industrial, bancária, científica ou militar, e ainda às metas a serem atingidas pele software que estiver em questão. Ou seja, este processo deve ser um conjunto especifico de características que sejam condizentes com os objetivos do software, influenciado por suas características básicas e próprias, e dependentes da área de aplicação, devendo ainda estar de acordo com as prioridades e importâncias estabelecidas claramente pelo usuário. Diante disto, o conjunto de características de qualidade pode variar para cada tipo de área de aplicação, e ainda para cada software dentre da mesma (PERRY, 1983).

Um alto nível de qualidade deveria ser um pré-requisito para a produção de qualquer software. Entretanto, quando a preocupação com qualidade implica em grandes investimentos para conscientização e organização da equipe, isto pode não ser considerado. Logo, caberá à instituição decidir a que nível um processo de Garantia de Qualidade ou de Controle da Qualidade poderá influenciar e participar do seu processo de desenvolvimento de software. Além disso, existem algumas razões mais fortes que podem ser consideradas nesta decisão, como as seguintes (MANNS, 1988):

- para se produzir um software com alto nível de

qualidade, simplesmente;

- por implicações legais devido ao risco de falhas do software, nos casos em que este envolver perigo ou atingir um grande número de pessoas e até mesmo a própria vida humana;
- por exigência do próprio usuário em ter uma qualidade satiçfatÓria;
- quando a adoção de um desses processos já tenha demonstrado a redução de custos na operação ou manutenção do software;
- por ser uma boa estratégia de mercado a comprovada qualidade de software.

1.2 - Objetivo da tese

Devido a crescente necessidade de qualidade em software, veio a preocupação e motivação sobre a área de Controle da Qualidade de Çoftware, mais especificamente, sobre o tema avaliacão da qualidade de programas, para elaboração desta tese. O fato de ser escolhido especificamente o produto programa é por ainda haver poucos trabalhas nesta área, por ser ele um produto final de desenvolvimento de software bem palpável, e por existirem várias técnicas e algumas métricas para programas já conhecidas e depuradas que podem ser utilizadas.

Para aquisição e solidificação das informações sobre o tema desta tese, o trabalho iniciou-se com o estudo sobre controle e avaliação da qualidade de programas, a adoção de um método para avaliação da qualidade, o estabelecimento e discussão de uma elassificacão das caracteristicas de

qualidade para programas relativos apenas à descrição, organização e representação dos mesmos, para então elaborar-se um Manual com as informações básicas sobre essas caracteristicas aplicadas especificamente para programas escritos na linguagem de programação FORTRAN 77 ANÇP. O fato de ser FORTRAN 77 ANSI é por ser ela uma linguagem original, conhecida e, ainda, largamente usada, principalmente no meio científico.

Diante disto, esta tese desfechou-se na proposta de uma ferramenta para execução automatizada apenas da análise estática de programas para determinadas caracteristicas de qualidade, de modo a fazer parte e apoiar o processo de avaliação da qualidade de programa. Esta ferramenta denomina-se APFOR - ANALISADOR ESTÁTICO PARA APOIO A AVALIAÇAO DA QUALIDADE DE PROGRAMAS FORTRAN.

1.3 - Organização da tese

Esta tese é composta por cinco Capítulos e dois Anexos, cujos respectivos conteúdos estão resumidamente descritos a seguir:

CAPITULO I - INTRODUÇÃO

Este capítulo apresenta noções gerais de Controle da Qualidade de Software e discute sua importância para o êxito de projetos de desenvolvimento de software.

Apresenta também a motivação para elaboração desta tese e uma descricão sucinta do trabalho realizado ao longo da mesma.

CAPÍTULO 11 - QUALIDADE DE PROGRAMAS

Apresenta definições e conceitos gerais de Controle da Qualidade de Programas, uma revisão dos principais trabalhos em qualidade de software/programas existentes na literatura e o método para avaliacão da qualidade de software adotado nesta tese. A partir deste método e da classificação das caracteristicas de qualidade de programas estabelecida, são apresentadas e discutidas as características referentes à forma de descrição, organização e representação de um programa.

CAPITULO III - APFOR - ANALISADOR ESTATICO PARA APOIO A AVALIAÇAO DA QUALIDADE DE PROGRAMAS FORTWAN

Apresenta a definicão e descricão da ferramenta APFOR para execução da análise estática, de acordo cam as características de qualidade consideradas, sobre programas codificados em FORTRAN 77 ANSI com o intuito de fazer parte e apoiar o processo de avaliação da qualidade do mesmo.

CAPÍTULO IV - CONCLUSÃO

Apresenta conclusões e observações finais sobre a elaboracão desta tese.

ANEXO ▮ - MANUAL PARA AVALIAÇÃO DA CONFIABILIDADE DA REPRESENTAÇÃO DE PROGRAMAS FBUTRAN

Apresenta o Manual para a avaliação da qualidade específico para programas codificados em FORTRAN 77 ANÇI referentes à sua descrição, organização e representação. Neste anexo são definidas e discutidas as características de qualidade, baseadas na classificacão adotada, referentes somente ao objetivo Confiabilidade da Representação, com

seus respectivos fatores, sub-fatores, critérios, processos de avaliação, interpretações e sugestões para procedimentos de revisão.

ANEXO II - MANUAL DO USUÁRIO DO APFOR - ANALISADOR EÇTATICO PARA APOIO A AVALIAÇAO DA QUALIDADE DE PROGRAMAS FBRTRAN

Apresenta o Manual para a devida utilização da ferramenta APFOR pelo usuário. Sendo explicados como acessá-lo, trabalhar dentro dele, e que informações são fornecidas. Apresenta também um exemplo da execução do mesmo.

CAPITULO II

QUALIDADE DE PROGRAMAS

2.1 - Introdução

O objetivo deste capitulo é apresentar definições e conceitos gerais sobre Controle da Qualidade de Software e métricas de qualidade, aprofundando-se na que se refere à Controle da Qualidade de Programas e suas características de qualidade associadas. Para isso, são apresentadas definições e noções gerais de Controle da Qualidade, uma revisão das principais trabalhos em qualidade de software/programas existentes na literatura e o método para avaliação da qualidade de software adotado nesta tese. A partir deste método e da classificação das características de qualidade de programas adotada, são definidas e discutidas as características referentes especificamente à forma de descrição, representação e organização de um programa.

2.2 - Definição e Discussão

Pensando em termos de passos sequenciais, realizar o processo de Controle da Qualidade de Software significa (BASILI, 1987) e (ARTHUR, 1985):

1- Selecionar o produto do software sobre o qual vai ser realizado o Controle da Qualidade;

2- Especificar as características e métricas de qualidade a serem aplicadas e satisfeitas, podendo também definir os valores a serem atingidos para estas métricas já estabelecidas;

3- avaliar, ou seja, aplicar as métricas especificadas sobre o produto selecionado;

4- Interpretar os resultados obtidos, comparando-os com os possíveis valores pré-estabelecidos, e indicando o nível de qualidade das características;

5- Aplicar, ou seja, utilizar as informações obtidas da interpretação para tomar uma ação corretiva a fim de melhorar a qualidade de produto à um nível satisfatório ou aceitável.

Existem alguns problemas a se enfrentar para se conseguir um software com alto nível de qualidade e também para a própria prática do processo de avaliação do software, como os seguintes (BUCKLEY, 1984) e (CARD, 1988):

- falta de uma definição largamente aceita sobre Qualidade de Software;
- dificuldade na determinação de métricas de qualidade;
- geralmente na implantação, operação ou manutenção é que se determina quão bom é o software. Nestes momentos, as alterações envolvem um alto custo, e normalmente o usuário acaba aceitando um software que não cumpre adequadamente seus objetivos e necessidades devido as várias falhas ocorridas e não resolvidas;
- muitas propostas de avaliação da qualidade de software não são práticas, pois assumem que a instituição mudará prontamente sua tendência técnica simplesmente para coletar novos dados;

- pouca preocupação e cultura dos profissionais envolvidos no desenvolvimento de software em relação à Qualidade ;
- resistência à avaliação, receando que ela gere um alto custo não justificável;
- entendimento incompleto do que as métricas de software são e como podem ser usadas. A avaliação não resolve os problemas, e sim ajuda a identificá-los e fornece indicações para solucioná-los.

As métricas de qualidade são orientadas para serem aplicadas sobre os produtos de software em determinados pontos de seu processo de desenvolvimento, não sendo normas rígidas e sim indicadores do nível de qualidade. Podem ser usadas, também, para prever a extensão de uma atividade futura num projeto de software (INCE, 1988).

Quanto mais cedo forem aplicadas essas métricas, mais cedo se terá uma avaliação inicial da qualidade do software, identificando e prevenindo-se, também, de falhas que podem ser superadas, resultando, assim, numa grande economia de custo para o software.

Existem alguns procedimentos que podem ajudar no sucesso do processo de Controle da Qualidade, que são (CARD, 1988):

- atribuir a responsabilidade do processo a uma pessoa ou equipe. Se não houver alguém claramente responsável, o trabalho será bem mais difícil;
- construir um Banco de Dados comum que armazene os resultados das avaliações realizadas, formando, assim, um histórico para possíveis comparações e

referências;

- gerar e distribuir relatórios periódicos, destacando alguns procedimentos chave. As pessoas desemponham-se melhor quando sabem que estão sendo avaliadas;
- proporcionar um conjunto básico de ferramentas para facilitar as atividades de coleta de dados.

Os resultados do processe de avaliação, mais especificamente, da quantificação das métricas de qualidade de software trazem algumas vantagens, como as seguintes (CERINO, 1986):

- podem ser usados para avaliar se um software satisfaz, ou não, seus objetivos contratuaisi
- podem ser usadas para gerenciar o desenvolvimento do software, pois proporcionam meios para guiar o processo de desenvolvimento e reconhecer áreas problemáticas;
- podem ser usados para depois prever os atributas de qualidade e o custo ao longo do ciclo de vida de um software;
- o próprio processo de Controle da Qualidade proporciona um meio de se estabelecer, de se expressar e de se dar prioridade aos objetivos de um software logo no inicio de seu processo de desenvolvimento. Isso influencia significativamente a maneira na qual o software será projetado e implementado.

Para que as métricas sejam realmente úteis, elas devem possuir algumas propriedades, como as seguintes (CERINO,

1986):

- ser confiável: fornecer os mesmos resultados depois de repetidas avaliações;
- ser válida: realmente avaliar a característica a que se propõe;
- ser sensível: responsabilizar-se por mudanças no produto.

No caso da aplicação das métricas de qualidade na execução de um processo em modo manual do Controle da Qualidade de Software, podem haver alguns problemas técnicos e administrativos, como os seguintes (RADC-TR-85-37):

- as métricas podem ser, em alguns casos, um pouco imprecisas, uma vez que envolvem julgamento subjetivo e especializado, onde os dados podem ser coletados por diferentes pessoas;
- podem haver confusões nas atribuições de responsabilidades e na estrutura organizacional de uma equipe para Qualidade de Software, uma vez que ainda falta uma definição adequada para tal.

As métricas também possuem algumas limitações, como as seguintes (CONTE, 1986):

- deve-se ter cuidado na análise de dados de instituições diferentes, ou de dados coletados sob diferentes regras numa mesma instituição, no sentido de que estes sejam medidos de maneira consistente, com o mínimo de subjetividade e tenham as mesmas definições, ou melhor, estejam de acordo com definições precisas de modo a

realmente avaliarem a mesma coisa;

- muitos ambientes requerem uma calibração, ou seja, um ajuste das métricas utilizadas para reduzir a possibilidade de falha no processo de avaliação, o que as leva a não poderem ser transportadas diretamente de um ambiente para outro sem uma recalibração, senão poderão falhar no processo de avaliação;
- as métricas são valiosas para tomadas de decisão, mas não subãtituem os gerentes;
- as métricas são usadas para avaliação de um software e não para o desempenho de uma equipe técnica. Assim, não devem ser usadas para comparação, opinião ou avaliação de desenvolvedores;
- deve-se ter cuidado nas estimativas de variáveis que não sejam muito conhecidas;
- existe uma certa dificuldade e custo na computação das métricas. As mais Úteis devem ser aquelas que podem ser coletadas automaticamente ou com um pequeno esforço.

Antes de se prosseguir mais detalhadamente em qualidade de Programas, existem definições importantes, como a de Programa e de Módulo, que devem ser esclarecidas.

Um **Programa** é um plano ou rotina de código fonte expresso através de uma sequência lógica de declarações, comandos ou instruções, com o objetivo de resolver um problema e obter um resultado. É escrita através de alguma linguagem de programação (HEHL, 1986).

Assim sendo, a principal tarefa de um programa é a de transformar os dadas de entrada em resultados através de funções de transformação que devem ser perceptíveis e inteligíveis aos prováveis usuários leitores do mesmo. Para cada uma dessas funções devem ser definidas assertivaç de entrada, que estabelecem as condições de entrada que deverão ser satisfeitas para que o programa funcione corretamente, e as assertivas de saída, que estabelecem as condições de saída que estarão satisfeitas caso o programa tenha funcionado corretamente (STAA, 1987).

Um programa geralmente pode ser dividido em partes menores, os Módulos, que podem ser tratados separadamente. Esta divisão muda e facilita a leitura e entendimento do programa.

Assim, um módulo é uma unidade que possui aí seguintes características (FAIRLEY, 1985):

- contém declaraçõeS, comandos, instruções, lógica de processamento e estruturas de dados;
- pode ser compilado separadamente e armazenado numa biblioteca;
- pode ser incluído num programa;
- seus segmentos podem ser usados para chamar um nome e alguns parâmetros;
- pode usar outros módulos.

Subrotinas, subprogramas, funções e procedimentos são exemplos de módulos que, dependendo da linguagem de programação, podem ser compilados separadamente e mantidos independentes um do outro. Quando bem projetado, o módulo pode, ainda, ser usado em outro software. Quando não ocorre

essa divisão do programa em módulos adequados, podem haver módulos muito grandes que geralmente são difíceis de serem mantidos e especializados demais para serem de uso geral (HEHL, 1986).

No processo de desenvolvimento de um software, baseado em fases, a implementação tem como objetivo construir programas a partir de uma especificação de projeto já existente. Com isso, sua principal atividade é a de Programação, que é basicamente a codificação e teste de programas e módulos já definidos no Projeto, ou seja, as especificações de projeto são traduzidas em código fonte, que quando integrados são os responsáveis pela execução do software.

A elaboração de um programa deve ser feita de modo que ele expresse sua definição de forma correta e completa, satisfazendo aos requisitos de qualidade estipulados, para que quando em operação/manutenção tenha uma vida útil longa e produtiva.

Existem algumas dificuldades que deve-se ter em mente na avaliação da qualidade de um programa, como as seguintes (BUCKLEY, 1984):

- uma mesma função ou algoritmo pode ser implementado de várias maneiras, e nem sempre fica claro qual a melhor forma;
- a complexidade e interações envolvidas no desenvolvimento de software de grande porte aumentam não linearmente com seu tamanho;
- a documentação deve ser considerada como parte integral do software.

Para uma devida avaliação da qualidade de um programa, pode ser levada em consideração tanto métricas quantitativas quanto exames subjetivos feitos diretamente no código fonte (ADAM, 1988).

Existem alguns fatores chaves que podem ajudar no desenvolvimento de programas com um alto nível de qualidade, como os seguintes (VINCENT, 1988):

- o uso de uma linguagem de programação adequada ao tipo da área de aplicação e ao software em questão;
- o uso de ambientes de suporte a programação adequados;
- a obediência a padrões estabelecidos pela instituição;
- ferramentas automatizadas de apoio ao desenvolvimento de software, como geradores de código fonte de programa;
- apoio gerencial;
- métodos para avaliação adequados;
- condições e recursos de trabalho adequados;
- comprometimento do desenvolvedor com a qualidade.

Todos esses fatores, além de objetivar o aumento da qualidade do programa, também ajudam a aumentar a produtividade dos desenvolvedores.

Com relação a dacumentacão de um programa, ela é composta por uma documentação externa e uma interna. A documentacão externa, entre outras, pode envolver os seguintes documentoã (STAA, 19871, (FAIRLEY, 1985) e (MARTIN, 19851 : Especificacão funcional do programa;

Modularidade funcional do programa; Gráfico de estrutura; Diagrama de Fluxo de Dados; Diagrama de Entidades e Relacionamentos; Diagrama de Transição de Estados; Especificações de processo (minispecs); Diagrama HIPO; Gráfico de Controle; Diagrama Warnier-Orr; Diagrama/Fluxograma de Constantine; Diagrama de ação; Quadro Nassi-Shneiderman; Diagrama de Jacksen; Algoritmo em Linguagem estruturada; Algoritmo em Pseudo-código; Arvore de decisão; Tabela de decisão; Fluxograma; Lay-outs de arquivos, registros, telas e relatórios; Dicionário de Dados; Mapa de Acesso Lógico; Diagrama de ação de Banco de Dados e Plane de testes.

A documentação interna de um programa, gerada na implementação, pode ter sido embutida na codificacão do código fonte ou gerada a tempo de compilação, e pode envolver os seguinteç (MARTIN, 1985), (FAIRLEY, 1985) e (VINCENT, 1988): comentários; uma listagem de referência cruzada; um prólogo padrão para cada módulo; assertivas de entrada/saída e aspectos de auto-documentacão próprios da linguagem de programação.

Além da documentação externa e interna de um programa, para completar a relação de sua documentação, inclui-se o dacumento mais importante, a listagem da Última versão do código fonte.

E primordial ressaltar a importância que possui a documentação do programa, tanto externa como interna, pois além de fornecer apoio para a avaliação das caracterásticas de qualidade do mesmo, é também a base para futuras manutenções do software ao longo de toda sua vida útil.

Isto porque o que é submetido para o processo de avaliação é toda sua documentação gerada, com a verificação de cada documento integrante por comparação com os requisitos de qualidade já pré-determinados, tanto com o fim de verificar se foram satisfeitos, somo para identificar uma qualidade inferior que poderia prejudicar o desempenho do software.

Esta tese parte do princípio que já está realizada a avaliação da qualidade da documentação externa do programa, considerando, assim, apenas a avaliação da documentação interna e de próprio código fonte do programa gerados na implementação.

2.3 - Revisão da literatura em Qualidade de Software

Existem alguns trabalhos sobre Qualidade de Software na literatura que tratam de diferentes enfoques. Os principais, brevemente descritos a seguir, pedem çer classificados em três tipos:

- Modelos Qualitativos;
- Ciência de Software;
- Modelos de Confiabilidade.

2.3.1 - Modelos Qualitativos

Modelos Qualitativos são trabalhos que tratam as caracteristicas de qualidade do software a partir de uma classificação hierárquica das mesmas, com a determinação da presença ou ausência de cada uma delas utilizando os resultados quantitativos de suas métricas. Possuem diferentes abrangências, podendo assim tratar tanto do

ciclo de vida do software por completo, como de algum produto especifico. Os principais são os seguintes:

- Modelo de Boehm (BOEHM, 1978);
- Modelo de McCall (MCCALL, 1978), (MCCALL, 1979) e (MCCALL, 1981).

Além destes, serão discutidas também:

- Modelo de Arthur (ARTHUR, 1985);
- Modelo de Rocha (ROCHA, 1987).

Estes Modelos são brevemente descritos a seguir, sendo que o de Rocha é abordado na Seção 2.4.

2.3.1.1 - Modelo de Boehm

Visando fornecer meios para medir a qualidade a nível de programa, dentro do processo de avaliação da qualidade de software, Boehm desenvolveu um conjunto relacional de características de qualidade e um conjunto de métricas associadas (BOEHM, 1978).

Este conjunta de características de qualidade de software está relacionado hierarquicamente através de um estrutura em árvore, mostrada na figura 1, onde a direção da seta representa uma implicação lógica. Para cada uma dessas caracteristicas, ele forneceu um conjunto de "checklists" de qualidade com uma ou mais métricas aplicáveis a programas FORTRAN.

Sua proposta é que dado um programa, as métricas forneçam uma medida quantitativa do yrau da característica associada a tal. A partir dai, a qualidade do software pode ser definida como uma função dos valores das métricas.

A definição das caracteristicas de qualidade, segundo

Boehm, também estão a seguir

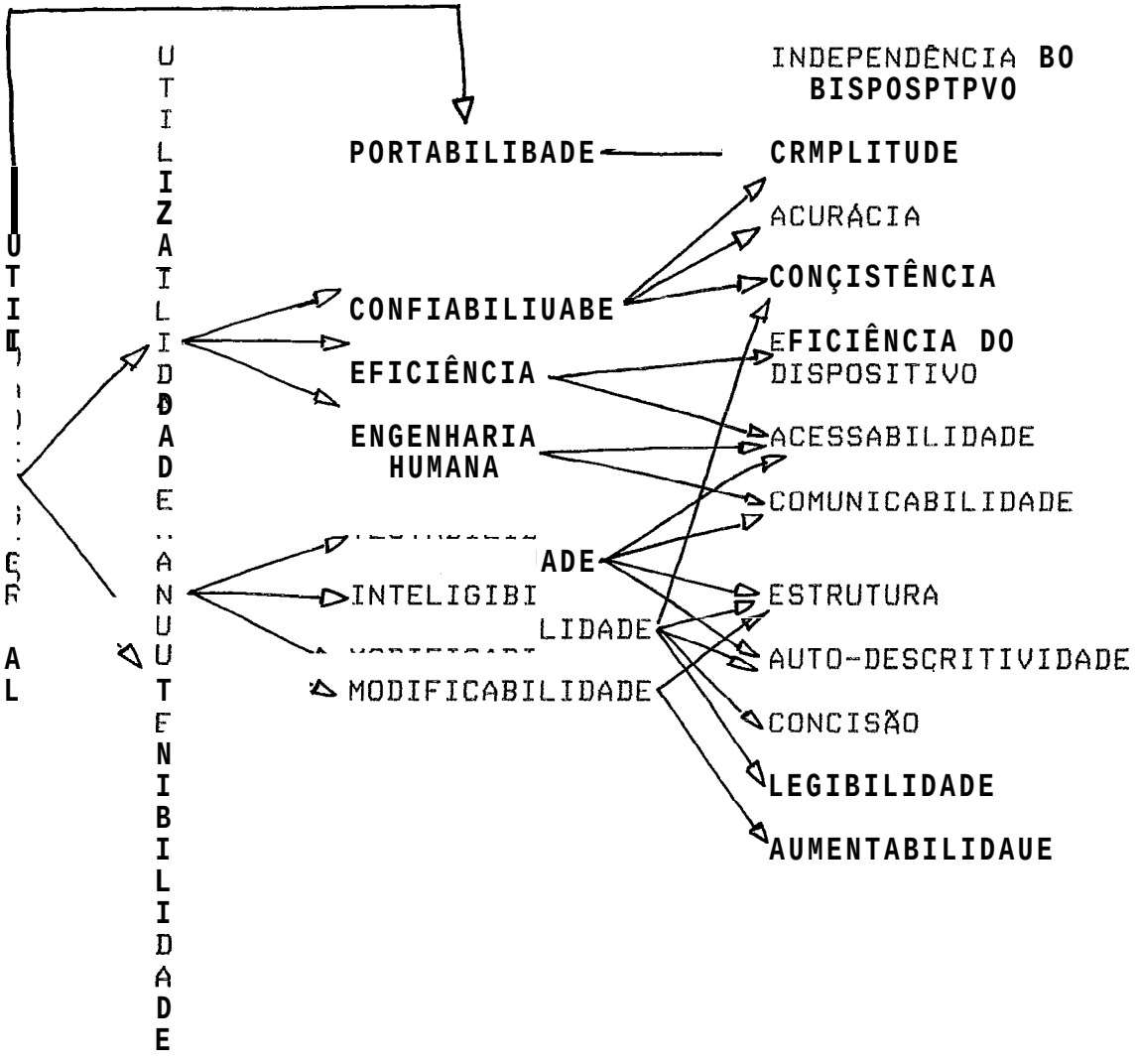


Fig. 1 - Arvore de características de qualidade de software
Fonte: (BOEHM, 1978)

- **PORTABELIDADE:** um software possui essa característica na medida em que pode ser fácil e adequadamente operado em outras configurações de computadur que não a corrente;
- UTILIZABILIDADE: um software possui essa característica na medida em que é conveniente e praticável sua utilização;
- **MANUTENIBILIBADE:** um software possui essa característica na medida em que é fácil sua atualização para

satisfazer a novas necessidadesi

- CONFIABILIDADE: um software possui essa caracteristica na medida em que pode-se esperar que efetue satisfatoriamente as funções pretendidas;
- EFICIÊNCIA: um software possui essa característiea na medida em que alcança seu objetivo sem desperdício de recursos;
- ENGENHARIA HUMANA: um software possui essa característica na medida em que atende a seu propósito sem desperdiçar a energia e o tempo de usuário eu degradar sua moral;
- TESTABILIDADE: um software possui essa earacteriítica na medida em que facilita o estabelecimento de critérios de aceitação e que suporta a avaliação de seu desempenho;
- INTELIGIBILIDADE: um seftware possui essa caracteristica na medida em que e propósito do produto está claro para o avaliador;
- MODIFICABILIDADE: um software possui essa caracteristica na medida em que facilita a incorporação de mudanças, uma vez que estas tenham sido determinadasi
- INDEPENBÉNCIA DO DISPOSITIVO: um software possui essa caracteristica na medida em que pode ser executado em configurações de hardware diferentes da corrente;
- COMPLITUDE: um software possui essa característica na medida em que todas as suas partes estão presentes e cada uma delas esta completamente desenvolvida;
- ACURACIA: um software possui essa característica na medida em que suas saídas são precisas o suficiente

para satisfazer a utilização pretendida;

- CONSISTÊNCIA: um ssftware possui essa caracteristica na medida em que contém notação, terminologia e simbologia uniformes e na medida em que o conteúdo está condizente com os requisitos;
- ACESSABILIDADE: um software possui essa caracteristica na medida em que facilita a utilização seletiva de seus componentes;
- COMUNECABILIDADE: um software possui essa caracteristica na medida em que facilita a especificação de entradas e permite saídas cuja forma e conteúdo são Úteis e fáceis de assimilar;
- ESTRUTURA: um software possui essa caracteristica na medida em que possui um relação definida de organização de suas partes interdependentes;
- AUTO-DESCRITIVIDADE: um software possui essa caracteristica na medida em que contém informação suficiente para o leitor determinar seus objetivos, suposições, restrições, entradaç, saídas, componentes e status;
- CONCISÃO: um software possui essa característica na medida em que não contém informação excessiva;
- LEGIBILIDADE: um software possui essa característica na medida em que sua função e a de suas instruções componentes são facilmente discerníveis pela leitura do código;
- AUMENTABILIDADE: um software possui essa caracteristica na medida em que facilmente permite expansões quando há necessidade de armazenamento de dados ou de funções

computacionais.

2.3.1.2 - Modelo de McCall

O conceito de Qualidade de Software para McCall, em plena prática conforme em (RADC-TR-85-37), baseia-se numa estrutura para medição da qualidade de software definida hierarquicamente em quatro níveis, que é mostrada na figura 2 e definida a seguir (MCCALL, 1978), (MCCALL, 1979) e (MCCALL, 1981).

Nível 1: **Orientações** - está relacionado com as atividades do ciclo de vida associadas ao produto final, sendo composto por: operação, revisão e transição do produto.

Nível 2: **Fator** - condição ou característica que contribui ativamente para a qualidade do software. Está agrupado de acordo com as orientações, e esta relação, feita através de perguntas, é mostrada na figura 3.

Nivel 3: **Critérios** - são atributos independentes e orientados ao software pelos quais a qualidade pode ser julgada, definida e medida. Definem melhor o fator de qualidade e ajudam a descrever os relacionamentos entre fatores, uma vez que um mesmo critério pode estar relacionado a mais de um fator. Assim, estão agrupados de acordo com os fatores, e esta relação é mostrada na figura 4 com as suas respectivas definições.

Nível 4: **Métricas** - proporcionam medidas quantitativas dos atributos de software, podendo ser aplicadas durante todas as fases da processo de desenvolvimento do software.

As métricas relacionadas aos onze fatores podem ser

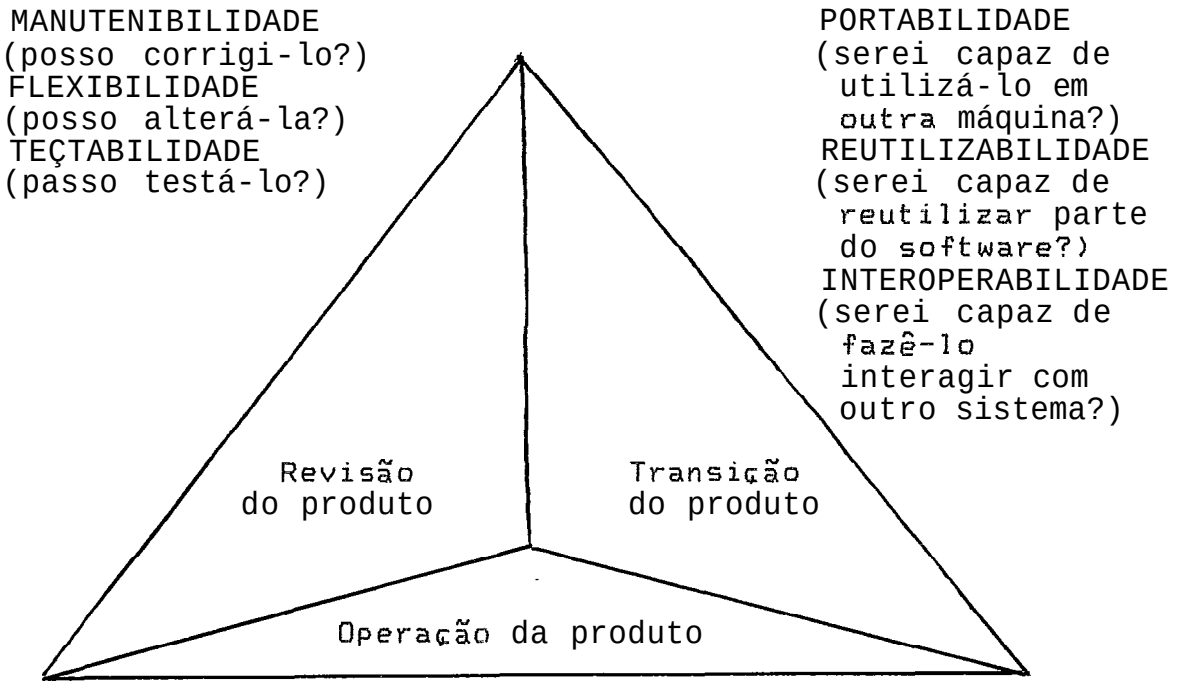
encontradas sob a forma de "checklists" ou em cálculos específicos, como exposto em (RADC-TR-85-37).

Fig. 2 - Fçtrutura da qualidade de software
Fonte: (MCCALL, 1979)

- UTILIZABILIDADE: esforço necessário para aprender, operar, preparar a entrada e interpretar a saída de um programa;
- INTEGRIDADE: grau em que o acesso ao software ou aos dados por pessoas não autorizadas pode ser controlado;
- EFICIÊNCIA: quantidade de recursos computacionais e de código necessários para um programa efetuar uma função;
- CORREÇÃO: grau em que um programa satisfaz suas especificações e atinge os objetivos de seus usuários;
- CONFIABILIDADE: grau em que pode-se esperar que um programa efetue a função pretendida com a precisão requerida;
- MANUTFNIBILIDADE: esforço neceçsário para localizar e identificar um erro em um programa já em operação;
- TESTABILIBABE: esforço necessário para testar um programa de modo a assegurar que ele efetua a função

pretendida;

- FLEXIBILIDADE: esforço para modificar um programa em operação;
- REUTILIZABILIDADE: grau em que um programa pode ser usada em outras aplicações;
- PORTABILIDADE: esforço necessário para transferir um programa de uma configuração de hardware e/ou de um ambiente do software do sistema (Sistema Operacional, Utilitários, Rotinas de I/O, extensões da linguagem, Sistema de Gerenciamento de Banco de Dados, etc) para outro;



CORREÇÃO (ele faz o que quero?)
CONFIABILIDADE (ele faz, sempre, de modo preciso?)
EFICIÊNCIA (ele executa no meu hardware da melhor forma possivel?)
INTEGRIDADE (ele é seguro?)
UTILIZABILPDADE (posso executá-lo?)

Fig. 3 - Relações das Orientações com seus respectivos Fatores
Fonte: (MCCALL, 1979)

- INTEROPERABILIDADE: esforço necessário para acoplar um sistema a um outro;
- OPERABILIDADE: atributos e procedimentos relacionados com a operação do software;
- TREINAMENTO: atributos do software que proporcionam a sua operação corrente ou sua a familiarização inicial;
- COMUNICABILIDADE: atributos do software que proporcionam entradas e saídas úteis que podem ser assimiladas;
- CONTROLE DE ACESSO: atributos do software que possibilitam o controle do acesso ao software e aos dados;
- AUDITABILIDADE DE ACESSO: atributos do software que possibilitam uma auditoria do acesso ao software e aos dados;
- EFICIÊNCIA DE ARMAZENAMENTO: atributos do software que proporcionam o mínimo de necessidades de armazenamento durante a operação;
- EFICIÊNCIA DE EXECUÇÃO: atributos do software que proporcionam tempo mínimo de processamento;
- RASTREABILIDADE: atributos do software que proporcionam uma ligação entre as necessidades e a implementação no que diz respeito ao desenvolvimento específico e ao ambiente operacional;
- COMPLETEZA: atributos do software que proporcionam uma completa implementação das funções requeridas;
- PRECISÃO: atributos do software que proporcionam a precisão requerida em cálculos e resultados;
- TOLERÂNCIA A ERROS: atributos do software que proporcionam continuidade de operação sob condições

anormais;

- CONSISTÊNCIA: atributos do software que proporcionam uniformidade de notação e técnicas de projeto e implementação;

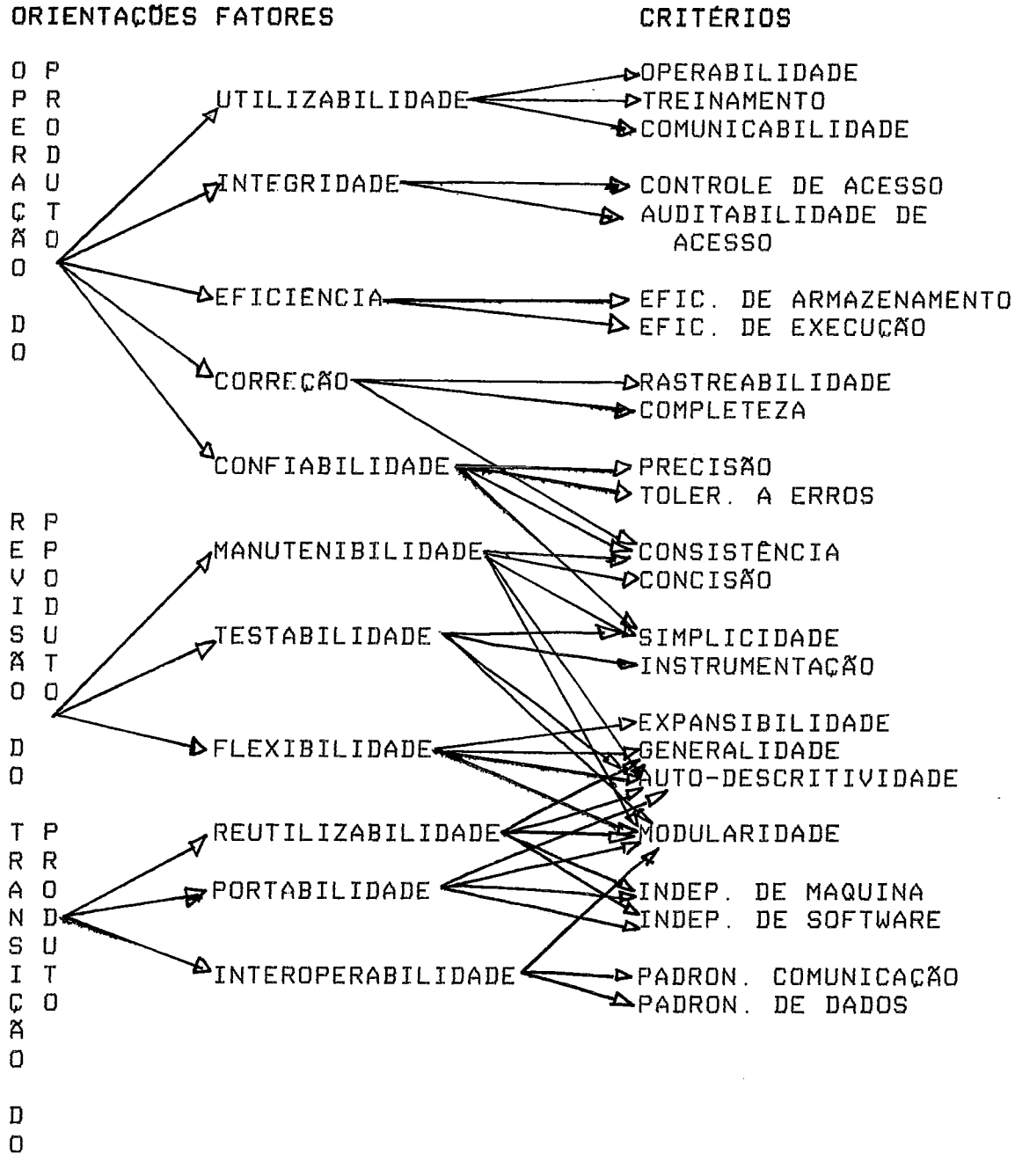


Fig. 4 - Relação dos Fatores com seus respectivos Critérios
Fonte: (MCCALL, 1979)

- SIMPLICIDADE: atributos do software que proporcionam a implementação das funções de uma maneira mais compreensível (geralmente evitando práticas que aumentem sua complexidade);
- CONCISÃO: atributos do software que proporcionam a implementação de uma função com a quantidade mínima de código;
- INSTRUMENTAÇÃO: atributos do software que proporcionam as medições do uso ou identificação de erros;
- EXPANSIBILIDADE: atributos do software que proporcionam a expansão das necessidades de armazenamento de dados ou funções computacionais;
- GENERALIDADE: atributos do software que proporcionam amplitude às funções desempenhadas;
- AUTO-DESCRITIVIDADE: atributos do software que proporcionam a explicação da implementação de uma função;
- MODULARIDADE: atributos do software que proporcionam uma estrutura de módulos altamente independentesi
- INBEPENDENCIA DE MAQUINA: atributos do software que determinam sua dependência ao hardware do sistema;
- INDEPENDÊNCIA DO SOFTWARE DO SISTEMA: atributos do software que determinam sua dependência ao ambiente do software do sistema (Sistema Operacional, Utilitários, Rotinas de I/O, extensões da linguagem, Sistema de Gerenciamento de Banco de Dados, etc);
- PADRONIZAÇÃO DA COMUNICAÇÃO: atributos do software que proporcionam o uso de protocolos e rotinas de interfaces padronizadas;

- PADRONIZAÇÃO DE DADOS: atributos do software que proporcionam s uso de padronização na representação dos dados.

2.3.1.3 - Modelo de Arthur

Arthur csnsidera que Qualidade de Çoftware é medível e varia de software para software, de programa para programa, sendo por isso definida através de uma equação que é uma função das diversas métricas de qualidade ds software. Assim, dependendo do ambiente existente e dos dados históricos, pode-se tanto eliminar alguma métrica como reduzir o seu peso na própria equação (ARTHUR, 1985).

A equação para qualidade do ssftware é a seguinte:

QUALIDADE = F(CORREÇÃO, EFICIÊNCIA, FLEXIBILIDADE, INTEGRIDADE, INTEROPERABILIDADE, MANUTENIBILIDADE, PORTABILIDADE, CONFEABILIDADE, REUTILIZABILIDADE, TESTABILIDADE, UTILIZABILIDADE)

As definições das métricas de qualidade são as seguintes:

- CORREÇÃO: a grau para o qual um programa satisfaz as especificações do usuário;
- EFPCIÊNCIA: quantidade de recursos computacionais requeridos para desempenhar uma função definida pelo usuário;
- FLEXIBILIDADE: quantidade de esforço para mudar o programa;
- INTEGRIDADE: grau que o software e os dados estão protegidos, controlados e segures;
- INTEROPERABILIDADE: quantidade de esforço requerido para

unir o programa ou software com um outro;

- **MANUTENIBILIDADE:** quantidade de esforço requerido para localizar e corrigir erros no programa;
- PORTABILIDADE: tipo de esforço tomado para transferir um programa de uma máquina para outra;
- **CONFIABILPDABE:** grau para o qual pode-se esperar que o software desempenhe sua função sem falha, ou seja, trabalhe de modo acurado sem falha;
- **REUTILIZABILZDADE:** extensão na qual o programa pode ser usado, total ou parcialmente, em autras aplicações para redução de custas;
- TESTABILIDADE: quantidade de esforço requerido para testar a estrutura e correção do programa;
- UTILIZABILIDADE: quantidade de esforço sue a usuário dispende para aprender e usar o software, ou seja, operá-lo facilmente.

Cada uma destas métricas de qualidade de software descreve um aspecto da qualidade, que por sua vez é composta por métricas de mais baixo nível, chamadas também de critério de qualidade de software, definidas a seguir.

- AUDITABILIDADE: atributos do software que proporcionam fácil auditoria do software, dados e resultados;
- ACURÁCIA: atributos do software que proporcionam precisão nos cálculos e nas saídas;
- PADRONIZAÇÃO DE COMUNICAÇÃO: atributos do software que usam protocolos e rotinas de interface padrão;
- COMPLETEZA: atributos do software que proporcionam uma completa implementação das funções requeridas;
- **COMPLEXIDADE:** atributos do software que diminuem a

clareza do programa ou do módulo;

- CONCISÃO: atributos do software que implementam uma função numa quantidade mínima de código;
- CONSISTÊNCIA: atributos do software que proporcionam documentação, técnicas de projeto e implementação uniformes;
- PADRONIZAÇÃO DE DADOS: atributos do software que proporcionam representações de dados padrão;
- TOLERÂNCIA A ERROS: atributos do software que proporcionam continuidade de operação sob condições adversas;
- EFICIÊNCIA DE EXECUÇÃO: atributos do software que minimizam o tempo de processamento;
- EXPANSIBILIDADE: atributos do software que proporcionam expansão dos dados ou funções do programa;
- GENERALIDADE: atributos do software que expandem a utilidade da função além do módulo existente;
- INDEPENDÊNCIA DE HARDWARE: atributos do software que determinam sua dependência ao hardware;
- INSTRUMENTAÇÃO: atributos do software que proporcionam medições do seu uso ou da identificação de erros;
- MODULARIDADE: atributos do software que proporcionam uma estrutura de módulos altamente independentes;
- OPERABILIDADE: atributos do software que determinam a facilidade ou dificuldade de operação do mesmo;
- SEGURANÇA: atributos que proporcionam controle e proteção ao software e aos dados;
- AUTO-DOCUMENTAÇÃO: atributos do software que explicam sua função;

- SIMPLICIDADE: atributos do software que proporcionam uma implementação das funções de uma maneira mais compreensível;
- INDEPENDÊNCIA DE SISTEMA DE SOFTWARE: atributos do software sue determinam sua dependência ao ambiente do software do sistema (Sistema Operacional, Utilitários, Rotinas de I/O, extensões da linguagem, Sistema de Gerenciamento de Banco de Dados, etc);
- RASTREABILIDADE: atributos do software que proporcionam uma ligação dos requisitos ao programa implementado;
- TREINAMENTO: atributos do software que proporcionam transições do ambiente corrente para o novo sistema.

Cada um desses critérios é derivado da presença ou ausência de certos operadores, operandos e tipos de dados no programa.

