


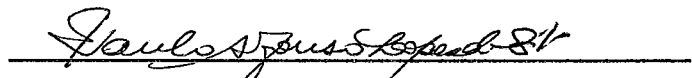
O USO DE MÉTRICAS DE COMPLEXIDADE
PARA O CONTROLE DA QUALIDADE DE SOFTWARE
CIENTÍFICO

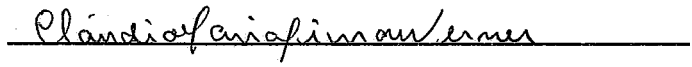
Alvaro de Sá Bahia

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

Aprovada por:


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


Paulo Afonso Lopes da Silva, PhD.


Cláudia Maria Lima Werner, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
MAIO 1992

BAHIA, ALVARO DE SÁ

O uso de métricas de complexidade para o controle da qualidade de software científico. [*Rio de Janeiro*] 1992,

X, 107 p. 29,7 cm (COPPE/UFRJ, M. Sc., Engenharia de Sistemas e Computação, 1992)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Controle de qualidade de software científico

I. COPPE/UFRJ

II. Título (série)

Às mulheres e aos homens que me deram e dão vida:

Beth, Fernanda, Anayde,

Daniel e Álvaro.

AGRADECIMENTOS

À minha orientadora Ana Regina pela paciência e estímulo ao longo de todo o processo e que soube compreender, sempre, as dificuldades para a conclusão deste trabalho em regime parcial de dedicação.

À PETROBRÁS por ter permitido que fossem alocados tempo e recursos para a realização deste trabalho.

Ao Hugo e Leonor pelo apoio constante, principalmente, junto ao Dani e depois com a Nanda.

Aos colegas da COPPE e Petrobrás que através de seu interesse e estímulo muito me incentivaram a não interromper este trabalho. Agradeço, em especial, aos companheiros do Setor de Apoio aos Usuários (SEAP) do Departamento de Exploração da Petrobrás que tanto me ajudaram na edição deste trabalho e no projeto da interface. Agradeço, ainda, aos colegas da Operação que foram sempre solidários, especialmente em minhas jornadas noturnas. Quero também registrar a compreensão de minhas Chefias que permitiram que eu me dedicasse a terminar este trabalho.

Ao Simplicio Freitas que primeiro me incentivou a aceitar este desafio e que muito ajudou a iniciá-lo.

Aos contemporâneos da COPPE com os quais compartilhei minha ignorância e saber, meu muito obrigado. Posso hoje dizer que foi duro mas valeu a pena.

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

O USO DE MÉTRICAS DE COMPLEXIDADE
PARA O CONTROLE DA QUALIDADE DE SOFTWARE CIENTÍFICO

Alvaro de Sá Bahia

Maio de 1992

Orientadora: Ana Regina Cavalcanti da Rocha

Programa : Engenharia de Sistemas e Computação

Para que sejam produzidos programas corretos dentro de prazos e custos determinados, é preciso transformar a criação de software num empreendimento cuidadosamente controlado, metódico e previsível. Para tanto, propõe-se a utilização de métricas de complexidade como meio de controlar e assegurar a qualidade do software produzido em determinado ambiente.

De modo a permitir a mensuração automática das métricas de complexidade sugeridas, foi desenvolvido, na linguagem C, um analisador estático para códigos escritos em FORTRAN com uma interface projetada para o ambiente X-Windows no padrão OSF/Motif. Através de parametrização pode-se, também, analisar o pseudocódigo com uma sintaxe similar.

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

USING COMPLEXITY METRICS TO ASSURE QUALITY CONTROL FOR
SCIENTIFIC SOFTWARE

Alvaro de Sá Bahia

May, 1992

Thesis Supervisor: Ana Regina Cavalcanti da Rocha

Department : Computing and Systems Engineering

In order to obtain reliable software within fixed budget and time, it's necessary to make the process of making software a carefully controlled, methodical and predictable enterprise. We propose the use of complexity metrics as a mean of controlling and assuring the quality of software obtained from an especific environment.

To allow the automatic measuring of the chosen complexity metrics it was developed and implemented, in C language, a static analyser for FORTRAN code with its interface designed for the X-Windows environment according to the OSF/Motif standard. Along with a customization process we can also analyse pseudocode with a similar syntax.

Índice

Lista de Figuras	ix
1. INTRODUÇÃO	1
1.1 - A Informática e a Indústria	1
1.2 - O Controle da Qualidade do Software	4
1.3 - Características do Software Científico	9
1.4 - Organização da Tese	12
2. AVALIAÇÃO DA COMPLEXIDADE DO SOFTWARE	14
2.1 - O Problema da Manutenção de Software	14
2.2 - Métricas de Complexidade e Manutenção de Software	16
2.3 - Motivação para a Tese	20
3. A FERRAMENTA PIÁCATÚ	25
3.1 - Especificação Informal da PIÁCATÚ	25
3.2 - Projeto da Interface com o Usuário	29
3.3 - Exemplo de Utilização	51
4. CONCLUSÕES	61
4.1 - Estratégias para a Utilização da PIÁCATÚ	61
4.2 - Sugestões para Futuros Trabalhos	64
REFERÊNCIAS	66

APÊNDICE A - Métricas de Complexidade	76
A.1 - Métricas de Tamanho	76
A.2 - Métricas de Estrutura Lógica	77
A.3 - Métricas de Estrutura de Dados	79
A.4 - Métricas de Halstead	79
A.5 - Métrica de Harrison e Cook	81
A.6 - Métrica de Card e Agresti	82
A.7 - Métrica de Hall e Preiser	83
A.8 - Métrica de Henry e Kafura	84
APÊNDICE B - Sintaxe Resumida	86
B.1 - Descrição da Sintaxe	86
APÊNDICE C - Relação de Métricas	92
C.1 - Apresentação	92
C.2 - Caracterização por Tipo de Complexidade	93
C.3 - Caracterização por Nível de Complexidade	95