Eles se relacionam com as métricas através de uma função. Logo, os objetivos da medição da qualidade de software é deduzir as seguintes equações:

CORREÇÃO = F(completeza,consistência,rastreabilidade)

EFICIÉNCIA = F(concisão,efic. de execução,operabilidade)

FLEXIBILIDADE = F(complexidade,concisão,consistência,
expansibilidade,generalidade,modularidade,auto-documentação,simplicidade)

INTEGRIDADE = F(auditabilidade,instrumentação,segurança)

INTEROPERBBILIBADE = F(padronização de comunic.,padronização de dados,generalidade,modularidade)

MANUTENIBILIDADE = F(concisão,consistência,modularidade,
instrumentação,auto-documentação,
simplicidade)

PORTABILIDADE = F(generalidade,indep. hardware,modularidade,auto-documentação,indep. software)

CONFIABILIDADE = F(acurácia,complexidade,consistência,
tol. erro,modularidade,simplicidade)

REUTILIZABILIDADE = F(generalidade,indep. hardware,modula-
ridade,auto-documentação,indep.
software)

TEÇTABILIDADE = F(auditabilidade,complexidade,instrumentação,
modularidade,auto-documentação,simplici-
dade)

UTILIZABILIDADE = F(operabilidade,treinamento)

2.3.2 - Ciência de Software

A Ciência de Çoftware é um enfoque proposto por Halstead (HALSTEAD, 1977) que se baseia no principio de que um programa é função apenas de seus operadores e operandss. Onde estes operadores são operadores de expressão e comandos da linguagem, e operandos são nomes de variável e constantes. Este conjunto de operadores e operandos, quando medidos e agregados de forma apropriada, fornecem o esforço requerido para implementar o programa, o volume do programa, o nível da linguagem do programa, o tamanho estimado para o programa e o tempo dedicado à programação.

Os operadores dividem-se em 3 classes: básico (operadores lógicos e aritméticos), palavras-chave (comandos e instruções) e especial (nomes de subr-etinas e funções) (LI, 1987).

As métricas de Halstead, expressas por equações, são mostradas na tabela 1 e esclarecidas a seguir.

O número de operadores não redundantes n1* poderia ser determinado como o conjunto de todos os operadores disponíveis na linguagem de programação utilizada.

As medidas de volume mais significativas de um programa

são n e N, que fornecem o volume do programa V. Este caracteriza o tamanho de uma implementação, que é o número de bits requerido para especificar o programa. Também é visto como a medida das comparações mentais.

O volume potencial V* indica a representação mínima da programa numa linguagem onde a operação requerida esteja embutida, resultando uma implementação numa forma mais sucinta.

SÍMBOLOS		EQUAÇÕES
n1	número de operadores distintas, únicos	
n2	número de operandos distintas, únicos	
n	Vocabulário da programa	$n = n1 + n2$
n1*	número de operadores não redundantes	
n2*	número de operandos não redundantes	
n*	Tamanho de voeabulária	$n^* = n1^* + n2^*$
N1	número total de operadores	
N2	número total de operandos	
N	Comprimento da programa	$N = N1 + N2$
V	Volume do programa	$V = N \log n$
V*	Volume potencial	$V^* = n^* \log n$
L	Nível de programa	$L = V^*/V$
D	Dificuldade de programa	$D = 1/L$
E	Esforce de programação	$E = V^2/V^* = V/L$

Tab. 1 - Equações da Ciência de Software
Fonte: (PAIGE, 1980)

Para avaliar o esforco de programação, propensão a erro e facilidade de entendimento, o nível de programa L de uma implementação tem o valor máximo de unidade e indica quão bem um programador usa a linguagem de programação adotada.

A dificuldade de um programa D é a dificuldade de codificação do mesmo, expressando, assim, a complexidade psicológica envolvida para tal.

O esforço de programação E, requisitado para gerar um programa, mede o esforço mental, ou a quantidade de tempo, requerido para implementar o programa. Assim, prevê o número de discriminações mentais elementares necessárias para se escrever ou entender um programa.

2.3.3- Modelas de Confiabilidade

Confiabilidade de software pode ser definida como a probabilidade de um software executar sem falhas, suas especificações e funções em um determinado ambiente durante um certo período de tempo. Para tanto, o software necessita estar correto apenas para as entradas projetadas. Também, se a saída está correta, com as tolerâncias especificadas, no caso de algum erro, este é ignorado. Logo, a Confiabilidade de software proporciona uma medida da confiança na correção operacional do software (BASTANI, 1982).

A classificação de alguns do5 Modelos de Confiabilidade de software existentes é mostrada na figura 5, estando esquematizada de acordo com a fase do ciclo de vida em que o Modelo é aplicável.

Na façe de Depuração, os modelos aplicáveis são também chamados de modelas de crescimento de confiabilidade. Dividem-se ainda em dois modelos, o de Contagem de erro, usado suando uma estimativa do número de erros restantes é necessária, e o de Contagem de não erro, usada se apenas a estimativa de canfiabilidade é requerida.

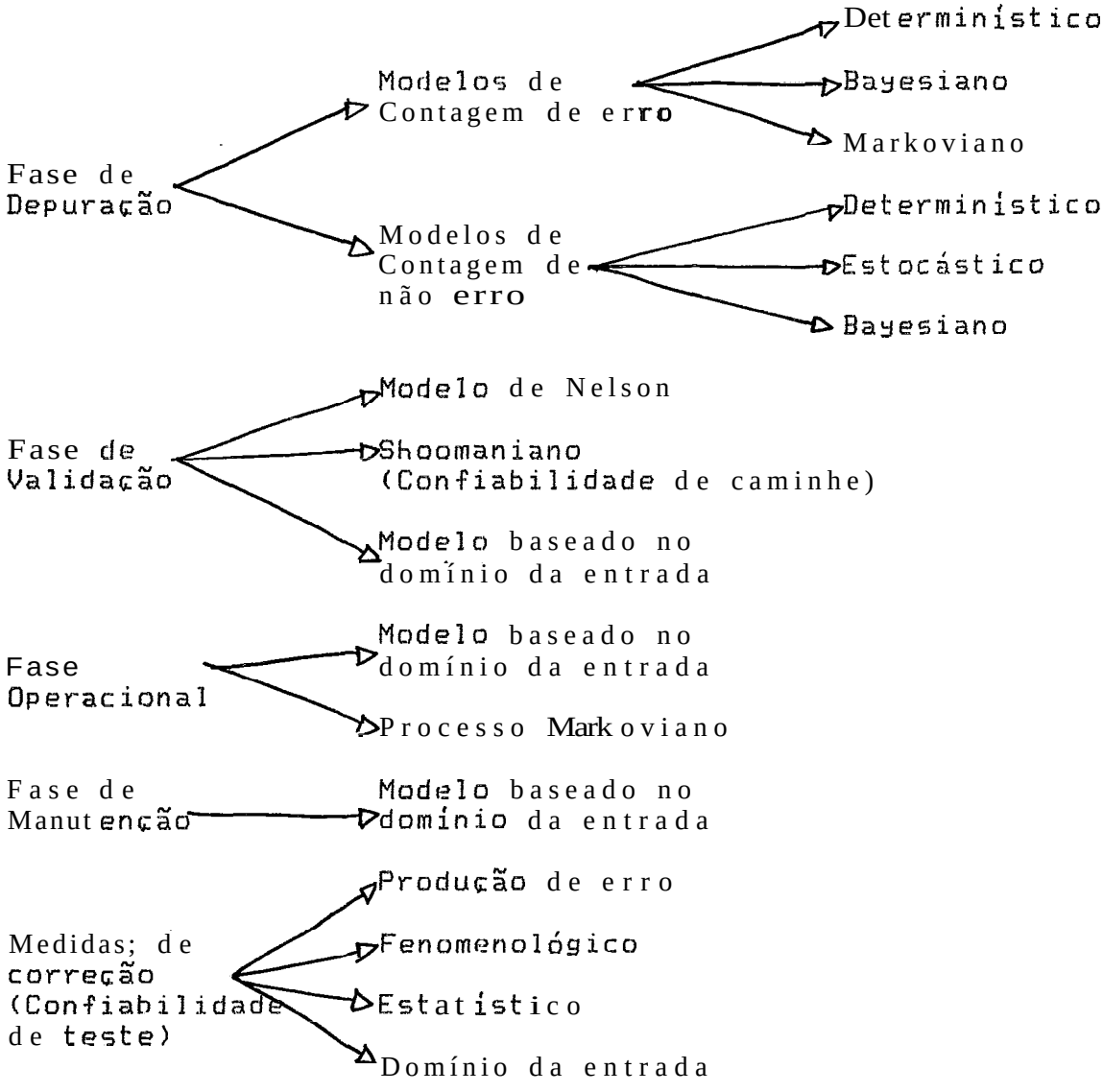


Fig. 5 - Classificação dos Modelos de Confiabilidade de software
Fonte: (BASTANI, 1982)

Na fase de Validação, o modelo de Nelson está baseado em princípios estatísticos e relacionado aos conceitos de classes de equivalência e continuidade no domínio da

entrada que são usados para estimar a probabilidade de correção do programa. Também usa a definicão operacional para obter a estimativa da eonfiabilidade de software (BASTANI, 1982).

Na fase Operacional, o processo de Markov modela e mecanismo de seleção de distribuição de entrada. O modelo baseado no domínio de entrada incorpora o aspecto de que o resultado de uma determinada execução pode depender dos resultados das execuções anteriores.

Na fase de Manutenção, a nova confiabilidade pode ser estimada usando os modelos da fase de Validação.

Na categoria de Medidas de correção estão os métodos que fornecem alguma medida para a eonfiabilidade de teste.

O aspecto de Produção de erro é usada para avaliar diferentes estratégias de teste, a confiabilidade do conjunto de casos de testes usados e a probabilidade de correção do programa. Também possibilita estimar o número de erres restantes num programa. O Fenomenológico é o próprio modelo de Ciência de Software de Halstead, já abordado anteriormente neste trabalho, que dá uma previsão do conteúdo dos erros. O Estatística fornece uma confiança na estimativa de confiabilidade baseada na teoria de exemplificação estatística. E o modelo de domínio de entrada estima diretamente a probabilidade de correção de um programa.

O problema destes Modelos de Confiabilidade é que eles podem falhar na incorporação de fatores de software intrínsecos, como a variabilidade do domínio de entrada/missão, complexidade funcional, tipo da área de

aplicação e parâmetros de desenvolvimento (CAVANO, **1985)**

2.4 - Método para avaliação da qualidade de programas

Um método para avaliação da qualidade de software é um conjunto de definições e procedimentos a serem seguidos para se poder avaliar as características de qualidade de um determinado produto resultante do processo de desenvolvimento do software.

Os métodos para avaliação da qualidade de software são necessários para (BACHE, 1988):

- entender os relacionamentoç entre as características de qualidade aplicadas num dado produto;
- explorar o efeito de alguma mudanca de uma característica de qualidade sobre outras;
- prever as características de qualidade de um prsduto;
- proporcionar procedimentos de avaliação, de padrões e de unidades que possam quantificar qualquer característica de qualidade;
- caracterizar uma dada situação e descrever propriedades individuais de uma caracteristica de qualidade.

Como abordado na seção 2.3 anterior, existem alguns trabalhos sobre Controle da Qualidade de Software na literatura, e para se escolher e usar algum método de avaliação, deve haver a preocupação com sua confiabilidade e se o mesmo é devidamente abrangente, pele menos para a

área de aplicação em que for empregado.

No caso desta tese, o método para avaliação da qualidade de software adotado para o produto programa é o proposto por ROCHA (1987). A motivação para esta escolha está no fato desta tese fazer parte de um conjunto de trabalhos da COPPE/SISTEMAS sobre Qualidade de Software que adotam este mesmo enfsque.

O método de ROCHA (1987) adotado foi influenciado pelos trabalhos de Boehm e McCall, e está baseado nos seguintes conceitos:

- **Objetivos de qualidade:** são as propriedadeç gerais que o produto deve possuir.
- **Fatores de qualidade:** são características que determinam a qualidade do ponto de vista dos diferentes usuários do produto (usuário final, mantenedores, etc).
- **Critérios de qualidade**: definem atributos primitivas possíveis de serem avaliados. Estes atributos são independentes entre si, e explicitamente definidos no nível mais baixo de detalhe sob o ponto de vista do produto.
- **Medidas:** indicam o grau de presença de um determinado critério, ou seja, são os resultados quantitativos da qualidade do produto segundo um determinado critério.
- **Processos de Avaliação**: determinam os procedimentos e as instrumentos a serem usados de forma a se medir o grau de presença no produto de um determinado atributo.
- **Medidas agregadas:** indicam o grau de presença de um determinado fator de qualidade, através da agregação das medidas obtidas a partir da avaliação segundo os

critérios

A estes conceitos, inclui-se mais um outro que é o de Sub-fator de qualidade. Este tipo é o fator de qualidade num nível mais detalhado de influência sobre o produto, sendo também atingido pelas medidas agregadas.

A figura 6 mostra a estrutura deste método adotado.

Os objetivos de qualidade são atingidos através dos seus respectivos fatores de qualidade, que por sua ver são atingidos por um respectivo conjunto de sub-fatores. Cada um desses sub-fatores são avaliados através dos respectivos critérios.

Objetivos, fatores e sub-fatores não são diretamente mensuráveis e só pedem ser avaliados através dos respectivos critérios. Com isso, nenhum critério isolado é uma descrição completa de um determinado sub-fator, nem um sub-fator isolado é uma descrição completa de um fator, e por sua ver, nenhum fator isolado define completamente um objetivo de qualidade.

Como base neste método de avaliação, a classificação das características de qualidade de programas também baseia-se no trabalho de ROCHA (1987), completando-se com o nível de critérios. Esta classificação, para o objetivo Confiabilidade da Representação tratado nesta tese, é mostrada na figura 7.

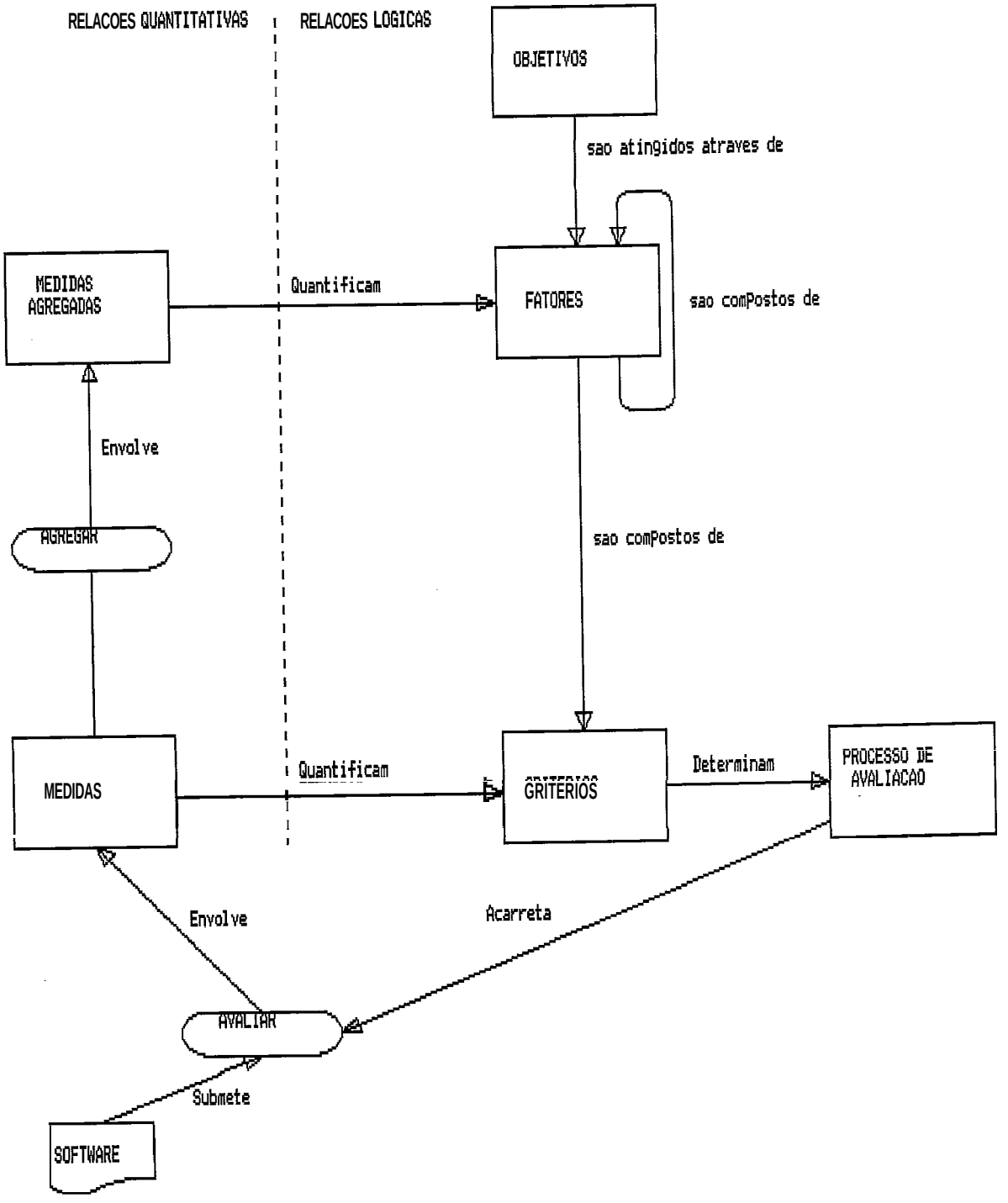


Fig. 6 - Método para avaliação da qualidade de software
Fonte: (ROCHA, 1987)

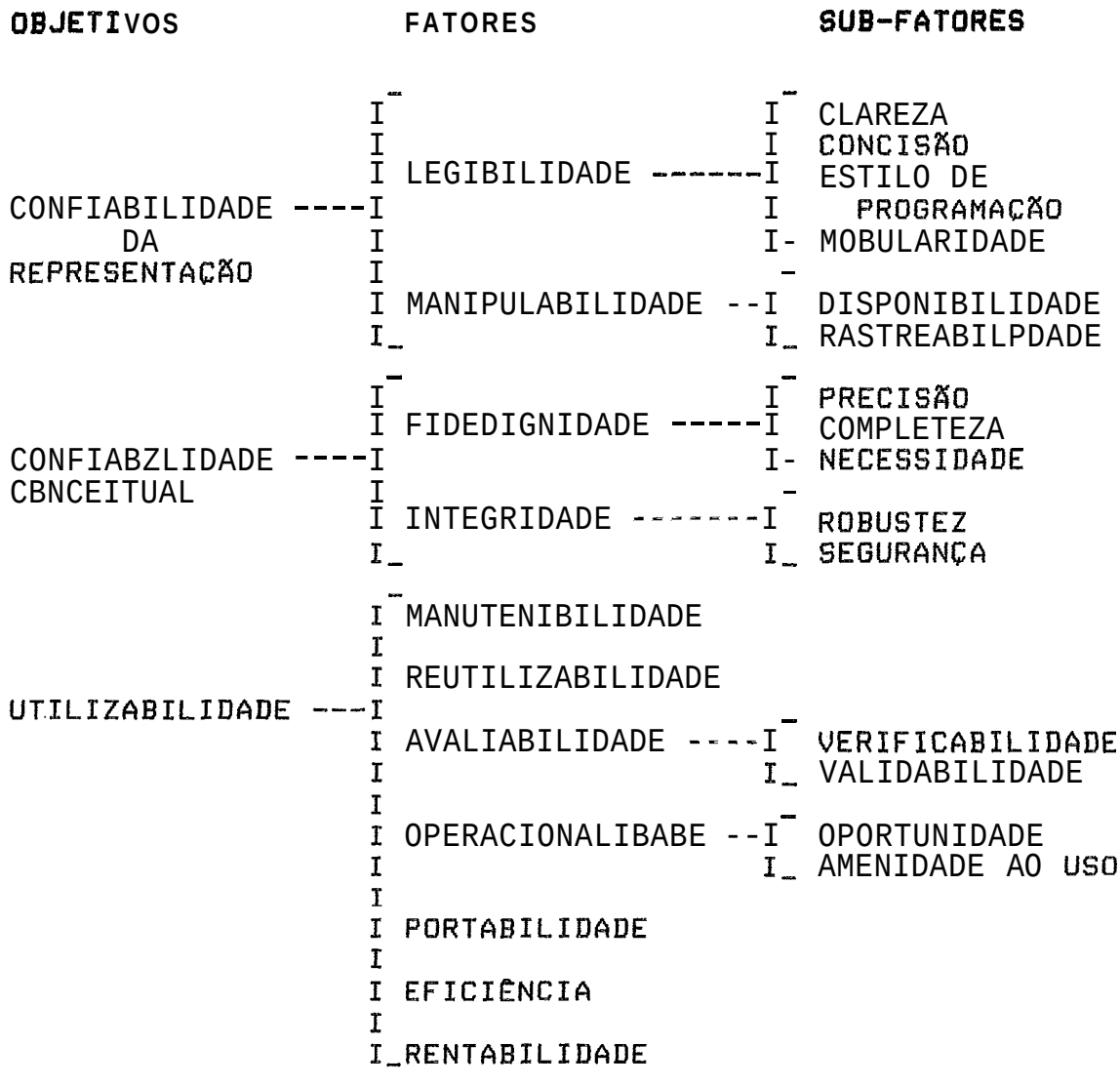


Fig. 7 - Classificação das características de qualidade de programas
Fonte: (ROCHA, 1987)

Como pode-se observar, as características de qualidade de programas podem ser agrupadas conforme o objetiva de qualidade à que estão relacionadas. Estes objetivos são definidos a seguir (FREITAS, 1985) e (ROCHA, 1987):

- **Confiabilidade da Representação**: refere-se às caracteristicas de entendimento e manipulação com relação à descrição, organização e representação da documentação interna e do código fonte do programa.
- **Confiabilidade Conceitual**: refere-se às características

de entendimento e correção com relação ao conteúdo da documentaeão interna e do código fonte do programa, de modo que satisfaça às suas especificações.

- **Utilizabilidade**: refere-se as características de utilização do programa sob as mais diversas formas, tanto durante a implementação como na operação ou manutenção. Requerendo, ainda, que já estejam satisfeitos os outros dois objetivos de qualidade, a Confiabilidade da Representação e a Confiabilidade Conceitual, para poder ser plenamente atingida e satisfeita.

Bentro do que se propõe esta tese, está envolvido apenas o objetivo Confiabilidade da Representação, discutido abaixo, juntamente com a definição, descrição e discussão de seus respectivos fatores, sub-fatores e critérios de qualidade. Estes critérios foram selecionados e definidos a partir de um estudo da literatura atual, e estão descritos à um nível geral, independente de uma linguagem de programação.

2.5 - Objetivo Confiabilidade da Representação

O objetivo Confiabilidade da Representação refere-se à necessidade de se ter facilidade no entendimento e na manipulação do programa com relação a descrição, organização e representação de sua documentação interna e de seu código fonte, de modo que possa ser manipulado pelas mais diferentes pessoas durante sua vida Útil (ROCHA, 1987).

A importância do objetivo Confiabilidade da Representação vem da necessidade das mais diferentes usuários leitores precisarem compreender e manipular o programa com facilidade. Estes usuários leitores não são necessariamente conhecidos entre si e, na maior parte das vezes, nem participaram do processo de desenvolvimento do programa ou do software (FREITAS, 1985).

A figura 8 apresenta a classificação das características de qualidade de programas çubordinadas ao objetivo Confiabilidade da Representação. Para definição destas caracteristicas, partiu-se das características já definidas em ROCHA (1987), acrescentando-se o nível de critérios de qualidade.

Para avaliar a presença deste objetiva Confiabilidade da Representação em um programa pode-se considerar os dois seguintes fatores de qualidade: Legibilidade e Manipulabilidade.

A seguir encontram-se aí definições e discussões relativos às características apresentadas na figura 8.

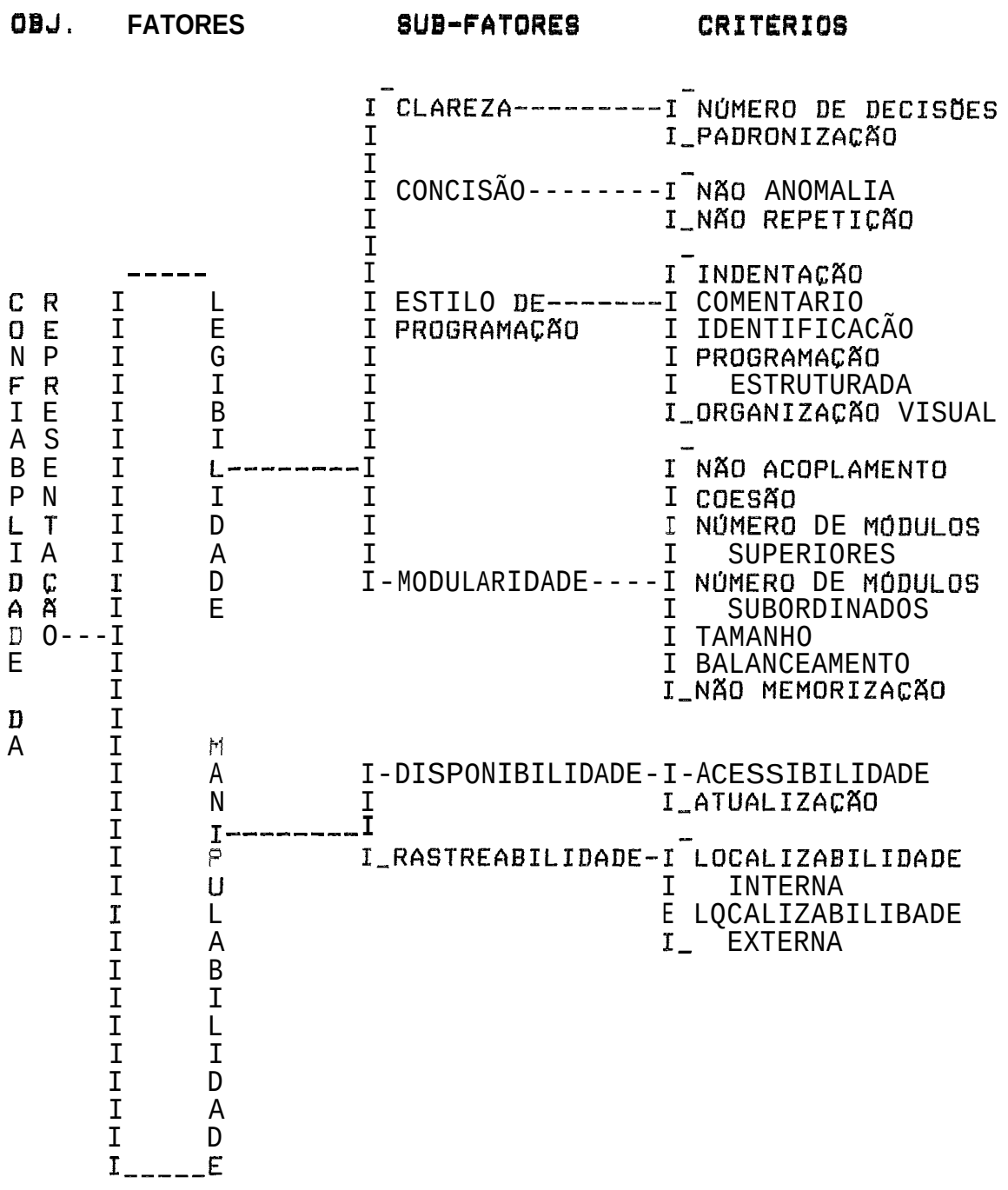


Fig. 8 - Classificação das características de qualidade de programas

2.5.1 - Fator Legibilidade

O fator **Legibilidade** refere-se a necessidade de se ter o programa codificado de maneira clara e legível em termos de sua forma e organização, de modo a ser facilmente lido e entendids por seus possíveis usuários leitores (ROCHA, 1987).

Um programa claro, simples e com explicações/relações nítidas e adequadas de seus componentes e de seu funcionamento, torna mais fácil e rápida sua leitura e consequente utilização. Com isso, seus custos de manutenção diminuem.

Para avaliar a presença do fator **Legibilidade** em um programa deve-se considerar os seguintes sub-fatores de qualidade: Clareza, Concisão, Estilo de Programação e Modularidade.

2.5.2 - Sub-fator Clareza

O sub-fator Clareza refere-se à necessidade de se ter as funções do programa codificadas da forma mais simples possível, isento de práticas que o possam tornar complexo ou de difícil leitura, entendimento e manipulação para os seus possíveis usuários leitores (ROCHA, 1987).

Com essa preocupação de esclarecimento e nitidez sobre a formatação do programa, fica bem mais fácil e rápido compreender o seu funcionamento e lógica.

Existe uma tendência natural do ser humano em complicar as coisas simples. Para evitar isso e alcançar a clareza dentro de um programa, é necessário disciplina, experiência

e raciocínio tanto do desenvolvedor como dos futuros manutenedores do mesmo (BARGUT, 1986).

Para avaliar a presença do sub-Fator Clareza em um programa deve-se considerar os seguintes critérios de qualidade:

- **Número de Decisões**

O critério **Número de Decisões** avalia se o programa possui uma quantidade reduzida de comandos de decisão segundo a Complexidade Ciclomática de McCabe. Com isso, o programa fica mais claro e diminui-se a complexidade de seu entendimento.

MCCABE **(1976)** propõe uma medida de complexidade de software **baseada** na teoria dos grafos, a Complexidade Ciclomática, que por sua vez baseia-se no número de caminhos básicos dos comandos de decisão existentes na estrutura de um grafo. Diante disso, considera o programa como um grafo de controle com um Único ponto de entrada e um único ponto de saída.

Existe também a complexidade cognitiva, que relaciona tanto as propriedades do software à cognição do programador quanto a cognição ao processo de desenvolvimento. Como esta complexidade cognitiva não é uma propriedade direta do software, mas relacionada à ele por meio de como as pessoas entendem os programas, para que a Complexidade Ciclomática possa então atingir esta complexidade cognitiva, pode-se aplicá-la à um fluxograma ou à um diagrama de estrutura de programa de Jackson (BACHE, **1988).** Esses casos também possuem toda a informação relevante sobre a estrutura lógica do programa e não deixam de ser um grafo com uma

Única entrada e uma única saída.

Assim, utiliza-se a seguinte fórmula para o cálculo da Complexidade Ciclomática V(G) sobre um grafo de controle, fluxograma ou diagrama de estrutura de programa de Jackson:

$$V(G) = e - n + 2p$$

onde:

e = número de arestas;

n = número de nós;

P = número de componentes conexos.

Para linguagens de programação que possuem comandos IF de decisão e operadores lógicos do tipo AND's, OR's e NOT's nas suas condições, como COBOL, FORTRAN e PL/I, esta fórmula pede ser simplificada. Para isço, considera-se apenas o número total de condições, ou seja, de operadores lógicos nos comandos IF existentes no programa, através da seguinte fórmula (ARTHUR, 1985):

$$V(G) = IF's + AND's + OR's + NOT's + 1$$

onde:

IF's = número total de comandos IF;

AND's = número total de operadores AND existentes nos comandos IF;

OR's = número total de operadores OR existentes nos comandos IF;

NOT's = número total de operadores NOT existentes nos comandos IF.

Quando o número de condições cresce, a Complexidade Ciclomática também cresce. De acordo com McCabe, para. tornar o programa mais claro, essa complexidade não deve ultrapassar o limite máximo de 10. Esse e um valor indicado

pela literatura, mas a prática e a observação de cada instituição podem alterá-lo.

Ainda segundo McCabe, a complexidade de um programa pode ser mais reduzida se ele tiver poucas violações de estrutura, como as seguintes:

- desvio para fora de uma decisão;
- desvio para dentro de uma decisão;
- desvio para fora de um "loop";
- desvio para dentro de um "loop".

Isto porque, quando acontecerem estes casos, haverá mais de um ponto de entrada ou mais de um ponto de saída, o que discorda do proposto por McCabe.

A avaliação de um programa segundo o critério Número de Decisões deve ser feita através de cálculo da fórmula da Complexidade Ciclomática simplificada de McCabe aplicada a cada um de seus módulos.

- Padronização

O critério **Padronização** avalia a obediência aos padrões técnicos de programação estabelecidos tanto pela linguagem de programação utilizada como pela instituição a que pertence o programa (PALERMO, 1988).

A verificação do programa com relação aoç padrões técnicos da linguagem de programação em que está codificado e feita durante a própria compilação ou interpretação do programa.

Quanto aos padrões técnicos de programação estabelecidos pela instituição, sua definição e obediência é uma tarefa importantíssima e necessária, pois é através destes que se

pode atingir uma maior homogeneidade dos programas, e consequentemente, do software, tornando-o mais claro, uniforme e de mais fácil entendimento.

Existem vários tipos de padrões técnicos de programação que podem ser estabelecidos por uma instituição, como os seguintes:

- orientações para estilo de programação;
- recomendações ou normas de programação especificas para a linguagem de programação utilizada;
- nomenclatura padrão;
- documentação do programa;
- técnicas para especificacão do programa, como gráfico de estrutura ou fluxograma;
- formulários padrão para definicão do programa.

A avaliação de um programa segundo o critério **Padronização** deve ser feita verificando se os padrões técnicos de programação adicionais estabelecidos pela instituição são obedecidos e se o mesmo já foi realmente submetido ao processo de compilação ou interpretação. No caso dos padrões estabelecidos pela instituição, eles todos devem ser incluídos um a um, ou neste critério ou em outro a que esteja mais relacionado.

2.5.3 - Sub-fator Concisão

O sub-fator **Concisão** refere-se à necessidade de se ter as funções do programa implementadas com uma quantidade mínima de código fonte, sem informação excessiva ou repetida (MCCALL, 1979) e (BOEHM, 1978).

Informações repetidas desviam a atenção do usuário leitor de detalhes importantes do fluxo lógico do programa. Isto leva o mesmo a dedicar mais tempo ao entendimento do programa, ainda mais se não participou de seu desenvolvimento, o que acontece na maioria dos casos. Logo, para facilitar e agilizar o entendimento do programa é que o mesmo deve ser conciso.

A linguagem de programação em que está codificado o programa é importante para a concisão, pois o número de declarações ou comandos é a próprio código fonte do programa, implementando sua lógica e funções. Quando um único comando representa o equivalente a muitas linhas de código, fica maiç fácil compreender seu significado e objetivo. Assim, linguagens de 20. geração, como ASSEMBLER, são menos concisas que linguagens de 3o. geração, como FORTRAN, COBOL e PL/I, que por sua vez são menos concisas que as de 40. geração (ARTHUR, 1985).

Para avaliar a presença do sub-fator **Concisão** em um programa deve-se considerar os seguintes critérios de qualidade:

- Não Anomalia

O critério **Não Anomalia** avalia se o programa possui práticas que o torne conciso, consequentemente, não possua práticas não recomendáveis que gerem apenas informações desnecessárias, redundantes, em excesso ou não utilizadas (BOEHM, 1978).

O excesso de código acaba cansando e desviando a atencão do usuário leitor. Assim, quanto maiç simplificado e

conciso o código, mais fácil e a leitura e acompanhamento do mesmo.

Existem muitas práticas não recomendáveis, dependenttes até mesmo da linguagem de programação utilizada, que só aumentam o tamanho do código do programa e são simplesmente desnecessárias ou não utilizadas. Exemplos de tais práticas são as seguintes:

- variáveis declaradas, mas não utilizadas;
- trechos do código nunca alcançados, gerando o que se chama de código morto;
- existência de subrotinas tipo "dummy" que só possuem comando de retorno, como por exemplo, no FORTRAN:

```
SUBROUTINE XXXX
RETURN
END
```

A avaliação de um programa segundo o critério **Não Anomalia** deve ser feita pela verificação de práticas que não aumentam a quantidade de código do mesmo.

- Não Repetição

O critério **Não Repetição** avalia se o programa não possui uma mesma sequência de código repetida em diferentes lugares desnecessariamente (BOEHM, 1978). Quando este tipo de sequência é transformada num único módulo, a quantidade de código do programa é reduzida, e consequentemente sua leitura torna-se mais rápida.

A presença de repetições, além de aumentar o número de linhas executáveis, pode ocasionar problemas durante sua manutenção, uma vez que é preciso fazer um mesmo tipo de alteração em mais de um lugar dentro do mesmo programa,

podendo nessa hora ocasionar erros e inconsistências. No caso de realmente haver necessidade da repetição explícita de um mesmo procedimento em diferentes lugares, é aconselhável o uso de mecanismos localizadores, como a Referência Cruzada, para que a manutenção seja feita consistentemente em todos os devidos lugares em que o mesmo for referenciado

Também, devido ao fato de diversas pessoas participarem da manutencão de um mesmo programa, muitas veres nem tendo participado de seu desenvolvimento, deve-se tomar cuidado para que um procedimento já descrito anteriormente não seja repetido desnecessariamente, com o mesmo grau de detalhe, em um outro ponto do programa (STAA, 1987).

A avaliação de um programa segundo o critério **Não Repetição** deve ser feita verificando se o mesmo não possui uma mesma sequência de código repetida em diferentes lugares desnecessariamente (BOEHM, 1978).

2.5.4 - Sub-fator Estilo de Programação

O sub-fator **Estilo** de **Programação** refere-se à necessidade de programa conter elementos de estilo, de forma que expresse seus objetivos e especificações de maneira clara, simples, elegante, organizada, direta, consistente e considerando as padronizações ou recomendações estabelecidas, sem precisar fazer referência à sua documentação externa (PRESSMAN, 1987).

A necessidade de se adotar um estilo de programação não é simplesmente para se obter um programa bonito e altamente

elaborado, mas principalmente para que ele seja legível e facilmente compreendido ao longo de sua vida útil.

Atualmente, na maior parte dos casos, a atividade de programação ainda é uma questão muito própria, de criatividade pessoal mesmo, oferecendo grande liberdade ao programador, sem regras que a orientem. Um exemplo disto é quando programadores, trabalhando juntos, reconhecem facilmente o estilo de programação de seus colegas de trabalho. Uma situação razoável seria que os programadores utilizassem suas características individuaiç mais adequadamente, ou seja, seguissem as recomendações ou regras de programação contidas na literatura técnica ou estabelecidas pela instituicão a que pertencer, considerando suas preferências pessoais onde fosse possível. Cem isso, haveria uma maior homogeneidade, em termos globais, nos produtos gerados por uma determinada instituição, considerados assim em bom estilo (FREITAS, 1985).

Os elementos de um estilo de programação podem envolver os seguintes:

- a documentacão interna do código fonte, com comentários adequados em relação aos seus conteúdos e localizações, com uma formação apropriada dos nomes dos identificadores ("labels" e variáveis) e com uma organização visual em relação à indentacão, linhas em branco e disposição dos comandos;
- recomendações para codificação, de acorda com a técnica de Programação Estruturada.

A aplicação de elementos de estilo de programação pode variar bastante, e o melhor estilo a ser adotado vai depender de cada situação e objetivo específico. De qualquer forma, o estilo de programação economiza muito tempo e esforço na manutencão de um programa que o tenha adotado.

Apesar da escolha e uso apropriado de uma boa linguagem de programação ser importantíssima, só isto não é suficiente para garantir a boa legibilidade de um programa. Além disso, a variedade de estilos adotados, ou a falta dos mesmos, tem prejudicado muito o trabalho dos analistas/programadores tanto na implementação de produtos de software grandes como na manutencão de programas incompreensíveis. Por tudo isso é que devem haver orientações para aplicação de um bom estilo de programação, fornecendo recomendações, regras e padrões de codificação sobre uma adequada documentação interna, o uso de indentação, comentários, linhas em branco, nomenclatura, parênteses e outros (BARGUT, 1986) e (STAHL, 1988).

No fundo, não existe um conjunto fixo e rígido de regras para se obter um bom estilo na codificação de um programa, e sim algumas recomendações gerais e dinâmicas que levam a uma certa homogeneidade na programação, ajudam no seu entendimento, são uma. Ótima documentação para sua lógica e podem ser aplicadas por qualquer programador.

Os próprios parâmetros envolvidos num estilo são variáveis e adaptáveis à realidade de cada instituição. Para generalizar, as abordagens do trabalho que se segue são consideradas para valores defaults.

Algumas recomendações gerais para um bom estilo de programação podem ser as seguintes (FAIRLEY, 1985), (KERNIGHAN, 1978) e (METCALF, 1985):

- usar a técnica de Programação Estruturada, com as estruturas básicas de controle de fluxo: de sequência (um bloco depois do outro indicando execução sequencial), de iteração (como os comandos DO-WHILE, DO-CONTINUE, FOR, REPEAT-UNTIL, ...), de seleção (como os comandos IF-THEN-ELSE, CASE, SELECT, GOTO COMPUTADO, ...), e de ativação de módulos (como os comandos CRLL, PERFORM, MACRO, ...);
- possuir um único ponto de entrada e um único ponto de saída para cada módulo;
- usar comandos GOTO quando extremamente necessário e de uma maneira disciplinada. O escopo de um comando GOTO deve ser limitado ao módulo em sue ele ocorre. Ou seja, o GOTO não deve ser usado para transferir controle de um módulo para outro, devendo ser usado apenas para desviar para o ponto de entrada ou para o ponto de saída da módulo a que pertence;
- proporcionar padrões de documentação;
- usar espaços em branco, linhas em branco, alinhamento das palavras-chave/atributos e margens em torno dos blocos de comentário para aumentar sua legibilidade;
- usar indentação de maneira consistente;
- usar parênteses de maneira consistente e para

evitar ambiguidades;

- simplificar, evitando complicações;
- evitar comandos associados THEN e ELSE nulos;
- evitar comandos TWEN-IF continuamente;
- não aninhar muito profundamente, usando, de preferência, até 3 níveis;
- examinar cuidadosamente os módulos que tenham mais de cinco parâmetros formais;
- não usar uma mesma variável para vários objetivos;
- declarar explicitamente todas as variáveis;
- declarar os atributos de cada variável;
- não usar nomes múltiplos para uma mesma área de memória;
- não comparar números reais;
- ser cauteloso quando em operações com inteiros;
- ser cauteloso ao usar módulos internos;
- evitar o uso de variáveis temporárias;
- ser cauteloso quando usar constantes como argumentos;
- evitar usar "labels" deçnecessarios e não usá-los como comentários;
- não ignorar as mensagens de advertência;
- evitar constantes literais;
- usar abreviações consistentes;
- usar, de preferência, um comando por linha;
- evitar misturar operações de tipos diferentes;
- usar um mesmo tipo de sintaxe por todo o programa

Para avaliar a presença do sub-fator Estilo de

Programação em um programa deve-se considerar os seguintes critérios de qualidade:

- Indentação

O critério Indentação avalia a utilização de afastamentos da margem esquerda ao longo do programa, para melhorar a estrutura lógica, destacar níveis de aninhamentos e blocos de comandos do mesmo (STAHL, 1988).

Os afastamentos da margem esquerda referem-se aos espaços em branco, deixados no começo da linha, geralmente entre a coluna um e a coluna onde se inicia o comando, dependendo da linguagem de programação. A indentacão é um tipo de formatação muito importante, pois permite organizar a disposição dos comandos, e mostrar a hierarquia, subordinação e aninhamento dos mesmos, tornando, assim, o programa mais fácil de ser entendido.

A indentacão é bastante usada, principalmente nas linguagens de programação estruturada e nas de formato livre.

Formatadores automáticos de programa podem ser usados tanto para indentar o programa como para ajustar as indentações inapropriadas ou incorretas, o que auxilia bastante o programador.

Assim, o uso da indentacão permite destacar e aumentar a visualização das declarações, "loops", procedimentos e blocos de comando importantes. Além disso, deve ser feita devidamente e quando necessária, ou seja, estando de acordo com as recomendações gerais para seu uso, pois quando utilizada de maneira exagerada, incorreta ou inconsistente, tende a levar o usuário leitor a ler de uma maneira

diferente o código do programa. Consequentemente, pode confundi-lo e induzí-lo a interpretar e tirar conclusões incorretas sobre a sua verdadeira lógica.

Existem algumas recomendações gerais para utilização da indentação, come as seguintes (MARTIN, 1983) e (KERNIGHAN, 1978):

- não codificar mais de uma declaração ou comanda numa mesma linha;
- evitar mais de três níveis de aninhamento;
- usar três espaços em branco para o afastamento;
- se um comando requer várias linhas, indentar suaç linhas de continuação;
- indentar a5 estruturas de controle do fluxo de iteração e de seleção, a fim de identificar mais claramente o seu conteúdo. Por exemplo, no comando IF, as palavras THEN e ELSE devem ficar em linhas separadas e geralmente indentadas em relação ao IF correspondente;
- estabelecer um esquema de indentação organizado e consistente para enfatizar a ordem de execução dentro das estruturas de controle do fluxo de sequência;
- alinhar e indentar a conteúdo dos blocos de código como BEGIN ... END e DO ... END;
- indentar as palavras em relação ao titulo, e as declarações em relação às palavras-chave;
- alinhar ítens de tipos similares numa mesma coluna;
- iniciar em linhas diferentes as claúsulas e

atributos para definicão de dados, mantendo um alinhamento vertical.

A avaliação de um programa segundo o critério **Indentação** deve ser feita pela verificacão da aplicação das recomendações gerais para utilização da indentação ao longe do mesmo.

- Comentário

O critério **Comentário** avalia os comentários existentes no programa, que são informações não executáveis que fornecem explicações sobre o seu funcionamento e lógica, ou seja, são o meio de comunicação textual do programa com os seus possíveis usuários leitores através de explicacões claras e diretas. Os comentários podem ser incluídos em qualquer lugar no programa, sendo a base de sua documentação interna. É composto de textos e explicacões que facilitam o entendimento das declarações, comandos e procedimentos ao longo do programa, devendo ser os mais completos e sucintos possíveis, ou seja, dizendo tudo em poucas e diretas palavras. A inclusão de comentários, além de facilitar e agilizar o entendimento do funcionamento do programa, reduz também a necessidade de referência à sua documentação externa, tornando-o, assim, auto-deserit ivo.

Os comentários devem proporcionar informações adicionais corretas, que não sejam prontamente obtidas do código fonte, sem atrapalhá-lo, reproduzí-lo simplesmente ou levar a conclusões erradas, dificultando a própria detecção de erro. Dessa maneira, seu intuito é simplesmente facilitar a leitura e entendimento dos programas, principalmente oç longos e complexos. Deve-se tomar cuidado, pois seu uso

excessivo, ou seja, a nível de cada declaração ou comando, ou o posicionamento indevido pode prejudicar enormemente seu percurso visual e até atrapalhar na interpretação apropriada e correta da lógica e das funções descritas no mesmo. Logo, não há necessidade de se ter um comentário para cada comando, para cada estrutura de controle ou para cada ponto de decisão do mesmo.

Os comentários podem ser de dois tipos, o comentário Introdutório, Prólogo ou Cabeçalho, localizado no inicio do programa ou do módulo, e o comentário Explicativo ou Descritivo, incluído ao longo do programa onde necessário.

O comentário tipo Cabeçalho/Prólogo é uma informação muito importante e valiosa, principalmente para os usuários leitores que vêem pela primeira ver o programa. Ele é uma incialização e preparação para o melhor entendimento da programa, e pode conter as seguintes informações, a serem preenchidas de acardo com sua realidade e necessidade (PRESSMAN, 1987):

- nome do código fonte;
- objetivos/funções realizadas;
- exemplo de uma chamada;
- argumentos ou parâmetros;
- assertivas de entrada e de saída;
- principais variáveis globais;
- principais estruturas de dados;
- subrotinas referenciadas (internas e externas);
- restrições de tempo;
- tratamento de excessõeç;
- nome do desenvolvedsr e data da codificação;

- indicacão de manutencão;
- nome do responsável pela manutencão;
- data da manutencão;
- motivo da manutencão.

O comentário tipo Explicativo/Descritivo pode conter, principalmente, os seguintes tipos (STAA, 1987):

- comentário a nível de declaração ou comando;
- comentário geral, que descreve os procedimentos do programa e os algoritmos utilizados, as restrições e/ou requisitos de uso;
- comentário de instrumentação, que refere-se à comandos desativados e facilmente reativáveis, com o intuito de auxiliar nos testes.

As assertivas de entrada e de saída são bons comentários, uma ver que as mesmas são informações bem esclarecedoras sobre os pré-requisitos de entrada e as condições de saída a serem satisfeitas de acordo com o correto funcionamento do programa.

Vale lembrar sue o mais importante é o conteúdo dos comentários e não seu volume.

O número desejável de comentários num programa é bem variado. Geralmente linguagens de mais baixo nível precisam de mais comentários que as de alto nível.

Existem algumas recomendações gerais para inclusão e utilização de comentários, que são as seguintes (STAHL, 1988):

- os comentários devem ser escritos ao mesmo tempo que o programa, para os detalhes não serem esquecidos;

- os comentários devem estar corretos e atualizados;
- os comentários devem concordar e transmitir informações sobre o código a que está relacionado;
- a utilização de palavras ou expressões em inglês só devem ser permitidas para aquelas já de usa geral, que não apresentem qualquer dúvida sobre seu significado;
- os comentários devem estar descritos em português simples, claro, preciso e conciso, evitando mnemônicos ou abreviações;
- os comentários devem conter, principalmente, o objetivo ou o resultado esperado;
- os comentários não devem ser uma repetição do que já estiver declarado no código;
- evitar o exceçso de comentários, procurando descrever blocos de código ao invés de comentar cada linha;
- os métodos usados devem ser explicados nos comentários quando forem especiais ou complexos;
- escrever e destacar o Cabeçalho/Prólogo;
- destacar visualmente os comentários de maneira consistente ao longo do programa, usando linhas em branco antes e depois;
- os comentários devem possuir um padrão para formatação;
- os comentários podem preceder os seguintes tipos de operações de módulo:

- ramo condicional: descrevendo porque cada ramo é escolhido;
- matemática/lógica: descrevendo o propósito de uma operação para algoritmos Únicos e complexos.

A avaliação de um programa segundo o critério Comentário deve ser feita pela verificação da aplicação das recomendações gerais para inclusão e utilização de comentários ao longo do mesmo.

- **Identificação**

O critério **Identificação** avalia se a formação dos nomes de identificadores, variáveis ou argumentos, referenciadao ao longo do programa, é feita de forma significativa, apropriada e mnemônica, e se segue algum tipo de padronização estabelecida pela instituição (BARGUT, 1986). Isto, consequentemente, reduz a necessidade de comentários específicos sobre os mesmos e agiliza muito o entendimento do programa.

Os nomes utilizados devem ser auto-explicativos e mnemônicos, ou **seja**, significativos, de forma a representar a função desempenhada, enfatiaar a estrutura lógica em que ***são*** empregados e ajudar a tornar o programa auto-documentado. Com isso, facilita o entendimento, reduz sensivelmente o esforço empregado nas manutenções, descreve e esclarece a função dos identificadores de maneira rápida e fácil, minimizando a necessidade de comentários sobre os mesmos. Esta auto-explicação pode levar a nomes compostos por sequências de caracteres longos, e aí deve-se usar o

bom senso para não se criar nomes longos desnecessariamente. Existem linguagens de programação que limitam os nomes a uns poucos caracteres, eomo o FOWTRAN 77 ANSI que 56 permite até 6 caractereç, o que dificulta o entendimento de seu significado. Neste caso, deve-se procurar utilizar abreviaçõeS o mais inteligíveis possível e fornecer uma relação das principais variáveis e seus significados. Assim, é muito importante que seja feita a devida criação de nomes (PRESSMAN, 1987).

O uso de nomes significativos pode também ser empregada para nomes de programas, subprogramas, módulos, procedimentos, funções, arquivos, relatórios ou telas.

Existem algumas recomendações gerais para a escolha de nomes de identificadores, como as seguintes (MARTIN, 1983):

- seguir a padronização de nomenclatura estabelecida pela instituição;
- escolher nomes que associem aos mesmos o objetivo, a propriedade física ou funcional que representam ou desempenham, de forma a serem mnemônicos ou significativos;
- evitar usar nomes sinônimos (duas variáveis com a mesmo objetivo);
- evitar usar nomes similares, ou semelhantes, que sejam difíceis de ler e estejam sujeitos a erros de digitação. Devem diferir de pelo menos doiç caracteres;
- evitar escrever os nomes de uma maneira não usual;
- escrever nomes Únicos, dificilmente confundíveis

com outros e fáceis de reconhecer;

- aproveitar as vantagens que a linguagem de programação utilizada oferecer para a criação de nomes;
- usar nomes tão longos quanto o necessário;
- usar nomes usuais à área de aplicação;
- usar nomes menores, abreviadas, para variáveis frequentemente usadas, como variáveis locais;
- usar nomes mais longos, mais descritivos, para variaveis menos usadas, como variáveis globais;
- usar um esquema consistente e pronunciável para abreviação dos nomes ou para indicar variáveis complexas;
- usar um prefixo/sufixo comum para identificar os nomes que estejam agrupados logicamente ou que indiquem ítens relacionados por algum motivo;
- evitar usar palavras-chave da linguagem de programação empregada;
- evitar usar nomes em inglês e português ao mesmo tempo, só usando aqueles já de usa geral;
- colocar os numerais de preferência no final dos nomes;
- tomar cuidado com os numerais 0,1,2 e 5, pois são facilmente confundidos com as letras O,I,Z e S.

A avaliação de um programa segunda o critério Identificação deve ser feita pela verificação da aplicação das recomendações gerais para a escolha de nomes ao longo do mesmo.

- **Programação Estruturada**

O critério **Programação Estruturada** avalia se o programa está escrito de acordo com as normas da técnica Programação Estruturada, que é utilizada para construir a lógica de um programa através de estruturas lógicas de controle e suas combinações (HEHL, 1986).

Os principais objetivos da programação Estruturada são os seguintes (JENSEN, 1981) e (STAA, 1987):

- minimizar o número de erros ocarridos durante o desenvolvimento do programa;
- minimirar o esforço requerido para corrigir erros em partes deficientes do código, atualizando essas partes com técnicas mais confiáveis, funcionais ou eficientes;
- minimizar os custos de manutenção do software;
- gerar programas simples, claros e bem definidos em termos estruturais;
- produzir código que possa ser lido sequencialmente em pequenas partes, permitindo a leitura dessas partes de forma top-down e visualizando nessas mesmas todos os seus caminhos de controle.

Para tanto, a aplicação da Programação Estruturada envolve as seguintes práticas (FAIRLEY, 1985), (SHOOMAN, 1983) e (MARTIN, 1983):

- uso disciplinado e restrito de comando GOTO, evitando-o sempre que possível. O uso de GOTO tende a violar o fluxo dinâmico de execução do programa, e pode também destruir a independência

de um módulo caso o GOTO desvie para fora dos limites do mesmo. Desvios para trás devem ser evitados, exceto quando se implementam construções DO-WHILE ou REPEAT-UNTIL em "loops" que não dependem de um contador. Desvios para frente devem ser usados para situações que tratem de condições de erro ou para realmente aumentar a clareza de algum procedimento. Não devem ser usados para transferir o controle de um módulo para outro, para dentro de um "loop" do comando DO, para dentro de um bloco do comando IF. Pedem ser usados, se necessário, para desviar para o ponto de entrada ou para o ponto de saída da módulo a que pertence;

- a codificação deve proporcionar uma solução simples e direta;
- uso de uma declaração ou comando por linha;
- uso de estruturas de controle de fluxo linear: devem ser usadas estruturas de **sequência** (a maneira natural de execução de um programa, que se realiza de um comando para o outra imediatamente seguinte), de **iteração** (ou repetição, implementada através de comandos como DO-WHILE, 80, FOR, REPEAT-UNTIL) e de **seleção** (para escolha entre caminhos de execução do programa, implementada através de comandas como IF-THEN-ELSE, CASE, SELECP, GOPO COMPUTADB, em que apenas uma única condição seja satisfeita). Estas estruturas de controle de fluxo diminuem a

complexidade do programa por reduzir o número de instruções necessárias e por impor um formato com um único ponta de entrada e um único ponto de saída;

- uso de Programação Modular, que é uma outra técnica de programação que consiste na divisão do programa em partes lógicas, os módulos, a fim de simplificar o seu desenvolvimento. Ela, por sua vez, reflete a estratégia de construção top-down, que é um método para decomposição de um programa em partes menores, por um processo de refinamentos sucessivos até obterem-se estruturas pequenas, de tarefa relevante e que podem ser resolvidas separadamente;
- uso de construções de entrada e saída Únicas: cada bloco de programa deve possuiu apenas um Único ponto de entrada e um Único ponto de saída.

No FORTRAN 77 ANSI, a estrutura de seleção é feita pele comando IF, e a de iteração, pelo comando DO, que depende de um contador para controle da execução. Para se implementar estruturas do tipo DO-WHILE ou REPEAT-UNTIL, pode-se usar as seguintes construções:

- construção simuladora do DO-WHILE:

```
n CONTINUE
  IF (condição) THEN
      comando 1

      comando m
  GO TO n
  END IF
```

- construção simuladora do REPEAT-UNTIL:

```
n CONTINUE
  comando 1

  comando m
  IF (not condição) GOTO n
  END IF
```

A avaliação de um programa segundo o critério **Programação Estruturada** deve ser feita pela verificação da obediência às práticas da Programação Estruturada.

- Organização Visual

O critério **Organização Visual** avalia **se** os comandos do programa estão dispostos de maneira organizada, de forma a melhorar a aparência e o percurso visual ao longo do mesmo.

Essa postura facilita e agiliza o entendimento do programa, permite que seu código evolua de maneira sistemática, oferece uma visão mais clara de sua estrutura e fica esteticamente mais elegante e agradável de se ver e ler.

A sequência de comandos codificados num programa determina todo o seu entendimento, e nem uma grande quantidade de comentários ou uma documentação suplementar pode substituir inteiramente estes comandos bem expressos (KERNIGHAN, **1978).**

A organização em um programa envolve as disposições e atribuições dos seguintes (METCALF, **1985):**

- labels;
- nomes;
- linhas de comando;
- declarações e comandos;
- estruturas de dados.

Dentro dos comandos de declaração, os próprios argumentos, quando for o caso, podem estar organizados, de acordo com o seu pretendida uso, na seguinte ordem (METCALF, 1985):

1o.- para entrada somentej

20.- para saída somente;

30.- para entrada e saída;

40.- como variáveis de controle;

5o.- como variáveis temporárias;

6o.- como nomes externos.

A organização de declarações e comandos pode se tornar mais simples e clara se forem consideradas as seguintes recomendações (PRESSMAN, 1987):

- evitar o uso de testes condicionais complicados;
- eliminar testes nas condições negativas;
- evitar aninhamento profundo de "loop" e de condições;
- usar parênteses nas expressões aritméticas e lógicas, para esclarecer ambiguidades e evitar erros;
- adotar colunas especificas para iniciar o comando, alinhando sempre pelas mesmas;
- usar espaços em branco e/ou símbolos de legibilidade para esclarecer o conteúdo;
- usar a padronização estabelecida;
- colocar todas as declarações de formato, como o comando FORMAT de FORPRAN juntas;
- pensar da seguinte forma: "Poderia entender isto se eu não fosse a própria pessoa que o

codificou ?".

Os espaços em branco ajudam a delinear os elementos sintáticos de uma declaracão ou comando. Podem ser usados depois de vírgulas, antes/depois de operadores de expressão e do sinal de atribuição (=). Não é aconselhável colocar muitos espaços num mesmo comando ou muitas linhas em branco juntas, pois acabam reduzindo a legibilidade do mesmo.

Quanto à disposição de nomes em listas, eles podem ser colocados em ordem alfabética e por colunas, facilitando, assim, a localização de um nome numa lista (STAHL, ***1988).***

A utilização de linhas em branco ao longo do programa, de forma não abusiva, também facilita a visualização das suas partes mais significativas. Com isso, permite destacar e aumentar a visualização dos comentários, cabeçalhos, títulos, declarações, "loops", procedimentos e comandes importantes.

Existem algumas recomendações para inclusão de linhas em branco, como as seguintes (METCALF, 1985).

- entre declarações e o corpo do programa;
- para grupos de comandos com funções lógicas distintas;
- precedendo e seguindo comentários;
- precedendo comandos indentados importantes;
- separando grupos de declarações ou camandos semelhantes.

BAECKER ***(1978)*** fez um trabalho sobre visualização de programas em linguagem C, onde identifica sete áreas fundamentais para uma boa apresentação do texto do código fonte:

- introdução automática de espaço em branco;
- boa quebra de linha automática;
- boa quebra de página automática;
- incorporação de intenções de formatação do programador;
- apresentação de padrões de formatação;
- avisos automáticos compreensíveis;
- anotações automáticas compreensíveis.

A avaliação de um programa segundo o critério **Organização Visual** deve ser feita pela verificação da organização e disposição dos comandos ao longo do mesmo.

2.5.5 - Sub-fator Modularidade

O sub-fator Modularidade refere-se à necessidade de se ter as funções do programa implementadas através de uma estrutura de módulos altamente independentes e particionados logicamente (ROCHA, 1987).

Um programa dividido em módulos funcionais, devidamente projetados, possibilita a redução de sua complexidade e torna-se bem mais simples e fácil de ser construido, modificado, tratado e compreendido, pois não há necessidade de se entender o programa inteiro para se alterar uma parte dele, podendo ser considerada separadamente cada uma de suas partes (CONTE, 1986). Uma consequência disso, é que quando uma biblioteca de módulos cresce, os programas podem ser implementades usando cada ver menos código novo e mais código reutilizável.

Assim, o objetivo da Modularidade é encontrar, num programa, uma tarefa complexa dividida em subtarefas

independentes. Este processo de decomposição em subtarefas, que são os módulos, possui portanto as seguintes finalidades (FAIRLEY, 1985):

- impor ordenação hierárquica;
- implementar abstrações de dados;
- desenvolver módulos independentemente Úteis;
- isolar as dependências de hardware;
- melhorar o desempenho;
- facilitar a depuração, teste, integração e manutenção do mesmo.

A definição e especificação da estrutura modular do programa, contida na sua documentação externa, deve ser feita durante a atividade de projeto do software, utilizando as considerações e técnicas do Projeto Estruturado (ROCHA,1987). Logo, este trabalho considera já realizada a avaliação da qualidade dessa definição e especificação, preocupando-se aqui em como ela está implementada.

Para avaliar a presença de sub-fator Modularidade em um programa deve-se considerar o5 seguintes critérios de qualidade:

- Não Acoplamente

O critério Não Acoplamento avalia se o programa possui um fraco relacionamento entre os módulos que se comunicam entre si. É um modo de particionamento da programa através do interrelacionamento ou interdependência entre dois módulos do mesmo (FREITAS, 1985).

Um forte acoplamento entre os módulos é prejudicial ao

software, pois torna-se mais difícil entender, mudar e corrigir apenas um Único módulo se ele estiver altamente relacionado com algum outro módulo. Já um fraco acoplamento indica um programa bem particionado, com um menor relacionamento entre os seus módulos, com a complexidade reduzida e altamente independente, diminuindo a chance de um erro em um dos módulos repercutir em outro.

O acoplamento entre dois módulos e influenciado por certos fatores, como complexidade da interface, tipo de conexão e tipo de comunicação entre os mesmos (FAIRLEY, 1985). Logo, depende do quão complicado é a associação entre os mesmos, do relacionamento referir-se ao próprio módulo ou algo dentro dele e do que está sendo enviado au recebido, ou seja, os dados e resultados intercambiados pelos mesmoç. Biante disto, o relacionamento avalia o tráfego de dados, que deve ser pequeno para que se tenha um fraco acoplamento (STAA, 1987).

Come o acaplamento depende dos dados passados entre os módulos, eles devem ser do mesmo tamanho e devem ter os mesmos atributos, ou seja, nunca deve-se passar dados alfabéticos em campos numéricos e nem passar dados binários em decimal compactado (ARTHUR, 1985).

Existem definidos na literatura cinco tipos de acoplamento que podem ocorrer entre dois módulos. Estes são os seguintes, em ordem decrescente de importância para que ocorra um fraco acoplamento: Acoplamento de dados, Acoplamento por dados estruturados, Acoplamento por controle, Acoplamento por área comum e Acoplamento por conteúdo (PAGE-JONES, 1980), (SHOOMAN, 1983) e (FAIRLEY,

1985).

- **Acoplamento de dados:** a comunicação entre os módulos é feita através de uma lista de parâmetros, onde o parâmetro pode ser tanto um dado elementar como uma tabela homogênea para passagem dos dados.

Esta é a melhor forma para um fraco acoplamento, pois acarreta em interfaces concisas e facilmente testáveis.

Segundo METCALF (1985), no FORTRAN 77, "se a eficiência está relacionada, o método mais rápido é normalmente usar blocos COMMBN ao invés de listas de parâmetros. Isto vem do fato de que os valores de variáveis COMMBN são armazenadas e recolhidos por referências diretas aos seus endereços, ao passa que para listas de parâmetros, os endereços são normalmente armazenadas como alguma forma de tabela que tenha sido construida e que seja referenciada indiretamente através de um ponteiro".

- **Acoplamento por dados estruturados:** a comunicação entre os módulos é feita pelo compartilhamento de uma mesma estrutura de dados interna e não simplesmente pelos dados necessários.

Este tipo tende a fornecer a um dos módulos mais. dados do que ele necessita.

- **Acoplamento por controle:** a comunicacão entre os módulos é feita quando um módulo passa para outro um campo de informação ("flag"), que é usado para controlar a lógica interna deste segundo módulo.

Este tipo indica um pobre particionamento, pois tanto uma função pode estar dividida em dois módulos, como um módulo tem de saber da lógica interna detalhada do outro.

Mas nem todas os "flags" são prejudiciais. Existe o "fias" descritiva que diz ao módulo chamador o que fazer e o que deve ser evitado.

- **Acoplamento por área comum**: a comunicação entre os módulos é feita quando eles se referem a uma mesma estrutura de dados global. Como por exemplo, através do comando COMMON do FORTRAN.

Existem algumas desvantagens no seu uso, como as seguintes:

- programas com muitos dados globais são extremamente difíceis de se fazer manutenção, pois é difícil saber que dados são usados por um módulo especifico;
- sem uma listagem de Referência Cruzada, é difícil encontrar os módulos que devem ser trocados se um dado for trocado;
- os dados globais não ficam em área protegida, logo, um erro em qualquer módulo pode alterar, incorretamente, um dos dados.

- **Acoplamento por conteúdo**: a comunicação entre os módulos é feita quando um dos módulos faz referência diretamente aos dados de dentro de um outro módulo.

Esta é a pior forma de acoplamento, pois qualquer módulo pode violar um outro. Somente a linguagem ASSEMBLER, através do uso de constantes de endereçamento, permite tal. prática. Linguagens de mais alto nível dificilmente oferecem alguma maneira de se implementar este tipo de acoplamento.

A avaliação de um programa segundo o critério Não

Acoplamento deve ser feita através da identificação dos tipos de acoplamento existentes entre as interfaces de cada par de módulos interrelacionados do mesmo (RADC-TR-85-37).

- Coesão

O critério **Coesão** avalia se o programa posçui um forte relacionamento entre os parâmetros ou argumentos de seus módulos. Com isso, avalia a força de associação funcional entre os parâmetros de cada módulo, ou seja, a força que mantém unidos estes elementos (FREITAS, 1985).

O alto nível de coesão no módulo é o fator mais significativo para um alto nível de modularidade, acarretando num módulo com parâmetros genuinamente relacionados, pois é um indicador da aproximação entre a estrutura do programa e a estrutura do problema. implementado. No caso dos parâmetros de um módulo estarem fortemente relacionados a parâmetros de outro módulo, ou seja, hauver um forte acoplamento, acarreta na diminuição da coesão de ambos (FREITAS, 1985).

A coesão está baseada nas diferentes maneiras que o tráfego de dados entre declarações ou comandos, ou melhor, a troca de informação por intermédio de um conjunto de dados, pode se relacionar ao fluxo de controle através destas mesmas declarações ou comandos (EMERSON, 1984). Este trafego deve ser alto para que se tenha um alto nível de coesão (STAA, 1987).

Existem definidos na literatura oito tipos de coesão que podem ocorrer num módulo. Estes são os seguintes, em ordem crescente de importância para que ocorra uma alta coesão: coincidental, lógica, temporal, procedural, comunicacional,

sequencial e funcional (PAGE-JONES, 1980), (SHOOMAN, 1983), (STAA, 1987), (ARTHUR, 1985) e (FAIRLEY, 1985).

- **Coesão coincidental**: os parâmetros do módulo não têm razão aparente para estarem juntos, uma vez que não existe relacionamento significativo entre os mesmos.

- **Coesão lógica**: existe algum tipo de relacionamento lógico entre os parâmetros do módulo, que contêm muitas funções comuns que estão numa mesma categoria e são executadas uma de cada vez de acordo com uma seleção de controle feita pelo próprio módulo. Coma por exemplo, um módulo que realiza todas as entradas e todas as saídas para o programa.

Este tipo parece-se muito com a Coesão coincidental, uma vez que as funções desta não estão relacionadas nem quanto ao fluxo de dados e nem quanto ao fluxo de controle, e já na Coesão lógica, suas funções estão, no mínimo, na mesma categoria.

- **Coesão temporal**: os parâmetros do módulo estão envolvidos em funções diferentes e relacionados pelo tempo, sem uma ordem lógica de execução. Por exemplo, um módulo que faz a inicialização do programa, pois seus elementos estão relacionados a tempo de inicialização

- **Coesão proeedural:** os parâmetros do módulo estão envolvidos em funções diferentes, onde o fluxo de controle, ou seja, a sequência lógica, segue de uma função para a próxima. Como por exemplo, ler arquivo e imprimir saída.

Módulos com Coesão procedural tendem a ser compostos por partes de funções sue tenham pouco relacionamento entre si, exceto quando são executadas numa ordem especifica por um

mesmo período de tempo.

Este tipo parece-se muito com a Coesão temporal, uma vez que os parâmetros desta são executados no mesmo período de tempo, e já na Coesão procedural, a ordem de execução é importante.

- **Coesão comunicacional**: os parâmetros do módulo estão relacionados por uma referência ao mesmo conjunto de dados de entrada e/ou de saída. O módulo geralmente executa muitas funções relacionadas pelo uso de seus dados, como por exemplo, imprimir e perfurar um arquivo de saída.

- **Coesão sequencial**: ocorre quando o dado de saída de um parâmetro do módulo serve come entrada para um outra elemento do mesmo. Como por exemplo, ler a próxima transação e autorizar o arquive mestre.

Este tipo parece-se muito com a Coesão comunicacional, só que para a Coesão sequencial, a ordem de execução é importante.

Este tipo normalmente tem uma forte coesão, pois pode conter várias funções ou simplesmente parte de uma função. A única desvantagem real é que ele não é prontamente reutilizável por outras partes do software.

- **Coesão funcional**: todos os parâmetros do módulo estão relacionados apenas para a execução de uma Única função. Como por exemplo, calcular a raiz quadrada. É o tipo mais alto de coesão, sendo prontamente reutilizável por outras partes do software.

Uma outra maneira de se identificar uma Coesão funcional, é verificando se todos os parâmetros de módulo contribuiram para atingir um único objetivo.

No caso de erro, um módulo deste tipo é facilmente localizado e alterado.

Para se verificar o tipo de coesão em um módulo, deve-se primeiro verificar o número de pontos de entrada e de saída do mesmo. Se houver mais de um ponto de entrada, o módulo geralmente pode possuir Coesão comunicacional, lógica ou coincidental. Se houver um único ponto de entrada e vários pontos de saída, o módulo geralmente pede possuir Coesão comunicacisnal, lógica ou coincidental. Se houver um Único ponto de entrada e um único ponto de saída, o módulo geralmente pode possuir Coesão funcional, procedural ou temporal (ARTHUR, 1985).

A avaliação de um programa segundo o critério Coesão deve ser feita através da identificação dos tipos de Coesão existentes em cada módulo do mesmo.

- Número de Módulos Superiores

O critério **Número de Módulos Superiores** avalia se o programa possui um adequado número de módulos superiores, que chamam um mesmo módulo subordinado. Neste caso, ocorre a concentração de ativação de um módulo, ou seja, o "fan-in" (NASCIMENTO, 1986). O valor adequado indicado pela literatura é de um a três, mas a pratica e a observação de cada instituição pode alterá-lo.

Quando isto acontecer, é uma indicação de uma mais fácil reutilização do módulo chamado, consequentemente, menor é a complexidade de organização do programa contendo menos módulos individuais e evitando-se a duplicação de código. Logo, também agiliza o entendimento do programa.

A avaliação de um programa segundo o critério **Número de Módulos Superiores** deve ser feita pela verificação da quantidade de módulos superiores sue chamam um mesmo módulo subordinado.

- **Número de Módulos Subordinados**

O critério **Número de Módulos Subordinados** avalia se o programa possui um baixo numero de módulos imediatamente subordinados chamados por um determinado módulo superior. Neste caso, ocorre a distribuição de ativação de um módulo, ou **seja**, o "fan-out" (NASCIMENTO, 1986). O valor adequado indicado pela literatura é de um a nove, mas a pratica e a observação de cada instituição pode alterá-lo.

Quando isto acontecer, é uma indicação de uma mais difícil reutilização do módulo que chamou, pois ele depende de muitas outros módulos, e que aumenta bastante a quantidade de módulos a serem reutilimados.

Um número muito alto ou muito baixo de módulos imediatamente subsrdinados ***são*** indicadores de um projeto pobre, embora um número alto seja mais perigoso do que um número baixo.

A avaliação de um programa segundo o critério **Número de Módulos Subordinados** deve ser feita pela verificação da quantidade de módulos imediatamente subordinadoa ao mesmo módulo.

- **Tamanho**

O critério **Tamanho** avalia se o programa não ultrapassa um número padrão de linhas de código executáveis estabelecido. Uma linha de código executável é qualquer

linha executável com comandos executáveis do código fonte de programa ou do módulo, excluindo linhas em branco, de comentários, de comandos de declaração e de comandos de designação de memória (VINCENT, 1988).

Além das linhas de código, existem outras tendências para medida do tamanho de código fonte, como as seguintes (CONTE, 1986) e (COOK, 1987):

- número de funções, grupando linhas que suportam funções bem definidas;
- número de comandos existentes;
- número de linhas fonte entregues, cantando o número de linhas físicas, e linhas que são de brancos ou exclusivamente de comentários.

Foram realizadas várias pesquisas para estabelecer um limite superior para o tamanho de um módulo. Baker considera que 50 comandes é um número adequado, por aproximar-se do número de linhas que podem ser impressas em uma página de listagem na impressora. Weimberg considera 30 comandos, para que não haja uma brusca queda da compreensão do módulo. Boehm já considera 100 comandos (FREITAS, 1985) e (NASCIMENTO, 1986). Arthur considera que o limite de tamanho de um módulo é de 100 linhas de código executável, exceto para módulos de edição (ARTHUR, 1985). Vincent também considera que cada módulo não deve exceder a um tamanho padrão de 100 comandos (VINCENT, 1988). Card, de uma pesquisa com 453 módulos escritas em FORTRAN, concluiu que (CARD, 1985):

- bons programadores não têm preferência por um tamanho especifico para os módulos;

- módulos grandes custam menos, por declaração executável, do que módulos pequenos;
- a incidência de erros não está relacionada ao tamanho do módulo.

Quando o programa necessitar de meios apropriados para reagir a situações hostis ou para evitar falhas sue provoquem consequências desastrosas, necessitando para tanto um procedimento para tratamento de erros ou de exceção bem detalhista e abrangente, ele provavelmente conterá mais comandos do que o realmente necessário, por uma questão de controle requisitado pelo software. Nestes casos, a complexidade e o tamanho aumentam.

Outra abordagem para verificar o tamanho de um programa é através do Comprimento de Halstead, baseado na hipótese de que um programa bem estruturado é uma função apenas de seus operadores e operandos Únicos para definir o conceito de comprimento. Onde os operadores são os próprios operadores de expressão e comandos da linguagem, e operandos são os nomes de variável e constantes (VINCENT, 1988).

Esta medida de Halstead é um indicador da qualidade de operadores e operandos existentes no programa, logo, quanto menor for o seu valor, menor o tamanho e mais conciso fica o programa.

A linguagem de programação utilizada também pode influenciar no Comprimento de um programa, pois uma vez que ela possua comandos que dêem uma explicação concisa do que farem, já economizam e substituem vários operadores, operandos e linhas de código.

A equação que define o Comprimento baseia-se no cálculo do Comprimento observado e do Comprimento calculado que encontram-se na tabela 2

SÍMBOLOS		EQUAÇÕES
n1	número de operadores únicos	
n2	número de operandos únicos	
Nc	Comprimento calculado	$Nc = n1\log_2 n1 + n2\log_2 n2$
N1	número total de operadores	
N2	número total de operandos	
No	Comprimento observado	$No = N1 + N2$

Tab. 2 - Equações sobre Comprimento da Ciência de Software
Fonte: (VINCENT, 1988)

Segundo Vincent (1988), para cada módulo do programa, o Comprimento é o valor médio dos valores do Comprimento de todos os módulos do mesmo, ou seja, $\frac{Nc - No}{No}$.

Diante de tudo isto, o limite máximo para tamanho adotado é de 100 linhas de código executáveis. Como é um valor originado da literatura, a prática e a observação de cada instituição podem alterá-la.

A avaliação de um programa segundo o critério Tamanho deve ser feita pela verificação do número de linhas de código executáveis do mesmo.

- **Balanceamento**

O critério **Balanceamento** avalia se o programa possui uma estrutura de módulos balanceada, ou seja, organizada de forma que apenas seus módulos de mais baixo nível envolvam as características físicas (NASCIMENTO, 1986).

Assim, objetiva-se uma estrutura balanceada onde os módulos de nível mais alto tratam os dados de uma forma lógica, de modo que estes sejam independentes de características físicas, e aos de nível mais baixo são atribuídas essas características físicas.

Diante disto, as estruturas balanceadas são mais fáceis de se implementar e fazer manutenções, principalmente porque estas envolvem os dispositivos físicos ou os formatos de Entrada/Saída.

A avaliação de um programa segundo o critério **Balanceamento** deve ser feita verificando se a estrutura em módulos do programa está balanceada.

- **Não Memorização**

O critério **Não Memorização** avalia se a programa não possui qualquer tipo de memória, ou seja, de armazenamento, requerido para manter o estado das variáveis entre ativações sucessivas de seus módulos (STAA, 1987). Este tipo pode ser uma memória de estado ou uma memória temporária.

A memória de estado é a memória prévia da existência do módulo. Geralmente quando um módulo é executado, ele não tem memória sobre sua existência anterior, ou seja, sobre o que aconteceu em outras execuções anteriores. Entretanto, existe um tipo de módulo que pode conter um registro de seu

passado, come se fosse um histórico, através de sua memória de estado. Um módulo possuidor desse tipo de memória torna-se imprevisível, pois para entradas idênticas comporta-se diferentemente, produzindo resultados inesperados a cada execução.

A memória temporária é a memória, ou melhor, a área de trabalho, compartilhada entre módulos (VINCENT, 1988). Neste caso, estes mesmos módulos correm o risco de perderem sua independência, tornando-se também imprevisíveis, pois a mudança da ordem de execução dos módulos pode provocar resultados inesperados. Logo, o módulo deve possuir memória própria, não compartilhável com outros módulos.

Para evitar esses tipos de problema com a interpretação dos resultados, principalmente durante a manutenção, e poder assim não se perder, entendendo o devido comportamento, e manter a independência de cada um dos módulos do programa, é que deve-se centralizar o tratamento de variáveis através de memória própria em cada módulo.

A avaliação de um programa segundo a critério **Não Memorização** deve ser feita pela verificação da não existência de algum tipo de memória de eçtado nos seus módulos ou de memória temporária entre seus módulos.

2.5.6- Fator Manipulabilidade

O fator **Manipulabilidade** refere-se à necessidade de se poder ter facilidade no acesso e na manipulação do programa pelos seus possíveis usuários leitores, ao longo de sua vida Útil (ROCHA, 1987).

Dentro do ciclo de vida do software, tanto ao longe do processo de desenvolvimento, na implementação, como na manutenção/operação de mesmo, o programa deve poder ser manipulado pelos seus possíveis usuários leitores, os quais não precisam necessariamente ter participado do seu desenvolvimento. A fim de atender à essa necessidade, o intuito desta característica é proporcionar ao usuário leitor um programa que seja facilmente manipulável.

Para avaliar a presença do fator **Manipulabilidade** em um programa deve-se considerar os seguintes sub-fatores de qualidade: Disponibilidade e Rastreabilidade.

2.5.7 - Sub-fator Disponibilidade

O sub-fator **Disponibilidade** refere-se à necessidade do programa estar disponível e pronto para poder ser acesçado e usado por seus possíveis usuários leitores na sua versão mais atualizada (ROCHA, 1987).

Assim, a Disponibilidade avalia a facilidade de acesso à versão mais atualizada do programa, consequentemente, de sua documentação interna que está contida no próprio código fonte. Quando houver necessidade de consulta à este, é primordial que o mesmo esteja guardado num meio físico de fácil apresentação, que pode ser uma biblioteca de programas fonte, exibida por meio de um terminal de vídeo, ou então uma biblioteca de listagens impressas em papel.

Uma preocupação que se deve ter é com a preservação da Disponibilidade, que influencia e exige as providências adequadas durante cada manutenção do programa, pois se este não estiver atualiradamente disponível, possivelmente não

acompanhará a vida Útil do software. Por isso, todas as cópias existentes do programa devem ser sempre devidamente alteradas e substituídas pelas versões mais recentes, mesmo çabendo que isto acarreta em mais trabalho.

Para avaliar a presença do áub-fator **Disponibilidade** em um programa deve-se considerar os seguintes critérios de qualidade:

- **Acessibilidade**

0 critério **Acessibilidade** avalia a facilidade de acesso ou consulta ao programa feita por qualquer usuário leitor autorizado. 0 próprio código fonte e a sua respectiva documentação interna devem ser utilizados, quando preciso, por diferentes usuários leitoreç, por isso é que o programa deve ser acessado de maneira rápida e fácil (RADC-TR-85-37).

Geralmente é vantajoso manter toda uma documentação em meio automatirado, pois isto permite que (BEAUFOND, 1987):

- a documentação esteja sempre disponível;
- a documentação possa ser facilmente reproduzida e armazenada após seu uso;
- os usuários possam ter rapidamente uma cópia sempre que desejarem;
- a consulta a qualquer um dos documentas armazenados seja fácil e ágil.

No caso do código fonte de um programa, ele geralmente está armazenado no próprio hardware que o executa, dentro de alguma biblioteca ou diretório apropriado.

A avaliação de um programa segundo o critério de

qualidade **Acessibilidade** deve ser feita pela verificacão dos locais de armanenamento do mesmo.

- Atualização

O critério **Atualização** avalia se o programa contém as informações mais recentes, ou seja, está na sua versão mais' atual e corrente.

É importante e vital para o sucesso de todo o software ter sempre toda sua documentação atualizada, pois ela serve de base tanta para fins de manutenção durante sua vida Útil come para reutilização/geração de outros produtos.

Assim, nos vários lugares em que o programa pode estar armazenado, sejam automatizados ou não, como as bibliotecas de fonte ou pastas de listagens, ele deve estar sempre em sua versão mais atual.

A utilização de ferramentas automatiradas, tanto na atualização da documentacão interna do programa como na controle do ambiente operacional a que o mesmo estiver ligado através do uso de sistemas de Gereneiamento de Configuração, auxilia muito na efetivação e divulgação de uma nova versão do programa.

A avaliação de um programa segundo o critério **Atualização** deve ser feita pela verificação da sua versão nos lugares designados para a mesma.

2.5.8- Sub-fator Rastreabilidade

O sub-fator **Rastreabilidade** refere-se ao fato de haver possibilidade de localização e acompanhamento, ou melhor, de percurso, dos componentes relacionados e referenciados

ao longo do programa. Estes componentes podem ser nomes de variáveis, "arrays", constantes, módulos, arquivos, relatórios ou telas. Com isso, é possível caminhar através da sequência de detalhes a um determinado aspecto ligado as programa, desde a sua visão mais geral até a mais detalhada, ou vice-versa (ROCHA, 1987).

Assim, um programa pode ser considerado rastreável se estiver clara e explícita a origem ou relação entre seus componentes.

Com isso, a presença da rastreabilidade num programa, seja automatizada ou não, facilita e agiliza o trabalho dos seus usuários leitores, principalmente para os responsáveis por sua manutenção, pois ajudando-os a localizar e percorrer os componentes que desejarem, esta, cansequentemente, facilitando o entendimento do programa.

Para avaliar a presença do sub-fator Rastreabilidade em um programa deve-se considerar os seguintes critérios de qualidade:

- Localizabilidade Interna

O critério **Localizabilidade Interna** avalia a possibilidade de se localizar os diversos componentes referenciados pelo programa dentro dele mesmo. Estes componentes pedem ser nomes de variáveis, "arrays", constantes, módulos, arquives, relatórios ou telas.

Para tal facilidade de rastreamento dos componentes referenciados dentro do próprio programa, existem mecanismos auxiliadores, que são basicamente os analisadores automáticos de código, que por sua vez fornecem as várias formas de Referência Cruzada dos seus

componentes. A Referência Cruzada é a referência entre as entidades por meios lógicos, que geralmente pode ser fornecida durante a compilação/depuração do programa, ou então por algum Utilitário/Programa específico para tal. Os relatórios de Referência Cruzada podem indicar (BROWN, 1987):

- relacão dos componentes do programa versus sua localização dentro do mesmo (linha do código fonte em que aparecem ou módulo em que são referenciados);
- relacão de hierarquia dos módulos existentes no programa;
- relacão de macros, que são rotinas em linguagem ASSEMBLER, versus os módulos em que são referenciadas;
- relacão de módulos versus correspondentes módulos que são chamados pelos mesmos, ou módulos versus correspondentes módulos que chamam os mesmos.

A avaliação de um programa segundo o critério **bocalizabilidade Interna** deve ser feita pela verificação da existência de algum tipo de Referência Cruzada sobre o mesmo.

- Localizabilidade Externa

O critério **Localizabilidade Externa** avalia se existe a posçibilidade de se localizar os componentes que estejam interrelacionadoç ao programa na documentação externa existente. Pode-se, assim, percorrer os componentes relacionados ao programa a partir da indicacão do mesmo.

Estes componentes podem ser nomes de módulos, arquivos, relatórios ou telas.

Existem mecanismos auxiliadores de percurso que facilitam a localização dos relacionamentos entre os componentes, o mais importante deles é o Dicionário de Dadas.

O Dicionário de Dados pode possibilitar associar, de forma top-down ou bottom-up, os componentes subordinados a outros componentes, facilitando a acesso aos componentes referenciados e o conhecimento de seus relacionamentos. Geralmente estas relações encontram-se na descrição do Dicionário de Bados de cada componente. Por exemplo, no Dicionario de dados de um arquivo podem estar relacionados tanto os registros e campos subordinados ao mesmo como os programas que o referenciam (ARTHUR, 1985).

A avaliação de um programa segundo o critério Localizabilidade **Externa** deve ser feita pela verificação da existência de Dicionário de Dados com interrelacionamentos entre os componentes do mesmo.

CAPÍTULO ZII

APFOR - ANALISADOR ESTATICQ PARA APOZQ À AVALIAÇÃO DA QUALIDADE DE PROGRAMAS FORTRAN

3.1 - Introdução

Qualidade é essencial para qualquer software, sendo por isso muito importante a execução de um processo de Controle da Qualidade ao longo do ciclo de vida do software, seja na sua especificação, projeta, implementação, operação ou manutenção.

Assim, diante da situação levantada nos capítulos anteriores sobre a necessidade de evolução da área de Qualidade de Software e de todo a estudo realizado sobre caracteristicas de qualidade de programa, abordados detalhadamente no capitulo II, é que nesta parte da tese é proposta e descrita a ferramenta **APFOR - ANALISADOR ESTÁTICO PARA APOIO À AVALIAÇÃO DA QUALIDADE DE PROGRAMAS FORTRAN**.

Esta ferramenta tem como objetivo principal realizar a tarefa de análise estática para determinadas características de qualidade, sobre um programa escrito na linguagem de programação FORTRAN 77 ANSI, já compilado especificamente no ambiente Microsoft FORTRAN versão 4.0 . Este programa é referenciado ao longo desta tese, então, como um programa MS FORTRAN *77* ANSI.

A motivação, proposta, definicão e implementação do APFOR relativos às informações sobre as características de qualidade dentro do processo de avaliação da qualidade de um programa, visa atingir os seguintes objetivos:

- servir como instrumento e com informações de apoio para execução parcial e consequente continuação de tal processo, influenciando, assim, nas tomadas de decisão;
- estar baseado no trabalho e nas informações desta tese abordados no capítulo II anterior e no anexo I que contém Manual para Avaliação da Confiabilidade da Representação de programas FORTRAN;
- tornar válida e viável a automatização, mesmo que parcial, deste processo;
- tornar economicamente viável a execução deste processo, reduzindo o processo manual de coleta e análise de dados;
- tornar mais confiavel os resultados intermediários e finais deste processa, reduzindo erros e inconsistências, próprios de uma tarefa manual;
- auxiliar na divulgação da interpretação dos resultados obtidos e nas sugestões de revisão sobre o programa;
- ser uçads nas atividades tanto de implementação de um programa, ao final de sua construcão, como na de operação ou manutenção do software;
- ser simples e prática, para ser prontamente utilizada;
- ser uma ferramenta interativa, com uma boa interface homem-máquina, de forma a possibilitar aos usuários executarem-na de maneira fácil e

amigável.