Lista de Figuras

Figura 1. Modelo de Rocha	5
Figura 2. Critérios para a Avaliação de Projeto	6
Figura 3. Critérios para a Avaliação de Programas (1/3)	7
Figura 4. Critérios para a Avaliação de Programas (2/3)	8
Figura 5. Critérios para a Avaliação de Programas (3/3)	9
Figura 6. Visão Esquemática da PIÁCATÚ	26
Figura 7. Tela Inicial	30
Figura 8. Opção Parâmetros	31
Figura 9. Diálogo Profile / Novo e Salva Como	32
Figura 10. Diálogo Profile / Abrir	33
Figura 11. Diálogo Arquivos	34
Figura 12. Diálogo Sintaxe / Cmd - Oper - Func	36
Figura 13. Diálogo Sintaxe / Comentários e Linha	37
Figura 14. Diálogo Componentes / Definição	39
Figura 15. Diálogo Componentes / Redefinição	40
Figura 16. Diálogo Componentes / Salva Como	40
Figura 17. Diálogo Métricas / Modif & Ativação	42
Figura 18. Diálogo Alerta / Comando	43
Figura 19. Diálogo Alerta / Labels	44
Figura 20. Diálogo Outlier / Def e Alerta	45
Figura 21. Diálogo Planilhas	46

Figura 22. Diálogo Geral	47
Figura 23. Tela Inicial - Base-Dados	48
Figura 24. Tela Inicial - Estatística	49
Figura 25. Tela Inicial - Visualização	51
Figura 26. Versão 1 - Programa MUTMETA	52
Figura 27. Versão 2 - Programa MUTMETA	53
Figura 28. Versão 3 - Programa MUTMETA	54
Figura 29. Programa principal (MAIN)	55
Figura 30. Sub-rotina LEPARA (Le-Parametros)	55
Figura 31. Sub-rotina CALNUM (Calcula-Numeros)	56
Figura 32. Sub-rotina IMPRES (Imprime-Resultados)	56
Figura 33. Sub-rotina ERRO	57
Figura 34. Programa principal (MAIN)	57
Figura 35. Sub-rotina LEPARA (Le-Parametros)	58
Figura 36. Sub-rotina CALNUM (Calcula-Numeros)	58
Figura 37. Sub-rotina IMPRES (Imprime-Resultados)	58
Figura 38. Sub-rotina ERRO	59
Figura 39. Resumo das métricas por programa	59
Figura 40. Uma estratégia de utilização da PIÁCATÚ	62
Figura 41. Outra estratégia para a utilização da PIÁCATÚ	63

1. INTRODUÇÃO

1.1 - A Informática e a Indústria

O uso de sistemas de informações na indústria é fato inconteste e não se pode mais conceber sua não utilização. Estimativas do Departamento de Comércio dos Estados Unidos indicavam que o mercado mundial de software para 1985 ficaria em torno de 40 bilhões de dólares, com uma taxa de crescimento anual avaliada em 30% (OCBE, 1984). Ao analisar a importância econômica de ganhos de produtividade no desenvolvimento de software, Boehm (1987) apresenta estimativas que diferem daquelas apresentadas pelo governo americano. Boehm registra que os custos com desenvolvimento de software, em 1985, nos Estados Unidos situavam-se na faixa de 70 bilhões de dólares, contra o valor de 140 bilhões a nível mundial. Segundo ele, admitindo-se que a taxa de crescimento anual vigente à época (12%) fosse mantida, chegaríamos a 1995 com os seguintes valores: nos Estados Unidos teríamos um gasto anual de 225 bilhões de dólares e em todo mundo algo em torno de 450 bilhões de dólares. Como se pode observar, independentemente de qual estimativa se considere, os números envolvidos com desenvolvimento de software são altamente expressivos.

Burril e Ellsworth (1986) mostram que os gastos diretos totais da indústria nos Estados Unidos em processamento de dados passaram de 2,1% do seu PNB

(Produto Nacional Bruto) em 1970, para 5,2% em 1980 e 8,3% em 1985. Para os anos 90 estima-se que estes gastos correspondem a cerca de 13,0%.

Em estudo elaborado pelo governo britânico (CO/ACARD, 1986), concluiu-se que um aumento real e significativo na produtividade da indústria de software só será obtido através da aplicação de princípios científicos, matemáticos e de gerência análogos ao da engenharia. Só assim será possível produzirem-se programas corretos, dentro de prazos e custos determinados e a níveis aceitáveis de preço e desempenho. Kafura (1985) salienta que é preciso transformar a criação de software de uma atividade considerada artesanal, pouco clara ou mesmo indisciplinada num empreendimento cuidadosamente controlado, metódico e previsível. Soat (1991), em artigo que analisa o desempenho das fábricas de software no Japão, diz que é necessário transformar métodos tradicionais de desenvolvimento de software utilizados nos Estados Unidos, tidos como não padronizados e pessoais num processo que tenha uma estrutura bem planejada, com um controle rigoroso e que resulte numa linha de produção de software eficaz. Soat diz ainda que a utilização destes princípios pelos japoneses, em suas fábricas de software, permite a obtenção de uma produção 70% maior e com a metade dos erros quando comparada com a conseguida pelos americanos.

É necessário, portanto, que cada ambiente de desenvolvimento identifique um conjunto de critérios que permita determinar os atributos mais relevantes do software por ele desenvolvido. Por mais relevantes, entende-se aqueles elementos que, uma vez quantificados, possam explicar ou estimar comportamentos futuros do software, em aspectos tais como: número de erros ainda existentes no software entregue ao usuário, tempo gasto na codificação e nos testes dos módulos, esforço necessário para efetuar manutenção nos módulos e outros.

Poore (1988) apresenta um esquema possível para a constituição de um grupo de trabalho bem definido que possa articular um conceito próprio de qualidade de software em termos operacionais e determinar métricas úteis para quantificar aquele conceito e ambiente específicos. Esta proposta requer a identificação e delimitação de um ambiente homogêneo de desenvolvimento de software, bem como o entendimento de qual é a visão de qualidade do grupo de trabalho. A partir daí, obtêm-se as métricas ajustadas ao software produzido por aquele ambiente.