Diante de todas estas propostas, o APFOR torna-se um analisador estático especial, pois além da verificação de características sobre o código fonte do programa, que, geralmente, é o que apenas fazem outros analisadores estáticos, ele fornece informações sobre o atendimento, ou não, das caracteristicas de qualidade consideradas e sugestões para procedimentos de revisão. Isto tudo, com a finalidade de fazer parte do processo de avaliação da qualidade de programa, e consequentemente, melhorar seu nível de qualidade.

A figura 9 mostra um Diagrama de Fluxo de Dados, no seu nível mais baixe, tom a indicação das principais funções exercidas pelo APFOR. De acorde com este Diagrama, a idéia é que o usuário primeiramente determine as caracteristicas (fatores e respectivos sub-fatores) de qualidade relativas à descrição, organização e representação da programa que devem ser consideradas sobre um programa MS FORTRAN 77 ANSI especifico, já compilado. Isto é uma responsabilidade da usuário, devendo ser feita de acordo com seus interesses e necessidades. A partir dai, começa a análise estática automática propriamente dita diretamente sobre o código fonte do programa em questão. Após este procedimento, podem ser feitas perguntas para o usuário sobre o que depender tanto de sua opinião referente ao código Conte do programa em questão como de informações sobre o ambiente em que este se encontrar. Com base em todas essas informações detectadas e coletadas, o APFOR interpreta e fornece para o usuário os resultados finais sobre as caracteristicas de

qualidade e sugestões de revisão sobre o programa. Dessa forma, a análise estática é feita através da interação tanto com o código fonte do programa em questão como com o próprio usuário.

E importante ressaltar que o **APFQR** não é um compilador, ele faz apenas verificações no código fonte de um programa MS FORTRAN 77 ANSI, não fazendo críticas sobre sua sintaxe. Para seu devido funcionamento, é que o programa precisa estar correto, sem erros de compilação.

A partir desta explicação geral sobre o APFOR, a próxima seção 3.2 contem as especificações, como a estrutura geral, capacidades, usuários, requisitos, escopo, funções exercidas e relatórios fornecidos pelo **APFQR** proposto, e a 3.4 contém os principais aspectos de sua implementação.

3.2 - Espeficicação

A especificação do **APFOR** envolve a abordagem de certos aspectos envolvidos com o mesmo, como sua estrutura geral, suas capacidades, seus usuários, seus requisitos de qualidade, seu escapo, suas funções e seus relatórios que são apresentados nas seções a seguir.

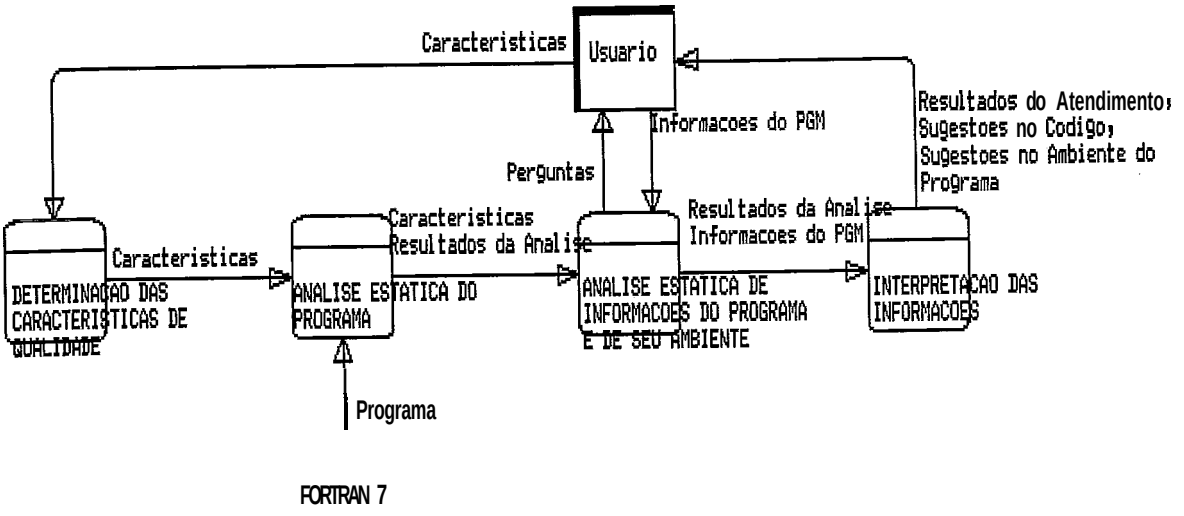


Fig. 9 - Diagrama de Fluxo de Dados do APFOR

3.2.1 - Estrutura geral

Dentro de sua estrutura geral, o APFOR é constituído de três componentes básicos e interrelacionados: o Módulo de Interface (MDI), o Analisador Estático de Código Fonte (AECF) e o Gerador de Informações de Apoio (GIA). Eles são mostrados na figura 10 e definidos a seguir.

- MÓDULO DE INTERFACE (MDI):

É a parte responsável pela comunicação que ocorre na APFOR, onde promove e controla a interface tanto com o usuário, como com o Analisador Estático de Código Fonte (AECF).

A interface com o usuário acontece para que o APFOR envie perguntas sobre o programa em questão, e, por sua, vez, receba as características de qualidade a serem consideradas e as reçpostas às perguntas sobre o programa feitas ao usuário. Este diálogo deve ser amigável e acessível, com uma boa capacidade de explicação; natural, com o mínimo de formalização; e flexível, para poder ser de fácil utilização por possíveis usuários com diferentes graus de experiência e conhecimentos.

Já a interface com o AECF acontece para sue o MDI envie solicitações para a execução da análise estática automática de determinadas medidas, relativas aos çub-fatores de qualidade (Clareza, Concisão, Estilo de Programação e Modularidade), diretamente no código fonte do programa, e, par sua ver, o MDI receba os resultados encontrados.

- ANALISADOR ESTÁTICO DE CODIGO FONTE (AECF):

É a paute que examina todo o código fonte do programa em questão, com o intuito de fazer certas verificações de medidas especificas, subordinadas aos critérios e çub-fatsres de qualidade, que forem solicitadas pelo usuário.

O AECF atua a partir da análise de código fonte do programa, que já **deve** ter sido compilado, estar livre de erros de compilação, livre de caracteres de controle (como os de tabulação) e com seu Último comando seguido de ⟨ENTER⟩ (consequentemente, com pelo menos uma linha em branco no seu final), caso contrário, os resultados da avaliasão não serão confiáveis. Ele analisa o código fonte

do programa a nivel de cada unidade de programa.

= GERADOR DE INFORMAÇÕES DE APOIO (GIA):

É a parte que organiza e dirige os passos tomados para fornecer ao usuário os resultados finais que servem de apoio ao processo de avaliação da qualidade do programa em questão. Logo, através do MDI, recebe as características de qualidade consideradas e as informações sobre o programa fornecidas tanto pelo usuário como pelo AECF.

Estes resultados referem-se à indicação da classificação das características coma ATENDIDA, NÃO ATENDIDA ou NÃO APLICAVEL, e às de sugestões para revisão que podem ser realizadas para melhorar o nível de qualidade do programa.

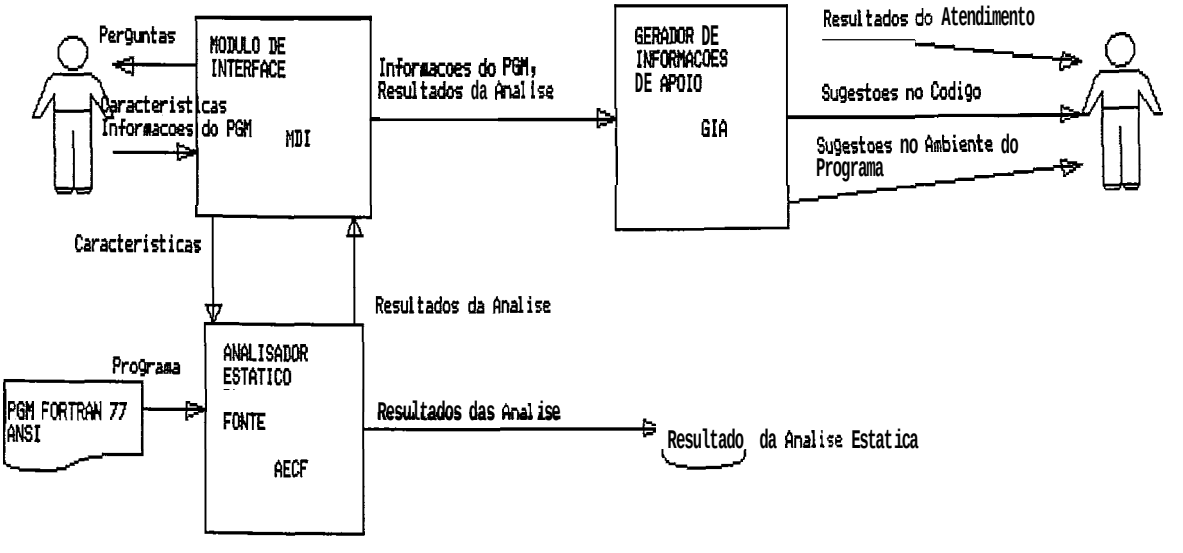


Fig. 10 - Estrutura geral do APFOR

3.2.2 - Capacidades

Considerando a estrutura geral proposta para o APFOR, ele deve possuir as seguintes capacidades:

- coleta automática de determinadas medidas de qualidade diretamente no código fonte do programa em questão;
- utilização de conhecimentos e informações sobre avaliação da qualidade de programas;
- sugestões para revisão do programa sobre as características de qualidade previamente selecionadas e consideradas, indicando também a classificação para as meçmas como ATENDIDA, NRO ATENBIDA e NAO APLICAVEL.

3.2.3 - Usuários

O APFOR pode ser usado, principalmente, pelos seguintes tipos de usuário envolvidos com o programa em questão:

- **Analista de sistemas ou Programador**: através da execução e dos relatórios finais do APFOR, pode tomar consciência e detectar especificamente as medidas que ***são*** ou não satisfeitas pelo programa em questão. Isto tudo de maneira rápida e simples, onde leva-se em consideração a dinamicidade e realidade dos ambientes de Processamento de Dados, mesmo os da área científica, onde o tempo é curtoe a cultura e experiência em avaliação da qualidade é pequena;
- **Engenheiro de Software, Analista ou Pesquisador da Controle da Qualidade ou de Garantia de Qualidade** de

Software: através da execução do APFOR, inicia o processo de avaliação da qualidade de um programa, e com as informações fornecidas nos seus relatórios finaiç, pode melhor indicar a nível de qualidade **que** o programa se encontra, e, também, dar sugestões abrangentes para elevar tal nível;

- **Gerente eu Coordenador de projetos**: através das relatórios finais do APFOU, pode verificar a quantidade de caracteristicas de qualidade que foram ou não atendidas, e a quantidade de medidas que foram ou não satisfeitas.

3.2.4 - Requisitos de qualidade

Devem ser enfatinadas, principalmente, as seguintes três características de qualidade durante o desenvolvimento do APFOR, em ordem decrescente de importância:

- Flexibilidade: para poder ser expandido através da inclusão da avaliação de outras características de qualidade de programas e, também, para poder permitir sua adaptação ao ambiente de programação da instituição em que for utilirado operacionalmente;
- **Interoperabilidade**: para poder ter interfaces com outras ferramentas, tanto de coleta de dados como de avaliação da qualidade;
- Portabilidade: para poder ser transportado para outros ambientes de software/hardware.

3.2.5 - Escopo

Os programas submetidos ao APFOR restringem-se apenas aos escritos em FORTRAN 77 ANSI. Sua escolha foi por ser esta uma linguagem bastante utilizada, principalmente para software científico.

As características de qualidade consideradas pelo APFOR baseiam-se no objetivo Confiabilidade da Representação e seus respectivos fatores, sub-fatores e critérios de qualidade, segundo o método de avaliação da qualidade de Rocha adotado nesta tese e a classificação para tais caracteristicas, abordados no capitulo II anterior. A escolha destas características foi no sentido do que seria mais simples e fácil para o usuário, e também o que seria possível e viável para ser automatizado. Logo, o escopo do APFOR para análise de caracteristicas é restrito, não abrangendo todas as possiveis caracteristicas determinantes da qualidade de um programa, formando uma classificação especifica para o APFOR, que estão relacionadas em ordem hierárquica por coluna na figura 11.

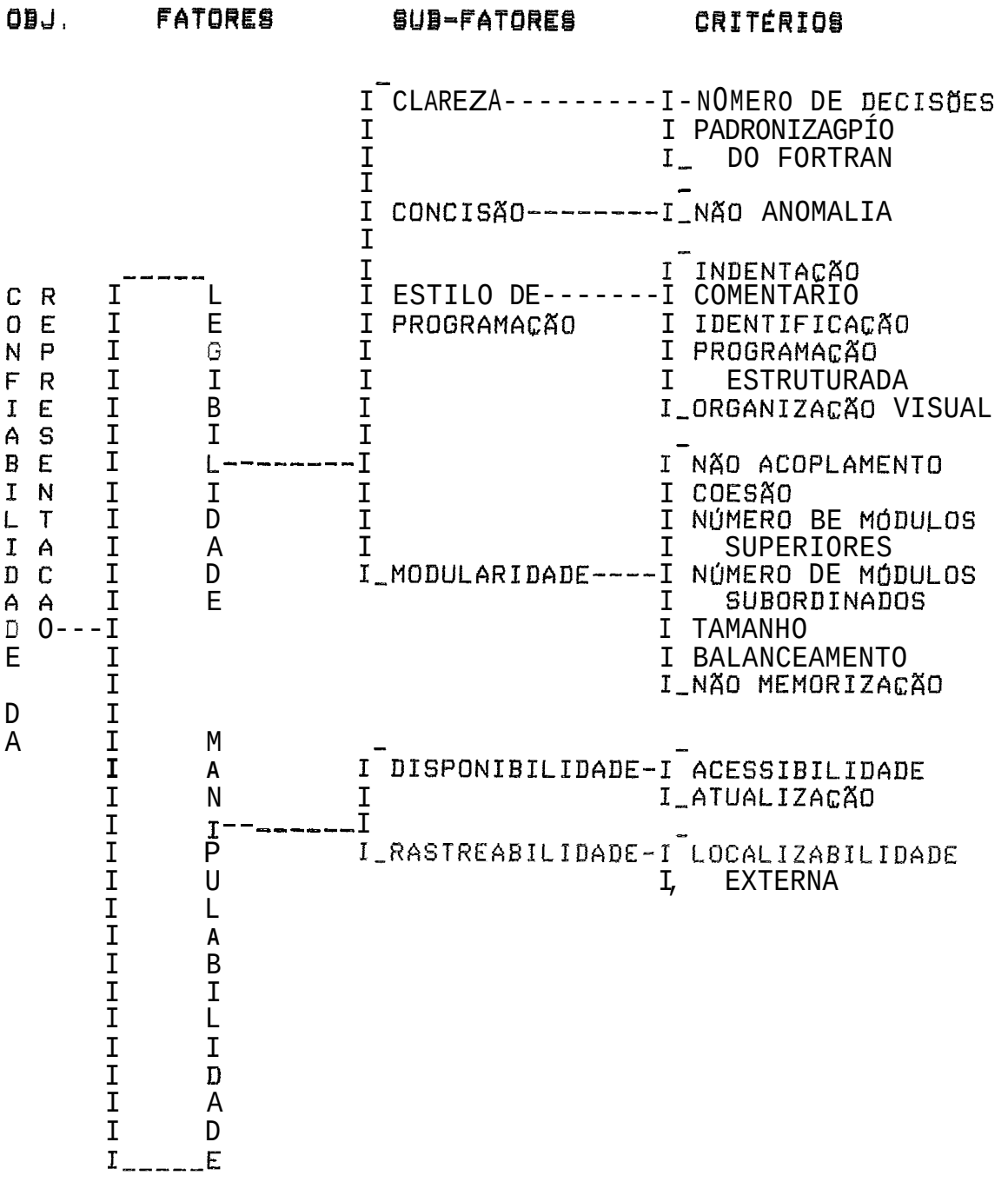


Fig. 11 - Classificação das características de qualidade consideradas no APFOR

3.2.6 = Funções

O APFOR executa as seguintes funções, relacionadas abaixo e numeradas hierarquicamente de acordo com a classificação adotada, para todas as possíveis medidas envolvidas com seus respectivos critérios de qualidade

Tudo isto baseando-se no trabalho descrito nos capítulos anteriores e principalmente no Manual para Avaliação da Confiabilidade da Representação de programas FORTRAN contidas no anexo I desta tese. Está descrito também a(s) respectiva(s) mensagem(ns) de sugestão que pode(m) ser gerada(s) quando seu correspondente critério de qualidade for classificado como característica NÃO ATENDIDA. Assim, estão apresentadas todas as funções envalvidas com as características de qualidade que podem ser consideradas, com as informações tanto de entrada (objetivo, fatores, sub-fatores, critérios e medidas de qualidade) como de saída (mensagens de sugestão de revisão).

Estas funções abaixo envolvem informações obtidas tanto do código fonte do programa como do usuário. Neste Último caso, ela será claramente indicada. Caso contrário, considera-se que tudo é feito direta e automaticamente pelo APFOR.

1- Analisar o objetivo do qualidade Confiabilidade da Representação:

Resultado dos fatores Legibilidade e Manipulabilidade.

1.1- Analisar o fator Legibilidade:

Resultado dos sub-fatores Clareza, Concisãu, Estilo de Programação e Modularidade.

1.1.1- Analisar o sub-fator Clareza:

Resultado dos critérios Número de Decisões e Padronização do FOWTRAN.

1.1.1.1- Analisar o critério Número de Decisões através do resultado da seguinte medida:

Medida: Complexidade ciclomática

Calcular o número de decisões para cada unidade de programa, baseando-se na formula da Complexidade ciclomática simplificada de McCabe e comparando cada resultado com o limite máximo permitido (igual a 10) para verificar se não o ultrapassa.

Mensagem de sugestão quando NÃO ATENDIDA:
>---> Devem ser simplificadas as condicoes

1.1.1.2- Analisar o critério Padronização de FORTRAN através dos resultados das seguintes medidas:

Medida i: Inexistência de erros da compilação no programa

Verificar se o programa está sem erros de compilação através de pergunta feita ao usuário. Este é um pré-requisita do APFOR, logo, se incialmente não for atendida, a execução e cancelada.

Medida E?: Inexistência de extensões próprias do Microsoft FORTRAN versão 4.0

Verificar se não existem no programa metacomandos (comandos que começam com $ na coluna 1) e nem declarações de tipos de dados como INTEGER*1, INTEGER*2, INTEGER*4, REAL*4, REAL*8, COMPLEX*8, COMPLEX*16, LOGICAL*1, LOGICAL*2 e LOGICAL*4), que não fazem parte do FORTRAN 77 ANSI. Se isto não ocorrer, a execução é cancelada.

1.1.2- Analisar O sub-fator Concisão:

Resultado do critério Não Anomalia.

Este sub-fator Concisão também é influenciado pelo resultado da critério Não Repetição, entretanto, este não é

considerado no escopo do APFOR, devido à dificuldade de sua automatização.

1.1.2.1- Analisar o critério Não Anomalia através dos resultados das seguintes medidas:

Medida 1: Inexistência de subprograma não referenciado

Verificar se todos os subprogramas declarados no programa são referenciados (chamados).

Mensagem de sugestão quando NÃO ATENDIDA::
)---) Deve ser retirado o subprograma nao referenciado

Medida 2: Inexistência de subprograma morto (sem comando executavel)

Verificar se não existem subprogramas mortos declarados no programa, ou seja, que possuam somente o comando END ou somente os comandos RETURN e END.

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve ser retirado o subprograma morto (sem comandos executaveis).

Medida 3: Inexistência de formato desnecessário

Verificar se todos os comandos FORMAT declaradaos no programa são necessários, ou seja, que não possuam label ou possuam label não referenciado pelos comandos de E/S formatada.

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve ser retirado o comando FORMAT nao referenciado.

1.1.3- Analisar o sub-fator Estilo de Programação:

Resultado dos critérios Indentacão, Comentário, Identificação, Programação Estruturada e Organização Visual.

1.1.3.1- Analisar o critério Indentação através dos resultados das seguintes medidas:

Medida 1: Existência de comando DO com bloco indentado

Verificar se todos os comandos DO do programa possuem os comandos do seu bloco subordinado indentados, em relação ao mesmo, com o espaçamento fornecido (igual a 3).

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve ser indentado o comando em relacao ao seu comando DO.

Medida 2: Existência ds comando IF com bloco da clausula THEN eu ELSE indentado

Verificar se todos os comandos IF do programa possuem os comandos do seu bloco subordinado, tanto da claúsula THEN como da ELSE, indentados, em relação ao mesmo, com o espaçamento fornecido (igual a 3).

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve ser indentado o comando em relacao ao seu comande IF.

1.1.3.2- Analisar o critério Comentário através dos resultados das seguintes medidas:

Medida 1: Existência de comentário de cabeçalho para explicar a função do programa antes ou depois do comando PROGRAM

Verificar se existe comentário antes ou depois da declaração do comando PROGRAM, que é o primeiro comando de um programa

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve ser incluido um comentario tipo cabecalho explicando a funcao do programa, no seu inicio, antes ou depois do comando PROGRAM.

Medida 2: Existência de comentário da subprograma para explicar a função de subprograma antes ou depois de sua declaração

Verificar se existe comentário antes ou depois da declaração de cada áubprograma

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve haver um comentario tipo cabecalho explicando a funsaa do subprograma, no seu inicio, anteç ou depois

de sua declaracao.

Medida 3: Existência de cabeçalho de manutencao padrão na inicio do programa

Verificar se existe cabecalho de manutenção padrão, que são comentários localizados antes ou depois do comando PROGRAM, no inicio do programa, com a indicação de MANUTENCAO, RESPONSAVEL, com sua respectiva string a seguir, DATA, com sua repectiva string a seguir e MOTIVO, com sua respectiva string a seguir

Mensagem de sugestão 1 quando NÃO ATENDIDA:
)---) Tambem não existe cabecalho de manuteneao padrao (com indicacao sobre MANUTENCAO, RESPONÇAVEL, DATA e MOTIVO) no seu inicia.
Caso ja' tenha ocorrido alguma manutencao no mesmo, inclua tambem este tipo de cabecalhe com as infsrmacoes mais recentes.

Mensagem de sugestão 2 quando NÃO ATENDIDA:
)---) Nao existe cabecalho de manutencao padrao (com indicacao sobre MANUTENCAO, RESPONÇAVEL, DATA e MOTIVO) no inicio do programa, antes ou depois do comando PROGRAM.
Caso ja' tenha ocorrido alguma manutencao no mesmo, inclua este tipo de cabecalho com as informacoes mais recentes.

Mensagem de sugestão 3 quando NÃO ATENDIDA:
)---) Existe um cabecalho de manutencao padrao (com indicacao sobre MANUTENCAO, RESPONSAVEL, DATA e MOTIVO) no inicis do programa, que esta' totalmente preenchido.
Deve ser verificado se a mesmo cantem realmente as informacoes mais recentes, pois seu conteudo nao e' analisado.

Mensagem de sugestão 4 quando NÃO ATENDIDA:
)---) Existe um cabecalho de manutencao padrao (com indicacao sobre MANUTENCAO, RESPONSAVEL, DATA e MOTIVO) no inicio do programa, que esta' parcialmente preenchido.
Devem ser devidamente incluidas no mesmo as informacoes mais recentes.

Medida 4: Inexistência de comentário, em linha de comando, depois da coluna 73

Verificar se não existe string depois da coluna 73 em cada linha do programa, exceto para a de comentário.

Mensagem de sugestão quando NRO ATENBIDA:
›---› Deve ser retirado o eomentario colocado depois da coluna 73.

Medida 5: Existência de comentários

Verificar se existe algum comentário no programa.

Mensagem de sugestão quando NÃO ATENDIDA:
›---› Devem ser incluidos, ao longo do programa, comentarios explicando seus principais procedimentos.

Medida 6: Existência de comentários claros e entendíveis

Verificar se os comentários no programa estão escritos de modo claro e entendível, através de pergunta ao usuário. Se não o estiverem, o próprio usuário indica as linhas em que acha que isto ocorre

Mensagem de sugestão quando NÃO ATENDIDA:
›---› Deve ser alterado o conteudo do bloco de comentarios para que a informacao seja clara, simples e sem abreviacae nao significativa.

1.1.3.3- **Analisar o critério Identificação** através do resultado da seguinte medida:

Medida: Existência de nomes de identificadores r variáveis significativos

Verificar se os nomes de identificadares e variáveis do programa são significativos, através de pergunta ao usuário. Se não o estiverem, o próprio usuário indica os nomes em que acha que isto ocorre

Mensagem de sugestão quando NÃO ATENDIDA:
›---› Devem ser substituidos os seguintes nomes referenciados ao longo do programa Por nomes considerados significativos:
XXXXXX, xxxxx, XXXXXX, xxxxx, xxxxxx

1,1.3.4- **Analisar e critério Programação Estruturada** através dos resultados das seguintes medidas:

Medida 1: Existência de subprograma cem um único pente de entrada (sem comando ENTRY)

Verificar se existe um Único ponto de entrada em cada

subprograma, que ocorre quando o mesmo não possui comando ENTRY.

Mensagem de sugestão quando NÃO ATENDIDA:
›---› Deve ser retirado o comando ENTRY, para se ter um unico ponto de entrada no subprograma.

Medida 2: Existência de subprograma com um único ponto de saída (sem mais de um comando RETURN)

Verificar se existe um único ponto de saída em cada subprograma, que ocorre quando o mesmo não possui mais de um comando RETURN.

Mensagem de sugestão quando NÃO ATENDIDA:
›---› Deve haver apenas um comando WETURN no subprograma, para se ter um unico ponto de saida.

Medida 3: Inexistência de desvie para fora do comando DO, através de comando O6 TO

Verificar se não existe comando GO TO desviando para um label de fora do bloco dos csmandos DO do programa

Mensagem de sugestão quando NÃO ATENDIDA:
›---› Deve ser retirado o comando GO TO.

Medida 4: Inexistência de desvio para fora do comando IF, através de comando GO TO

Verificar se não existe comando GO TO desviando para um label de fora do bloco das claúsulas THEN ou ELSE dos csmandos IF do programa.

Mensagem de sugestão quando NÃO ATENBIDA:
›---› Deve ser retirado o comando GO TO.

1.1.3.5- Analisar e critério Organização Visual através dos resultados das seguintes medidas:

Medida 1: Existência de label ordenado em relação à ordem crescente do label anterior

Verificar se todos os labels do programa estão ordenados em ordem numérica crescente em relação ao label anterior declarado.

Mensagem de sugestão quando NAO ATENDIDA:
)---) Deve ser ordenado o label em ordem crescente em relacao ao label anterior.

Medida 2: Existência de comando alinhado em relação ao csmando PROGRAM

Verificar se no programa a coluna inicial de declaração dos comandos é igual à coluna inicial de declaração do csmando PROGRAM, excluindo os blocos subordinados dos comandos DO e IF.

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve ser alinhado o comando de acordo com a coluna do comando PROGRAM.

Medida 3: Existência de operador de expressão em comandos executáveis com espaço em branco antes e depois do mesmo

Verificar se no programa todoç os operadores de expressão em comandos executáveis possuem espaço em branco antes e depois do mesma

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve haver pelo menos um espaco em branco antes e depois do operador de expressao.

Medida 4: Existência da sinal ₅ de atribuição com espaço em branco antes e depois do mesmo

Verificar se no programa todos os sinais "=" de atribuição possuem espaço em branco antes e depois do mesmo.

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve haver pelo menos um espaco em branco antes e depois do sinal = de atribuicao.

Medida 5: Existência de comando de declaração com espaço em branco depois da vírgula separadora de variáveis

Verificar se no programa todos os comandos de declaração possuem espaço em branca depois da vírgula que separa suas variáveis.

Mensagem de sugestão quando NÃO ATENDIBA:
)---) Deve haver pelo menos um espaco em branco depois da virgula separadora de variaveis.

Medida 6: Existência de bloco de comentários com linha em branco antes e depois da mesmo

Verificar se existe linha em branco antes e depois dos blocos de comentário do programa.

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve haver pelo menos uma linha em branco antes e depois do bloco de comentarios.

Medida 7: Existência de subprograma com linha em branco antes de sua declaração

Verificar se existe linha em branco antes da declaração de cada subprograma.

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve haver pelo menos uma linha em branco antes de subprograma.

1.1.4- Analisar o sub-fator Modularidade:

Resultado dos critérios Não Acoplamento, Coesão, Número de Módulos Superiores, Número de Módulos Subordinados, Tamanho, Balanceamento e Não Memorização.

1.1.4.1- Analisar o critério Não Acoplamento através do resultado da seguinte medida:

Medida: Inexistência de subprograma com acoplamento por área comum (possui comando COMMON) ou com acoplamento por controle (possui argumento da entrada como flag de controle na condição de comando IF ou no incremento de comando DO)

Verificar se em cada subprograma não existe acoplamento por área comum, que ocorre quando não possui comando COMMON, ou não existe acoplamento por controle, que ocorre quando não possui argumento de entrada como flag de controle na condição de comando IF ou no incremento de comando DO.

Mensagem de sugestão 1 quando NÃO ATENDIDA:
)---) Deve ser retirado o comando COMMON do subprograma

Mensagem de sugestão *2* quando NÃO ATENDIDA:
›---› Deve ser mudado o tipo de argumento de entrada que e' um flag de controle.

1.1.4.2- Analisar e critério Coesão através do resultado da, seguinte medida:

Medida: Existência de subprograma com um único ponto de entrada (sem comando ENTRY) ou com um Único ponto de saída (sem mais de um comande RETURN)

Verificar se em cada subprograma existe um Único ponto de entrada, que ocorre quando não possui comando ENTRY, ou se existe um único ponto de saída, que ocorre quando não possui mais de um comando RETURN, não existindo coesão lógica e nem coesão coincidental.

Mensagem de sugestão 1 quando NÃO ATENBIDA:
›---› Deve ser retirado o comando ENTRY, para se ter um unico ponto de entrada no subprograma.

Mensagem de sugestão *2* quando NÃO ATENDIDA:
›---› Deve haver apenas um comando RETURN no subprograma, para se ter um unico ponto de saida.

1.1.4.3- Analisar e critério Número de Módulos Superiores através do resultado da seguinte medida:

Medida: Quantidade de unidades de programa que chamam subprograma

Calcular para cada subprograma a quantidade de unidades de programa que o chama, comparando cada reçultado com o limite máximo permitido (igual a 3) para verificar se não o ultrapassa.

Mensagem de sugestão suando NÃO ATENDIDA:
›---› Deve ser dividido o subprograma.

1.1.4.4- Analisar o critério Número de Módulos Subordinados através do resultado da seguinte medida:

Medida: Quantidade de subprogramas chamados por unidades de programa

Calcular para cada unidade de programa a quantidade de subprogramas chamados pela mesma, e comparando cada resultado com o limite máximo permitido (igual a 9) para verificar se não o ultrapassa

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Devem ser diminuidas as chamadas feitas pela unidade de programa.

1.1.4.5- Analisar o critério Tamanho através do resultado da seguinte medida:

Medida: Quantidade de linhas executáveis

Calcular para cada unidade de programa a quantidade de linhas executaveis, comparando cada resultado com o limite máximo permitido (igual a 100) para verificar se não o ultrapassa.

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve ser diminuida ou dividida a unidade de programa.

1.1.4.6- Analisar o critério Balanceamento através do resultado da seguinte medida:

Medida: Inexistência de unidades de programa não de baixo nível com características físicas (possui comando de E/S formatada)

Verificar se todos os comandos de E/S formatada do programa, que indicam características físicas, estão em unidades de programa de mais baixo nível (que são as que não chamam outro subprograma)

Mensagem de sugestão quando NÃO ATENDIDA:
)---) Deve ser transferido o comando de E/S formatado do subprograma.

1.1.4.7- Analisar o critério Não Memorização através do resultado da seguinte medida:

Medida: Inexistência de subprograma com memória temporária (possui comando COMMON ou EQUIVALENCE)

Verificar se em cada subprograma não existe o comando COMMON ou EQUIVALENCE, que são indicadores de memória temporária.

Mensagem de sugestão 1 quando NÃO ATENDIDA:
›---› Deve ser retirado o comando COMMON do subprograma

Mensagem de sugestão 2 quando NÃO RTENRIDA:
›---› Deve ser retirado o comando EQUIVALENCE do subprograma.

2.2- Analisar o Fator Manipulabilidade:

Resultado dos sub-fatores Disponibilidade e Rastreabilidade.

1.2.1- Analisar o sub-fator Disponibilidade:

Resultado dos critérios Acessibilidade e Atualização

1.2.1.1- **Analisar o critério Acessibilidade** através do resultado da seguinte medida:

Medida: Armazenamento de programa em lugares designados para tal

Verificar se existem lugares próprios designados para o armazenamento do programa (como bibliotecas, diretórios, diskettes, pastas ou armários) e se o programa em questão está realmente nesses lugares, através de pergunta ao usuário

Mensagem de sugestão 1 quando NÃO ATENDIRA:
›---› Devem ser estabelecidos lugares proprios para armazenar programas, como bibliotecas, diretorios, diskettes, pastas ou armarios especificos. E nesses, armazenar o programa.

Mensagem de sugestão 2 quando NWO ATENDIDA:
›---› Deve ser armazenado o programa nos lugares designados para tal.

1.2.1.2- **Analisar o critério Atualização** através do resultado da seguinte medida:

Medida: Atualização do programa nos lugares designados para seu armazenamento

Verificar se nos lugares designados para armazenamento de programa está a versão mais atual do programa que estiver em questão, através de pergunta ao usuário

Mensagem de sugestão quando NNO ATENDIDA:
›---› Deve ser armazenada a versao mais atual do programa nas seus devidos lugares.

1.2.2- Analisar o sub-fator Rastreabilidade:

Resultado do critério Localirabilidade Externa

Este sub-fator Aastreabilidade também é influenciado pelo resultado do critério Localinabilidade Interna, entretanto, este não é considerada no escopo do APFOR, devido à dificuldade de sua automatização.

1.2.2.1- Analisar o critério Localizabilidade Externa

através do resultado da seguinte medida:

Medida: Existência de Dicionário de dados eu outra documentação externa similar com os interrelacionamentoã do programa

Verificar se existe Bicionaria de dados ou outra documentação externa similar com informação atualirada sobre os interrelacionamentoç de programa, através de pergunta ao usuário.

Mensagem de sugestão 1 quando NÃO ATENDIDA:
›---› Deve ser criado um Dicionario de Dados contendo para cada componente (como unidade de programa, arquivo, registro, campo, relatorio e tela) do programa, a indicacao dos respectivos componentes interrelacionados.

Mensagem de sugestão 2 quando NÃO ATENDIDA:
›---› Deve ser atualizado o Bieionaria de Dados ou a documentacao externa que contem os interrelacionamentoã de componentes de programa.

3.2.7 = Relatórios

Existem quatro tipos de relatórios gerados pelo APFOR:

i- Resultados sobre as Medidas da Análise Estática no Programa;

2- Resultados sobre o Atendimento das Características de Qualidade;

3- Indicações sobre as Medidas com Sugestões para Revisão no Programa;

4- Sugestões para Revisão no Ambiente de Trabalho do Programa.

1= Resultados sobre no Medidas da Análise Estática no Programa

Relatório para indicar os resultados obtidos para cada respectiva medida subordinada ao critério e ao sub-fator de qualidade sendo considerado, que são as próprias características de qualidade. Só não é gerado se apenas o sub-fator Manipulabilidade for o único considerado.

Çeu "lay-out" geral, com todas os possiveis resultados que podem aparecer, em forma reduzida, já que seu tamanho normal é de 80 colunas, está na figura 12.

2= Resultados sobre o Atendimento das Características de Qualidade

Relatdrio para indicar a classificação em um dos três tipos de resultado (CARACTERISTICAS ATENDIDAS, CARACTERIÇTPCAS NÃO ATENDIDAS e CARACTERISTICAS NÃO APLICAVEIS) para cada característica de qualidade (objetivo, fator, çub-fator e critério de qualidade) sendo considerada.

Seu "lar-out" geral, com todos os possíveis resultados que podem aparecer, em forma reduzida, já que seu tamanha normal é de *77* colunas, está na figura 13.

3- Indicações sobre as Medidas com Sugestões para Revisão no Programa

Relatório para indicar sugestões sobre o código fonte do programa, a fim de atender às medidas das características de qualidade com classificação NÃO ATENDIDA no relatório anterior de Resultados sobre o Atendimento das Caracteristicas de Qualidade, exceto às referentes ao fator Manipulabilidade, que por sua vez estão relacionadas apenas ao ambiente de trabalho do programa.

Estas sugestões são feitas através de uma mensagem, numa linha imediatamente seguinte à linha em que foi detectado o não atendimento, que por sua vez origina-se das informações contidas no relatório "Resultadas sobre as Medidas para Análise Estática sobre o Programa" e de perguntas feitas ao usuário quando necessário.

Seu "lay-out" geral, com todos os possíveis resultados que podem aparecer, em forma reduzida, já que seu tamanho normal é de *77* colunas, está na figura 14.

4- Sugestões para Revisão no Ambiente de Trabalho do Programa

Relatório para indicar sugestões sobre o ambiente de trabalho do programa em questão, quando um dos critérios Acessibilidade ou Atualização do sub-fator Disponibilidade, ou então o critério Localizabilidade Externa do sub-fator Rastreabilidade (ambos subordinados ao fator

Manipulabilidade) for classificado como NÃO ATENDIDA no relatório anterior de Resultados sobre o Atendimento das Características de Qualidade. Fornece também um aviso de atenção quando o usuário não souber responder alguma pergunta.

Estas sugestões são feitas através de uma mensagem que origina-se das respostas obtidas de perguntas feitas ao usuário quando necessário.

Çeu "lay-out" geral, com todos os possíveis resultados sue podem aparecer, em forma reduzida, já que seu tamanho normal é de 77 colunas, está na figura 15.

```
-------------------------------------------------------------------------------
          RESULTADOS SOBRE AS MEDIDAS DA ANALISE ESTATICA NO PROGRAMA
                          XXXXXXXX.FOR EM  DD/MM/AA
                                       'AS HH:MM HS
-------------------------------------------------------------------------------

                        ************************************
                        * ANALISE PARA O SUB-FATOR CLAREZA *
                        ************************************

CRITERIO: NUMERO DE DECISOES
.........................

   MEDIDA: COMPLEXIDADE CICLOMATICA (limite maximo permitido para o numero de
   ------  decisoes = 10)

   -> TODAS AS UNIDADES DE PROGRAMA NAO ULTRAPASSAM ESTE LIMITE PERMITIDO

   -> NAO EXISTE COMANDO IF PARA SER VERIFICADO O NUMERO DE DECISOES

      UNIDADE DE PROGRAMA          LINHA      NUMERO DE DECISOES
      ULTRAPASSANDO O LIMITE

      XXXXXX                       X          XX


-------------------------------------------------------------------------------

                        ************************************
                        * ANALISE PARA O SUB-FATOR CONCISAO *
                        ************************************

CRITERIO: NAO ANOMALIA

   MEDIDA 1: INEXISTENCIA DE SUBPROGRAMA NAO REFERENCIADO

   -> TODOS OS SUBPROGRAMAS SAO REFERENCIADOS

   -> NAO EXISTE SUBPROGRAMA

      SUBPROGRAMA NAO REFERENCIADO                    LINHA

      XXXXXX                                          XX

   MEDIDA 2: INEXISTENCIA DE SUBPROGRAMA MORTO (sem comando executavel)

   -> TODOS OS SUBPROGRAMAS NAO SAO MORTOS

   -> NAO EXISTE SUBPROGRAMA
      SUBPROGRAMA MORTO                            LINHA

      XXXXXX                                       XX
```

Fig. 12 - **Relatório dos Resultados sobre as Medidas da Análise Estática** no **Programa**

```
   MEDIDA 3:  INEXISTENCIA DE FORMATO DESNECESSARIO
   --------

   -> TODOS OS COMANDOS FORMAT SAO NECESSARIOS

   -> NAO EXISTE COMANDO FORMAT

      LABEL DO COMANDO FORMAT DESNECESSARIO          LINHA

      XXXXX                                          XX

-------------------------------------------------------------------------------

                  ****************************************
                  * ANALISE PARA O SUB-FATOR DISPONIBILIDADE *
                  ****************************************

CRITERIO: ATUALIZACAO
.....................

   MEDIDA: EXISTENCIA DE CABECALHO DE MANUTENCAO PADRAO NO INICIO DO PROGRAMA
   ------

   -> O CABECALHO DE MANUTENCAO PADRAO ESTA' TOTALMENTE PREENCHIDO

   -> NAO EXISTE TAL CABECALHO DE MANUTENCAO PADRAO

   -> O CABECALHO DE MANUTENCAO PADRAO ESTA' PARCIALMENTE PREENCHIDO

-------------------------------------------------------------------------------

               ************************************************
               * ANALISE PARA O SUB-FATOR ESTILO DE PROGRAMACAO *
               ************************************************

CRITERIO 1: INDENTACAO
......................

   MEDIDA 1: EXISTENCIA DE COMANDO DO COM BLOCO INDENTADO
   --------

   -> TODOS OS COMANDOS DO POSSUEM SEU BLOCO INDENTADO

   -> NAO EXISTE COMANDO DO

      LINHA COM COMANDO NAO INDENTADO

      XX

   MEDIDA 2:  EXISTENCIA DE COMANDO IF COM BLOCO DA CLAUSULA THEN OU ELSE
   --------   INDENTADO

   -> TODOS OS COMANDOS IF POSSUEM SEU BLOCO INDENTADO

   -> NAO EXISTE COMANDO IF

      LINHA COM COMANDO NAO INDENTADO

      XX
```

Fig. 12 - Relatório dos Resultados sobre as Medidas da Análise Estática no Programa (continuação)

CRITERIO 2: COMENTARIO
.....

MEDIDA 1: EXISTENCIA DE COMENTARIO DE CABECALHO PARA EXPLICAR A FUNCAO DO PROGRAMA ANTES OU DEPOIS DO COMANDO PROGRAM

-> EXISTE COMENTARIO DE CABECALHO

-> NAO EXISTE COMENTARIO DE CABECALHO

MEDIDA 2: EXISTENCIA DE COMENTARIO DE SUBPROGRAMA PARA EXPLICAR SUA FUNCAO ANTES OU DEPOIS DE SUA DECLARACAO

-> TODOS OS SUBPROGRAMAS POSSUEM COMENTARIO DE SUBPROGRAMA

-> NAO EXISTE S U B P R O G W

S U B P R O G W SEM COMENTARIO	LINHA
XXXXXX	XX

MEDIDA 3: INEXISTENCIA DE COMENTARIO,EM LINHA DE COMANDO,DEPOIS DA COLUNA 73

-> NAO EXISTE STRING DEPOIS DA COLUNA 73

LINHA COM COMENTARIO DEPOIS DA COLUNA 73

XX

CRITERIO 3: PROGRAMACAO ESTRUTURADA

MEDIDA 1: EXISTENCIA DE SUBPROGRAMA COM UM UNICO PONTO DE ENTRADA (sem comando ENTRY)

-> TODOS OS SUBPROGRAMAS NAO POSSUEM COMANDO ENTRY
-> NAO EXISTE SUBPROGRAMA

SUBPROGRAMA	LINHA CQM COMANDO ENTRY
XXXXXX	XX

MEDIDA 2: EXISTENCIA DE SUBPROGRAMA COM UM UNICO PONTO DE SAIDA (sem mais de um comando RETURN)

-> TODOS OS SUBPROGRAMAS POSSUEM APENAS UM COMANDO RETURN

-> NAO EXISTE SUBPROGRAMA

SUBPROGRAMA COM MAIS DE UM COMANDO RETURN	LINHA
XXXXXX	XX

Fig. 12 - Relatório dos Resultados sobre as Medidas da Análise Estática no Programa (continuação)

```
   MEDIDA 3: INEXISTENCIA DE DESVIO PARA FORA DO COMANDO DO, ATRAVES DE COMANDO
   --------  GO TO

   -> TODOS OS COMANDOS DO NAO POSSUEM COMANDO GO TO DESVIANDO PARA FORA

   -> NAO EXISTE COMANDO DO .

      LINHA COM COMANDO GO TO

      XX


   MEDIDA 4: INEXISTENCIA DE DESVIO PARA FORA DO COMANDO IF, ATRAVES DE COMANDO
   --------  GO TO

   -> TODOS OS COMANDOS IF NAO POSSUEM COMANDO GO TO DESVIANDO PARA FORA

   -> NAO EXISTE COMANDO IF

      LINHA COM COMANDO GO TO

      XX


CRITERIO 4: ORGANIZACAO VISUAL

   MEDIDA 1: EXISTENCIA DE LABEL ORDENADO EM RELACAO `A ORDEM CRESCENTE DO
   --------  LABEL ANTERIOR

   -> TODOS OS LABELS ESTAO ORDENADOS EM ORDEM CRESCENTE

   -> NAO EXISTE LABEL

      LABEL DESORDENADO                          LINHA
      XXXXX                                      XX


   MEDIDA 2: EXISTENCIA DE COMANDO ALINHADO EM RELACAO AO COMANDO PROGRAM
   -------

   -> TODOS OS COMANDOS ESTAO ALINHADOS

      LINHA COM COMANDO DESALINHADO

      XX


   MEDIDA 3: EXISTENCIA DE OPERADOR DE EXPRESSAO EM COMANDOS EXECUTAVEIS COM
   --------  ESPACO EM BRANCO ANTES E DEPOIS DO MESMO

   -> TODOS OS OPERADORES DE EXPRESSAO POSSUEM ESPACO EM BRANCO ANTES E
      DEPOIS DO MESMO

   -> NAO EXISTE OPERADOR DE EXPRESSAO

      LINHA COM OPERADOR SEM ESPACO EM BRANCO ANTES OU DEPOIS

      XX
```

Fig. 12 – Relatório dos Resultados sobre as Medidas da Análise Estática no Programa (continuação)

MEDIDA 4: EXISTENCIA DE SINAL = DE ATRIBUICAO COM ESPACO EM BRANCO ANTES E DEPOIS DO MESMO

-> TODOS OS SINAIS = DE ATRIBUICAO POSSUEM ESPACO EM BRANCO ANTES E DEPOIS DO MESMO

-> NAO EXISTE SINAL = DE ATRIBUICAO

LINHA COM SINAL = DE ATRIBUICAO SEM ESPACO EM BRANCO ANTES OU DEPOIS

XX

MEDIDA 5: EXISTENCIA DE COMANDO DE DECLARACAO COM ESPACO EM BRANCO DEPOIS DA VIRGULA SEPARADORA DE VARIAVEIS

-> TODOS OS COMANDOS DE DECLARACAO POSSUEM ESPACO EM BRANCO DEPOIS DA VIRGULA SEPARADORA DE VARIAVEIS

-> NAO EXISTE COMANDO DE DECLARACAO

LINHA SEM ESPACO EM BRANCO DEPOIS DA VIRGULA SEPARADORA

XX

MEDIDA 6: EXISTENCIA DE BLOCO DE COMENTARIOS COM LINHA EM BRANCO ANTES E DEPOIS DO MESMO

-> TODOS OS BLOCOS DE COMENTARIO POSSUEM LINHA EM BRANCO ANTES E DEPOIS DO MESMO

-> TODOS OS BLOCOS DE COMENTARIO POSSUEM LINHA EM BRANCO ANTES E DEPOIS DO MESMO

LINHA FINAL DO COMENTARIO SEM LINHA EM BRANCO ANTES OU DEPOIS

XX

MEDIDA 7: EXISTENCIA DE SUBPROGRAMA COM LINHA EM BRANCO ANTES DE SUA DECLARACAO

-> TODOS OS SUBPROGRAMAS POSSUEM LINHA EM BRANCO ANTES DE SUA DECLARACAO

-> NAO EXISTE SUBPROGRAMA

SUBPROGRAMA SEM LINHA EM BRANCO ANTERIOR	LINHA
NCHAM	91

****
* ANALISE PARA O SUB-FATOR MODULA,RIDA.DE*
****

CRITERIO 1: NAO ACOPLAMENTO

MEDIDA: INEXISTENCIA DE SUBPROGRAMA COM ACOPLAMENTO COMUM (possui comando COMMON) OU COM ACOPLAMENTO POR CONTROLE (possui argumento de controle na condicao de comando IF ou no incremento de comando DO)

Fig. 12 - Relatório dos Resultados sobre as Medidas da Análise Estática no Programa (continuação)

-> TODOS OS SUBPROGRAMAS NAO POSSUEM ACOPLAMENTO COMUM E NEM ACOPLAMENTO POR CONTROLE

-> NAO EXISTE SUBPROGRAMA

SUBPROGRAMA COM COMMON	LINHA
XXXXXX	XX
SUBPROGRAMA COM ARGUMENTO DE CONTROLE	LINHA
XXXXXX	XX

CRITERIO 2: COESAO

MEDIDA: EXISTENCIA DE SUBPROGRAMA COM UM UNICO PONTO DE ENTRADA (sem comando ENTRY) OU COM UM UNICO PONTO DE SAIDA (sem mais de um comando RETURN)

-> TODOS OS SUBPROGRAMAS POSSUEM UM UNICO PONTO DE ENTRADA E UM UNICO PONTO DE SAIDA

-> NAO EXISTE SUBPROGRAMA

SUBPROGRAMA	LINHA DO COMANDO ENTRY
XXXXXX	XX
SUBPROGRAMA COM MAIS DE UM COMANDO RETURN	LINHA
XXXXXX	XX

CRITERIO 3: NUMERO DE MODULOS SUPERIORES

MEDIDA: QUANTIDADE DE UNIDADES DE PROGRAMA QUE CHAMAM SUBPROGRAMA (limite maximo de superiores = 3)

-> TODOS OS SUBPROGRAMAS NAO ULTRAPASSAM ESTE LIMITE PERMITIDO

-> NAO EXISTE SUBPROGRAMA

SUBPROGRAMA ULTRAPASSANDO O LIMITE	LINHA	UNIDADES DE PROGRAMA SUPERIORES
XXXXXX	XX	XX

CRITERIO 4: NUMERO DE MODULOS SUBORDINADOS

MEDIDA: QUANTIDADE DE SUBPROGRAMAS CHAMADOS POR UNIDADE DE PROGRAMA (limite maximo de superiores = 9)

-> TODOS AS UNIDADES DE PROGRAMA NAO ULTRAPASSAM ESTE LIMITE PERMITIDO

UNIDADE DE PROGRAMA ULTRAPASSANDQ O LIMITE	LINHA	SUBPROGRAMAS SUBORDINADOS
XXXXXX	XX	XX

Fig. 12 - Relatório dos Resultados sobre as Medidas da Análise Estática no **Programa** (continuação)

```
CRITERIO 5: TAMANHO
-----------------

   MEDIDA: QUANTIDADE DE LINHAS EXECUTAVEIS (limite maximo = 100)

   -> TODAS AS UNIDADES DE PROGRAMA NAO ULTRAPASSAM ESTE LIMITE PERMITIDO

      UNIDADE DE PROGRAMA          LINHA      LINHAS EXECUTAVEIS
      ULTRAPASSANDO O LIMITE
      XXXXXX                       XX         XX


CRITERIO 6: BALANCEAMENTO

   MEDIDA: INEXISTENCIA DE UNIDADES DE PROGRAMA NAO DE BAIXO NIVEL COM
   ------  CARACTERISTICAS FISICAS (possui comando de E/S FORMATADA)

   -> TODOS OS COMANDOS DE E/S FORMATADA ESTAO EM UNIDADES DE PROGRAMA DE
      MAIS BAIXO NIVEL

   -> NAO EXISTE COMANDO DE E/S FORMATADA

      UNIDADE DE PROGRAMA NAO DE BAIXO NIVEL        LINHA
      COM COMANDO DE E/S FORMATADA

      XXXXXX                                        XX


CRITERIO 7: NAO MEMORIZACAO

   MEDIDA: INEXISTENCIA DE SUBPROGRAMA COM MEMORIA TEMPORARIA
   ------  (possui comando COMMON ou EQUIVALENCE)

   -> TODOS OS SUBPROGRAMAS NAO POSSUEM MEMORIA TEMPORARIA

   -> NAO EXISTE SUBPROGRAMA

      SUBPROGRAMA COM COMANDO COMMON                LINHA

      XXXXXX                                        XX

      SUBPROGRAMA COM COMANDO EQUIVALENCE           LINHA

      XXXXXX                                        XX

-------------------------------------------------------------------------------


   ..........................................
   *                                        *
   *   ANALISE ESTATICA NAO REQUISITADA PARA O:  *
   *                                        *
   *      SUB-FATOR CLAREZA                 *
   *      SUB-FATOR CONCISA0                *
   *      SUB-FATOR DISPONIBILIDA.DE        *
   *      SUB-FATOR ESTILO DE PROGRAMACAO   *
   *      SUB-FATOR MODULARIDADE            *
   *                                        *
   ******************************************
```

Fig. 12 - Relatório dos **Resultados sobre as Medidas da Análise Eçtática no Programa** (continuação)

```
.....................................................................
      RESULTADOS SOBRE O ATENDIMENTO DAS CARACTERISTICAS DE QUALIDADE
                     XXXXXXXX.FOR EM   DD/MM/AA
                                  'AS  HH:MM HS
.....................................................................

CARACTERISTICAS ATENDIDAS:
.........................

OBJETIVO:          CONFIABILIDADE DA REPRESENTACAO

   FATOR:          LEGIBILIDADE

      SUB-FATOR:   CLAREZA
         CRITERIOS:

                   NUMERO DE DECISOES
                   PADRONIZACAO DO FORTRAN

      SUB-FATOR:   CONCISAO
         CRITERIO:

                   NAO ANOMALIA

      SUB-FATOR:   ESTILO DE PROGRAMACAO
         CRITERIOS:

                   INDENTACAO
                   COMENTARIO
                   IDENTIFICACAO
                   PROGRAMACAO ESTRUTURADA
                   ORGANIZACAO VISUAL

      SUB-FATOR:   MODULARIDADE
         CRITERIOS:

                   NAO ACOPLAMENTO
                   COESAO
                   NUMERO DE MODULOS SUPERIORES
                   NUMERO DE MODULOS SUBORDINADOS
                   TAMANHO
                   BALANCEAMENTO
                   NAO MEMORIZACAO

   FATOR:          MANIPULABILIDADE

      SUB-FATOR:   DISPONIBILIDADE
         CRITERIOS:

                   ACESSIBILIDADE
                   ATUALIZACAO

      SUB-FATOR:   RASTREABILIDADE
         CRITERIO:

                   LOCALIZABILIDADE EXTERNA
```

Fig. 13 - Relatório dos **Resultados sobre** s **Atendimento das** Características **de Qualidade** sobre **o programa**

```
-----------------------------
CARACTERISTICAS NAO ATENDIDAS:
-----------------------------

OBJETIVO:            CONFIABILIDADE DA REPRESENTACAO
   FATOR:            LEGIBILIDADE
      SUB-FATOR:     CLAREZA
         CRITERIOS:

                     NUMERO DE DECISOES
                     PADRONIZACAO DO FORTRAN
      SUB-FATOR:     CONCISAO
         CRITERIO:
                     NAO ANOMALIA
      SUB-FATOR:     ESTILO DE PROGRAMACAO
         CRITERIOS:
                     INDENTACAO
                     COMENTARIO
                     IDENTIFICACAO
                     PROGRAMACAO ESTRUTURADA
                     ORGANIZACAO VISUAL
      SUB-FATOR:     MODULARIDADE
         CRITERIOS:
                     NAO ACOPLAMENTO
                     COESAO
                     NUMERO DE MODULOS SUPERIORES
                     NUMERO DE MODULOS SUBORDINADOS
                     TAMANHO
                     BALANCEAMENTO
                     NAO MEMORIZACAO

   FATOR:            MANIPULABILIDADE
      SUB-FATOR:     DISPONIBILIDADE
         CRITERIOS:

                     ACESSIBILIDADE
                     ATUALIZACAO
      SUB-FATOR:     RASTREABILIDADE
         CRITERIO:

                     LOCALIZABILIDADE EXTERNA

-------------------------------
CARACTERISTICAS NAO APLICAVEIS:
-------------------------------
```

Fig. 13 - Relatório dos Resultados sobre o Atendimento das **Características de Qualidade sobre o programa** (continuação)

```
OBJETIVO:          CONFIABILIDADE DA REPRESENTACAO

   FATOR:          LEGIBILIDADE

      SUB-FATOR:   CLAREZA
         CRITERIOS:

                   NUMERO DE DECISOES

      SUB-FATOR:   CONCISAO
         CRITERIO:

                   NAO ANOMALIA

      SUB-FATOR:   ESTILO DE PROGRAMACAO
         CRITERIOS:

                   INDENTACAO
                   COMENTARIO
                   IDENTIFICACAO
                   PROGRAMACAO ESTRUTURADA
                   ORGANIZACAO VISUAL

      SUB-FATOR:   MODULARIDADE
         CRITERIOS:

                   NAO ACOPLAMENTO
                   COESAO
                   NUMERO DE MODULOS SUPERIORES
                   NUMERO DE MODULOS SUBORDINADOS
                   BALANCEAMENTO
                   NAO MEMORIZACAO


   FATOR:          MANIPULAJ3ILIDA.DE

      SUB-FATOR:   DISPONIBILIDADE
         CRITERIOS:

                   ACESSIBILIDADE
                   ATUALIZACAO

      SUB-FATOR:   RASTREABILIDADE
         CRITERIO:

                   LOCALIZABILIDADE EXTERNA
```

Fig. 13 - Relatório **dos** Resultados sobre **o Atendimento** das **Características de Qualidade sobre o programa** (continuação)

```
-------------------------------------------------------------------------
     INDICACOES SOBRE AS MEDIDAS COM SUGESTOES PARA REVISAO NO PROGRAMA
                        XXXXXXXX.FOR EM   DD/MM/AA
                                      'AS  HH:MM HS
-------------------------------------------------------------------------

)___) Mensagem de sugestão para IDENTIFICAÇÃO

)___) Mensagem de sugestão para COMENTARIO (MEDIDA 1)

)___) Mensagem de sugestão para COMENTARIO (MEDIDA 3)

)___) Mensagem de sugestão para COMENTARIO (MEDIDA 5)

------------) LISTAGEM DQ CODIGO FONTE DO
              C:\APFOR\XXXXXXXX.FOR

)---) Mensagem de sugestão para COMPLEXIDADE CICLOMATICA

)---) Mensagem de sugestão para NÃQ ANOMALIA (MEDIDA 1)

)---) Mensagem de sugestão para NÃO ANOMALIA (MEDIDA 2)

)---) Mensagem de sugestão para NÃO ANOMALIA (MEDIDA 3)

)---) Mensagem de sugestão para INDENTACBO (MEDIDA 1)

)---) Mensagem de sugestão para INDENTAÇAO (MEDIDA 2)

)---) Mensagem de sugestão para COMENTARIO (MEDIDA 2)

)---) Mensagem de sugestãa para COMENTARIO (MEDIDA 4)

)---) Mensagem de sugestão para COMENTARJQ (MEDIDA 6)

)---) Mensagem de sugestão para PRQGRAMACBO ESTRUTURADA
      (MEDIDA 1)

)---) Mensagem de sugestão para PROGRAMACWO ESTRUTURADA
      (MEDIDA 2)

)---) Mensagem de sugestão para PROGRAMACAO ESTRUTURADA
      (MEDIDA 3)

)---) Mensagem de sugestão para PROGRAMACAO ESTRUTURADA
      (MEDIDA 4)

)---) Mensagem de sugestão para ORGANIZAÇÃO VISUAL
      (MEDIDA 1)

)---) Mensagem de sugestão para ORGANIZAÇÃO VISUAL
      (MEDIDA 2)
```

Fig. 14 - Relatório das Indicações sobre as Medidas com Sugestões para Revisão no Programa

```
-------------------------------------------------------------------------
     INDICACOES SOBRE AS MEDIDAS COM SUGESTOES PARA REVISA0 NO PROGRAMA
                        XXXXXXXX.FOR EM   DD/MM/AA
                                     'AS  HH:MM HS
-------------------------------------------------------------------------

)___> Mensagem de sugestão para ORGANIZAÇÃO VISUAL
      (MEDIDA 3)

)___> Mensagem de sugestão para ORGANIZAÇÃO VISUAL
      (MEDIDA 4)

)___> Mensagem de sugestão para ORGANIZAÇÃO VISUAL
      (MEDIDA 5)

)___> Mensagem de sugestão para ORGANIZACAO VISUAL
      (MEDIDA 6)

)___> Mensagem de sugestão para ORGANIZAÇÃO VISUAL
      (MEDIDA 7)

)___> Mensagens de sugestão para NÃO ACOPLAMENTQ

)___> Mensagens de sugestão para COESÃO

)---> Mensagem de sugestão para NUMERO DE MODULOS
      SUPERIORES

)___> Mensagem de sugestão para NUMERO DE MODULOS
      SUBORDINADOS

)___> Mensagem de sugestão para TAMANHO

)___> Mensagem de sugestão para BALANCEAMENTO

)---> Mensagens de sugestão para NÃO MEMORIZAÇÃO
```

Fig. **14** - Relatório das Indicações sobre as Medidas com Sugestões para Revisão no Programa (continuação)

```
--------------------------------------------------------------------------
        SUGESTOES PARA REVISAO NO AMBIENTE DE TRABALHO DO PROGRAMA
                    XXXXXXXX.FOR EM   DD/MM/AA
                                 'AS  HH:MM HS
--------------------------------------------------------------------------

)---> Mensagem de sugestão para ACESSIBILIDADE

)---> Mensagem de sugestão para ATUALIZAÇÃO

)---> Mensagem de sugestão para LOCALIZABILIDADE EXTERNA

)---> Mensagem de atenção usuário
```

Fig. 15 - Relatório das Sugestões para Revisão no Ambiente de Trabalho do Programa

3.3 - Implementação

A descrição do que foi utilizado para implementação do APFOR e .uma visão geral de como ele está implementado, através de seu fluxo operacional, segue abaixo.

Como complementação desta documentação, encontra-se no Anexo II desta tese o Manual do Usuário com explicações sobre sua execução, funcionamento e operação, juntamente com um exemplo de ativação e o "lay-out" do5 seus resultados.

3.3.1 - Ambiente de desenvolvimento

Para a implementação do Módulo de Interface (MDI) e do Gerador de Informações de Apoio (GIA) foi utilizada a linguagem de programação Clipper SUMMER 87, e para a Analisador Estático de Código Fonte (AECF), foi utilizada a linguagem de programação Microsoft C versão 5.1 que também possibilita a interface com o Clipper. Estes foram

utilizados num microcomputador compatível com IBM PC **XT**

3.3.2 - Fluxo Operacional

O Fluxo Operacional do **APFOR** indicando seu código executável, denominado **APFOR.EXE**, com suas entradas e saídas, tanto com as principais informações fornecidas on-line, como com arquivos de trabalho e de resultados, estão expressos na figura 16 com suas respectivas descrições a seguir.

= **INFORMAÇOES DE ENTRADA**:

- **Características**: indica as características de qualidade a serem consideradas.
- Informações **PGM**: pode indicar se os comentários são claros e entendíveis (senão, quais as linhas do programa em que isso não ocorre), se os nomes de identificadores e variáveis são significativos (senão, quais os nomes não significativos), se o ambiente de trabalho do programa possui lugares designados para seu armarenamento com a versão mais atual do programa, e se existe Dicionário de Dados ou outro tipo de documentação externa com interelacionamentoç atualizados dos componentes do programa.

= **ARQUIVOS DE ENTRADA**:

- **XXXXXXXX.FOR**: código fonte de programa MS FORTRAN 77 **ANSI**.
- **INICIA.DAT**: arquiva de inicialização com os limites permitidos, obtidos da literatura para verificações de certas medidas. Com esse arquivo, o APFOR torna-se

flexível no sentido de permitir a alteração de alguns valores tanto para uma adaptação ou padronizacão em função da instituição em sue for utilizado, como por experiência e observação própria que demonstre outra realidade. Estes valores estão na seguinte ordem:

VARIAVEIS	VALORES
para o cabeçalho de manutenção padrão:	
sequência de caracteres para manutenção	**MANUTENCAO**
sequência de caracteres para responsável	**RESPONSAVEL**
sequência de caracteres para data	**BATA**
sequência de caracteres para motivo	**MOTIVO**
quantidade de espaços para indentação	3
limite para a complexidade ciclomática	10
limite para o tamanho	100
limite para o número de módulos superiores	3
limite para o número de módulos subordinados	9

- **ARQUIVOS DE TRABALHO:**

- **CARACTER.IMP**: arquivo com as características de qualidade sendo consideradas

- **LISPRG.LIN**: código fonte do programa **MS FORTRAN** 77 **ANSI** XXXXXXXX.FOR com as linhas numeradas sequencialmente.

- **STR_CON.MCS** e CONTROL2.MCS: arquivos de controle

- **ANALISE.CDF**: arquivo com os resultados das medidas da analise feita pelo **AECF**, onde podem possuir o valor 0 (nenhuma atende), 1 (todas atendem), 2 (apenas algumas atendem) ou 3 (não aplicável?.

- **LINREV.CDF**: arquivo com as linhas do programa que não atendem às medidas da análise feita pelo **AECF**.

- **LINREV.DBF**: arquivo LINREV.CDF no formato Clipper.

- ARQVARIA.DBF: arquivo com os identificadores e variáveis considerados não significativos pelo usuário.

= **ARQUIVOS DE RESULTADOS FINAIS/INFORMAÇÕES DE SAIDA:**

- ANALISE.RL1: arquivo com o relatório Resultados sobre as Medidas da Análise Estática no Programa (**Resultados das Medidas**).
- XXXXXXXX.RL2: arquivo com os três relatórios finais: Resultados sobre o Atendimento das Características de Qualidade (**Resultados do Atendimento**); Indicações sobre as Medidas com Sugestões para Revisão no Programa (**Sugestoes no Código**) e Sugestões para Revisão no Ambiente de Trabalho do Programa (**Sugestoes no Ambiente**).

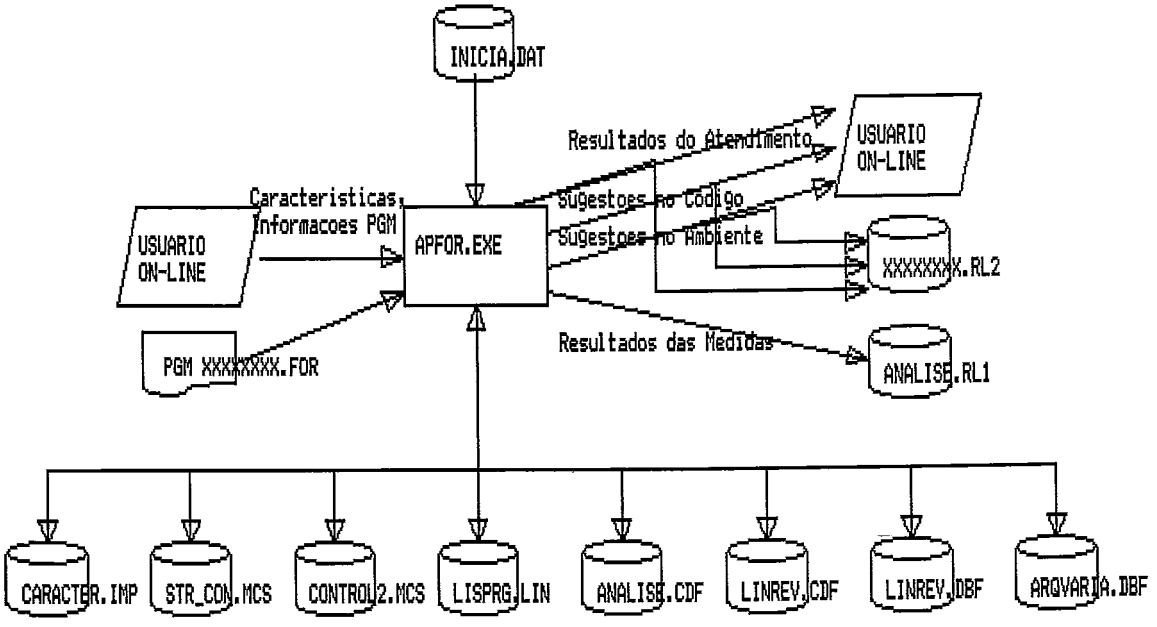


Fig. 16 - Fluxo Operacional do APFOR

CAPÍTULO IV

CONCLUSÃO

Considerando-se que a objetivo principal dessa tese e lidar com o processo de avaliação da qualidade de programas e tratar de sua automatização, foi proposta uma ferramenta para fazer parte e dar apoio à execução deste processo de avaliação, que resultou na definição da ferramenta APFOR – ANALISADOR ESTRTICO PARA APQIQ A AVALIAÇÃO DA QUALIDADE DE PROGRAMAS FORTRAN, especifica para programas FORTRAN 77 ANSI sobre o ambiente Microsoft FORTRAN versão 4.0 Com isso, camprova-se a validade e viabilidade do objetivo desta tese.

A utilização do APFOR traz as seguintes vantagens para o processo de avaliação da qualidade de programas:

- redução dos custos empregados, tornando economicamente mais viável este processo de avaliação;
- automatização deste processo de avaliação;
- incorporação das informações sobre este processa de avaliação nas funções exercidas pelo APFOR;
- apresentação de resultados finais da análise estática de uma maneira mais rápida e uniforme;
- melhoria na aplicação (coleta e análise de dados) deste processo de avaliacão, a nível de sua manutenibilidade e produtividade;
- utilização por diferentes tipos de usuário;
- divulgação da área de Controle da Qualidade de Programas, ainda pouco difundida e conhecida.

Com relação ao crescimento do APFOR num futuro, existem as seguintes possibilidades que poderão ser exploradas:

1- a avaliação dos outros dois objetivos de qualidade, segundo o método para avaliação da qualidade adotado, que são a CONFIABILIDADE CONCEITUAL e a UTILIZABILIDADE;

2- a aplicação para outras linguagens de programação, como Pascal, Assembler, Ada, C, PL/I ou Cobol;

3- a realização de uma abrangente pesquisa de campo para levantamento de opiniões envolvendo vários profissionais da área de Informática, das mais diversas e diferentes instituições, a fim de levantar graus de importância atribuídos às características de qualidade. Com isso, a opinião pratica de pessoas que têm mais e maior contato com a atividade de programação estaria registrada, sendo, por sua vez, o conhecimento de especialistas em programação utilizado para diferenciar e validar as características de qualidade.

4- a proposta de uma ferramenta completa, que realize totalmente o processo de avaliação da qualidade de programas através da inclusão de um **Assistente Especialista de Programas**, que possuiria aspectos de um sistema de apoio a decisão, de forma a indicar sua opinião e o motivo sobre o nível de qualidade do programa submetido à avaliação da qualidade, com a finalidade de ajudar o usuário a tomar decisões sobre futuras alterações no mesmo, concretizando, assim, o processo de Controle da Qualidade.

As vantagens trazidas por este Assistente Especialista de Programas, seriam principalmente os

seguintes:

- auxiliar o usuário na tarefa de Controle da Qualidade de Programas, não apenas apresentando os resultados finais obtidos, mas também proporcionando informações intermediárias que levaram à conclusão de tais resultados, consequentemente, auxiliaria na elaboração de programas de boa qualidade;
- proporcionar explicações e justificativas no desenrolar do processo de avaliação.

Isto se justifica pelo fato de que a aplicação do processo de avaliação da qualidade de programas possui algumas características, como as seguintes:

- é um problema não solucionado algoritmicamente;
- é um problema restrito, delimitado;
- envolve conhecimento especializado e escasso;
- serve de suporte para o processo de tomada de decisões sobre manutenções no programa;
- é uma área pouco difundida e explorada.

Pensando mais longe, esta ferramenta completa trabalharia com tecnologia de ponta através da ligação da Engenharia de Çoftware, onde encontram-se as áreas de Qualidade de Çoftware, e a Inteligência Artificial, resultando assim na idéia de se construir um Sistema Especialista aplicado à área de avaliação da qualidade de programas.

Finalizando, uma preocupação formal com a área de Controle da Qualidade de Çoftware no Brasil, atualmente,

ainda é pequena. Porém, é crescente a consciência çobre a importância e o acompanhamento de métodos, técnicas, auditorias e revisões de software. A experiência em termos de instituições dos EUA tem demonstrado que apesar da aplicação de um processo efetivo de avaliação da qualidade de software requerer um certo tempo e custo, ganhos reais podem ser obtidos com tal investimento ao longo da vida útil do software, como o trabalho abordado em (RADC-TR-85-372.

Para que então a área de Controle da Qualidade de Software seja mais divulgada e aplicada, *e* preciso haver um maior envolvimento das instituições e, principalmente, dos seus administradores, e onde possível, proporcionar meios automatitados para facilitar o processo de entendimento e operação da avaliação da qualidade de um programa. Uma ferramenta automatizada geralmente causa um impacto benéfico aos usuários, pois os estimula a se inteirar do assunto e poder, de uma maneira rápida, fácil e confiável, executar seu trabalho.

Assim, diante da própria proposta do APFOR - ANALISADOR ESTATICO PARA APOIO A AVALIAÇÃO DA QUALIDADE DE PROGRAMAS FORTRAN ao longo desta tese, e das indicações acima sobre suas implementações futuras, esta ferramenta pode ser considerada a base para futuras expansões dentro do processo de Controle da Qualidade de Programas, podendo até entrar em produção através da sua adaptação aos padrões e normas do ambiente de programação da instituição que o empregar.

REFERÊNCIAS BIBLIOGRÁFICAS

ADAM, M.F. e outros (1988), "Towards an observatory aiming at controlling the Software Quality", Second IEE/BCS Conference

ARTHUR, L.J. (1985), "Measuring programmer productivity and software quality", John Wiley & Sons

BACHE, R. e outros (1988), "Software Quality Assurance, a Rigorous Engineerins Practice", Second IEE/BCS Conference

BAECKER, R. (1988), "Enhancing Program Readability and Comprehensibility with Tools for Program Visualization", 10th International Conference on Software Engineerins, abril

BARGUT, M.F. (1986), "Qualidade de software para microcomputadores", Tese de mestrado, IME, Ria de Janeiro

BASILI, V.R. e Rombach, H.B. (1987), "Implementing Quantitative SQA: R Practical Model", IEEE Software, september

BASTANI, F.B. e Ramamoorthy, C.V. (1982), "Software Reliability - Statuç and Perspectives", IEEE Transactions on Software Engineering, vol. SE-8, no. 4, julho

BEAUFOND, C.E.C. (1987), "Verificação e Validação de software na fase de especificação de requisitos", Tese de mestrado, COPPE/UFRJ, Rio de Janeiro

BOEHM, B.W. e outrsç (1978), "Characteristics of Software Quality", North-Holland

BROWN, B.J. (1987), "Static Analysis and Dynamic Analysis applied to Software Quality Assurance", Handboak of Software Quality Assurance, Van Nostrand Reinhold Company

BUCKLEY, F.J. e Poston,R. (1984), "Software Quality Assurance", IEEE Transactions on Software Engineering, Vol. SE-10, no. ▮, janeiro

CARD, D.N., Page, G.T. e McGarry, F.E. (1985), "Criteria for Software Modularization", Proceedings 8th International Conference on âoftware Engineering, august

CARD, D.N. (1988), "The role of measurement in Software Engineering", Second IEE/BCS Conference, julho

CAVANO, J.P. (1985), "Toward High Confidence Software", IEEE Transactions on Software Engineering, vol. SE-11, tio. 12, dezembro

CERINO, D.A. (1986), "Software Quality Measurernent Tools and Techniques", Proceedings 10th Computer Software and Application Conference (COMSAC), IEEE

CONTE, S.D. (1986), Dunsmore,H.E. e Shen,V.Y., "Software Engineering Metrics and Models", The Benjamin/Cummings Pub Company

COOK, C. R. e Nanja, M. (1987), "Prototype Software Complexity Metrics Tool",. ACM SIGSOFT, vol. 12, no. 3, julho

EMERSON, T.J. (1984), "A Discriminant metric for Module Cohesion", Proceedings 7th International Conference on Software Engineering

FAIRLEY, R.F. (1985), "Software Engineering Concepts", New Yor-k, McGraw-Hill

FREITAS, A.C.C.T., Bargut, M.F. e Rocha, R.R.C.(19851, "Características de Qualidade de programas", Relatório Técnico ES-83/85, COPPE/UFRJ, Rio de Janeiro

HALSTEAD, M.H. (1977), "Elements *of* Software Science", Elsevier Publisher

HAUSEN, H.L. e Mullerburg, M. (1984), "An Introduction to Quality Assurance ané Control of Software", Software Validation, North-Holland, Elsevier Science Publishers

HEHL, M.E. (1986), "Fortran 77", McGraw-Hill

HIRAYAMA, M. e outros (1990), "Practice of quality modeling and measurement on Softwat-e Life Cycle", Proceedings 12th International Conference on Software Engineering

INCE, B.C. e Sheppard, M.J. (1988), "System Design Metrics: a Review and Perspective", Çecond IEE/BCS Conference, julho

JANUSZ, P.E. e Eckert, P.T. (1986), "Software Quality Assessment Measure", Proceedings Annual Reliability and Maintainability Symposium, IEEE

JENSEN, R.W. (1981), "Çtructured Programming", Computer, march

KELLER, R. (1987), "Expert System Technology - Development and Application", Yourdon Press

KERNIGHAN, B.W. e Plauger, P.J. (1978), "The elements of Programming Style", McGraw-Hill

LI, H.F. e Cheung, W.K. (1987), "An Empirical study of Software Metrics", IEEE Transactions on Software Engineering, vol. SE-13, no. 6, junho

MANNS, T.S. e Coleman, M.J. (1988), "Software Quality Assurance"

MARTIN, J. e Mc'Clure, C. (1983), "Software Maintenance", Prentice-Hall

MARTIN, J. e Mc'Clure, C. (1985), "Metodologias para Análise de Projeto de Sistemas", Compucenter

MCCABE, T.J. (1976), "A Complexity Measure", IEEE Transactisns on Software Engineering, vol. SE 2, no. 4, dezembro

MCCALL, J.A. e Cavano, J.P. (1978), "A framework for the measurement of Çoftware Quality", Proceedings ACM Software Quality Assurance Workshop, novembro

MCCALL, J.A. (1979), "An Introduction to Software Quality Metrics", Software Quality Management, J.D. Cooper, M.J. Fisher (eds), Petrocelli

MCCALL, J. (1981), Markhana,D., Stosick,M. e McGindly,R., "The Automated Measurement of Software Quality", Proceedings 5th International Computer Çoftware and Applications Conference, IEEE

MCMANUS, J.I. (1987), "How does software quality assurance fit in ?", Handbook of Software Quality Assurance, Van Nostrand Reinhold C.

METCALF, M. (1985), "Effective FORTRAN 77"

NASCIMENTO, E.M. e Rocha, A.R.C. (1986), "Manual para controle da qualidade sub-fator modularidade", Relatório Técnico ES-112/86, COPPE/UFRJ, Rio de Janeiro

PAGE-JONES, M. (1980), "The Practical Guide to Structured Systems Design", Yourdon Press

PAPGE, M. (1980), "A Metric for Çoftware Test Planning", Proceedings 4th International Computer Software and Applications Conference (COMPSAC 80)

PALERMO, S. e Rocha, A.R.C. (1988), "A Proposal ta evaluate program quality for the Delphi Off-line Environment", Relatório Técnico ES-180/88, COPPE/UFRJ, Rio de Janeiro

PERRY, W.E. (1983), "Effective methods of EDP Quality Assurance", Prentice-Hall

PRESSMAN, R.S. (1987), "Software Engineering", McGraw-Hill

(RADC-TR-85-37), "Specification of Çoftware Quality Attributes", Final Technical Report RADC-TR-85-37 (Volumes I, II e III), US Rome Air Development Center/Boeing Aerospace Company, 1985

ROCHA, A.R.C. (1987), "Análise e Projeto Estruturado de Sistemas", Rio de Janeiro

ÇNEEB, H.M. e Merey, A. (1985), "Automated Software Quality Assurance", PEEE Transacitons on Çoftware Engineering, vol. SE-iI, no. 9, setembro

SHOOMAN, M.L. (1983), "Software Engineering", McGraw-Hill

STAA, A.V. (1987), "Engenharia de programas", Livros Técnicas e Científicos Editora

STAHL, M.M. (1988), "Qualidade de programas: Estilo de Programação", Relatório Técnico ES-168/88, COPPE/UFRJ, Rio de Janeiro

UINCENT, J. e outros (1988), "Software Quality Assurance", Vol.1 Practice and Implementation, Prentice-Hall

ANEXO E

MANUAL PARA AVALIAÇÃO DA CONFIABILIDADE DA REPRESENTAGÃO DE PROGRAMAS FORTRAN

INDICE

ÍNDICE

I.1 - Introdução

Com o advento e evolução da Informática na vida atual para automatização e desenvolvimento das mais diversas atividades, existe uma crescente necessidade de se ter um software de boa qualidade, ou seja, um çoftware que esteja dentro de níveis satisfatórios de qualidade, já pré-determinados, de acordo com as solicitações do usuário e da área de aplicação em si. Isto influencia diretamente no nível de qualidade de cada um dos produtos gerados pelo software ao longo do seu processo de desenvolvimento. Para que esse processo seja verificado e controlado, é precise realizar um processo de Controle da Qualidade de Software, que identifique e avalie as várias características de qualidade envolvidas nos produtos de software, indicando se as mesmas são atingidas e satisfeitas de acordo com os requisitos de qualidade previamente estabelecidos.

Diante disto, o intuito deste Manual é tratar do processo de avaliação da qualidade de um produto bem específico que são programas codificados na linguagem de programação FORTRAN 77 ANSI, limitando-se às características de qualidade relacionadas à sua descrição, organização e representação.

Para facilitar o entendimento e a interpretação deste Manual, é preciso deixar claro algumas definições próprias do FORTRAN 77 ANSI. Logo, os seus principais termos são os seguintes (HEHL, 1986):

- **Argumente:** é um parâmetro passado entre unidades de programa para prover um meio de comunicação entre a unidade de programa que chama e a que é chamada.

- **Comanda:** e a unidade básica de um programa FORTRAN que classifica-se em executável e não executável. Um comando executável e aquele que especifica uma ação a ser realizada. Um comando não executável é aquele que descreve a natureza e as características dos dados ou das informações sobre o próprio código fonte.

- Função: é um subprograma que retorna um Único valor para a unidade de programa que a chama.

- **Função de comando;** é uma função, definida de forma similar a um comando de atribuição, que aparece interna à unidade de programa que a chama.

- **Função intrínseca:** é uma função, suprida pelo compilador FORTRAN, que realiza operações matemáticas ou sobre caracteres.

- **"Loop":** é a execução repetitiva de um comando ou grupo de comandos.. Normalmente controlado por um comando DO.

- Módulo: é o mesmo que unidade de programa.

- "Label": é um número composto de 1 a 5 dígitos decimais que é usado para identificar um comando.

- **Procedimento:** é um conjunto sequencial de comandos que pode ser usado em um ou mais pontos de um ou mais programas de computador, e sue usualmente recebe um ou mais parâmetros de entrada e retorna um ou mais parâmetros de saída. Consiste de subrotinas, subprogramas função e funções intrínsecas.

- **Procedimento externo:** é um subprograma subrotina ou subprograma função escrito em FORTRAN.

- **Programa principal:** é uma unidade de programa que pede chamar outras unidades de programa, mas que não pode ser chamada por estas. O programa principal é o primeiro a receber o controle no inicio da execução, e seu primeiro comando é o PROGRAM.

- **Subprograma:** é uma unidade de programa que é chamada por outra unidade de programa. Tem como primeiro comando, um comando FUNCTIBN, SUBROUTINE ou BLOCK DATA.

- **Subrotàna:** é um subprograma SUBROUTINE.

- **Unidade de programa:** é uma sequência de comandos constituindo um programa principal ou um subpragrama, terminando com um comando ENB.

Depois de todas essas definições, o próximo capitulo 1.2 contém a estrutura do método para avaliação da qualidade proposto por Rocha (1987) adotado. O capitulo 1.3 contém a estrutura do Manual através da descricão do seu "lar-out" e em seguida apresenta o Manual propriamente dita com cada característica de qualidade no seu respectivo formulário e na seguinte ordem:

- objetivo de qualidade Confiabilidade da Representação;
- fator- de qualidade Legibilidade;
- respectivos sub-fatores de qualidade da Legibilidade, onde, após cada um desses, segue

imediatamente seus respectivos critérios de qualidade, antes de começar o próximo sub-fator;

- fator de qualidade Manipulabilidade;
- respectivos sub-fatores de qualidade da Manipulabilidade, onde, após cada um desses, segue imediatamente seus respectivos critérios de qualidade, antes de começar e próximo sub-fator.

I.2 - Método para avaliação da qualidade de software

O Método para avaliação da qualidade de software adotado é o proposto por Rocha (1987), que se baseia nos seguintes conceitos:

- Objetivos de qualidade: são as propriedades gerais que o produto deve possuir.
- Fatores de qualidade: são características que determinam a qualidade do ponto de vista das diferentes usuários do produto (usuário final, mantenedores, etc).
- Critérios de qualidade: definem atributos primitivos possíveis de serem avaliadas.
- Medidas: indicam o grau de presença de um determinado critério, ou seja, são os resultadas quantitativos da avaliação do produto segundo um determinado critério.
- Processes de Avaliação: determinam os procedimentos e os instrumentos a serem usados de forma a se medir o grau de presença na produto de um determinado atributo.
- Medidas agregadas: indicam o grau de presença de um determinado fator de qualidade, atraves da agregação das medidas obtidas a partir da avaliação segundo os

critérios

À estes conceitos, inclui-se mais um outro sue é o de Çub-fator de qualidade. Este tipo é o fator de qualidade num nível mais detalhado de influência sobre a produto, sendo também atingido pelas medidas agregadas.

Objetivos, fatores e sub-fatores não são diretamente mensuráveis e só podem ser avaliados através dos respectivos critérios. Com isso, nenhum critério isolado é uma descrição completa de um determinado sub-fator, nem um sub-fator isolado é uma descrição completa de um fator, e por sua vez, nenhum fator isolado define completamente um objetiva de qualidade.

Com base neste método de avaliação, a classificação das características de qualidade de programas também baseia-se no trabalho de Rocha (1987), completando-se com o nível de critérios. Esta classificação, para. o objetivo Confiabilidade da Representação tratado neste Manual, é mostrada na figura AI.1 .