Os fatores de qualidade podem ser influenciados pela linguagem de programação utilizada, pelo ambiente de testes, pelo tipo de gerência e pela prática e experiência da equipe desenvolvedora. Segundo Arthur (1985), os fatores que caracterizam a qualidade de um software podem variar de sistema a sistema e de programa a programa. Kearney et alli (1986) afirmam que os fatores que contribuem para dificultar a tarefa de produzir software podem ser bastante diferentes quando se consideram programadores novatos ou programadores experientes. Sugerem, ainda, que antes que o uso de determinada métrica seja incorporada a um ambiente de desenvolvimento de software o usuário deve certificar-se que ela é apropriada para o objetivo que se pretende alcançar.

Uma vez identificado o conjunto de medidas que melhor descreve o comportamento do software desenvolvido por determinado ambiente, não se pretende que elas sejam utilizadas por outro grupo ou até pelo mesmo em situações diferentes. Na seleção do conjunto adequado para determinado ambiente, é necessário considerar suas características individuais. Espera-se, entretanto, que uma vez escolhidas elas sejam confiáveis e de fácil uso e obtenção.

1.2 - O Controle da Qualidade do Software

Para que a qualidade do software possa ser assegurada, é preciso que o ambiente de desenvolvimento tenha: a) uma definição não ambígua de qualidade; b) um processo de julgamento que possa de maneira consistente aplicar tal definição; c) um conjunto de premissas ou regras que permitam uma caracterização adequada da definição; d) uma função de pontuação que torne o processo de estimar qualidade uma atividade viável.

A literatura apresenta diversos modelos para o controle da qualidade de software, dentre os quais pode-se citar o de Boehm (1978), McCall (1979), Arthur (1985) e Rocha (1987). Este trabalho se insere no contexto do Modelo de Rocha, que descreve-se sucintamente a seguir.

Rocha apresenta um modelo no qual através da identificação de **critérios** (atributos primitivos de um software passíveis de serem medidos de forma objetiva), são realizadas **medidas** através de **processos de avaliação** bem definidos. A **agregação destas medidas** permite quantificar a presença de **fatores** (ou **subfatores**) de qualidade necessários à obtenção dos **objetivos de qualidade** definidos previamente para o software em questão. A Figura 1 apresenta uma visão esquemática do método descrito.

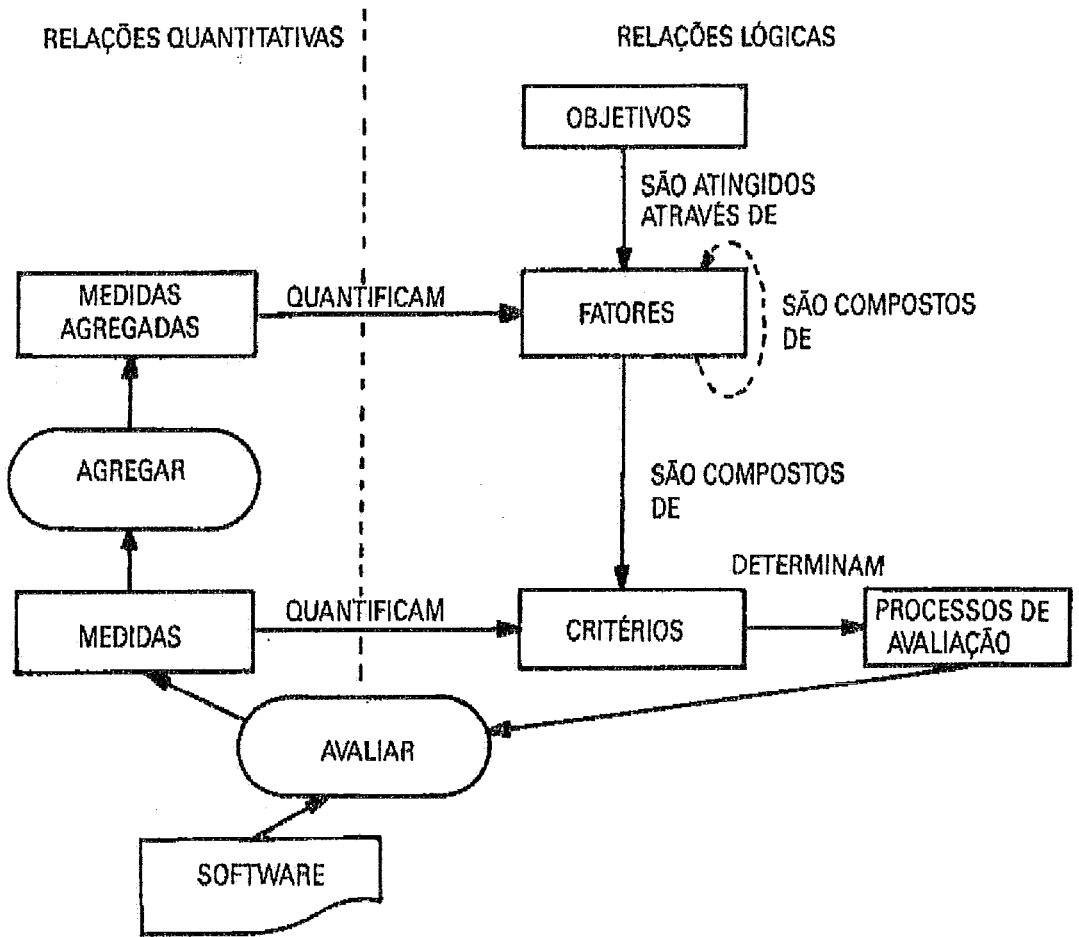


Figura 1. Modelo de Rocha: Modelo proposto por Rocha para o controle da qualidade do software (Rocha, 1987).

O Modelo de Rocha pode ser aplicado na avaliação da qualidade de software nas seguintes fases do seu desenvolvimento: especificação, projeto e codificação de programas. Para cada caso são definidos seus objetivos de qualidade, bem como os fatores e os subfatores através dos quais aqueles são atingidos. Por fim, temos os critérios que permitem a quantificação dos subfatores (fatores).