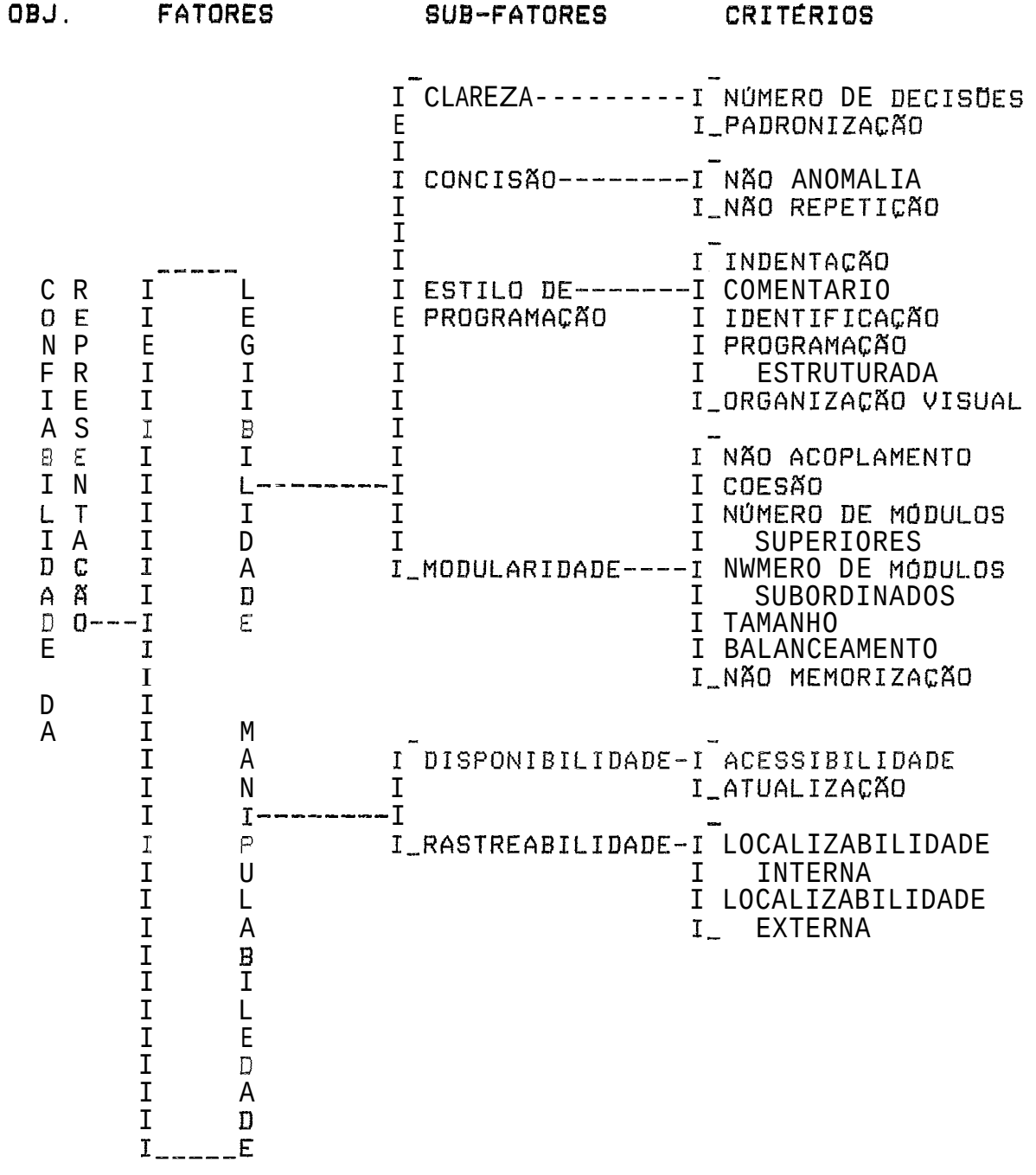


Fig. AI.l - Classificação das características de qualidade

I.3 - Estrutura do Manual

Este Manual define e descreve os procedimentos necessários para serem aplicados às avaliações do objetiva de qualidade Confiabilidade da Representação e seus respectivos fatores, sub-fatores e critérios de qualidade, divididos hierarquicamente. Para tanta, cada um desses encontra-se especificado num formulário pré-definido, descrito a seguir.

A descricão de objetivo, fatores e sub-fatores de qualidade contém, num mesmo tipo de formulário, os seguintes ítens:

- **Definição**: apresenta a definicão precisa e sucinta da característica de qualidade em questão.
- Processo de avaliação: identifica as respectivas características de qualidade, subordinadas hierarquicamente, que permitem a avaliação da característica de qualidade em questão.
- Interpretação: através de um quadro, expressa o resultado da avaliação da característica de qualidade em questão pela classificacão ATENDE, NÃO ATENDE e NÃO APLICÁVEL na coluna RESULTADO, juntamente com o que deve estar satisfeito para se obter tal resultado na conluna CONDIÇÃO.

Já a descricão de critérios de qualidade contém, num mesmo tipo de formulário, os seguintes ítens:

- **Definição**: apresenta uma definição precisa e sucinta do critério em questão.
- Discussão: apresenta uma fundamentação teórica resumida

do critério em questão.

- **Processo de avaliação**: descreve a avaliação do critério em questão através de uma das seguintes formas:
  - um questionário;
  - um cálculo especifico.
- **Interpretação**: através de um quadro, expressa o resultado da avaliação do critério de qualidade em questão **pela** classificação ATENDE, NÃO ATENDE e NÃO APLICAVEL na coluna RESULTADO, juntamente com o que deve estar satisfeito para se obter tal resultado na conluna CONDIÇÃO.
- **Sugestão de revisão:** sugere algum procedimento a ser aplicado no caso de haver um resultado classificado como NÃO ATENDE no quadra da Interpretação.

Objetivo : Confiabilidade da Representação

Definição:

É a caracteristica de um programa ser facilmente entendido e manipulado com relação à sua descrição, organização e representação.

Processo de avaliação:

Este objetivu é avaliado através dos seguintes fatores:

- Legibilidade
- Manipulabilidade

Interpretação:

RESULTADO	CONDIÇÃO
atende a Confiabilidade da Representação	atende à Lesibilidade e atende à Manipulabilidade
não atende à Confiabilidade da Representação	não atende à Legibilidade ou não atende à Manipulabilidade
não aplicável à Confiabilidade da Representação	não aplicável à Legibilidade e não aplicável à Manipulabilidade

---

Fator: Legibilidade **Objetivo: Confiabilidade da Representação**

Definição:

É a característica de um programa estar codificado de maneira clara e legível, de modo a ser facilmente lida e entendido.

Processo de avaliação:

Este fator é avaliado através dos seguintes sub-fatores

- Clareza
- Concisão
- Estilo de Programação
- Modularidade

Interpretação:

RESULTADO	CONDIÇÃO
atende à Legibilidade	atende à Clareza e atende à Concisão e atende ao Estilo de Programação e atende à Modularidade
não atende à Legibilidade	não atende à Clareza ou não atende à Concisão ou não atende ao Estilo de Programação ou não atende à Modularidade
não aplicável à Legibilidade	não aplicável à Clareza e não aplicável à Concisão e não aplicável ao Estilo de Programação e não aplicável à Modularidade

.....

Sub-fator: Clareza Objetivo: Confiabilidade da Representação
Fator: Legibilidade

Definição:

É a característica de um programa ter suas funções codificadas da forma mais clara e simples possível.

Processo de avaliação:

Este sub-fator é avaliado através dos seguintes critérios:

- **Número de Decisões**
- **Padronização**

Interpretação:

RESULTADO	CONDIÇÃO
atende a Clareza	atende ao Número de Decisões e atende à Padronização
não atende a Clareza	não atende ao Número de Decisões ou não atende à Padronização
não aplicável à Clareza	não aplicável ao Número de Decisões e não aplicável à Padronização

---

Critério: Número de Decisões Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Çub-fator: Clareza

Definição:

O programa possui o critério **Número de Decisões** na medida em que possua uma quantidade mínima de comandos de decisão segundo a Complexidade Ciclomática de McCabe.

Discussão:

McCabe propõe uma medida de complexidade de software baseada na teoria dos grafos, a Complexidade Ciclomática, que por sua ver baseia-se no número de caminhos básicos dos comandos de decisão existentes na estrutura de um grafo. Diante disso, considera o programa como um grafo de controle com um Único ponto de entrada e um Único ponto de saída.

Assim, utiliza-se a fórmula proposta para a cálculo da Complexidade Ciclomática simplificada V(G) considerando apenas o número total de condicões, ou seja, de operadores lógicas nos comandos IF de decisão existentes.

Quando o número de decisões creçce, a Complexidade Ciclomática também cresce. De acordo com McCabe, para tornar o programa mais claro, essa complexidade não deve ultrapassar o limite de 10. Consequentemente, com uma Complexidade Ciclomática mínima, diminui-se a complexidade de seu entendimento.

Processo de avaliação:

Este critério pode ser avaliado através dos seguintes cálculos sobre a Complexidade Ciclomatica simplificada, baseada em McCabe, sobre o código fonte do programa em questão:

Para cada unidade de programa i, calcular:

Vi =
IF´s + NOT´s + AND´s + OR´s + EQV´s + XOR´s + NEQV's + 1

onde: IF´s = número total de comandos IF de i
NOT's = número total de operadores .NOT. existentes na condição dos comandos IF
AND´s = número total de operadores .AND. existentes condição dos comandos IF
OR's = número total de operadores .OR. existentes na condição dos comandos IF
EQV´s = número total de operaderes .EQV. existentes na condição dos comandos IF

XOR´s = número total de operadores .XOR. existentes na condição dos comandos IF
NEQV's = número total de operadores .NEQV. existentes na condição dos comandos IF

Critério: Número de Decisões **Objetivo: Confiabilidade da Representação**
Fator: Legibilidade
Çub-fator: Clareza

Limite máximo para Vi indicado por McCabe = **10**

Interpretação:

RESULTADO	CONDIÇÃO
atende ao Número de Decisões	**todos os Vi não ultrapassam o limite máximo**
não atende ao Número de Decisões	todos os Vi ultrapassam o limite máximo
não aplicável ao Número de Decisões	não existe comando IF

Sugestão de revisão:

O programa deve ser revisto, no sentido de simplificar sua5 respectivas condições **ou até mesmo ser dividido em** subprogramas.

Critério: Padronização — Objetivo: Confiabilidade da Representação
Fator: Legibilbdade
Sub-fator: Clareza

Definição:

O programa possui o critério **Padronização** na medida em que obedece aos padrões técnicos de programação estabelecidos tanto pelo FORTRAN 77 ANSI como pela instituição a que pertence.

Discussão:

A verificação do programa com relação aos padrões técnicos do FORTRAN 77 ANÇP é feita durante a própria compilação do programa.

Quanto aos padrões técnicos de programação estabelecidos pela instituição, sua definição e obediência é uma tarefa importantíssima e necessária, pois é através destes que se pode atingir uma maior homogeneidade dos programas, e consequentemente, do software, tornanda-o mais claro, uniforme e de mais fácil entendimento.

Existem vários tipos de padrães técnicos de programação que podem ser estabelecidos por uma instituição, como os seguintes:

- orientações para estilo de programação;
- recomendações ou normas de programação específicas para a linguagem de programação utilizada;
- nomenclatura padrão;
- documentação do programa;
- técnicas para especificação do programa, como gráfico de estrutura ou fluxograma;
- formulários padrão para definição do programa.

Processo de avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário:

	S	N	NA
Q1- Todos as padrões técnicos de programação adicionais estabelecidos pela instituição, incluídos nesta avaliação, são obedecidos ?	--	--	--
	S	N	
Q2- O programa está sem erros de compilação ?	--	--	

Critério: Padronização **Objetivo:** **Confiabilidade da** Representação
Fator: Legibilidade
Sub-fator: **Clareza**

Interpretação:

RESULTADO	CONDIÇÃO
atende à Padronização	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Padronização	respostas do questionário são N (Não) ou NA (Não aplicável)

Sugestão de revisão:

O programa **deve** ser **revisto, no sentido de devidamente seguir os padrões** técnicos **de programação da** instituição **ou estar realmente compilado.**

.....

Sub-fator: Concisão Objetivo: Confiabilidade da Representação
Fator: Legibilidade

Definição:

É a característica de um programa ter suas funções implementadas com uma quantidade mínima de código fonte, sem informação excessiva ou repetida

---

Processo de avaliação:

Este sub-fator é avaliado através dos seguintes critérios:

- **Não Anomalia**
- **Não Repetição**

Interpretação:

RESULTADO	CONDIÇÃO
atende a Concisão	atende à Não Anomalia e atende a Não Repetição
não atende à Concisão	não atende à Não Anomalia ou não atende à Não Repetição
não aplicável à Concisão	não aplicável à Não Anomalia e não aplicável à Não Repetição

---

.....

Critério: Não Anomalia

Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Concisão

Definição:

O programa possui o critério Não Anomalia na medida em que não possua práticas desnecessárias ou procedimentos não utilizados.

.....

Discussão:

O excesso de código acaba cansando e desviando a atenção do usuário leitor. Assim, quanto mais simplificado e conciso o código, mais fácil é a leitura e acompanhamento do mesmo.

Para sue isso não ocorra, e preciso evitar certas práticas e procedimentos que só aumentam o tamanha do código do programa e são simplesmente desnecessários ou não utilizados.

Processo de avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

	S	N	NA
Q1- Todo código é usado e atingido, evitando-se código morto ?	--	--	--
Q2- Evitou-se subprograma tipo "dummy" ?	--	--	--
Q3- Existe algum subprograma que não seja chamado ?	--	--	--
Q4- Evitou-se comando FORMAT não referenciado e desnecessário ?	--	--	--
Q5- Evitou-se superdimensionar os "arrays" ?	--	--	--
Q6- Todaç as variáveis declaradas são utilizadas ?	--	--	--
Q7- Todos os argumentos de subrotinas são utilizados ?	--	--	--
Q8- Todos os subprogramas possuem uma função, um único objetivo de processamento, ou seja, são funcionais ?	--	--	--
Q9- Todas as condições de erro, de critica, ou de exceção previstas são aplicáveis e apenas necessárias ?	--	--	--
Q10- Evitou-se modificar o índice da "loop" DO ?	--	--	--

Critério: Não Anomalia

Objetavo: Confiabilidade **da** Representação
Fator: Legibilidade
Sub-fator: Concisão

Q11- Evitou-se expressões de tipos diferentes ?
-- -- --

Q12- Evitou-se overlay ~/o usegmentação ?
-- -- --

Q13- Os comandos especificas para teste/depuração foram retirados ?
-- -- --

Interpretação:

RESULTADO	CONDIÇÃO
atende à Não Anomalia	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Não Anomalia	respostas do questionário são N (Não) ou NA (Não aplicável)
não aplicável à Não Anomalia	respostas do questionário são NA (Não aplicável)

Sugestão de revisão:

O programa deve ser revisto, no sentido de retirar a parte do código desnecessária ou não utilizada.

Critério: Não Repetição Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Concisão

Definição:

O programa possui o critério **Não Repetição** na medida em que não possua uma mesma sequência de código repetida em diferentes lugares desnecessariamente.

.....

Discussão:

Quando uma sequência de código repetida desnecesariamente é transformada num Único subprograma, a quantidade de código do programa é reduzida, e consequentemente sua leitura torna-se mais rápida.

A presença de repetições, além de aumentar o número de linhas executáveis, pode ocasionar problemas durante sua manutenção, uma ver que é preciso fazer um mesmo tipo de alteração em mais de um lugar dentro do mesmo programa, podendo nessa hora ocasionar erros c inconsistências. No caso de realmente haver necessidade da repetição explícita de um mesmo procedimento em diferentes lugares, é aconselhável o uso de mecanismos localizadores, como a Referência Cruzada, para que a manutenção seja feita consistentemente em todos os devidos lugares em sue o mesmo for referenciado.

Processo de avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

	S	N	NA
Q1- Evitou-se repetir um mesmo procedimenta desnecessariamente ?	--	--	
Q2- Um mesmo subprograma não é chamado desnecessariamente em mais de um lugar ?	--	--	--
Q3- Evitou-se recalcular/reprocessar mais de uma ver um mesmo valor em expressões aritméticas desnecessariamente?	--	--	--
Q4- Evitou-se usar variáveis intermediárias de trabalho desnecessariamente ?	--	--	--

---

Critério: Não Repetição Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Concisão

Interpretação:

RESULTADO	CONDIÇÃO
atende à Não Repetição	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Não Repetição	respostas do questionário são N (Não) ou NA (Não aplicável)

Sugestão de revisão:

O programa deve ser revisto, no sentido de transformar cada procedimento repetido desnecessariamente num Único subprograma.

Sub-fator: Estilo de Programação — **Objetivo:** Confiabilidade da Representação — **Fator:** Legibilidade

---

Definição:

É a característica de um programa conter elementos de estilo, de forma que expresse seus objetivos e especificações de maneira clara, elegante, organizada, direta, consistente e padronizada.

Processo de avaliação:

Este sub-fator é avaliado através dos seguintes critérios:

- **Indentacão**
- **Comentário**
- **Identificação**
- **Programação Estruturada**
- **Organização Visual**

Interpretação:

RESULTADO	CONDIÇÃO
atende ao Estilo de Programação	atende à Indentação e atende ao Comentário e atende à Identificação e atende à Programação Estruturada e atende à Organização Visual
não atende ao Estilo de Programação	não atende à Indentação ou não atende ao Comentário ou não atende à Identificação ou não atende à Programação Estruturada ou não atende à Organização Visual

---

Critério: **Indentação**

Objetiva: Csnfiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Definição:

O programa possui o critério Indentação na medida em que utiliza afastamentos da margem esquerda para melhorar sua estrutura lógica, destacar níveis de aninhamentos e blocos de comando.

Discussão:

Este tipo de postura melhora a estrutura lógica, destaca níveis de aninhamentos e blocos de comandos da mesmo.

Os afastamentos da margem esquerda referem-se aos espaços em branco, deixados no começo da linha, geralmente entre a coluna um e a coluna onde se inicia o comando, dependendo da linguagem de programação. R indentação é um tipo de formatação muito importante, pois permite organizar a disposição doç comandos, e mostrar a hierarquia, subordinação e aninhamento das mesmos, tornando, assim, o programa mais fácil de ser entendido.

Assim, o uso da indentação permite destacar e aumentar a visualização das declarações, "loops", procedimentos e blocos de comanda importantes. Além disso, deve ser feita devidamente e quando necessária, ou seja, estando de acorda com as recomendações gerais para seu uso, pois quando utilizada de maneira exagerada, incorreta ou inconsistente, tende a levar o usuário leitor a ler de uma maneira diferente o código do programa. Consequentemente, pode confundi-lo e induzi-lo a interpretar e tirar conclusões incorretas sobre a sua verdadeira lógica.

Processo de avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

	S	N	NA
Q1- São utilizadas consistentemente indentações de três espaces em branco ?	--	--	--
Q2- Nos comandos DO, seu correspondente bloco interno esta indentado ?	--	--	--
Q3- Nos comandas IF, os correspondentes blocos do THEN e do ELSE estão indentados ?	--	--	--
Q4- As linhas de continuação de um comando estão indentadas de pelo menos com um espaço em branco ?	--	--	--

---

Critério: Indentação

Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Interpretação:

RESULTADO	CONDIÇÃO
atende à Indentação	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Indentação	respostas do questionário são N (Não) ou NA (Não aplicável)
não aplicável à Indentação	respostas do questionário são todas NA (Não aplicável)

Sugestão de revisão:

O programa deve ser revisto, no sentido de indentar os comandos onde necessário.

Critério: Comentário — Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Definição:

O programa possui o critério Comentário na medida em que contenha comentários com explicações sobre o seu funcionamento e lógica.

.....

Discussão:

Os comentários são o meio de comunicacão textual do programa com os seus possíveis usuários leitores através de explicações claras e diretas. Os comentários podem ser incluídos em qualquer lugar no programa, sendo a base de sua documentação interna. É composto de textos e explicações que facilitam o entendimento das declarações, comandos e procedimentos ao longa do programa, devendo ser os mais completos e sucintos possíveis, ou seja, dizendo tudo em poucas e diretas palavras. A inclusão de comentários, além de facilitar e agilizar o entendimento do funcionamento do programa, reduz também a necessidade de referência à sua documentação externa, tornando-o, assim, auto-descritivo.

Os comentários devem proporcionar informações adicionais corretas, que não sejam prontamente obtidas do código fonte, sem atrapalhá-lo, reproduzí-lo simplesmente ou levar a conclusões erradas, dificultando a própria detecção de erro. Dessa maneira, seu intuito é simplesmente facilitar a leitura e entendimento do5 programas, principalmente as longos e complexos. Deve-se tomar cuidado, pois seu uço excessivo, ou seja, a nível de cada declaração ou comando, ou o posicionamento indevide pode prejudicar enormemente seu percurso visual e até atrapalhar na interpretação apropriada e correta da lógica e das funções descritas no mesmo. Logo, não há necessidade de se ter um comentário para cada comanda, para cada estrutura de controle ou para cada ponto de decisão do mesmo.

Os comentários podem ser de dois tipos, o comentário Cabecalho, localizado no início do programa principal, e o comentário Explicativo, localizado ao longo de todo o programa.

O comentário tipo Cabecalho é uma informação muito importante e valiosa, principalmente para os usuários leitores que vêem pela primeira vez o programa. Ele é uma incialização e preparação para o melhor entendimento do programa, e pode conter as diversas informações, a serem preenchidas de acordo com sua realidade e necessidade.

Vale lembrar que o mais importante é o conteúdo dos comentários e não seu volume.

Critério: **Comentário** Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Processo **de avaliação**:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre a código fante do programa em questão:

	S	N	NA
Q1- Os comentários estão escritos em português simples, claro, preciso e conciso, sem detalhamentos exagerados e sem abreviações não signifieativas ?	--	--	--
Q2- Os comentários estão servindo realmente para esclarecimento ?	--	--	--
Q3- Os comentários contêm as explicações necessárias para o entendimento da código a que se referem ?	--	--	--
Q4- A quantidade de comentário5 é suficiente para facilitar o entendimento da programa ?	--	--	--
Q5- Existe comentário tipo Cabeçalho/Prólogo no início de código fonte ?	--	--	--
Q6- Existem um comentário tipo Explicativo para cada subprograma ?	--	--	--
Q7- Evitou-se colocar comentários na mesma linha que um comando (entre as colunas 73 e 80) ?	--	--	--
Q8- Os conteúdos dos comentários estão atualizados ?	--	--	--
Q9- Existem comentários que contém as asçertivas de entrada de programa ?	--	--	--
Q10- Existem comentários que contêm as assertivas de saída do programa ?	--	--	--
Q11-comentários servem como uma auto-documentação efetiva do programa ?	--	--	--
Q12- Existe consistência no uso de comentários ?	--	--	--
Q13- Cada subrotina, correspondente a uma unidade reutilizável, que pode ser desvineulada da programa, possui um comentário tipo Cabeçalho/Prólogo ?	--	--	--

Critério: Comentário — Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Q14- As palavras ou expressões em inglês usadas são apenas as de uso geral ? -- -- --

Q15- O comentário é para blocos de código e não para cada linha ? -- -- --

Q16- Os procedimentos para exceção ou recuperação de erros são comentados ? -- -- --

Q17- Os atributos das variáveis de entrada/saída estão comentados ? -- -- --

Q18- Todo código dependente de máquina está comentado ? -- -- --

Q19- Os atributos não óbvios das variáveis declaradas estão comentados ? -- -- --

Q20- Os comentários estão nos principais pontos do programa ? -- -- --

Q21- Evitou-se colocar linhas de comentário entre eontinuações ? -- -- --

Interpretação:

RESULTADO	CONDIÇÃO
atende ao Comentário	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende ao Comentário	respostas do questionário são N (Não) ou NA (Não aplicável)

Sugestão de revisão:

O programa deve ser revisto, no sentido de alterar os comentários para fornecerem maiores e melhores esclarecimentos, e se necessário, seguirem a padronização de formatação estabelecida para os mesmos.

Critério: Identificação Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Definição:

O programa possui o critério **Identificação** na medida em que possua nomes significativos, apropriados e mnemônicos.

Discussão:

Eçte tipo de postura reduz a necessidade de comentários específicos sobre os mesmos e agiliza muito o entendimento do programa.

Os nomes utilizados devem ser auto-explicativos e mnemônicos, ou seja, significativos, de forma a representar a função desempenhada, enfatizar a estrutura lógica em que são empregados e ajudar a tornar o programa auto-documentado. Com isso, facilita o entendimento, reduz sensivelmente a esforço empregado nas manutenções, descreve e esclarece a função dos identificadores de maneira rápida e fácil, minimizando a necessidade de comentários sobre os mesmas. Esta auto-explicação pode levar a nomes compostos por sequências de caracteres longos, *e* ai deve-se usar o bom senso para não se criar nomes longos desnecessariamente, pois o FORTRAN 77 ANÇI só permite até 6 caracteres, o que dificulta o entendimento de seu significade. Neste caso, deve-se procurar utilizar abreviações o mais inteligíveis possível e fornecer uma relação das principais variáveis e seus significados. Assim, é muito importante sue seja feita a devida criação de nomes.

O uso de nomes significativos pode também ser empregado para nomes de programas, subprogramas, arquivos, relatórios ou telas.

Processo de avaliação:

Eçte critério pode ser avaliado através das respostas ao seguinte questionário sobre os nomes dos identificadores e variáveis do código fonte do programa em questão:

	S	N	NA
Q1- Os nomes obedecem a padronização de nomenclatura estabelecida ?			
	--	--	--
	S	N	
Q2- Os nomes associam a si mesmo o objetiva, a propriedade física ou funcional que desempenham, ou seja, são mnemônicos e significativos ?			
	--	--	
Q3- Todas as variáveis estão declaradas explicitamente ?			
	--	--	

Critério: Identificacão — Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Q4- Evitou-se escrever os nomes de maneira não usual ?	--	--	
Q5- Os nomes são Únicos, fáceis de reconhecer ?	--	--	
Q6- São usados nomes comuns à área de aplicação ?	--	--	
Q7- É usado um esquema consistente e pronunciável para abreviação dos nomes ?	--	--	
Q8- São evitadas palavras-chave como nome de variável ?	--	--	
Q9- Um identificador com um objetivo próprio é representado por um mesmo nome e não por mais de um nome ?	--	--	
Q10- Evitou-se usar uma mesma variável para contextos diferentes ?	--	--	
Q11- O significado de cada variável é consistente ao longo de todo programa ?	--	--	
	S	N	NA
Q12- Os nomes semelhantes diferem de pele menos dois caracteres ?	--	--	--
Q13- É usado um prefixo/sufixo comum para identificar nomes que estão agrupados lógicamente, indicando ítens subordinados que estão relacionados por algum motivo ?	--	--	--
Q14- São evitadas nomes em inglês e português ao mesmo tempo, só usando aqueles já de uso geral ?	--	--	--
Q15- Os numerais nos nomes são colocados de preferência mais para o final ?	--	--	--

---

Critério: Identificação — Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Interpretação:

RESULTADO	CONDIÇÃO
atende à Identificação	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Identificação	respostas do questionário são N (Não) ou NA (Não aplicável)

Sugestão de revisão:

O programa deve ser revisto, no sentido de utilizar nomes mnemônicos e se necessário, seguirem a padronização da nomenclatura estabelecida para os mesmos.

.....

Critério: Programa~%~ Estruturada

Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Definição:

O programa possui o critério **Programação Estruturada** na medida em que obedeça às normas da técnica de Programação Estruturada.

Discussão:

A técnica Programação Estruturada é utilizada para construir a lógica de um programa através da estruturas lógicas de controle e suas combinações.

Os principais objetivos da Programação Estruturada são os seguintes:

- minimizar o número de erros ocorridos durante o desenvolvimento do programa;
- minimizar o esforça requerido para corrigir erros em partes deficientes do código, atualizando essas partes com técnicas mais confiáveis, funcionais ou eficientes;
- minimizar os custos de manutenção do çoftware;
- gerar pregramaç simples, claros e bem definidos em termos estruturais;
- produzir código que possa ser lido sequencialmente em pequenas partes, permitindo a leitura dessas partes de forma top-down e visualizando nessas mesmas todos os seus caminhos de controle.

Processo de avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

	S	N	NA
Q1- Na construção do programa foi aplicado conscientemente as normas da Programação Estruturada ?	__	__	
Q2- Evitou-se usar o comando EMTRY nos subprogramas ?	__	__	__
Q3- Evitou-se colocar mais de um comande RETURN num subprograma ?	__	__	__

Critério: **Programação** Estruturada

Objetivo: Confiabilidade da Representação

Fator: Legibilidade

Sub-fator: Estilo de Programação

Q4- Evitou-se colocar dentro do comando IF um comando GO TO desviando para fora ?

-- -- --

Q5- Evitou-se colocar dentro do comando IF um comando GO TO desviando para fora ?

-- -- --

Interpretação:

RESULTADO	CONDIÇÃO
atende à Programação Estruturada	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Programação Estruturada	respostas do questionário são N (Não) ou NA (Não aplicável)
não aplicável à Programação Estruturada	respostas do questionário são todas **NA** (Não aplicável)

Çugestão de revisão:

0 programa deve ser revisto, no sentido de realmente obedecer as normas da técnica de Programação Estruturada.

Critério: **Organização** Visual Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Definição:

O programa possui o critério Organização Visual na medida em que seus names e comandos estejam dispostos de maneira organizada.

.....

Discussão:

Este tipo de postura facilita e agiliza o entendimento do programa, permite que seu código evolua de maneira sistemática, oferece uma visão mais clara de sua estrutura e fica esteticamente mais elegante e agradável de se ver e ler.

A sequência de comandos codificados num programa determina todo o seu entendimento, e nem uma grande quantidade de comentários ou uma documentação suplementar pode substituir inteiramente estes comandos bem expressos.

A organização em um programa envolve as disposições e atribuições dos seguintes:

- labels;
- nomes;
- linhas de comando;
- declarações e comandos;
- estruturas de dados.

Dentro doç comandos de declaração, os próprios argumentas, quando for o caso, podem estar organizados, de acordo com o seu pretendido uso.

Os espaços em branca ajudam a delinear os elementos sintáticos de uma declaração ou comando. Podem ser usados depois de vírgulas, antes/depois de operadores de expressão e do sinal de atribuieão (=). Não é aconselhável colocar muitos espaços num mesmo comando ou muitas linhas em branco juntas, pois acabam reduzindo a legibilidade do mesmo.

Quanto a disposição de nomes em listas, eles podem ser colocados em ordem alfabética e por colunas, facilitando, assim, a localização de um nome numa lista.

A utilizacão de linhas em branco ao longo do programa, de forma não abusiva, também facilita a visualização das suas partes mais significativas. Com isso, permite destacar e aumentar a visualização dos comentários, cabeçalhos, títulos, declarações, "loops", procedimentos e comandos importantes.

Critério: **Organização** Visual — Objetivo: Csnfiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Processo de **avaliação**:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

	S	N	NA
Q1- Os "labels" estão dispostos em ordem crescente ?	--	--	--
Q2- É usado espaço em branco antes e depois dos operadores ?	--	--	--
Q3- É usado espaço em branco antes e depois da sinal = de atribuição ?	--	--	--
Q4- É usado um espaço em branco depois de vírgulas que separam variáveis na lista dos comandos de declaração ?	--	--	--
Q5- São utilizadas linhas em branco antes de depois de subprogramas ?	--	--	--
Q6- São utilizadas linhas em branco antes e depois dos comentários ?	--	--	--
Q7- Os comandas estão alinhados na mesma coluna inicial do comando PROGRAM ?	--	--	--
Q8- São utilizadas linhas em branco para separar os comandos de especificação do corpo do programa ?	--	--	--
Q9- Os "labels" do comando FORMAT possuem ~ & r i @ numérica diferente da usada para os outros comandos ?	--	--	--
Q10- Os comandos FORMAT de cada unidade de programa eçtão agrupados juntos ?	--	--	--
Q11- Cada "loop" do comando DO termina num comando CONTINUE ?	--	--	--
Q12- Os subprogramas estão em ordem alfabética ?	--	--	--
Q13- As linhas de continuação de um comando, exceto a Última, não parecem terminar legalmente o comanda ?	--	--	--

---

Critério: Organização Visual — Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Q14- Os "labels" estão posicionados mais para a direita de seu campo ? -- -- --

15- Os comandos DATA estão depois de todos os outros comandos de especificação e antes de qualquer função ou comandos executáveis ? -- -- --

Q16- Cada declaração de um dado bloco CBMMOM é idêntica em todos os aspectos ? -- -- --

Q17- Os blocos COMMON estão em ardem alfabética ? -- -- --

Q18- Os argumentos estão ordenados numa lista de acordo com seu pretendido uso ? -- -- --

Q19- São usados parênteses nas expressões aritméticas ou lógicas (comando IF composto) ? -- -- --

Q20- Evitou-se testes condicionais complicados ? -- -- --

Q21- Evitou-se testes nas condições negativas ? -- -- --

Q22- As listas nos comandos de declaração estão arrumadas em ordem alfabética ? -- -- --

Q23- Os itens de tipos similares estão alinhados numa mesma coluna ? -- -- --

Q24- As variáveis definidas nos comandos de especificação, que são iniciadas em linhas diferentes, mantém um alinhamento vertical ? -- -- --

Q25- São utilizadas linhas em branco para destacar as partes mais significativas e importantes do programa ? -- -- --

Q26- São utilizadas linhas em branco para destacar grupos de comandos com funções lógicas distintas ? -- -- --

Q27- São utilizadas linhas em branco antes de importantes comandos indentados ? -- -- --

Q28- São utilizadas linhas em branco para separar grupos de comandos semelhantes ? -- -- --

Q29- São utilizadas linhas em branco de maneira consistente ? -- -- --

---

Critério: Organização Visual Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Estilo de Programação

Q30- Evitou-se usar mais de três níveis de aninhamento ?

-- -- --

Interpretação:

RESULTADO	CONDIÇÃO
atende à Organização Visual	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Organização Visual	respostas do questionário são N (Não) ou **NA** (Não aplicável)

Sugestão de revisão:

O programa deve ser reviçto, na sentido de dispor melhor e mais organizadamente seus nomes e comandos onde necessário.

Sub-fator: Modularidade Objetivo: Confiabilidade da Representação
Fator: Legibilidade

Definição:

É a cat-acterística de um programa ter suas funções implementadas através de uma estrutura de subprogramas altamente independentes e particionados logicamente.

Processo de avaliação:

Este sub-fator é avaliado através dos seguintes critérios:

- Não Acoplamento
- Coesão
- Número de Módulos Superiores
- Número de Módulos Subordinados
- Tamanho
- Balanceamento
- Não Memorização

Sub-fator: Modularidade — Objetivo: Confiabilidade da Representação
Fator: Legibilidade

Interpretação:

RESULTADO	CONDIÇÃO
atende à Modularidade	atende ao Não Acoplamento e atende à Coesão e atende ao Número de Módulos Superiores e atende ao Número de Modulos Subordinados e atende ao Tamanho e atende ao Balanceamento e atende à Não Memorização
não atende à Modularidade	não atende ao Não Acoplamento ou não atende a Coesão ou não atende ao Número de Módulos Superiores ou não atende ao Número de Módulos Subordinadas ou não atende ao Tamanho ou não atende ao Balanceamento ou não atende à Não Memorização

---

Critério: Não Acoplamento — Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Modularidade

Definição:

O programa possui o critério Não Acoplamento na medida em que possua um fraeo interrelarionamento entre as suas subrotinas.

Discussão:

Acoplamento é um modo de particionamento do programa através do interrelacionamento ou interdependência entre duas subrotinas.

Um forte acoplamento entre os módulos é prejudicial ao software, pois torna-se mais difícil entender, mudar e corrigir apenas uma Única subrutina se ela estiver altamente relacionada com outra. Ja um fraco acoplamento indica um programa bem partbcianado, com um menor relacionamento entre as suas subrotinaç, com a complexidade reduzida e altamente independente, diminuindo a chance de um erro em um das subrotinas repercutir em outra.

Existem definidos na literatura quatro tipos de acoplamento que podem ocorrer entre duas subrotinas. Estes são os seguintes, em ordem decrescente de importância para que ocorra um fraco acoplamento: Acoplamento de dados, Acoplamento por dados estruturados, Acoplamento por controle e Acoplamento por área comum.

- Acoplamento de dados: a comunicação entre as subrotinas é feita através de uma lista de parâmetros, onde o parâmetro pode ser tanto um dada elementar como uma tabela homogênea para passagem dos dados.

Esta é a melhor forma para um fraco acoplamento, pois acarreta em interfaces concisas e facilmente testáveis.

- Acoplamento por dados estruturados: a comunicação entre as subrotinas é feita pelo compartilhamento de uma mesma estrutura de dados interna e não simplesmente pelos dados necessários.

Este tipo tende a fornecer a um dos módulos mais dadas do que ele necessita.

- Acoplamento por controle: a comunicação entre as subrotinas é feita quando uma subrotina passa Para outra um campo de informação ("flag"), que é usado para controlar a lógica interna.

Este tipo indica um pobre particionamento, pois tanto uma função pode estar dividida em duas subretinas, come uma subrotina tem de saber da lógica interna detalhada da outra.

Mas nem todos os "flags" são prejudiciais. Existe o "flag" descritivo que diz ao módulo chamador o que fazer e o que deve ser evitado.

- Acoplamento por área comum: a comunicacão entre os módulos é feita quando eles se referem à uma mesma estrutura de dados global.

Criteria: **Não Acoplamento** Objetivo: Confiabilidade da Repreçentação
Fator: Legibilidade
Sub-fator: Modularidade

Processo **de avaliação:**

Este critério é avaliado através das seguintes verificações sobre o código fonte do programa em questão:

Para cada subrotina i, verificar se existe:

- Acoplamento de dados (comunicação por dados elementares)
- Acoplamento por dados estruturados (comunicação por estrutura de dados)
- Acoplamento por controle (comunicação com flag de controle)
- Acoplamento por área comum (comunicação por COMMON)

Interpretação:

RESULTADO	CONDIÇÃO
atende ao Não Acoplamento	todas as i's possuem Acoplamento de dadas ou por dados estruturados
não atende ao Não Acoplamento	todos as i's possuem Acoplamento por controle ou por área comum
não aplicável ao Não Acoplamento	não existe subprograma

Sugestão de revisão:

O par de subrotinas em que foi detectado o Acoplamento por área comum ou por controle deve ser revisto, no sentido de mudar o tipo de comunicação para que ocorra um fraco acoplamento.

Critério: **Coesão** Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Modularidade

Definição:

O programa possui o critério **Coesão** na medida em que possua um forte relacionamento entre os argumentos dos seus subprogramas.

Discussão:

Com isso, avalia a força de associação funcional entre os argumentos de cada subprograma, ou seja, a força que mantém unidos estes elementos.

O alto nível de coesão no subprograma é o fator mais significativo para um alto nível de modularidade, acarretando num subprograma com elementos genuinamente relacionados, pois é um indicador da aproximação entre a estrutura do programa e a estrutura do problema implementado. No caso dos elementos de um subprograma estarem fortemente relacionados a elementos de outro subprograma, ou seja, houver um forte acoplamento, acarreta na diminuição da coesão de ambos.

Existem definidos na literatura oito tipos de coesão que podem ocorrer num subprograma. Estes são os seguintes, em ordem crescente de importância para que ocorra uma alta coesão: coincidental, lógica, temporal, procedural, comunicacional, sequencial e funcional.

- **Coesão coincidental**: os argumentos não têm razão aparente para estarem juntos, uma vez que não existe relacionamento significativo entre os mesmos.
- **Coesão lógica**: existe algum tipo de relacionamento lógico entre os argumentos, que contém muitas funções comuns que estão numa mesma categoria e são executadas uma de cada vez de acordo com uma seleção de controle feita pelo próprio subprograma.

Este tipo parece-se muito com a Coesão coincidental, uma vez que as funções desta mão estão relacionadas nem quanto ao fluxo de dados e nem quanto ao fluxo de controle, e já na Coesão lógica , suas funções estão, no mínimo , na mesma categoria.

- **Coesão temporal**: os argumentos estão envolvidos em funções diferentes e relacionados pelo tempo, sem uma ordem lógica de execução.
- **Coesão procedural**: os argumentos estão envolvidos em funções diferentes, onde o fluxo de controle, ou seja, a sequência lógica, segue de uma função para a próxima.

Subprogramas com Coesão procedural tendem a ser compostos por partes de funções que tenham pouco relacionamento entre si, exceto quando são executadas numa ordem específica por um mesmo período de tempo.

Este tipo parece-se muito com a Coesão temporal, uma vez que os elementos desta são executados no mesmo período de tempo, e já na Coesão procedural, a ordem de execução é importante.

Critério: Coesão — Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Çub-fator: Modularidade

- Coesão comunicacional: os argumentos estão relacionado~ por uma referência ao mesmo conjunto de dados de entrada e/ou de saída. 0 subprograma geralmente executa muitas funções relacionadas pelo uso de seus dados.
- Coesão sequencial: ocorre quando um argumento serve como entrada e depois saída num subprograma.

Este tipo parece-se muito com a Coesão comunicacional, só que para a Coesão sequencial, a ordem de execução é importante.

Este tipo normalmente tem uma forte coesão, pois pode conter várias funções ou simplesmente parte de uma. função. A Única desvantagem real é que ele não é prontamente reutilizável por outras partes do software.

- Coesão funcional: todos os argumentos estão relacionados apenas para a execução de uma Única função.

É o tipo mais alto de coesão, sendo prontamente reutilizável por outras partes do software.

Processo de **avaliação**:

Este critério é avaliada através das seguintes verificações sobre o código fonte do programa em questão:

Para cada subprograma i, verificar:

Se existe mais de um ponto de entrada
então Coesão lógica ou coincidental

Se existe um Único ponto de entrada e vários pontos de saída
então Coesão lógica ou coincidental
senão Coesão funcional, sequencial, comunicacional, procedural ou temporal

---

Interpretação dos resultados:

RESULTADO	CONDIÇÃO
atende à Coesão	todas as i's possuem Coesão funcional, sequencial, comunicacional, procedural ou temporal
não atende à Coesão	todos as i's possuem Coesão lógica ou coincidental
não aplicável à Coesão	não existe subprograma

Critério: Coesão **Objetivo: Confiabilidade da Representação**
Fator: Legibilidade
Sub-fator: Modularidade

Sugestão de revisão:

O subprograma **com coesão** lógica **ou** coincidental **deve ser revisto, no sentido de procurar fornecer** aos **seus** elementos **internos um outro tipo de** relacionamento **com uma mais** forte **coesão.**

Critério: Número de Módulos Superiores

Objetivo: Confiabilidade da Representação

Fator: Legibilidade

Sub-fator: Modularidade

Definição:

O programa possui o critério Número de Módulos Superiores na medida em que possua para cada um de seus subprogramas chamados, um adequado número de unidadeç de programa que respectivamente os chamem.

Discussão:

Um adequado número de unidades de programa que chamam um mesma çubprograma subordinado, é de 1 a **3.** Neste caso, ocorre a concentração de ativação de um subprograma, ou seja, o "fan-in".

Quando isto acontecer, é uma indicação de uma mais fácil reutilização do subprograma chamado, consequentemente, menor é a complexidade de organização do programa cantenda menos subprosramas individuais e evitando-se a duplicação de código. Logo, também agiliza o entendimento do programa.

Processo de avaliação:

Eçte critério pode ser avaliado através dos seguintes cálculos sobre o código fonte do programa em questão:

NMSPi = para cada subprograma i,
contar o número de subrotinas, função e programa principal que a chamam

Limite máximo para NMSPi = 3

Interpretação:

RESULTADO	CONDIÇÃO
atende ao Número de Módulos Superiores	todos os NMSPi são menores ou iguais a 3
não atende ao Número de Módulos Superiores	todos os NMSPi são maiores que 3
não aplicável ao Número de Módulos Superiores	não existe subprograma

.....

Critério: Número de Módulos Superiores

Objetivo: Confiabilidade da Representação

Fator: Legibilidade

Çub-fator: Modularidade

.....

Sugestão de revisão:

Deve ser revista, no sentido de verificar a possibilidade de subdividir algum subprograma, pois pode conter mais de uma função com diferentes chamadores.

Critério: Número de Módulos Subordinadas

Objetivo: Confiabilidade da. Representação
Fator: Legibilidade
Çub-fator: Modularidade

Definição:

O programa possui o critério **Número de Módulos Subordinados** na medida em que possua para cada unidade de programa, um baixo número de subprogramas respectivamente chamados pelas mesmas.

Discussão:

Um baixo número de subprogramas chamados por uma determinada unidade de programa, é de 1 a 9. Neste caso, ocorre a distribuição de ativação de uma unidade de programa, ou seja, o "fan-out".

Quando isto acontecer, é uma indicação de uma mais difícil reutilização da unidade de programa que chamou, pois ela depende de muitos outros subpragramas, o que aumenta bastante a quantidade de subprogramas a serem reutilirados.

Um número muito alto ou muito baixo de subprogramas chamados são indicadores de um projeto pobre, embora um número alto seja mais perigoso do que um número baixo.

Processo de avaliação:

Este critério pode ser avaliado através dos seguintes cálculos sobre o código fonte do programa em questão:

NMSBi = para cada unidade de programa i,
contar o número de subrotina e função chamadas pela mesma

Limite máximo para NMSBi = 9

Interpretação:

RESULTADO	CONDIÇÃO
atende ao Número de Módulos Subordinados	todos os NMSBi não ultrapassam o limite máximo
não atende ao Número de Módulos Subordinados	todos os NMSPi ultrapassam o limite máximo

---

Sugestão de revisão:

O programa deve ser revisto, no sentido de diminuir a quantidade de subprogramas sendo chamados.