Nesta tese consideram-se métricas (medidas) associadas à Avaliação de Projetos e à Avaliação de Programas. A Figura 2 apresenta a descrição dos objetivos de qualidade, fatores e subfatores para a Avaliação de Projetos. Assinalou-se em **negrito** os elementos que podem ser avaliados por métricas consideradas neste trabalho.

-
- Confiabilidade Conceitual
 - Fidedignidade
 - ▲ Completeza
 - ▲ Consistência
 - ▲ Fidelidade
 - ▲ Não-Ambiguidade
 - ▲ Necessidade
 - Integridade
 - ▲ Robustez
 - ▲ Segurança
 - Confiabilidade da Representação
 - Comunicabilidade
 - ▲ Concisão
 - ▲ Correção no uso da linguagem de projeto
 - ▲ **Modularidade**
 - ▲ Uniformidade de terminologia
 - ▲ **Uniformidade no nível de abstração**
 - Manipulabilidade
 - ▲ Disponibilidade
 - ▲ Estrutura
 - ▲ Rastreabilidade
 - Utilizabilidade
 - Avaliabilidade
 - ▲ Validabilidade
 - ▲ Verificabilidade
 - Flexibilidade
 - Implementabilidade
 - ▲ Viabilidade de Cronograma
 - ▲ Viabilidade de Mão-de-Obra
 - ▲ Viabilidade de Recursos de Suporte
 - ▲ Viabilidade Econômica
 - ▲ Viabilidade Financeira
 - ▲ Viabilidade Social
 - ▲ Viabilidade Técnica
 - Manutenibilidade
 - ▲ Evolutibilidade
 - ▲ Modificabilidade
 - Reutilizabilidade

Figura 2. Critérios para a Avaliação de Projeto: Critérios para a fase de projeto (Rocha, 1987).

Para a Avaliação de Programas temos os critérios descritos nas Figuras 3 a 5.

-
- Confiabilidade Conceitual
 - Fidedignidade
 - ▲ Completeza
 - ▲ Necessidade
 - ▲ Precisão
 - Integridade
 - ▲ Robustez
 - ▲ Segurança

Figura 3. Critérios para a Avaliação de Programas (1/3): Critérios para a fase de codificação de programas - Confiabilidade Conceitual (Andrade, 1991).

O que se pretende é a identificação de um amplo leque de critérios que, podendo ser medidos de forma clara e objetiva, possam ser utilizados para: caracterizar determinado ambiente; comparar um ambiente com outro; monitorar o "status" de um projeto em andamento e permitir a elaboração de estimativas, tendo por base os dados coletados em projetos anteriores. Idealmente, as métricas escolhidas devem possibilitar o seu uso no início do desenvolvimento e ter forte correlação com as variáveis dependentes que sejam de interesse (Basili, 1985).

Apesar da reconhecida necessidade de uma contínua melhoria de produtividade na geração de software, um aspecto nem sempre considerado é que poucas organizações sabem quantificar sua produtividade atual e são portanto incapazes de medir os resultados decorrentes de mudanças em métodos e ferramentas. Desta forma, tão importante quanto efetuar-se medidas que quantifiquem os diversos aspectos do software produzido, é fundamental um acompanhamento ao longo do tempo para que se possa determinar se uma mudança efetuada no processo de desenvolvimento resultou em ganho ou perda de qualidade.

-
- **Confiabilidade da Representação**
 - **Legibilidade**
 - ▲ **Clareza**
 - △ **Número de decisões**
 - △ **Padronização**
 - ▲ **Concisão**
 - △ **Não anomalia**
 - △ **Não repetição**
 - ▲ **Estilo de programação**
 - △ **Comentário**
 - △ **Identificação**
 - △ **Indentação**
 - △ **Organização visual**
 - △ **Programação estruturada**
 - ▲ **Modularidade**
 - △ **Balanceamento**
 - △ **Coesão**
 - △ **Não acoplamento**
 - △ **Não memorização**
 - △ **Número de módulos inferiores**
 - △ **Número de módulos superiores**
 - △ **Tamanho**
 - **Manipulabilidade**
 - ▲ **Disponibilidade**
 - △ **Acessibilidade**
 - △ **Atualização**
 - ▲ **Rastreabilidade**
 - △ **Localizabilidade interna**
 - △ **Localizabilidade externa**

Figura 4. Critérios para a Avaliação de Programas (2/3): Critérios para a fase de codificação de programas - Confiabilidade da Representação (Andrade, 1991).

Uma pergunta que se coloca é: Quais métricas considerar? Acreditamos que as métricas de complexidade, descritas no Capítulo 2 e no Apêndice A, são aquelas que devem ser abordadas pois permitem, de modo geral, que se automatize seu processo de medição, podendo em certos casos serem usadas já na fase de projeto. Ducan (1988) descreve a utilização de métricas de software como meio para quantificar e descrever a qualidade existente no software produzido por um determinado grupo de engenharia de software da Digital Equipment Corporation (DEC).

Cabe destacar que não se pretende dizer que outros fatores não possam ou não devam ser considerados. Card et alli (1985), na tentativa de elaborar critérios para a modularização de software, estudaram os efeitos da coesão e tamanho dos módulos nos custos de desenvolvimento e na quantidade de erros entregues no produto. Perceberam então que bons programadores tendem a construir módulos com alta coesão e que tais módulos apresentam uma taxa de erros menor do que aqueles que possuem baixa coesão. Advertem, entretanto, que fatores tais como: coesão, acoplamento e encapsulamento de dados são elementos difíceis de identificar mecanicamente e que por enquanto a alternativa é determinar sua ocorrência através da elaboração de uma lista de verificações. Palermo e Rocha (1988) apresentam um estudo interessante para avaliação da qualidade de programas, mas que requer em muitos casos a inspeção manual do código para verificar se determinado critério foi alcançado ou não.

-
- Utilizabilidade
 - Avaliabilidade
 - ▲ Validabilidade
 - ▲ Verificabilidade
 - Eficiência
 - Manutenibilidade
 - Operacionalidade
 - ▲ Amabilidade ao uso
 - ▲ Oportunidade
 - Portabilidade
 - Rentabilidade
 - Reutilizabilidade

Figura 5. Critérios para a Avaliação de Programas (3/3): Critérios para a fase de codificação de programas - Utilizabilidade (Andrade, 1991).

1.3 - Características do Software Científico

A produção de software científico, tanto quanto em outras áreas, vem enfrentando obstáculos para obter produtos de qualidade em tempo hábil. A

dificuldade, ainda existente, para conceituar e medir certos aspectos relativos à qualidade de software soma-se àquela decorrente da inexistência de consenso a respeito da forma mais adequada para a elaboração de software científico ou numérico. O software enquadrado nesta categoria é caracterizado por possuir um conjunto de algoritmos, por sua vez compostos de uma série de algoritmos menores, agrupados normalmente numa estrutura complexa onde a iteração é a principal característica (Bahia, 1988b). A quantidade de dados gerados, de um modo geral, pode não ser substancial, mas uma proporção significativa do total é operada um número expressivo de vezes. A execução do software numérico requer equipamentos com grande memória e alta capacidade computacional.

A produção de software científico requer duas fases distintas. A primeira, de pesquisa, é necessária para o equacionamento do problema. Muitas vezes a solução não é conhecida e deve ser procurada. Os algoritmos necessários à solução numérica do problema devem, ainda, ser estabelecidos e testados. Uma vez chegado à concepção da solução para o problema proposto, é necessário que ela seja validada através de um protótipo. Verificar-se-á, assim, se o conjunto de algoritmos pode ser implementado de forma satisfatória. Uma vez avaliada a factibilidade da solução, passa-se à fase de construção do software.

Cross (1986) afirma que os técnicos responsáveis pela fase de pesquisa se veem, via de regra, mais como matemáticos, engenheiros ou cientistas do que como analistas de sistemas. Isto, apesar de gastarem cerca de 80% do seu tempo projetando, confeccionando e mantendo software. Outro fator presente é que o esforço requerido para implementar com sucesso determinados algoritmos numéricos é tão substancial, que uma vez terminado, a tendência é deixá-lo do jeito que está, desconsiderando-se que o produto obtido é um protótipo, e portanto, sem os requisitos necessários para ser considerado um software produto.

Através de uma pesquisa realizada na Inglaterra, por Medes et alli (1987), verificou-se que a linguagem predominante é o FORTRAN, o que seria, em parte, explicado pelo fato de ser uma linguagem disponível mundialmente, por ser bastante portátil (compatível), e também por dispor de uma grande variedade de bibliotecas de subprogramas numéricos (Bell, 1986).

Sistemas para esta área são confeccionados sem que atendam adequadamente o usuário final e são, na maioria das vezes, elaborados de tal forma que a sua manutenção é difícil e onerosa. A pesquisa de Medes, citada anteriormente, mostra que cerca de 37% dos respondentes não se consideravam usuários de técnicas de programação estruturada. Cross (1986) chega a afirmar que a metodologia de desenvolvimento utilizada na maioria das organizações envolvidas com software numérico é informal.

Chivers e Clark (1985) citam também os seguintes aspectos para justificar porque o FORTRAN ainda é uma linguagem largamente utilizada: a) existe uma enorme quantidade de software já produzido e não seria trivial sua conversão para outra linguagem; b) existe um padrão amplamente aceito e obedecido pelos compiladores disponíveis no mercado; c) a implementação da linguagem é bastante eficiente e gera código com alto desempenho; d) os elementos da linguagem são facilmente assimilados pelos seus usuários que frequentemente vêem a atividade de programação como secundária, já que estão mais diretamente interessados em solucionar problemas em suas próprias disciplinas. Davis (1987), além de concordar com os aspectos acima mencionados, indica que pesquisadores estimam que cerca de 25% dos ciclos de máquina disponíveis mundialmente executam código gerado por algum compilador FORTRAN. Segundo ele o FORTRAN reina absoluto entre os pacotes desenvolvidos para área científica ou de engenharia.

Nomura (1987) mostra, com dados provenientes de três pesquisas realizadas entre 200 empresas japonesas, que a linguagem FORTRAN é utilizada por 54% dos respondentes ficando atrás apenas do COBOL (60%) e ASSEMBLER (72%). Indica, também, que pesquisa similar realizada nos Estados Unidos apresenta resultados semelhantes.

1.4 - Organização da Tese

Organizou-se esta tese em quatro capítulos e três apêndices. O primeiro capítulo contém a Introdução onde é enfatizada a importância econômica da atividade de desenvolvimento de software para a indústria como um todo, fala-se da necessidade de adotar-se um modelo para a garantia de qualidade do software a ser produzido, constata-se o uso da linguagem FORTRAN pela indústria, no desenvolvimento de software científico e, por fim, apresenta-se a forma em que este trabalho está organizado.

No Capítulo 2 descrevem-se os problemas habitualmente encontrados no desenvolvimento de software, notadamente na fase de manutenção. Em seguida apresenta-se a alternativa de usar-se métricas de complexidade como mecanismo para o controle da qualidade e descrevem-se os elementos motivadores desta tese.

No Capítulo 3 faz-se uma descrição da especificação e uma apresentação detalhada do projeto da interface da ferramenta PIÁCATÚ sugerida no capítulo 2. Ao final do capítulo mostra-se um estudo de caso, analisando-se um pseudocódigo e um código FORTRAN, através do uso do protótipo da ferramenta.

No Capítulo 4 resume-se a proposta desta tese, o controle da qualidade obtido com o uso de métricas de complexidade, com a apresentação de duas estratégias

para a utilização da PIÁCATÚ. Ao final identificam-se algumas alternativas que possam dar continuidade a este trabalho.

No Apêndice A descrevem-se as métricas de complexidade citadas no texto e mais significativas para este trabalho.

No Apêndice B apresenta-se, de forma resumida, os comandos e a sintaxe considerada na elaboração do analisador.

No Apêndice C tem-se a lista das métricas passíveis de serem analisadas pela PIÁCATÚ.