Critério: **Tamanho**

Objetive: Confiabilidade da Representacão
Fator: Legibilidade
Sub-fator: Modularidade

Definição:

O programa possui o critério **Tamanho** na medida em que suas unidades de programa não ultrapassem um número padrão de linhas de código executáveis.

.....

Discussão:

Uma linha de código executável é qualquer linha executavel com declaração ou comandos do código fonte do programa ou do módulo, excluindo linhas em branco, de comentários, de comandos de declaração, de comandos de designação de memória e de inicialização não executável.

Foram realizadas várias pesquisas para estabelecer um limite superior para o tamanho de um módulo. Baker considera que 50 comandos é um número adequada; por aproximar-se do número de linhas sue podem ser impressas em uma página de listagem na impressora. Weimberg considera 30 comandos, para que não haja uma brusca queda da compreensão do mbdulo. Boehm já considera 100 comandoç. Arthur considera que o limite de tamanho de um módulo é de 100 linhas de código executável, exceto para módulos de edição. Vincent também considera que cada módulo não deve exceder a um tamanho padrão de 100 comandos. Card, de uma pesquisa com 453 módulos escritos em FORTRAN, concluiu que:

- bons programadores não têm preferência por um tamanho especifico para os módulos;
- módulos grandes custam menos, por declaração executável, do que módulos pequenos;
- a incidência de erros não está relacionada ao tamanho da módulo.

Quando o programa necessitar de meios apropriados para reagir a situações hostis ou para evitar falhas sue provoquem consequências desastrosas, necessitando para tanto um procedimento para tratamento de erros ou de exceção bem detalhista e abrangente, ele provavelmente conterá mais comandos do que o realmente necessário, por uma questão de controle requisitado pelo software. Nestes casos, a complexidade e o tamanho aumentam.

Processo de Avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

Para cada unidade de programa i, calcular:

Ti = quantidade de linhas de código executáveis

Limite máxima para Ti = 100

Critério: Tamanho

Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Modularidade

Interpretação:

RESULTADO	CONDIÇÃO
atende ao Tamanho	todos os Ti não ultrapassam o limite máximo
não atende ao Tamanho	todos os Ti ultrapassam o limite máximo

---

Sugestão de revisão:

O programa deve ser revisto, no sentido de suas unidades de programa serem separadas, divididas em partes menores.

Critério: Balanceamento

Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Çub-fator: Modularidade

Definição:

O programa possui o critério **Balanceamento** na medida em que apenas suas subrotinas de mais baixo nível envolvam as características físicas.

Discussão:

Assim, objetiva-se uma estrutura balanceada onde os módulos de nível mais alto tratam os dados da uma forma lógica, de modo que estes sejam independentes de características físicas, e aos de nivel mais baixo são atribuídas essas características físicas.

Diante disto, as estruturas balanceadas são mais fáceis de se implementar e fazer manutenções, principalmente porque estas envolvem **os** dispositivos **físicos** ou **os** formatos de Entrada/Saída.

Processo de avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

	S	N	NA
Q1- Somente as subrotinas de mais baixo nível ou programa principal possuem os comandos READ, WRITE, PRINT, OPEN, CLOSE, INQUIRE, REWIND, BACKSPACE, ENB FILE ou FORMAT ?	--	--	--

Interpretação:

RESULTADO	CONDIÇÃO
atende ao Balanceamento	resposta do questionário é S (Sim)
não atende ao Balanceamento	resposta do questionário é N (Não)
não aplicável ao Balanceamento	resposta do questionário é NA (Não aplicável)

Critério: Balanceamento

Objetivo: **Confiabilidade da Representação**

Fatar: Legibilidade

Sub-fator: Modularidade

.....

Sugestão de revisão:

O programa deve ser revisto, no sentido de colocar nas sua5 subrotinas de mais baixo nível ou no programa principal os comandos de E/S e de especificação de formato.

Critério: Não **Memorização** Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Modularidade

Definição:

O programa possui o critério Não **Memorização** na medida em que não possua qualquer tipo de memória para manter o estado das variáveis entre ativações sucessivas de suas rubrotinar.

Discussão:

Este tipo pode ser uma membria de estado ou uma memória temporária.

A memória de estado é a memória prévia da existência do módulo. Geralmente suando um módulo é executado, ele não tem memória sobre sua existência anterior, ou seja, sobre o que aconteceu em outras execuções anteriores. Entretanto, existe um tipo de módulo que pode conter um registro de seu passado, como se fosse um histórico, através de sua memória de estado. Um módulo possuidor desse tipo de memória torna-se imprevisível, pois para entradas idênticas comporta-se diferentemente, produzindo resultados inesperados a cada execução.

A memória temporária é a memória, ou melhor, a área de trabalho, compartilhada entre módulos. Neste caso, estes mesmos módulos correm o risco de perderem sua independência, tornando-se também imprevisíveis, pois a mudança da ordem de execução dos mbdulos pode provocar resultados inesperados. Logo, o módulo deve possuir memória própria, não compartilhável com outros módulos.

Para evitar esses tipos de problema com a interpretação das resultados, principalmente durante a manutenção, e poder assim **não** se perder, entendendo o devida comportamento, e manter a independência de cada um dos módulos do programa, é que deve-se centralizar o tratamento de variáveis através de memória própria em cada módulo.

Processo de avaliação:

Este critério pode ser avaliado através das respostaç ao seguinte questionário sobre as subrotinas do código fonte do programa em questão:

	S	N	NA
Q1- Evitou-se usar o comando EQUIVALENCE ?	--	--	--
Q2- Evitou-se usar o comando COMMON ?	--	--	--

Critério: Não Memorização Objetivo: Confiabilidade da Representação
Fator: Legibilidade
Sub-fator: Modularidade

Interpretação:

RESULTADCI	CONDIÇÃO
atende à Não Memorização	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Não Memorização	respostas do questionário são N (Não) ou NA (Não aplicável)
não aplicável à Não Memorização	não existe subprograma

Sugestão de revisão:

Os subprogramas devem ser revistos, no sentido de se evitar a utilização dos comandos EQUIUALENCE ou COMMON.

Fator: Manipulabilidade **Objetiva: Confiabilidade da Representação**

Definição:

É **a característica para um programa ser facilmente** acessado **e manipulado pelos seus** possíveis **usuários leitores**

Processo de avaliação:

Este fator é **avaliado** através **dos seguintes sub-fatores:**

- **Disponibilidade**
- **Rastreabilidade**

Interpretação:

RESULTADO	CONDIÇÃO
atende à Manipulabilidade	atende à Disponibilidade **e atende à Rastreabilidade**
não atende à Manipulabilidade	não atende à Disponibilidade ou não atende à Rastreabilidade
não aplicável à Manipulabilidade	**não aplicável à Disponibilidade e não aplicável à Rastreabilidade**

Çub-fator: Disponibilidade Objetivo: Confiabilidade da Representação
Fator: Manipulabilidade

Definição:

É a característica de um programa estar com seu código fonte disponível, pronto e atualizado para ser usado

Processo de avaliação:

Este sub-fator é avaliado através dos seguintes critérios:

- Acessibilidade
- Atualização

Interpretação:

RESULTADO	CONDIÇÃO
atende à Disponibilidade	atende à Acessibilidade e atende à Atualização
não atende à Disponibilidade	não atende à Acessibilidade ou não atende à Atualização
não aplicável à Disponibilidade	não aplicável à Acessibilidade e não aplicável à Atualização

---

Critério: Acessibilidade

Objetivo: Confiabilidade da Representação
Fator: Mãnipulabilidade
Sub-fator: Disponibilidade

Definição:

O programa possui o critério **Acessibilidade** na medida em que seja facilmente aeessado ou consultado.

Discussão:

O próprio código fonte e a sua respectiva documentacão interna devem ser utilizados, quando preciso, por diferentes usuários leitores, por isso e que o programa deve ser aeessado de maneira rápida e fácil.

Geralmente é vantajoso manter toda uma documentação em meio automatizado, pois isto permite que:

- a documentação esteja sempre disponível;
- a documentação possa ser facilmente reproduzida e armazenada após seu uso;
- os usuários possam ter rapidamente uma cópia sempre que desejarem;
- a consulta a qualquer um dos documentoç armazenados seja fácil c ágil.

Na raso da código fonte de um programa, ele geralmente está armazenado no próprio hardware que o executa, dentro de alguma biblioteca ou diretório apropriado.

Processa de avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

	S	N	NA
Qi- Está armazenado nos lugares designados para tal, como bibliotecas, diretórios, pastas ou armários ?	--	--	--

	S	N
Q2- **Está** armazenado em seu próprio meio automatizado, ou melhor, no hardware que o executa ?	--	--
Q3- Pode ser obtida uma cópia da sua listagem ?	--	--

Critério: Acessibilidade — Objetivo: Confiabilidade da Representação; Fator: Manipulabilidade; Çub-fator: Disponibilidade

Interpretação:

RESULTADO	CONDIÇÃO
atende à Acessibilidade	respostas do questionário são S (Sim) ou NA (Não aplicável)
não atende à Acessibilidade	respostas do questionário são N (Não) ou NA (Não aplicável)
não aplicável à Acessibilidade	respostas do questionário são NA (Não aplicável)

Sugestão de revisão:

A maneira de armazenamento do programa deve ser revista, no sentido de torná-la de mais fácil acesso.

Critério: Atualização

Objetivo: Confiabilidade da Representação
Fator: Manipulabilidade
Çub-fator: Disponibilidade

Definição:

O programa possui o critério **Atualização** na medida em que está na sua versão mais atual.

Discussão:

É importante e vital para o sucesso de todo o software ter sempre toda sua documentação atualizada, pois ela serve de base tanto para fins de manutenção durante sua vida útil como para reutilização/geração de outros produtos.

As informações sobre manutenções no programa devem vir no seu comentário tipo Cabeçalho/Prólogo, no início do mesmo, contendo: indicação de manutenção; responsável pela manutenção; data da manutencão e motivo da manutencão.

Consequentemente, o5 correspondentes comentários descritivos devem ser incluídos ao longo do programa onde necessário.

Assim, nos vários lugares em que o programa pode estar armazenado, sejam automatinadoç ou não, como as bibliotecas de fonte ou pastas de listagens, ele deve estar sempre em sua versão mais atual.

A utilização de ferramentas automatizadas, tanto na atualização da documentação interna do programa como no controle do ambiente operacional a que o mesmo estiver ligado através do usa de sistemas de Gerenciamento de Configuração, auxilia muito na efetivação e divulgação de uma nova versão do programa.

Processo de avaliação:

Este critério pode ser avaliado através das respostas ao seguinte questionário sobre o código fonte do programa em questão:

	S	N	NA
Q1-Esta é a sua versão mais atual, ou seja, a versão em teste mais recente ou em produção/ operação ?	--	--	
Q2- Todas as suas cópias, nos lugares designadas para tais, estão na sua versão mais atual ?	--	--	--
Q3- Foi utilizada alguma ferramenta automatizada para gerar sua documentacão interna ?	--	--	--
Q4- Foi utilizada alguma ferramenta para Gerenciamento de Configuração ?	--	--	--

Critério: Atualização

Objetiva: Confiabilidade da Representação
Fator: Manipulabilidade
Sub-fator: Disponibilidade

Interpretação:

RESULTADO	CONDIÇÃO
atende à Atualizacão	respostas do questionário são S (Sim) ou **NA** (Não aplicável)
não atende à Atualização	respostas do questionário são N (Não) ou NA (Não aplicável)
não aplicável à Atualização	respostas do questionário são NA (Não aplicável)

Sugestão de revisão:

Deve ser revisto, no sentido de sua documentação interna e seu próprio código fonte conter as informações mais recentes ou estar armazenada a sua versão mais atual.

Çub-fator: Rastreabilidade **Objetivo: Confiabilidade da Representação**
Fator: Manipulabilidade

Definição:

É **a característica** de **haver** possibilidade de localização **e acompanhamento dos componentes referenciados por um** programa.
o

Processo de avaliação:

Este sub-fator é avaliado através dos seguintes critérios:

- **Localizabilidade Interna**
- **Localizabilidade Externa**

Interpretação:

RESULTADO	CONDIÇÃO
atende à Rastreabilidade	atende à Localizabilidade Interna e atende à Localizabilidade Externa
não atende à Rastreabilidade	**não atende** à Localizabilidade **Interna** ou **não atende** à Localizabilidade **Externa**
não aplicável à Rastreabilidade	não aplicável à Localizabilidade Interna e não aplicável à Localizabilidade Externa

---

Critério: Localizabilidade Interna

Objetivo: Confiabilidade da Representação
Fator: Manipulabilidade
âub-fator: Rastreabilidade

Definição:

O programa possui o critério Localizabilidade **Interna** na medida em que haja possibilidade de localizar seus componentes referenciados dentro de si mesmo.

Discussão:

Estes componentes podem ser nomes de variáveis, "arrays", constantes, unidades de programa, arquivos, relatórios ou telas.

Para tal facilidade de rastreamento dos componentes referenciados dentro do próprio programa, existem mecanismos auxiliadoreç, que são basicamente os analisadores automáticos de código, que por sua vez fornecem as várias formas de Referência Cruzada dos seus componentes. A Referência Cruzada é a referência entre as entidades por meios lógicos, que geralmente pode ser fornecida durante a compilação/depuração do programa, ou então por algum Utilitário/Programa específico para tal. Os relatórios de Referência Cruzada podem indicar:

- relação dos componentes do programa versus sua localização dentro do mesmo (linha da código fonte em que aparecem ou unidade de programa em que são referenciados);
- relação de hierarquia das unidades de programa existentes;
- relação de unidades de programa versus correspondentes unidades de programa que são chamadas pelas mesmas, ou unidades de programa versus correspondentes unidades de programa que chamam as mesmas.

Processo de avaliação:

Este critério pode ser avaliado através da resposta à seguinte pergunta:

	S	N	NA
Q1- O programa possui algum tipo de Referência Cruzada que relacione todos os seus componentes ?	--	--	--

---

Critério: Localizabilidade Interna

Objetivo: Confiabilidade da Representação
Fator: Manipulabilidade
Sub-fator: Rastreabilidade

Interpretação:

RESULTADO	CONDIÇÃO
atende à Localizabilidade Interna	resposta do questionário é S (Sim)
não atende à Localizabilidade Interna	resposta do questionário é N (Não)
não aplicável à Localizabilidade Interna	resposta do questionário é NA (Não aplicável)

Sugestão de revisão:

Deve ser introduzida algum tipo de ferramenta de Referência Cruzada que forneça a relação de todos os componentes do programa.

Critério: Localizabilidade Externa

Objetivo: Confiabilidade da Representação
Fator: Manipulabilidade
Sub-fator: Rastreabilidade

Definição:

O programa possui o critério **Localirabilidade Externa** na medida em que haja possibilidade de se localizar os componentes interrelacionados ao mesmo na documentação externa existente.

Discussão:

Pode-se, assim, percorrer os componentes relacionados ao programa a partir da indicação do mesmo. Estes componentes podem ser nomes de unidades de programa, arquives, relatórios ou telaç.

Existem mecanismos auxiliadores de percurso que facilitam a localização dos relacionamentos entre os componentes, o mais importante deles e o Dicionário de Dados.

O Dicionário de Dados pode possibilitar associar, de forma top-down ou bottom-up, os componentes subordinados a outros componentes, facilitando o acesso aos componentes referenciados e o conhecimento de seus relacionamentos. Geralmente estas relações encontram-se na descrição do Bicionário de Dados de cada componente. Por exemplo, no Dicionário de dadoç de um arquivo podem estar relacionados tanto os registros e campos subordinados ao mesmo como os programas que o referenciam.

Processo de avaliação:

Este critério pode ser avaliado através da resposta à seguinte pergunta:

	S	N	NA
Q1- Existe Dicionário de Dados que contenha os interrelacionamentos entre componentes do programa ?	--	--	--

---

Critério: Localizabilidade Externa

Objetivo: Confiabilidade da Representação

Fator: Manipulabilidade

Sub-fator: Rastreabilidade

Interpretação:

RESULTADO	CONDIÇÃO
atende à Localizabilidade Externa	resposta do questionário é S (Sim)
não atende à Localimabilidade Externa	resposta do questionário é N (Não)
não aplicável à Localizabilidade Externa	resposta do questionário é NA (Não aplicável>

Sugestão de revisão:

Deve ser introduzido um Dicionário de Dados com os interrelacionamentos entre as componentes referenciados dentro do mesmo.

ANEXO II

MANUAL DO USUÁRIO DO APFQW - ANALISADOR ESTÁTICO PARA APOIO A AVALIAÇÃO DA QUALIDADE DE PROGRAMAS FORTRAN

ÍNDICE

II.1 - Introdução

Este Manual é referente à ferramenta APFOR - ANALISADOR ESTÁTICO PARA APOIO A AVALIACÃQ DA QUALIDADE DE PROGRAMAS FORTRAN, fornecendo explicações sobre seu funcionamento, escopo, execução e operação. As definições dos termos envolvidos no mesmo são encontradas no próprio APFOR, através das suas telas de Ajuda. Para finalizar o Manual, é fornecido um exemplo prático de todo um procedimento de execução no APFOR. Com isso, o usuário entenderá melhor e estará mais capacitado a utilizar adequadamente a ferramenta.

O objetivo principal do APFOR é realizar a tarefa de análise estática, para determinadas características de qualidade, sobre um programa MS FORTRAN 77 ANSI, ou seja, um programa escrito na liguagem de programação FORTRAN 77 ANSI e já compilado no ambiente Microsoft FORTRAN versão 4.0, a fim de servir como instrumento e com informações de apoio para a execução do processo de avaliação da qualidade do programa em questão.

Estas caracteristicas (divididas em objetivo, fator, sub-fator e critério) de qualidade consideradas fazem parte de todo um conjunto de caracteriçticas que podem estar envolvidas sob o ponto de vista da descrição, organização e representação do programa. Assim, o escopo do APFOR para análise de caracteristicas é restrito, não abrangendo todas as possíveis caracteriçticas determinantes da qualidade de um programa. Logo, estas características formam uma classificação específica para o APFOR, estando relacionadas

em ordem hierárquica por coluna na figura AII.1

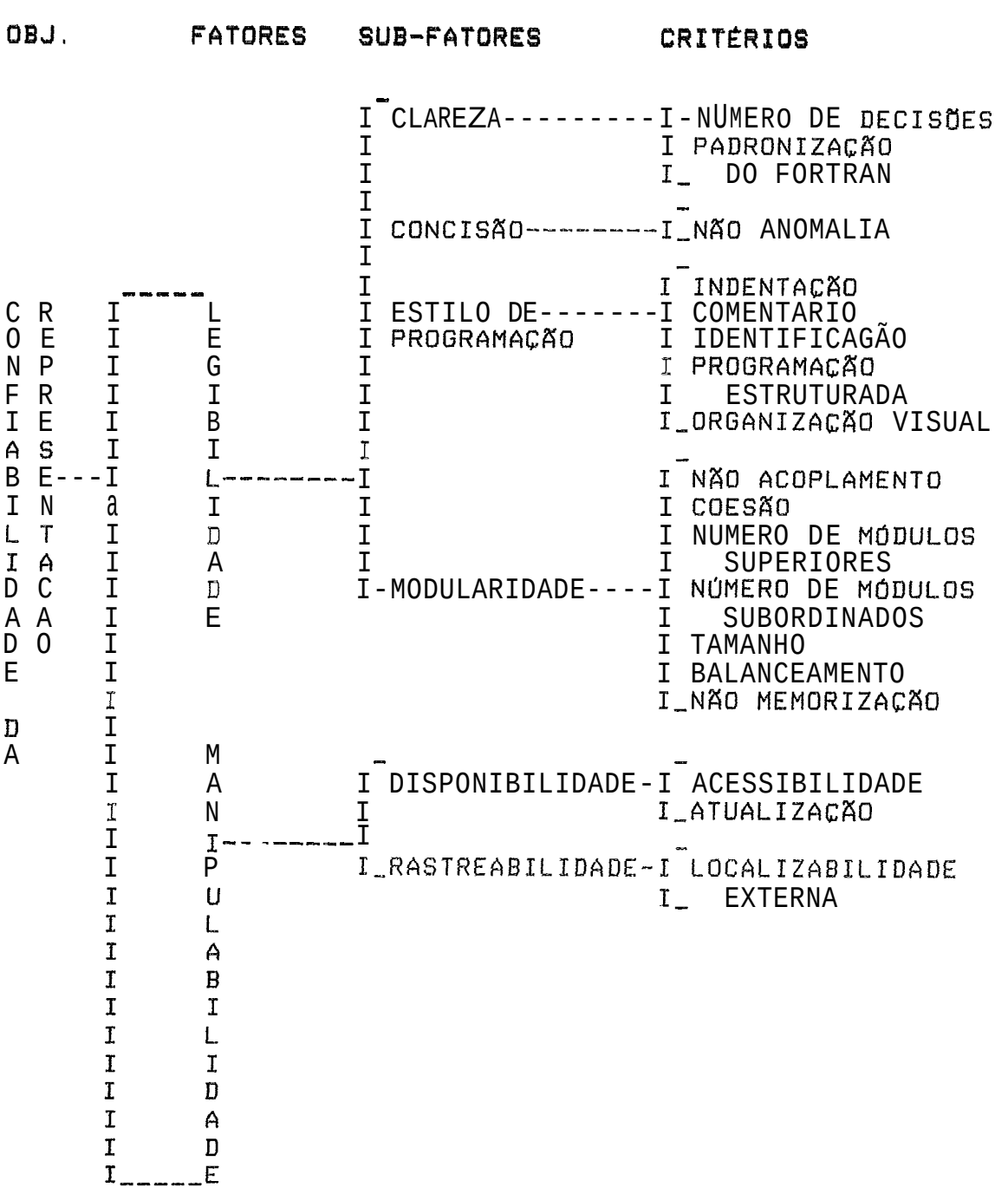


Fig. AII.1 - Classificação **das** características **de qualidade consideradas no** APFOR

11.2 - Funcionamento e Escopo

A partir da determinacão das características (fatores e respectivos sub-fatores) de qualidade que devem ser consideradas sobre um programa MS FORTRAN 77 ANÇI especifico , já compilado, o APFOR promove, inicialmente, a análise estática automática destas características (para os sub-fatore5 Clareza, Concisão, Estilo de Programação ou Modularidade com seus respectivos critérios de qualidade) diretamente sobre o código fonte do programa em questão, sem o envolvimento do usuário.

Após o sucesso deste procedimento, dependendo da característica (apenas para os sub-fatores Estilo de Programação, Disponibilidade ou Rastreabilidade) considerada, faz algumas perguntas ao usuário, tanto sobre o programa em questão como sobre o ambiente de trabalho em que o mesmo se encontra. Isto tudo para finalmente fornecer uma classificação para a5 características como ATENDIDA, NÃO ATENDIDA ou NÃO APLICAVEL, em relação ao programa em questão. Também fornece sugestões para procedimentos de revisão através de mensagens geradas nas devidas linhas do próprio código fonte do programa, e mensagens para modificações no ambiente de trabalho do mesmo.

A seguir, estão descritas todas as possíveis medidas envolvidas pelo APFOR, numeradas hierarquicamente de acordo com a classificação adotada, juntamente com sua respectiva mensagem(ns) de sugestão que pode(m) ser gerada(s) quando a medida for classificada como NÃO ATENDIDA. Assim, é apresentado o escopo de informações sobre as

características de qualidade que podem ser consideradaç pelo **APFOR,** tanto as de entrada (objetivo, fatores, sub-fatores, critérios e medidas de qualidade) como as de saída (mensagens de sugestão).

```
==========================================================
1 - CONFIABILIDADE DA REPRESENTAÇÃO:
==========================================================
1.1 - LEGIBILIDADE:
----------------------------------------------------------
1.1.1 - CLAREZA:
----------------------------------------------------------
1.1.1.1 - COMPLEXIDADE CICLOMÁTICA:

Medida: Complexidade ciclomática (limite maximo permitido
        para o numero de decisoes = 10)
Mensagem de sugestão:
)---) Devem ser simplificadas as condições.
----------------------------------------------------------
1.1.1.2 - PADRONIZAÇÃO DO FORTRAN:

Medida 1: Inexistência de erros de compilação

Medida 2: Inexistência de extensões próprias do Microsoft
          FORTRAN versão 4.0


1.1.2 - CONCISÃO:

1.1.2.1 - NÃO ANOMALIA:

Medida 1: Inexistência de subprograma não referenciado
Mensagem de sugestão:
)---) Deve ser retirado o subprograma não referenciado.

Medida 2: Inexistência de subprograma morto (sem comando
          executavel)
Mensagem de sugestão:
)---) Deve ser retirado o subprograma morto (sem comandos
      executáveis).

Medida 3: Inexistência de formato desnecessário
Mensagem de sugestão:
)---) Deve ser retirado o comando FORMAT não referenciado.
==========================================================
1.1.3 - ESTILO DE PROGRAMAÇÃO:
----------------------------------------------------------
1.1.3.1 - INDENTAÇÃO:

Medida 1: Existência de comando DO com bloco indentado
Mensagem de sugestão:
)---) Deve ser indentado o comando em relação ao seu
      comando DO.
```

Medida 2: Existência de comando IF com bloco da claúsula THEN ou ELSE indentado

Mensagem de sugestão:
)---) Deve ser indentado o comando em relação ao seu comando IF.

1.1.3.2 - COMENTARIO:

Medida 1: Existência de comentário de cabecalho para explicar a função do programa antes ou depois do comando PROGRAM

Mensagem de sugestão:
)---) Deve ser incluido um comentario tipo cabeçalho explicando a função de programa, no seu inicio, antes ou depois do comando PROGRAM.

Medida 2: Existência de comentário de subprograma para explicar sua função antes ou depois de sua declaração

Mensagem de sugestão:
)---) Deve haver um comentario tipo cabecalho explicando a função do subprosrama, no seu inicio, antes ou depois de sua declaração.

Medida 3: Existência de cabeçalho de manutencão padrão no inicio do programa

Mensagem de sugestão 1:
)---) Tambem não existe cabecalho de manutenção padrao (com indicacão sobre MANUTENCAO, RESPONSAVEL, DATA e MOTIVO) no seu inicio.
Caso já tenha ocorrido alguma manutenção no mesmo, inclua tambem este tipo de cabeçalho com as informações mais recentes.

Mensagem de sugestão 2:
)---) Nao existe cabecalho de manutenção padrao (com indicação sobre MANUTENCAO, RESPONÇAVEL, DATA e MOTIVO) no inicio do programa, antes ou depois do comando PROGRAM.
Caso já tenha ocorrido alguma manutencão no mesmo, inclua este tipo de cabeçalho com as informacões mais recentes.

Mensagem de sugestão 3:
)---) Existe um cabecalho de manutenção padrão (com indicação sobre MANUTENCAO, RESPONÇAVEL, DATA e MOTIVO) no inicio do programa, que está totalmente preenchido.
Deve ser verificado se o mesmo contém realmente as informações mais recentes, pois seu conteúdo não é analisado.

Mensagem de sugestão 4:
)---) Existe um cabeçalho de manutenção padrão (com indicação sobre MANUTENCAO, RESPONSAVEL, DATA e MOTIVO) no inicio do programa, que está parcialmente preenchido.

Devem ser devidamente incluídas no mesmo as informações mais recentes.

Medida 4: Inexistência de comentário, em linha de comando, depois da coluna 73
Mensagem de sugestão:
)---) Deve ser retirado o comentário colocado depois da coluna 73.

Medida 5: Existência de comentários
Mensagem de sugestão:
)---) Devem ser incluídos, ao longo do programa, comentários explicando seuç principais procedimentos.

Medida *6:* Existência de comentários claros e entendiveis
Mensagem de sugestão:
)---) Deve ser alterado o conteudo do bloco de comentários para que a informacão seja clara, simples e sem abreviação não significativa.

1.1.3.3 - **IDENTIFICAÇÃO:**

Medida: Existência de nomes de identificadores e variáveis significativos
Mensagem de sugestão:
)---) Devem ser substituídos os seguintes nomes referenciados ao longo do programa por nomes considerados significativos:
XXXXXX, XXXXX, XXXXXX, XXXXXJ XXXXXX

1.1.3.4- PROGRAMAÇÃO ESTRUTURADA:

Medida 1: Existência de subprograma com um único ponto de entrada (sem comando ENTRY)
Mensagem de sugestão:
)---) Deve ser retirado o comando ENTRY, para se ter um unico ponto de entrada no subprograma.

Medida 2: Existência de subprograma com um Único ponto de saída (sem mais de um comando RETURN)
Mensagem de sugestão:
)---) Deve haver apenas um comando RETURN no subprograma, para se ter um Único ponto de saida.

Medida 3: Inexistência de desvio para fora do comando DO, através de comando GO TO
Mensagem de sugestão:
)---) Deve ser retirado o comando GO TO.

Medida 4: Inexistência de desvio para fora do comande IF, através de comando GO TO
Mensagem de sugestão:
)---) Deve ser retirado o comando GO TO.

1.1.3.5 - ORGANIZAÇÃO VISUAL:

Medida 1: Existência de label ordenado em relação à ordem crescente do label anterior
Mensagem de sugestão:
)---) Deve ser ordenado o label em ordem crescente em relação ao label anterior.

Medida 2: Existência de comando alinhado em relação ao comando PROGRAM
Mensagem de sugestão:
)---) Deve ser alinhado o comando de acordo com a coluna do comande PROGRAM.

Medida 3: Existência de operador de expressão em comandos executáveis com espaço em branco antes e depois do mesmo
Mensagem de sugestão:
)---) Deve haver pelo menos um espaco em branco antes e depois do operador de expressão.

Medida 4: Existência de sinal = de atribuição com espaço em branco antes e depois do mesmo Mensagem de sugestão:
)---) Deve haver pelo menos um espaço em branco antes e depois do sinal = de atribuição.

Medida 5: Existência de comando de declaração com espaço em branco depois da vírgula separadora de variáveis
Mensagem de sugestão:
)---) Deve haver pelo menos um espaco em branco depois da vírgula separadora de variáveis.

Medida 6: Existência de bloco de comentários com linha em branco antes e depois do mesmo
Mensagem de sugestão:
)---) Beve haver pelo menos uma linha em branco antes e depois do bloco de comentários.

Medida 7: Existência de subprograma com linha em branco antes de sua declaração
Mensagem de sugestão:
)---) Deve haver pelo menos uma linha em branco antes do subprograma.

==========

1.1.4 - MODULARIDADE:

1.1.4.1 - NÃO ACOPLAMENTO:

Medida: Inexistência de subprograma com acoplamento por área comum (possui comando COMMON) ou com acoplamento por controle (possui argumento de entrada como flag de controle na condição de comando IF ou no incremento de comando DO)
Mensagem de sugestão 1:
)---) Deve ser retirada o comando COMMON do subprograma.

Mensagem de sugestão 2:
)---) Deve ser mudado o tipo de argumento de entrada que é um flag de controle.

1.1.4.2- COESSO:

Medida: Existência de subprograma com um Único ponta de entrada (sem comando ENTRY) ou com um único ponto de saída (sem mais de um eomando RETURN)
Mensagem de sugestão 1:
)---) Deve ser retirado o comando ENTRY, para se ter um Único ponto de entrada no subprograma.

Mensagem de sugestão 2:
)---) Deve haver apenas um comando RETURN no subprograma, para se ter um Único ponto de saida.

.....

1.1.4.3 - **NUMERO DE MODULOS SUPERIORES:**

Medida: Quantidade de unidades de programa que chamam subprograma (limite maximo de superiores = 3)
Mensagem de sugestão:
)---) Deve ser dividido o subprograma.

.....

1.1.4.4 - NUMERO DE MÓDULOS SUBORDINADOS:

Medida: Quantidade de subprogramas chamados por unidades de programa (limite maximo de superiores = 9)
Mensagem de sugestão:
)---) Deve ser diminuída as chamadas feitas pela unidade de programa.

1.1.4.5- TAMANHO:

Medida: Quantidade de linhas executáveis (limite maximo = 100)
Mensagem de sugestão:
)---) Deve ser diminuída ou dividida a unidade de programa.

1.1.4.6 - BALANCEAMENTO:

Medida: Inexistência de unidades de programa não de baixo nível com características físicas (possui comando de E/S formatada)
Mensagem de sugestão:
)---) Deve ser transferido o comando de E/S formatada do subprograma.

1.1.4.7 - **NÃO MEMORIZAÇÃO:**

Medida: Inexistência de subprograma com memória temporária (possui comando COMMON ou EQUIVALENCE)
Mensagem de sugestão 1:
)---) Beve ser retirado o comando COMMON do çubprograma.

Mensagem de sugestão 2:
›---> Deve ser retirado o comando EQUIVALENCE do subprograma.

==========

1.2 - MANIPULABILIDADE:

.....

1.2.1 - DISPONIBILIDADE:

.....

1.2.1.1 - ACESSIBILIDADE:

Medida: Armazenamento do programa em lugares designados para tal
Mensagem de sugestão 1:
›---› Devem ser estabelecidos lugares próprios para armazenar programas, como bibliotecas, diretorios, diskettes, pastas ou armarios especificoç. E nesses, armazenar este programa.

Mensagem de sugestão 2:
›---› Deve ser armazenado este programa nos lugares designados para tal.

1.2.1.2 - **ATUALIZAÇÃO:**

Medida: Atualização do programa nos lugares designados para seu armazenamento
Mensagem de sugestão:
›---› Deve ser armazenada a versão mais atual deste programa nos seus devidos lugares.

==========

1.2.2 - **RASTREABILIDADE:**

1.2.2.1 - **LOCALIZABILIDADE EXTERNA:**

Medida: Existência de Dicionário de dados ou outra documentação externa similar com informação atualizada sobre os interrelacionamentos deste programa
Mensagem de sugestão 1:
›---› Deve ser criado um Dicionário de Dados contendo para cada componente (como unidade de programa, arquivo, registro, campo, relatorio e tela) deste programa, a indicação dos respectivos componentes interrelacionados.

Mensagem de sugestão 2:
›---› Deve ser atualizado o Dicionário de Dados ou a documentação externa que contém os interrelaeionamentos de componentes do programa.

==========

==========

Ação: no caso de haver alguma resposta **Nao sei** fornecida pelo usuário durante a execução do APFOR

Mensagem de atenção usuário:

ATENCAO USUARIO:

ANTES DE ATIVAR ESTA FERRAMENTA NUMA PROXIMA VEZ, CONHECA MAIS O AMBIENTE DE TRABALHO ENVOLVIDO COM O PROGRAMA A SER SUBMETIDO 'A ANALISE EÇTATPCA, OBTENDO INFORMACOES SOBRE O **QUE** EXISTE OU NAQ.

==========

O conhecimento de t~das essas informações fornecidas serve tanto para apoio ao andamento e continuação do processo de avaliação da qualidade do programa como para uma rápida e simples verificação do programa. O conhecimento das sugestões forneridas, por sua vez, serve para conscientizar e usuário sobre a necessidade de aumento do nível de qualidade do programa. De uma maneira ou de outra, o intuito é ter-se na final um programa cem um bom nível de qualidade.

Assim, o APFOR destina-se tanto à um programador que deseja simplesmente localizar os pontos de seu programa MS FORTRAN 77 ANSI, que não atendem às medidas de qualidade como à um especialista em avaliação da qualidade que precisar de informações sobre o atendimento às características de qualidade para continuar a realização do processo de avaliação da qualidade e dar no final sua opinião sobre o nível de qualidade do programa em questão

II.3 - Ativação

Todos os passos que podem ocorrer no fluxo normal de ativação e durante uma execução do APFOR, sem erros de digitação por parte do usuário e sem faltas na instalação por parte do APFOR, são descriteç a seguir através de apresentações e ações de ambos as lados envolvidas, ou seja, do Usuário e do próprio APFOR, após sua instalação num microcomputador compatível com a linha IBM PC.

A sua instalação resume-se na criação de um diretório, o **C:\APFOR**, nas cópias do seu codigo executável, o **APFOR.EXE**, da seu arquivo de entrada **INICIA.DAT**, das arquives de ajuda HLP_XXX.TXT, e do programa a ser submetido à análise estática **XXXXXXXX.FOR** neste diretório criado e específica para a execução do APFOR.

EXECUÇÃO DO APFOR:

1- Usuário: Entrar no diretório C:\APFOR .

2- Usuário: Executar o APFOR, digitando: C:\APFOR-›**APFOR**

Dentro de cada tela do APFOR, utilizar as teclas de função, indicadas na última linha de cada tela, para **sua** correta e pretendida operação. Geralmente aparecem os seguintes:

F1 = Ajuda => para acessar as telas de Ajuda

F2 = Fim => para abortar, retornanda ao Sistema Operacional

‹ESC› ou **‹ENTER›** => para continuar ou cancelar

F4 = Examinar **programa** => para ver o código fonte do programa em questão com a5 linhas numeradas sequencialmente

F5 = Imprimir => para imprimir on-line a tela em questão

↓↑ = Percorrer texto => para percorrer o texto na tela de acordo com a direção da seta

3- APFOR: mostra a tela de abertura.

4- **Usuário**: teclar <ENTER> para continuar.

5- **APFQR:** moçtra a tela para entrada do nome do programa MS FORTRAN 77 ANSI a ser submetido à análise estática.

6- **Usuário:** digitar apenas o nome do programa XXXXXXXX desejado, a extensão .FOR já está indicada.

Fora do APFOR, este programa XXXXXXXX.FOR com as linhas numeradas sequencialmente pode ser acessado na arquivo C:\APFOR\LISPRG.LIN .

7- APFOR: mostra a tela para confirmar que o programa em questão está sem erros de compilação, sem caracteres de controle (como os de tabulação) e com seu último comando seguido de <ENTER> (cansequentemente, com pelo menos uma linha em branco no seu final).

8- **Usuário**: responder **Sim** ou **Nao** sobre a confirmação destes pré-requisitos.

9- **APFQR:** com a confirmação positiva dos pré-requisitos, mostra a tela para a escolha de como considerar as características de qualidade.

10- **Usuário:** responder **Todas as caracteristicas** ou **Apenas algumas caracteristicas,** conforme pretendido.

11- APFOR: considerando a respoçta **Todas as características,** mostra a tela com o objetivo, todos os fatores, respectivos sub-fatores e critérios de qualidade envolvidos na Classificação das características de qualidade. Depois vai para o passo 20.

Considerando a resposta de Apenas algumas características, moçtra a tela para a escolha de qual fator de qualidade deve considerar.

12- Usuário: pode responder **Legibilidade** e/ou Manipulabilidade, conforme pretendido.

23- APFQR: se a resposta for Legibilidade, mostra a tela para a escolha dos seus respectivos sub-fatores de qualidade sue, por sua vez, num nível hierárquico menor, deve considerar.

14- Usuário: pode responder Clareza, Concisao, Estilo **de** Programacao e/ou Medularidade. **Ao** terminar, escolher a opção Termina selecao para continuar a escolha de características.

15- APFOR: se a resposta for Manipulabilidade, mostra a tela para a escolha dos seus respectivos sub-fatores de qualidade que, par sua vez, num nível hierárquica menor, deve considerar.

16- Usuário: pode responder **Disponibilidade** e/ou **Rastreabilidade**. Ao terminar, escolher a opção Termina selecao para continuar a escolha das características.

17- APFOR: mostra novamente a tela para a escolha de qual fator de qualidade deve considerar, só que dessa vez já marcado.

18- Usuário: para terminar a escolha das características e continuar, escolher a opção Termina selecao.

19- APFOR: mostra a tela com o objetivo, fatores, respectivos sub-fatores e critérios de qualidade escolhidos.

20- **Usuário**: Após ver as caracteristicas que serão

consideradas pelo APFOR, teclar <ESC> para continuar.

Fora do APFOR, a relação destas características sendo consideradas pede ser acessada no arquivo C:\APFOR\CARACTER.IMP .

21- APFOR: mostra a tela EXECUTANDO A ANALISE ESTATICA PARA OS SUB-FATORES DE QUALIDADE REQUISITADOS SOBRE O PROGRAMA.

22- Usuário: aguardar.

23- APFOR: mostra a tela FINAL BEM SUCEDIDO DA ANALISE ESPATICA.

24- Usuário: aguardar.

Fora do APFOR, o relatório Resultados das medidas para análise estática sobre o programa em questão, resultante desta análise eçtática, pode ser acessado no arquivo C:\APFOR\ANALISE.RL1 .

25- APFOR: se não é considerada Todas as caracteristicas eu o sub-fator Estilo de Programacao, vai para o Passo 29. Se sim, mostra a tela com a pergunta sobre como estão escritas os comentários ao longo do programa em questão.

26- Usuário: responder Todos estao claros e entendiveis, Apenas alguns estao claros ou entendiveis ou Todos estao confusos.

Antes de responder, pode ver todo o código fonte do programa ao teclar F4, examinando, assim, os comentários para melhor poder dar sua opinião.

27- APFOR: se a resposta for Todos estao claros e entendiveis, vai para o passo 29. Se for Apenas alguns estao claros ou entendiveis ou Todos estao confusos, mestra a tela para entrada de linhas do programa que contenham comentário confuso ou não claro.

28- **Usuário**: digitar as linhas do programa em que achar o comentário confuso ou não claro. Ao terminar, teclar <ESC> para continuar.

Antes ou durante esta digitação, pode ver todo o código fonte do programa ao teclar F4, examinando, assim, os comentários para indicar corretamente a linha com tal característica.

29- **APFOR**: se não é considerada Todas as **caracteristicas** ou o sub-fator Estilo de Programacao, vai para o Passa **33.** Se sim, mostra a tela com a pergunta sobre como estão escritas as nomes dos identificadores e variáveis referenciados ao longo do programa em questão.

30- Usuário: responder Todos **sao significativos**, Apenas alguns sao significativos ou Nenhum **e'** significativa.

Antes de responder, pode ver todo o código fonte do programa ao teclar F4, examinando, assim, os nomes dos identificadores e variáveis para melhor poder dar sua opinião.

31- APFQR: se a resposta for Todos sao significativos, vai para o passo 33. Se for Apenas alguns sao significativos ou Nenhum e' significativo, mostra a tela para entrada de nomes de identificadores de programa que estejam não significativos.

32- **Usuário**: digitar os nomes do identificador ou variável que achar não significativo. Ao terminar, teclar (ESC) para continuar.

Antes ou durante esta digitação, pode ver todo o código fonte do programa ao teclar F4, examinando, assim, os identificaderes e variáveis para indicar corretamente o

nome com tal característica.

33- **APFOF?:** se não é considerado **Todas as caracteristicae** ou o sub-fator **Disponibilidade,** vai para a Passo 39. Se sim, mostra a tela com a pergunta sobre a existência de lugares designados para o armazenamento de programa.

34- **Usuário:** responder **Sim, existem, Nao existem** ou **Nao sei.**

35- **APFOR:** se a resposta for **Nao existem** ou **Nao sei,** vai para o passo 39. Se for **Sim, existem,** mostra **a** tela com a pergunta sobre como está o armarenamento do programa **em** questão nestes lugares.

36- Usuário: responder **Em todos os lugares, Em alguns desses lugares, Em nenhum desses lugares** ou **Nao sei.**

37- **APFOF?:** se a resposta for **Em nenhum desses lugares ou Nao sei,** vai para o passo **39.** Se for **Em todos os lugares** ou **Em alguns desses lugares,** mostra **a** tela com a pergunta çobre como estão atualizadas as cópias do programa em questão nestes lugares.

38- Usuário: responder **Todos estao na versao mais atual, Apenas algumas estao na versao mais atual, Nenhuma delas esta' na versao mais atual** ou **Nao sei.**

39- **APFOR:** se não é considerada **Todas as caracteristicas** ou o sub-fator **Rastreabilidade,** vai para o Passo **43.** Se sim, mostra a tela com a pergunta sobre a existência de Dicionário de Dados ou alguma outra documentação externa similar à este.

40- **Usuário:** responder **Sim, existe, Nao existe** ou **Nao sei.**

41- **APFOR: se** a resposta for **Nao existe** ou **Nao sei,** vai para o passo 43. Se for **Sim, existe,** mostra **a** tela com a

pergunta sobre como estão estes tipos de informação sobre os interrelacionamentos do programa.

42- **Usuário**: responder **Todos estao atualizados**, **Apenas** alguns estao atualizados, **Nenhum esta** atualizado ou **Nao sei**.