2. AVALIAÇÃO DA COMPLEXIDADE DO SOFTWARE

2.1 - O Problema da Manutenção de Software

Hoje em dia, quando se pensa em medir a qualidade de software, pensa-se nos efeitos que tal qualidade terá sobre a manutenção futura que certamente ocorrerá. Schneidewind (1987), em seu artigo sobre o estado atual da manutenção de software, apresenta alguns aspectos relevantes ao procurar responder perguntas do tipo: onde reside o problema da manutenção do software? Por que esta atividade é considerada difícil? Segundo o autor, o maior problema é que não se pode dar manutenção a um sistema que não foi projetado para ser mantido com facilidade. O próprio fato de instalar-se um sistema altera o ambiente, e subsequentes pressões operam no sentido de alterar o problema e a solução tecnológica anteriormente apresentada.

Programas grandes dificilmente são completados, eles simplesmente continuam se desenvolvendo. Novas funções são sempre adicionadas e nunca substituídas, devendo ser adequadamente incorporadas ao sistema corrente. Sistemas em produção raramente são totalmente substituídos, exceto quando há fortes razões econômicas ou técnicas. Outro complicador reside no fato de que as organizações estão

sempre procurando obter compatibilidade entre seus sistemas e não necessariamente a perfeição.

É inegável que o entendimento dos requisitos necessários não é obrigatoriamente obtido ao final da fase de definição de requisitos. É, inclusive, mais razoável esperar que tal compreensão advenha mais com a experiência no uso do sistema, do que a partir de uma especificação abstrata obtida na fase inicial do sistema.

Ao longo do tempo, os sistemas se tornam mais difíceis de serem mantidos, porque também se tornam crescentemente mais complexos. Muitas vezes a manutenção se torna complexa pois há a dificuldade em determinar o processo que gerou o produto, as mudanças não são adequadamente documentadas e sempre há o efeito de propagação de erros ao tentar-se consertar erros detectados.

Uma pesquisa realizada por Lientz e Swanson (1981), abrangendo 487 empresas revelou que:

- a) 50% do esforço total de desenvolvimento eram dedicados à manutenção;
- b) 55% do esforço de manutenção eram voltados para a extensão de funcionalidade ou melhoramentos no desempenho, 25% para adaptar-se à mudanças nos dados ou no ambiente de processamento, e apenas 20% para correção de erros;
- c) a idade média das aplicações situava-se entre 3 a 4 anos;
- d) o tamanho médio dos sistemas ficava em torno de 23.000 linhas de código, distribuídos em torno de 55 programas;
- e) cerca de 0,5 homem/ano era dedicado à manutenção de um sistema típico.

Fairley (1985) afirma que cerca de 40 a 60% do tempo no ciclo de vida total de um software são dedicados à manutenção, podendo em certos casos chegar a 90%. Outro aspecto a considerar é que as fases de programação, codificação e testes chegam

a representar cerca de 60% do tempo total de desenvolvimento do software (Fairley, 1985; Conte, 1986).

É preciso então:

- a) desenvolver software tendo em vista futuras alterações e depois mantê-lo tendo em vista as subseqüentes manutenções;
- b) estabelecer uma boa gerência de mudanças;
- c) envolver mantenedores de software nas fases de projeto e testes;
- d) obter ferramentas que auxiliem o processo de manutenção, em atividades tais como: a elaboração de gráficos de estrutura; controle de versões; "trace" de variáveis e de fluxos de controle etc;
- e) usar métricas que avaliem a estabilidade dos módulos, com respeito ao efeito da propagação de erros e a aspectos relativos a parte lógica e ao desempenho;
- f) privilegiar a simplicidade e, constantemente, avaliar o sistema para identificar excessiva complexidade.

2.2 - Métricas de Complexidade e Manutenção de Software

Gremillion (1984) ao avaliar a relação entre número de erros entregues e tamanho, complexidade, frequência de uso e idade dos programas, constatou que o número de erros é fortemente relacionado à complexidade. Vários autores - Basili (1984, 1985), Berns (1984), Gill e Kemerer (1991), Gremillion (1984), Henry (1984), Kafura (1985, 1987), Li (1987), Lind e Vairavan (1989) e Weyuker (1988) - têm tentado investigar a correlação entre métricas de complexidade e o esforço necessário para manter determinado software.

Pesquisas indicam que, ao longo da vida útil de um programa, os módulos que o constituem são manipulados, de uma forma ou de outra, por um tempo sempre superior à metade do tempo total. Durante todo este tempo, ele terá que ser compreendido de maneira a ser testado ou alterado. Sua complexidade deve ser controlada para não prejudicar seu entendimento a ponto de impedir que seja adequadamente modificado ou testado. Quanto mais complexo um sistema for mais difícil é entendê-lo e portanto mantê-lo. Gibson e Senn (1989) citam pesquisas empíricas mostrando que programas complexos tendem a requerer mais manutenção corretiva ao longo de sua vida útil. O problema acaba entrando num círculo vicioso: a manutenção é difícil por causa da complexidade e por causa da complexidade mais manutenção se torna necessária.

A literatura apresenta uma variedade de métricas. Um estudo interessante é aquele efetuado por Cote et alli (1988) onde mais de 120 publicações a este respeito foram analisadas e classificadas. As métricas de complexidade existentes podem ser agrupadas em três classes:

- 1) métricas de código (ou micro), que visam determinar o nível de complexidade interna do módulo com relação ao tamanho, à estrutura lógica e à utilização das estruturas de dados. Como exemplo, podemos citar: número total de linhas de código, número ciclomático de McCabe (1976), métricas de Halstead (1977), etc. Há autores que ainda subdividem esta classe em métricas de volume ou de controle (Li, 1987);
- 2) métricas de estrutura, que procuram avaliar a complexidade do sistema devida à interação e à interconexão entre os módulos, neste grupo estariam aquelas propostas por Card e Agresti (1988), Kafura (1981) e Henry (1981);
- 3) métricas híbridas, que são aquelas que pretendendo considerar a complexidade total do sistema, agrupam valores relativos à complexidade do módulo e aqueles decorrentes da interação entre os diversos componentes

do sistema, como exemplo temos as métricas propostas por Harrison e Cook (1987), Hall e Preiser (1984).

Para conhecer mais detalhes a respeito destas métricas sugere-se a leitura do Apêndice A.

Kafura e Canning (1985), através de um estudo com dez métricas, concluíram que:

- a) a natureza dos sistemas e do processo de desenvolvimento de software só pode ser perfeitamente entendida se vários recursos utilizados em sua construção forem considerados e também se vários fatores relativos ao software forem quantificados;
- b) um conjunto específico de métricas pode ser usado para identificar os componentes que conterão uma quantidade excessiva de erros ou que demandarão mais esforço para serem codificados e testados, sendo interessante usar métricas das três classes para garantir que vários dos aspectos associados ao desenvolvimento do software possam ser considerados;
- c) as métricas de estrutura e de código são complementares, sendo que as de estrutura tem a vantagem de poderem ser utilizadas já na fase de projeto;
- d) a escolha sobre quais e quantas métricas utilizar não é uma tarefa simples.

Kafura e Reddy (1987), em outro estudo onde avaliaram o comportamento de sete métricas, ao longo do período de manutenção de um software de médio porte, constataram que:

- a) o aumento da complexidade do sistema se reflete nas métricas;
- b) é possível identificar a integração inadequada de funções, através da análise das métricas de complexidade;

c) módulos excessivamente complexos podem gerar distorções na estrutura do sistema, pois futuras manutenções tendem a evitar alterações em tais módulos;

d) as métricas permitem identificar "*outliers*" (módulos cuja diferença entre o valor da métrica e o valor médio, é superior a duas vezes o desvio padrão), indicando desta forma onde os esforços devem ser concentrados, para que sua complexidade possa ser reduzida ou para que se possa dedicar-lhes mais atenção nas etapas de codificação e testes.

Ao estudar o comportamento de trinta e uma métricas de complexidade de código, Li (1987) constatou a consistência existente entre os valores obtidos para métricas de uma mesma classe. Verificou, também, que ao se dispor de medidas de complexidade do software, pode-se estabelecer e quantificar a qualidade pretendida, bem como atestar o cumprimento das obrigações no software contratado. É possível, então, verificar a adequação do produto final às especificações e obter-se elementos que permitam avaliar alternativas na fase de desenvolvimento e manutenção.

De forma a permitir a comparação entre as várias métricas de complexidade existentes, particularmente as de código, Weyuker (1988) apresenta um conjunto de propriedades que qualquer métrica deveria possuir. O número ciclomático de McCabe, ao ser analisado de acordo com aquelas propriedades, apresenta o inconveniente de não distinguir programas que realizam pouca computação daqueles que realizam muita, se ambos tiverem uma estrutura de decisão similar. Tanto o número ciclomático quanto o número de linhas de código não conseguem satisfazer a propriedade relacionada ao posicionamento dos comandos, isto é, não consideram o contexto mas apenas a complexidade intrínseca dos comandos. Lakshmanan et alli (1991) apresentam certas propriedades que qualquer métrica que avalie o fluxo de controle deve possuir. Em seu trabalho, revelam os pontos fortes e fracos de algumas das métricas existentes.

Por fim existem trabalhos, como o de Berns (1984a), que sugerem uma abordagem alternativa para determinar a complexidade dos módulos. Segundo o autor, as métricas de código (número de linhas, número ciclomático de McCabe e métricas de Halstead) consideram apenas aspectos isolados do código e por isso não têm se mostrado confiáveis para medir a complexidade de um programa em todos os seus aspectos. Tal opinião é compartilhada por outros autores Kafura (1985 e 1987), Li (1987) e Weyuker (1988). Em seu estudo, Berns, sugere a atribuição de pesos para cada comando da linguagem (FORTRAN), assumindo que cada comando possui um nível de complexidade específico e que a soma total, dos pesos de cada comando em um programa, representa melhor a dificuldade em entendê-lo. O grande problema aqui é que os pesos atribuídos foram determinados de forma empírica, dando margem a questionamentos e exigindo a obtenção de valores para cada linguagem em que se queira usar esta métrica.

2.3 - Motivação para a Tese

Um dos grandes problemas para a escolha das métricas de complexidade a utilizar é que não há consenso a respeito de quais métricas devam ser selecionadas - Bahia (1988a), Basili (1985), Berns (1984), Henry (1984), Kafura (1985 e 1987), Li (1987) e Poore (1988). Não havendo consenso, a adequação de qualquer métrica de complexidade não pode ser avaliada com precisão e segurança. Há que acrescentar que pessoas diferentes vêem software diferentemente e interpretam sua complexidade também diferentemente.

A complexidade do software é causada por tantos fatores distintos que a medição de apenas alguns deles não é suficiente para a obtenção de resultados confiáveis, quando aplicadas de forma generalizada. Por exemplo, as métricas de tamanho podem ser utilizadas para classificar programas dentro de faixas, mas não

distinguem bem programas dentro da mesma faixa. As métricas baseadas no fluxo de controle pecam por só considerarem este aspecto. Uma alternativa poderia ser a classificação de programas tendo por base seu tamanho e então diferenciá-los através da avaliação do seu fluxo de controle. Gill e Kemerer (1991) e Sagri (1989) argumentam que a complexidade ciclomática de McCabe isoladamente não consegue expressar todos os aspectos relativos à complexidade de um módulo e sugerem extensões alternativas àquela métrica.

Desta forma, concordamos com a proposta de que cada ambiente identifique quais as métricas que melhor traduzem o comportamento do software por ele produzido. Uma ferramenta que se destine a efetuar a análise de aspectos de complexidade deve ser capaz de medir uma vasta gama de elementos associados à complexidade do software.