43- APFOR: mostra a tela **GERANDO OS RESULTADOS DAS CARACTERISTICAS DE QUALIDADE E AS SUGESTOES PARA REVISAO DO PROGRAMA XXXXXXXX.FOR**, ande XXXXXXXX.FOR é o nome do programa em questão.

44- Usuário: aguardar.

45- **APFOR**: mostra a Última tela com os três relatórios finais: Resultados para as características de qualidade sobre o programa; Sugestões para procedimentos de revisão sobre e código fonte do programa e Sugestões para procedimentos de revisão sobre o ambiente de trabalho de programa.

46- **Usuário**: verificar os resultados fornecidos pelo APFOR que podem estar contidos nos três relatúrios finais apresentados. Ao terminar, teclar ⟨ESC⟩ ou F3 para finalizar e retarnar ao Sistema Operacional C:\APFOR-⟩ .

Fora do **APFOR**, estes relatórios podem ser acessados no arquivo C:\APFOR\XXXXXXXX.RL2 , onde XXXXXXXX.FOR é o nome da programa em questão.

Tanto as mensagens que podem aparecer ao longo da execução do **APFOR**, como estes relatórios são abordados mais detalhadamente no5 capítulos a seguir.

II.4 - Ajuda

Exiçte a opção da função Ajuda no APFOR, indicada e acessada através da tecla F1, a fim de fornecer explicações sobre as seguintes telas:

- tela de abertura do APFOR: com a explicação do objetivo do APFOR;
- telas posteriores com pergunta: com explicações sobre os objetivos da pergunta corrente e descrições dos termos envolvidos pela mesma.

II.5 - Mensagens

Existem dois tipos de mensagens que podem aparecer na execução do APFOR:

1- Mensagens de pré-requisitos para execução do APFOR;

2- Mensagens de operação no APFOR.

1- Mensagens de pré-requisitos para execução do APFOR:

Estas mensagens aparecem durante a execução do APFOR, indicando algum pré-requisito necessário para continuar corretamente a execução do mesmo. Exceto as duas primeiras mensagens relacionadas abaixo, que permitem retorno para a continuação da execução do APFOR, as outras são abortivas, ou seja, cancelam a execução do APFOR e retornam ao Sistema Operacional C:\APFOR-) .

Os textos das mensagens desse tipo, em forma reduzida, pois o tamanho original e de 77 colunas, são os seguintes na ordem em que podem aparecer no APFOR:

LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

O APFOR precisa de um programa MS FORTRAN 77 ANSI para analisar.

LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

O APFOR precisa de um programa MS FORTRAN 77 ANSI para analisar.
Este programa XXXXXXXX.FOR nao consta do diretorio C:\APFOR\ .

PRE-REQUISITO PARA EXECUCAO DO APFOR:

Compile este programa MS FORTRAN 77 ANSI, localizado em C:\APFOR com o nome de XXXXXXXX.FOR no ambiente MS FORTRAN versao 4.0, retirando os erros de compilacao indicados, os caracteres de controle que por acaso existam e, se necessario, tecle <ENTER> no ultimo comando do programa.

Nao se preocupe com as mensagens de alerta que aparecerem, pois elas sas ignoradas pelo APFOR.

Entao, ative novamente o APFOR.

LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: Nao consegui abrir o arquivo com impressao das caracteristicas
de qualidade C:\APFOR\CARACTER.IMP .

ACAO: Contacte o responsavei pela instalacao do APFOR.

==========

LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: Nao consegui abrir o arquivo com programa e linhas
numeradas C:\APFOR\XXXXXXXX.LIN .

ACAO: Contacte o responsavel pela instalacao do APFOR.

==========

```
==========================================================================

                    LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: Nao consegui abrir o arquivo de inicializacao com os limites
      permitidos C:\APFOR\INICIA.DAT .

ACAO: Contacte o responsavel pela instalacao do APFOR.

==========================================================================

                    LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: Nao consegui abrir o programa  MS FORTRAN 77 ANSI C:\APFOR\
      XXXXXXXX.FOR a ser submetido a analise estatica pelo APFOR.

ACAO: Copie o programa desejado para o diretorio C:\APFOR e ative
      novamente o APFOR.

--------------------------------------------------------------------------

                    LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: Espaco insuficiente no disco.

ACAO: Organize seu winchester, apagando arquivos desnecessarios, a
      fim de ter espaco disponivel para execucao do APFOR.

--------------------------------------------------------------------------

                    LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: O programa C:\APFOR\XXXXXXXX.FOR nao esta' no padrao  MS
      FORTRAN 77 ANSI, pois possui metacomandos proprios do MS
      FORTRAN Versao 4.0 .

ACAO: Retire os metacomandos do  C:\APFOR\XXXXXXXX.FOR, compile-o
      novamente, retirando os erros de compilacao indicados e ati-
      ve novamente o APFOR com esta nova versao do programa.

==========================================================================
```

==========

LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: O programa C:\APFOR\XXXXXXXX.FOR nao esta no padrao MS FORTRAN 77 ANSI, pois possui comandos de declaracao com *n proprios do MS FORTRAN Versao 4.0

ACAO: Retire esses comandos de declaracao com o *n do C:\APFOR\XXXXXXXX.FOR, compile-o novamente, retirando os erros de compilacao indicados e ative novamente o APFOR, com esta nova versao do programa.

==========

LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: Nao consegui criar o arquivo de relatorio da analise estatica C:\APFOR\ANALISE.RL1 .

ACAO: Contacte o responsavel pela instalacao do APFOR.

LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: Nao consegui criar o arquivo de linhas para revisao C:\APFOR\LINREV.CDF .

ACAO: Contacte o responsavel pela instalacao do APFOR.

==========

LEIA ATENTAMENTE AS INSTRUCOES ABAIXO

ERRO: Nao consegui abrir o arquivo de resultados da analise estatica C:\APFOR\ANALISE.CDF .

ACAO: Contacte o responsavel pela instalacao do APFOR.

==========

```
==========================================================================
ERRO: Valor invalido nos parametros de entrada da interface no APFOR.
ACAO: Contacte o responsavel pela instalacao do APFOR.
==========================================================================
```

2- Mensagens de operação na APFOR:

Estas mensagens aparecem durante a execução do APFOR, informando sobre algum tips de processamento interno demorado, daí a razão de sua existência, ou indicando alguma critica aos dados de entrada fornecidos on-line pelo usuário.

Os textos das mensagens desse tipo, em forma reduzida, pois o tamanho original é de 77 colunas, são os seguintes na ordem em que pedem aparecer no APFOR:

```
==========================================================================
Nao pode sair do menu sem escolher uma opcao. tecle <ENTER> = Continuar
==========================================================================
EXECUTANDO A ANALISE ESTATTCA PARA OS SUB-FATORES DE QUALIDADE
REQUISITADOS SOBRE O PROGRAMA XXXXXXXX.FOR .
==========================================================================
FINAL BEM SUCEDIDO DA ANALISE ESTATICA.
==========================================================================
ERRO: Alguma linha deve ser digitada.
      Senao, com as linhas em branco, tecle <ESC> = Cancelar
==========================================================================
ERRO: Algum identificador deve ser digitado.
      Senao, com os identificadores em branco, tecle <ESC> = Cancelar
==========================================================================
```

```
==========================================================================

ERRO: Linha Inexistente, tecle <ENTER> = Redigitar

==========================================================================

GERANDO OS RESULTADOS DAS CARACTERISTICAS DE QUALIDADE E
AS SUGESTOES PARA REVISAO DO PROGRAMA XXXXXXXX.FOR .

==========================================================================
```

II.6 - Relatórios

Existem quatro tipos de relatórios que podem ser gerados na execução do RPFOR:

1- Resultados sobre as Medidas da Análise Estática no Programa;

2- Resultados sobre o Atendimento das Características de Qualidade;

3- Indicações sobre as Medidas com Sugestões para Revisão no programa;

4- Sugestões para Revisão no Ambiente de Trabalho do Programa

1- Resultados sobre as Medidas da Análise Estática no Programa:

E gerado pelo APFOR quando a análise estática realizada sobre o programa em questão, de acordo com as características de qualidade requisitadas e consideradas, obtém um final bem sucedido. Só não é gerado se apenas o fator de qualidade Manipulabilidade for o único a ser considerado.

E um relatório intermediário com informações básicas para a geração de relatórios finais do APFOR.

Indica os resultados obtidos para cada respectiva medida

subordinada ao critério e ao sub-fator de qualidade sendo considerado, que são as próprias características de qualidade.

Só pode ser acessado depois e fora da execução do APFOR através do arquivo C:\APFOR\ANALISE.RL1 .

2- Resultados sobre o Atendimento das Características de Qualidade:

E gerado pelo APFOR como seu primeiro relatório final.

Indica a classificação em um dos três tipos de resultado (CARACTERÍSTICAS ATENDIDAS, CARACTERÍSTICAS NÃO ATENDIDAS e CARACTERÍSTICAS NÃO APLICÁVEIS) para cada característica de qualidade (objetivo, fator, sub-fator e critério de qualidade) sendo considerada.

E apresentado na última tela do APFOR. Fora desse, pode ser acessado através do arquivo C:\APFOR\XXXXXXXX.RL2, onde XXXXXXXX.FOR é o nome do programa MS FORTRAN 77 ANSP em questão

3- Indicações sobre as Medidas com Sugestões para Revisão no programa:

Pode ser gerado pelo APFOR como seu segundo relatório final, quando houver alguma característica com classificação NAO ATENDIDA, indicada no relatório anterior de Resultados sobre o Atendimento das Características de Qualidade, exceto às referente ao fator Manipulabilidade.

Indica sugestões sobre o código fonte do programa, através de uma mensagem, numa linha imediatamente seguinte à linha em que foi detectado o respectivo não atendimento, que por sua vez origina-se das informações contidas no

relatório Resultados das Medidas para Analise Estática no Programa e de perguntas feitas ao usuário quando necessário. Assim, estas devem ser realizadas a fim de atender à característica de qualidade classificada como NÃO ATENDIDA, e consequentemente, melhorar o nível de qualidade do programa em questão.

E apresentado na Última tela do APFOR. Fora desse, pode ser acessado através do arquivo C:\APFOR\XXXXXXXX.RL2, onde XXXXXXXX.FOR é o nome da programa MS FORTRAN 77 ANSI em questão.

4- Sugestões para Revisão no Ambiente de Trabalho do Programa:

Pode ser gerado pelo APFOR como seu terceiro e último relatório final, quando um dos critérios Acessibilidade ou Atualização do sub-fator Disponibilidade, ou então do critério Localizabilidade Externa do sub-fator Rastreabilidade (ambos subordinados ao fator Manipulabilidade) for classificada como NÃO ATENDIDA no relatório anterior de Resultados sobre o Atendimento das Características de Qualidade.

Indica sugestões sobre o ambiente de trabalho do programa em questão, através de uma mensagem, que origina-se das respostas obtidas de perguntas feitas ao usuário quando necessário. Assim, estas devem ser realizadas a fim de atender ao fator Manipulabilidade e seus respectivos sub-fatores e critérios classificados como NÃO ATENDIDA, e eonsequentemente, melhorar o nível de qualidade do programa em questão.

E apresentado na Última tela do APFOR. Fora desse, pode ser acessado através do arquivo C:\APFOR\XXXXXXXX.RL2, onde XXXXXXXX.FOR e o nome do programa MS FORTRAN 77 ANSI em questão.

II.7 - Exemplo de uma execução

05 passos ocorridos no fluxo normal de ativação e execução do APFOR para um programa exemplo, sem erros de digitação por parte do usuário e sem faltas na instalação por parte do APFOR, são descritos a seguir através de apresentações e ações de ambos os lados envolvidos, ou seja, do Usuário e do próprio APFOR.

EXECUÇÃO DO APFOR:

1- **Usuário:** Entrar no diretório C:\APFOR .

2- **Usuário:** Executar o APFOR, digitando: C:\APFOR->**APFOR** .

Dentro de cada tela do APFOR, utilizar as teclas de função, indicadas na última linha de cada tela, para sua correta operação.

3- **APFQR:** mostra a tela de abertura.

4- **Usuário:** teclar <ENTER> para continuar.

5- **APFOR:** mestra a tela para entrada do nome do programa MS FORTRAN 77 ANSP a ser submetido à análise estática.

6- **Usuário:** digitar **EXPTES**

Fora do APFOR, este programa EXPTES.FOR com as linhas numeradas sequencialmente pode ser acessado no arquivo C:\APFOR\LISPRG.LIN . Ele está na figura AII.2 .

7- APFOR: mostra a tela para confirmar que o programa exemplo C:\APFOR\EXPTES.FOR está sem erros de compilação,

sem caraeteres de controle (como os de tabulação) e com seu Último comando seguido de ⟨ENTER⟩ (consequentemente, com pelo menos uma linha em branco no seu final).

8- **Usuário:** responder **Sim** sobre a confirmação destes pré-requisitos.

9- **APFOR:** mostra a tela para a escolha de como considerar as características de qualidade.

10- **Usuário:** responder **Todas as características**

11- **APFOR:** mostra a tela com o objetivo, todos os fatores, respectivos sub-fatores e critérios de qualidade envolvidoç na Classificação das características de qualidade.

12- **Usuário:** Após v as características que serão consideradas pelo APFOR, teclar ⟨ESC⟩ para continuar.

Fora do APFOR, a relação destas características sendo consideradas pode ser acessada no arquiva C:\APFOR\CARACTER.IMP.

13- **APFOR:** mostra a tela **EXECUTANDO A ANALISE ESTATICA PARA OS SUB-FATORES DE QUALIDADE REQUISITADOS SOBRE O PROGRAMA.**

14- **Usuário:** aguardar.

15- **APFOR:** mostra a tela **FINAL BEM SUCEDIDO DA ANALISE ESTATICA**.

16- **Usuário:** aguardar.

Fora do APFOR, o relatório Resultados das medidas para análise estática sobre o programa exemplo, resultante desta análise estática, pode ser acessado no arquivo C:\APFOR\ANALISE.RL1 . Ele está, de forma reduzida, na figura AII.3 .

17- **APFOR:** mostra a tela com a pergunta sobre como estão escritos os comentários ao longo do programa exemplo.

18- Usuário: responder **Apenas alguns estao claros ou entendiveis.**

19- **APFOR:** mostra a tela para entrada de linhas do programa que contenham comentario confuso ou não claro.

20- **Usuário:** digitar a linha 26 e teclar <ESC> para continuar.

21- **APFOR:** mostra a tela com a pergunta sobre coma estão escritos os nomes dos identificadores e variaveis referenciados ao longo do programa.

22- **Usuário:** responder **Apenas alguns sao significativos.**

23- APFOR: mostra a tela para entrada de nomes de identificadores do programa que estejam não significativos.

24- Usuário: digitar os nomes **SOMA1** e **SOMA2**, e teclar <ESC> para continuar.

25- **APFOR:** mostra a tela com a pergunta sobre a existência de lugares designados para o armazenamento do programa.

26- Usuário: responder **Sim, existem.**

27- **APFOR:** mostra **a** tela com a pergunta sobre como está o armazenamento do programa exemplo nestes lugares.

28- **Usuário:** responder **Em todos os lugares.**

29- APFOR: mostra a tela com a pergunta sobre como está o armazenamento do programa nestes lugares.

30- **Usuário:** responder **Apenas algumas estao na versao mais atual.**

31- **APFOR:** mostra a tela com a pergunta sobre a existência Dicionário de Dados ou alguma outra documentação externa similar à este.

32- Usuário: responder **Sim, existe.**

33- APFOR: mostra **a** tela com a pergunta sobre como estão

estes tipos de informação sobre os interrelacionamentos do programa.

34- **Usuário:** responder **Nao sei.**

35- **APFOR:** mostra a tela **GERANDO OS RESULTADOS DAS CARACTERISTICAS DE QUALIDADE E AS SUGESTOES PARA REVISAO DO PROGRAMA EXPTES .FOR** .

36- **Usuário:** aguardar.

37- **APFOR:** moçtra a Última tela com os três relatórios finais: Resultados para as características de qualidade sobre o programa; Sugestões para procedimentos de revisas sobre o código fonte do programa e Sugestões para procedimentos de revisão sobre o ambiente de trabalho do programa.

38- **Usuário:** verificar os resultados fornecidos pela APFOR que podem estar contidas nos três relatórios finais apresentados. Ao terminar, teclar <ESC> ou F3 para finalizar *e* retornar ao Sistema Operacional C:\APFOR-> .

Fora do APFOR, estes relatórios podem ser acessados no arquivo C:\APFOR\EXPTES.RL2 . Eles estão, de forma reduzida, nas figuras AII.4, AII.5 e AII.6 .

```
LINI CODIGO FONTE
.....................................................
  1|        PROGRAM PPEXP
  2|        EXTERNAL SPNH, COSH, TANH
  3|        INTEGER SOMA1
  4|        REAL ANGULO
  5|        BIMENSEON X(10),Y(10)
  6|
  7|* FUNCAO 1: CALCULO DOS VALORES DA TANGENTE, SENO E COS
ENO PARA ANGULO
  8|
  9|        PRINT *, 'Entre com o numero +para ser calculad
o= o valor da
 10|     *tangente, seno ou coseno, ou entae CTRL Z para sa
ir'
 11|
 12|        READ *,ANGULO
 13|
 14|        TANHX = TANH(ANGULO)
 15|          SINHX = SINH(ANGULO)
 16|        COSHX = COSH(ANGULO)
 17|
 18|        WRITE(*,100) TANHX, SINHX, COSHX
                  IMP...
 19|
 201108     FORMAT("1VALORES ENCONTRADOS PARA TAN-SIN-COS =
",3F3.0)
 21|
 221120     FORMAT(//1X,'VALORES ENCONTRADOS:')
 23|
 24|10      CONTINUE
 25|
 26|* FUNCAO 2: FAZ O SOMATOWIO DOS NUMEROS INVERTIDOS DE 2
0 A 30
 27|
 28|        CALL CALCULO(30,SOMA1)
 29|
 30|* FUNCAO 3:
 31|
 32|        READ (4,REC = 47)(X(IND1), IND1 = 1,10)
 33|        READ (4,REC=47)(Y(IND2), IND2 = 1,10)
 34|
 35|        PWODXY = 0.0
 36|
 37|        PRLNT *,'VALOR INTERMEDIARIO =
 38|        DO 130 IND1=1,10
 39|
 40|           IF((X(IND1) .EQ. 0) .AND. (Y(IND1) .EQ. 0))
THEN
 41|              GO TO 130
 42|           ELSE
 43|                PRODXY = PRODXY+X(IND1)*Y(IND2)
 44|           END IF
 45|
 46|              PRINT *,PRODXY
```

Fig. AII.2 - Programa MS FORTRAN 77 ANSI exemplo com as linhas numeradas

```
LINI CODIGO FONTE
-----------------------------------------------------------------
 47|
 481130      CONTINUE
 49|
 50|
 51|         STOP
 52|
 531         END
                        FIM.
 54|
 55|C
 56|C CALCULO DA FUNCAO SENO SINH, COSENO COSH E TANGENTE T
ANH
 57|C
 58|
 59|         REAL FUNCTION TANH(X)
 60|C
 61|C FUNCAQ DE COMANDO PARA CALCULAR DUAS VEZES 0 SENO SIN
H
 62|C
 63|         DVSINH(X) = EXP(X) - EXP(-X)
 64|C
 65|C FUNCAO DE COMANDO PARA CALCULAR DUAS VEZES A COSH
 66|         DVCOSH(X) = EXP(X) + EXP( - X)
 67|
 68|            TANH = DVSINH(X)/DVCOSH(X)
 69|         RETUHN
 70|
 71|C
 72|C CALCULO DO SENO SINH
 73|         ENTRY SINH(X)
 74|         SINH = DVSINH(X)/2.0
 75|         RETURN
 76|C
 77|C CALCULO DO COÇH
 78|C
 79|         ENTRY COSH(X)
 80|         COSH = DVCOSH(X)/2.0
 811         RETURN
 82|         END
 83|
 84|C
 85|C EXEMPLO DE ÇUBROTINA DUMMY, OU SEJA, MORTA (SEM COMAN
DO EXECUTAVEL)
 86|         SUBROUTLNE DUMMY
 87|
 88|         RETURN
 891         END
 90|C EXEMPLO DE ÇUBROTINA NAO REFERENCIAUA
 91|         ÇUBROUTINE NCHAM(A,B,C)
 92|         CHARACTER M1*18, M2*10
 93|         EQUIVALENCE (M1, M2)
 94|         INTEGER A,B,C
 95|
```

Fig. A11.2 - Programa MS FORTRAN 77 ANSI exemplo com as linhas numeradas (continuação)

```
LINI CODPGO FONTE

 96|         C = (A*4) + B
 97|
 98|         RETURN
 98|         END
1001
101|*
102|* EXEMPLO DE SUBROTINA NAO DE BAIXO NIVEL
1031% FAZ O SOMATORIO DE UM CALCULO COM O NUMERO INVERTIDO
DO LIMITE ESPECIFICADO
104|*
105|         SUBROUTINE CALCULO(CONTR,RESULT)
106|         INTEGER RESULT, CONTR
107|
108|         DO 500 I = 20, CONTR
109|
110|            RESULT = RESULT + INVERT(I)
111|
112|500      CONTINUE
113|
114|         WRLTE (*,550)RESULT
115|550      FORMAT(I5)
116|
117|         RETURN
118|         END
119|
120|         FUNCTTON INVERT(NUM)
121|         COMMON TEMP
122|
123|         INTEGER DIG1,DIG2
124|
125|         IF (NUM .LT. 10 .OR. NUM .GE. 100) THEN
126|            PRINT *, 'ERRO NO ARGUMENTO', NUM
127|            GO TO 600
128|         END IF
129|
130|         DIG1 = NUM /10
131|         DIG2 = MOD(NUM,10)
132|         TEMP = DIG2 H 10
133|
134|* RETORNA UM NUMERO INTEIRO DE DOIS DIGITOS EM ORDEM IN
VERSA AO "NUM"
135|* DE ENTRADA
1361
137|         INVERT = TEMP +DIG1
1381600     RETURN
139|         END
140|
141|
```

Fig. AII.2 – Programa MS FORTRAN 77 ANSI exemplo com as linhas numeradas (continuação)

```
-------------------------------------------------------------------------------
          RESULTADOS SOBRE AS MEDIDAS DA ANALISE ESTATICA NO PROGRAMA
                          EXPTES  .FOR EM  26/02/91
                                       'AS  20:22 HS
-------------------------------------------------------------------------------

                        ************************************
                        * ANALISE PARA O SUB-FATOR CLAREZA *
                        ************************************

CRITERIQ: NUMERO DE DECISOES

   MEDIDA: COMPLEXIDADE CICLOMATICA (limite maximo permitido para o numero de
   -------  decisoes = 10)

   -> TODAS AS UNIDADES DE PROGRAMA NAO ULTRAPASSAM ESTE LIMITE PERMITIDO

-------------------------------------------------------------------------------

                        ************************************
                        * ANALISE PARA O SUB-FATOR CONCISAO *
                        ************************************

CRITERIO: NAO ANOMALIA
.....................

   MEDIDA 1: INEXISTENCIA DE SUBPROGRAMA NAO REFERENCIADO

      SUBPROÇRAMA NAO REFERENCIADO                    LINHA

      DUMMY                                           86

      NCHAM                                           91

   MEDIDA 2: INEXISTENCIA DE SUBPROGRAMA MORTO (sem comando executavel)

      SUBPROGRAMA MORTO                               LINHA

      DUMMY                                           86

   MEDIDA 3: INEXISTENCIA DE FORMATO DESNECESSARIO

      LABEL DO COMANDO FORMAT DESNECESSARIO           LINHA

      120                                             22

-------------------------------------------------------------------------------
```

Fiy. ATI.3 - Relatório dos Resultados sobre as Medidas da Análise Estática na Programa exemplo

```
            ************************************************
            * ANALISE PARA O SUB-FATOR ESTILO DE PROGRAMACAO *
            ************************************************

CRITERIO 1: INDENTACAO

   MEDIDA 1: EXISTENCIA DE COMANDO DO COM BLOCO INDENTADO

      LINHA COM COMANDO NAO INDENTADO

      46

   MEDIDA 2: EXISTENCIA DE COMANDO IF COM BLOCO DA CLAUSULA THEN OU ELSE
             INDENTADO

      LINHA COM COMANDO NAO INDENTADO

      43

CRITERIO 2: COMENTARIO
.....

   MEDIDA 1: EXISTENCI  DE COMENTARI  DE C      PARA      I A FUNCAO DO
             PROGRAMA ANTES O  EPOI  DO CC LU   PROGRAM  C  )

   -> NAO EXISTE COMENTMIO DE CABECALHO

   MEDIDA 2: EXISTENCIA DE COMENTARIO DE SUBPROGRAMA PARA EXPLICAR SUA FUNCAO
             ANTES OU DEPOIS DE SUA DECLARACAO

      SUBPROGRAMA SEM COMENTARIO                    LINHA

      INVERT                                        120

   MEDIDA 3: EXISTENCIA DE CABECALHO DE MANUTENCAO PADRAO NO INICIO DO PROGRAMA

   -> NAO EXISTE TAL CABECALHO DE MANUTENCAO PADRAO

   MEDIDA 4: INEXISTENCIA DE COMENTARIO,EM LINHA DE COMANDO,DEPOIS DA COLUNA 73

      LINHA COM COMENTARIO DEPOIS DA COLUNA 73

      18

      53
```

Fig. AII.3 - Relatório dos Resultadas sobre as Medidas da Análise Estática no Programa exemplo (continuação)

```
CRITERIO 3: PROGRAMACAO ESTRUTURADA
...................................

   MEDIDA 1: EXISTENCIA DE SUBPROGRAMA COM UM UNICO PONTO DE ENTRADA (sem
             comando ENTRY)

      SUBPROGRAMA                                   LINHA COM COMANDO ENTRY

      TANH                                          73

      TANH                                          79

   MEDIDA 2: EXISTENCIA DE SUBPROGRAMA COM UM UNICO PONTO DE SAIDA (sem mais de
             um comando RETURN)

      SUBPROGRAMA COM MAIS DE UM COMANDO RETURN     LINHA

      TANH                                          59

   MEDIDA 3: INEXISTENCIA DE DESVIO PARA FORA DO COMANDO DO, A T W E S DE COMANDO
             GO TO

   -> TODOS OS COMANDOS DO NAO POSSUEM COMANDO GO TO DESVIANDO PARA FORA

   MEDIDA 4: INEXISTENCIA DE DESVIO PARA FORA DO COMANDO IF, ATRAVES DE COMANDO
             GO TO

      LINHA COM COMANDO GO TO

      41

      127

CRITERIO 4: ORGANIZACAO VISUAL

   MEDIDA 1: EXISTENCIA DE LABEL ORDENADO EM RELACAO `A ORDEM CRESCENTE DO
   --------  LABEL ANTERIOR

      LABEL DESORDENADO                             LINHA

      10                                            24

   MEDIDA 2: EXISTENCIA DE COMANDO ALINHADO EM RELACAO AO COMANDO PROGRAM

      LINHA COM COMANDO DESALINHADO

      15

      68
```

Fig. **AII.3** – Relatório dos **Resultados** sobre **as Medidas da** Análise Estática no Programa **exemplo** (continuação)

MEDIDA 3: EXISTENCIA DE OPERADOR DE EXPRESSA0 EM COMANDOS EXECUTAVEIS COM ESPACO EM BRANCO ANTES E DEPOIS DO MESMO

LINHA COM OPERADOR SEM ESPACO EM BRANCO ANTES OU DEPOIS

43

63

68

74

80

96

130

137

MEDIDA 4: EXISTENCIA DE SINAL = DE ATRIBUICAO COM ESPACO EM BRANCO ANTES E DEPOIS DO MESMO

LINHA COM SINAL = DE ATRIBUICAO SEM ESPACO EM BRANCO ANTES OU DEPOIS

33

38

MEDIDA 5: EXISTENCIA DE COMANDO DE DECLARACAO COM ESPACO EM BRANCO DEPOIS DA VIRGULA SEPARADORA DE VARIAVEIS

LINHA SEM ESPACO EM BRANCO DEPOIS DA VIRGULA SEPARADORA

5

94

123

MEDIDA 6: EXISTENCIA DE BLOCO DE COMENTARIOS COM LINHA EM BRANCO ANTES E DEPOIS DO MESMO

LINHA FINAL DO COMENTARIO SEM LINHA EM BRANCO ANTES OU DEPOIS

65

72

85

Fig. AII.3 - Relatório dos Resultados sobre as Medidas da Análise Estática no Programa exemplo (continuação)

```
      90

   MEDIDA 7: EXISTENCIA DE SUBPROGRAMA COM LINHA EM BRANCO ANTES DE SUA
   --------  DECLARACAO

      SUBPROGRAMA SEM LINHA EM BRANCO ANTERIOR      LINHA

      NCHAM                                         91

-------------------------------------------------------------------------------

                  ****************************************
                  * ANALISE PARA O SUB-FATOR MODULARIDADE *
                  ****************************************

CRITERIO 1: NAO ACOPLAMENTO

   MEDIDA: INEXISTENCIA DE SUBPROGRAMA COM ACOPLAMENTO COMUM (possui comando
   ------  COMMON) OU COM ACOPLAMENTO POR CONTROLE (possui argumento de controle
           na condicao de comando IF ou no incremento de comando DO)

      SUBPROGRAMA COM COMMON                        LINHA

      INVERT                                        120

      SUBPROGRAMA COM ARGUMENTO DE CONTROLE         LINHA

      CALCULO                                       105

      INVERT                                        120

CRITERIO 2: COESAO
------------------

   MEDIDA: EXISTENCIA DE SUBPROGRAMA COM UM UNICO PONTO DE ENTRADA (Sem comando
   ------  ENTRY) OU COM UM UNICO PONTO DE SAIDA (sem mais de um comando RETURN)

      SUBPROGRAMA                                   LINHA DO COMANDO ENTRY

      TANH                                          73

      TANH                                          79

      SUBPROGRAMA COM MAIS DE UM COMANDO RETURN     LINHA

      TANH                                          59
```

Fig. AII.3 - **Relatório dos Resultados** sobre a5 **Medidas da** Análise Estática no Programa exemplo (continuação)

```
CRITERIO 3: NUMERO DE MODULOS SUPERIORES
.....................................

   MEDIDA: QUANTIDADE DE UNIDADES DE PROGRAMA QUE CHAMAM SUBPROGRAMA
   ------  (limite maximo de superiores = 3)

   -> TODOS OS SUBPROGRAMAS NAO ULTRAPASSAM ESTE LIMITE PERMITIDO


CRITERIO 4: NUMERO DE MODULOS SUBORDINADOS
.....................................

   MEDIDA: QUANTIDADE DE SUBPROGRAMAS CHAMADOS POR UNIDADE DE PROGRAMA
   ------  (limite maximo de superiores = 9)

   -> TODOS AS UNIDADES DE PROGRAMA NAO ULTRAPASSAM ESTE LIMITE PERMITIDO


CRITERIO 5: TAMANHO
-------------------

   MEDIDA: QUANTIDADE DE LINHAS EXECUTAVEIS (limite maximo = 100)

   -> TODAS AS UNIDADES DE PROGRAMA NAO ULTRAPASSAM ESTE LIMITE PERMITIDO


CRITERIO 6: BALANCEAMENTO
.........................

   MEDIDA: INEXISTENCIA DE UNIDADES DE PROGRAMA NAO DE BAIXO NIVEL COM
   ------  CARACTERISTICAS FISIÇAS (possui comando de E/S FORMATADA)

      UNIDADE DE PROGRAMA NAO DE BAIXO NIVEL        LINHA
      COM COMANDO DE E/S FORMATADA

      CALCULO                                       105


CRITERIO 7: NAO MEMORIZACAO
...........................

   MEDIDA: INEXISTENCIA DE SUBPROGRAMA COM MEMORIA TEMPORARIA
   ------  (possui comando COMMON ou EQUIVALENCE)

      SUBPROGRAMA COM COMANDO COMMON                LINHA

      INVERT                                        120

      SUBPROGRAMA COM COMANDO EQUIVALENCE           LINHA

      NCHAM                                         91

--------------------------------------------------------------------------------
```

Fig. AEI.3 – Relatório dos **Resultados** sobre as **Medidas da Análise Estática no Programa** exemplo (continuação)

```
-------------------------------------------------------------------------
       RESULTADOS SOBRE O ATENDIMENTO DAS CARACTERISTICAS DE QUALIDADE
                        EXPTES .FOREM  26/02/91
                                'AS  20:23 HS
-------------------------------------------------------------------------

CARACTERISTICAS ATENDIDAS:

      SUB-FATOR:    CLAREZA
         CRPTERIQS:
                    NUMERO DE DECISOES
                    PADRONIZACAO DO FORTRAN

      SUB-FATOR:    MODULARIDADE
         CRITERIOS:
                    NUMERO DE MODULOS SUPERIORES
                    NUMERO DE MODULOS SUBORDINADOS
                    TAMANHO

      SUB-FATOR:    DISPONIBILIDADE
         CRITERIOS:
                    ACESSIBILIDADE

.............................
CARACTERISTICAS NAO ATENDIDAS:
.............................

OBJETIVO:           CONFIWILIDA.DE DA REPRESENTACAO
   FATOR:           LEGIBILIDADE
      SUB-FATOR:    CONCISAO
         CRITERIO:
                    NAO ANOMALIA
      SUB-FATOR:    ESTILO DE PROGRAMACAO
         CRITERIOS:
                    INDENTACAO
                    COMENTARIO
                    IDENTIFICACAO
                    PROGRAMACAO ESTRUTURADA
                    ORGANIZACAO VISUAL
      SUB-FATOR:    MODULARIDADE
         CRITERIOS:
                    NAO ACOPLAMENTO
                    COESAO
                    BALANCEAMENTO
                    NAO MEMORIZACAO
```

Fig. AIE.4 - Relatórios **dos** Resultados **çobre** o Atendimento **das Características de Qualidade para** o **Programa** exemplo

```
   FATOR:              MANIPULABILIDADE

      SUB-FATOR:       DISPONIBILIDADE

         CRITERIOS:

                       ATUALIZACAO

------------------------------
CARACTERISTICAS NAO APLICAVEIS:
------------------------------

      SUB-FATOR:       RASTREABILIDADE

         CRITERIO:

                       LOCALIZABILIDADE EXTERNA
```

Fig. AII.4 - **Relatórios dos Resultados sobre o Atendimento** das Características de Qualidade para o **Programa** exemplo (continuação)

```
------------------------------------------------------------------------
     INDICACOES SOBRE AS MEDIDAS COM SUGESTOES PARA REVISA0 NO PROGRAMA
                          EXPTES  .FOR EM   26/02/91
                                    'AS   20:23 HS
------------------------------------------------------------------------

>---> Devem ser substituidos os seguintes nomes referenciados ao longo deste
>      programa por nomes considerados significativos:
>      SOMA1 , SOMA2 .

>---> Deve ser incluido um comentario tipo cabecalho explicando a funcao
>      deste programa, no seu inicio, antes oy depois do comando PROGRAM.
>
>      Tambem nao existe cabecalho de manutencao padrao (com indicacao sobre
>      MANUTENCAO, RESPONSAVEL, DATA e MOTIVO) no seu inicio.
>      Caso ja tenha ocorrido alguma manutencao no mesmo, inclua tambem este
>      tipo de cabecalho.

        PROGRAM PPEXP
        EXTERNAL SINH, COSH, TANH
        INTEGER SOMA1
        REAL ANGULO
        PIMENSION X(10),Y(10)
>---> Deve haver pelo menos um espaco em branco depois da virgula separadora
>      de variaveis.


* FUNCAO 1: CALCULO DOS VALORES DA TANGENTE, SENO E COSENO PARA ANGULO

        PRINT *, 'Entre com o numero +para ser calculado= o valor da
     *tangente, sen9 ou coseno, ou entao CTRL Z para sair'

        READ *,ANGULO

        TANHX = TANH(ANGULO)
          SINHX = SINH(ANGULO)
>---> Deve ser alinhado este comando de acordo com a coluna do comando
>      PROGRAM.

        COSHX = COSH(ANGULO)

        WRITE(*,100) TANHX, SINHX, COSHX                                IMP...
>---> Deve ser retirado o comentario colocado depois da coluna 73.


100     FORMAT("1VALORES ENCONTRADOS PARA TAN-SIN-COS =",3F3.0)

120     FORMAT(//1X,'VALORES ENCONTRADOS:')
>---> Deve ser retirado este comando FORMAT nao referenciado.


10      CONTINUE
>---> Deve ser ordenado este label em ordem crescente em relacao ao label
      anterior.


* FUNCAO 2: FAZ O SOMATORIO DOS NUMEROS INVERTIDOS DE 20 A 30
>---> Deve ser alterado o conteudo deste bloco de comentarios para que a
>      informacao seja clara, simples e sem abreviacao nao significativa.


        CALL CALCULO(30,SOMA1)

* FUNCAO 3:
```

Fig. AII.5 - Relatórios das Indicações sobre as Medidas com Sugestões para Revisão no Programa exemplo

```
        READ (4,REC = 47)(X(IND1), IND1 = 1,10)
        READ (4,REC=47)(Y(IND2), IND2 = 1,10)
>---> Deve haver pelo menos um espaco em branco depois do sinal = de
>     atribuicao.


        PRODXY = 0.0

        PRINT *,'VALOR INTERMEDIARIO = '
        DO 13Q IND1=1,10
>---> Deve haver pelo menos um espaco em branco depois do sinal = de
>     atribuicao.


           IF ((X(IND1) .EQ. 0) .AND. (Y(IND1) .EQ. O)) THEN
              GO TO 130
>---> Deve ser retiradó este comando GO TO.

           ELSE
                PRODXY = PRODXY+X(IND1)*Y(IND2)
>---> Deve ser indentado este comando em relacao ao seu comando IF.
>---> Deve haver pelo menos um espaco em branco antes e depois do operador
>     de expressao.

           END IF

              PRINT *,PRODXY
>---> Deve ser indentado este comando em relacao ao seu comando DO.


130     CONTINUE


        STOP

        END                                                          FIM.
>---> Deve ser retirado o comentario colocado depois da coluna 73.


C
C CALCULO DA FUNCAO SENO SINH, COSENO COSH E TANGENTE TANH
C

        REAL FUNCTION TANH(X)
>---> Deve haver apenas um comando RETURN neste subprograma, para se ter um
>     unico ponto de saida.

C
C FUNCAO DE COMANDO PARA CALCULAR DUAS VEZES O SENO SINH
C
        DVSINH(X) = EXP(X) - EXP(-X)
>---> Deve haver pelo menos um espaco em branco antes e depois do operador
>     de expressao.

C
C FUNCAO DE COMANDO PARA CALCULAR DUAS VEZES A COSH
>---> Deve haver pelo menos uma linha em branco antes e depois deste bloco de
>     comentarios.

        DVCOSH(X) = EXP(X) + EXP( - X)

           TANH = DVSINH(X)/DVCOSH(X)
>---> Deve ser alinhado este comando de acordo com a coluna do comando
```

Fig. AII.5 - Relatórios das Indicações sobre as Medidas com Sugestões para Revisão no Programa exemplo (continuação)

```
>     de expressao.

        RETURN
        END

*
* EXEMPLO DE SUBROTINA NAO DE BAIXO NIVEL
* FAZ O SOMATORIO DE UM CALCULO COM O NUMERO INVERTIDO DO LIMITE ESPECIFICADO
*
        SUBROUTINE CALCULO(CONTR,RESULT)
>---> Deve ser mudado o tipo de argumento de entrada que e um flag de
>     controle.
>---> Deve ser transferido o comando de E/S formatado deste subprograma.

        INTEGER RESULT, CONTR

        DO 500 I = 20, CONTR

           RESULT = RESULT + INVERT(I)

500     CONTINUE

        WRITE (*,550)RESULT
550     FORMAT(I5)

        RETURN
        END

        FUNCTPON INVERT(NUM)
>---> Deve haver um comentario tipo cabecalho explicando a funcao deste
>     subprograma, no seu inicio, antes ou depois de sua declaracao.
>---> Deve ser retirado o comando COMMON deste subprograma.
>---> Deve ser mudado o tipo de argumento de entrada que e um flag de
>     controle.

        COMMON TEMP

        INTEGER DIG1,DIG2
>---> Deve haver pelo menos um espaco em branco depois da virgula separadora
>     de variaveis.

        I F (NUM .LT. 10 .OR. NUM .GE. 100) THEN
           PRINT *, 'ERRO NO ARGUMENTO', NUM
           GO TO 600
>---> Deve ser retirado este comando GO TO.

        END I F

        DIG1 = NUM /10
>---> Deve haver pelo menos um espaco em branco antes e depois do operador
>     de expressao.

        DIG2 = MOD(NUM,10)
        TEMP = DIG2 * 10

* RETORNA UM NUMERO INTEIRO DE DOIS DIGITOS EM ORDEM INVERSA AO "NUM"
* DE ENTRADA

        INVERT = TEMP +DIG1
>---> Deve haver pelo menos um espaco em branco antes e depois do operador
>     de expressao.
```

Fig. AII.5 - Relatórios das Indicações sobre as Medidas com Sugestões para **Revisão** no **Programa** exemplo (continuação)

```
>      PROGRAM.
>---> Deve haver pelo menos um espaco em branco antes e depois do operador
>      de expressao.

        RETURN

C
C CALCULO DO SENO SINH
>---> Deve haver pelo menos uma linha em branco antes e depois deste bloco de
>      comentarios.

        ENTRY SINH(X)
>---> Deve ser retirado este comando ENTRY, para se ter um unico ponto de
>      entrada neste subprograma.

        SINH = DVSINH(X)/2.0
>---> Deve haver pelo menos um espaco em branco antes e depois do operador
>      de expressao.

        RETURN
C
C CALCULO DO COSH
C
        ENTRY COSH(X)
>---> Deve ser retirado este comando ENTRY, para se ter um unico ponto de
>      entrada neste subprograma.

        CQSH = DVCOSH(X)/2.0
>---> Deve haver pelo menos um espaco em branco antes e depois do operador
>      de expressao.

        RETURN
        END

C
C EXEMPLO DE SUBROTINA DUMMY, OU SEJA, MORTA (SEM COMANDO EXECUTAVEL)
>---> Deve haver pelo menos uma linha em branco antes e depois deste bloco de
>      comentarios.

        SUBROUTINE DUMMY
>---> Deve ser retirado este subprograma nao referenciado.
>---> Deve ser retirado este subprograma morto (sem comandos executaveis).


        RETURN
        END
C EXEMPLO DE SUBROTINA NAO REFERENCIADA
>---> Deve haver pelo menos uma linha em branco antes e depois deste bloco de
>      comentarios.

        SUBROUTINE NCHAM(A,B,C)
>---> Deve ser retirado este subprograma nao referenciado.
>---> Deve haver pelo menos uma linha em branco antes deste subprograma.
>---> Deve ser retirado o comando EQUIVALENCE deste subprograma.

        CHARACTER M1*18, M2*10
        EQUIVALENCE (M1, M2)
        INTEGER A,B,C
>---> Deve haver pelo menos um espaco em branco depois da virgula separadora
>      de variaveis.


        C = (A*4) + B
>---> Deve haver pelo menos um espaco em branco antes e depois do operador



600     RETURN
        END
```

Fig. AII.5 - Relatórios das Indicações sobre as Medidas com Sugestões para Revisão no Programa exemplo (continuação)

```
------------------------------------------------------------------------------
        SUGESTOES PARA REVISA0 NO AMBIENTE DE TRABALHO DO PROGRAMA
                      EXPTES  .FOR EM   26/02/91
                                'AS   20:23 HS
------------------------------------------------------------------------------

>----> Deve ser armazenada a versao mais atual deste programa nos seus devidos
>      lugares.

                          ---------------
                          ATENCAO USUARIO:
                          ---------------

   ANTES DE ATIVAR ESTA FERRAMENTA NUMA PROXIMA VEZ,

   CONHECA MAIS O AMBIENTE DE TRABALHO ENVOLVIDQ COM O PROGRAMA A SER SUBME-

   TIDO A ANALISE ESTATICA, OBTENDO INFORMACOES SOBRE O QUE EXISTE OU NAO.
```

Fig. AIP.6 - Relatórios das Sugestões para Revisão no Ambiente de Trabalho do Programa exemplo