Ao dispor das características básicas do software produzido e de dados referentes à taxa de erros e ao esforço gasto no desenvolvimento e manutenção dos módulos, o responsável pelo controle da qualidade pode determinar o conjunto de métricas que melhor reflete tais comportamentos. É mais importante que a ferramenta identifique, por exemplo, a quantidade de operadores e operandos (distintos e total), do que calcular todas as métricas de Halstead (esforço, volume, número de erros previstos etc). Isto por dois motivos: primeiro porque as métricas de Halstead, em seu conjunto, podem não traduzir adequadamente o comportamento do software analisado e segundo porque a obtenção destes valores advém da agregação dos valores anteriormente citados.

O estudo de métricas de complexidade aplicado ao código de programas traz o inconveniente de ser efetuado após a fase de codificação, quando então todas as decisões de projeto já terão sido tomadas, tenham elas sido boas ou más. Nesta fase a eliminação das distorções apontadas pelas métricas será sempre mais dispendiosa.

Bahia (1988a) assinala que vários autores entendem ser necessário dispor de métricas que possam ser utilizadas na fase de projeto e que levem em consideração a adequação da estrutura modular do sistema. A literatura apresenta alguns exemplos de analisadores estáticos para códigos, dentre eles podemos citar: SMAP (Software Metrics Analysis Program) e MSTAT (Metrics Statistical Analysis) para códigos em ASSEMBLER (Blaine e Kemmerer, 1985); o MAE (Metrics Analysis Environment) para códigos escritos em COBOL (Harrison, 1988); o MAT (Maintainability Analysis Tool - Berns, 1984b), AUTOMARK E ASSESS (Redish e Smyth, 1986) e SAP (Static Code Analysing Program - Basili et alli, 1983), estes últimos para códigos em FORTRAN.

McCabe e Butler (1989) sugerem a utilização de métricas para avaliar a complexidade do projeto, a estrutura modular e a complexidade para realizar a integração dos módulos. Afirmam dispor de uma ferramenta automatizada que vem sendo aplicada com sucesso em diversos projetos.

Henry e Kafura (1984) usaram uma métrica de complexidade baseada no fluxo de informação entre os módulos, para avaliar o projeto e implementação de um software básico. Desta forma, conseguiu-se detetar vários defeitos no projeto e na fase de implementação. Em certos casos a adição de código, visando introduzir um nível de abstração que faltava (ausência apontada pela métrica) permitiu reduzir a complexidade do sistema como um todo. Terminaram concluindo ser necessário dispor-se de avaliadores de métricas automatizados que possibilitem uma avaliação rápida, objetiva e quantitativa a respeito da estrutura do sistema. Em outro trabalho, Henry e Goff (1989) detalham a utilização de métricas de complexidade aplicadas às especificações de projetos, descritas através de uma linguagem gráfica própria. Objetivam substituir o ciclo **projeto - codificação - análise de métricas - novo projeto** por um outro em que as métricas possam ser avaliadas antes da fase de codificação. Teríamos, então, um novo ciclo definido pelas fases: **projeto - análise das métricas - novo**

projeto e uma vez satisfeitos com o projeto passaríamos à codificação. Desta forma, pode-se eliminar a geração de código desnecessário uma vez que as distorções seriam percebidas em etapa anterior à codificação. Reynolds (1989) mostra a utilização de um analisador estático, PMS (Partial Metrics System), que pode ser aplicado à pseudocódigos. O que se pretende é avaliar como as decisões tomadas ao longo do projeto contribuem para aumentar/diminuir a complexidade estrutural do programa. Desta forma, obtem-se subsídios para que se possa detetar e corrigir problemas durante o desenvolvimento do projeto.

Basili e Perricone (1984), em um estudo para analisar a relação entre a frequência e distribuição de erros com vários fatores ambientais (complexidade, experiência com a aplicação e reutilização de código e projeto), concluíram que independentemente do ambiente e tamanho dos módulos, a maioria dos erros detetados deriva também de uma especificação apresentada de forma inadequada ou mal entendida.

Em função das peculiaridades de cada linguagem, seria tarefa complexa a confecção de uma única ferramenta, parametrizável para qualquer sintaxe, sendo portanto necessária a fixação de uma linguagem alvo. Dadas as características já mencionadas, existentes nos ambientes de desenvolvimento científico, entende-se que a linguagem escolhida deve ser o FORTRAN, deixando-se para uma outra oportunidade a elaboração de ferramentas similares que considerem outras linguagens também utilizadas naqueles ambientes. Para que tal ferramenta possa ser usada já na fase de projeto basta apenas que a sintaxe da linguagem de pseudocódigo utilizada, esteja contida naquela do FORTRAN. Para melhor atender a este objetivo a ferramenta deve permitir uma customização nos elementos da sintaxe, de forma que, um comando READ possa ser percebido, por exemplo, como um comando Leia_Arquivo; um comando CALL por Chama_Rotina, etc.

Propõe-se então a implementação da ferramenta PIÁCATÚ¹, um analisador estático de métricas de complexidade para códigos escritos em FORTRAN (ou pseudocódigos cuja sintaxe esteja contida na do FORTRAN, exceto por algumas extensões). A ferramenta dá também suporte para o gerenciamento de uma base de dados de erros e de custos, permitindo a realização de correlações e regressões estatísticas sobre qualquer par de valores (métricas e erros/custos) especificado pelo usuário.

De modo a garantir a confecção de uma interface amigável com o usuário ela foi projetada em ambiente X-Windows segundo o padrão OSF/Motif. O código do analisador foi escrito na linguagem C e desenvolvido numa estação de trabalho IBM RS/6000 modelo 320H com sistema operacional IBM AIX Version 3 for RISC System/6000.

No próximo capítulo descreve-se, em detalhes, a ferramenta PIÁCATÚ.

¹ piácatú em Tupi significa: simples e pacífico (Masucci, 1979); situação a ser alcançada com sua utilização, já que ela procura identificar módulos complexos.

3. A FERRAMENTA PIÁCATÚ

3.1 - Especificação Informal da PIÁCATÚ

A ferramenta proposta tem como objetivo básico a construção de uma base de dados de métricas de complexidade (BDM) para programas escritos em FORTRAN segundo a sintaxe definida pela IBM (1987) para o produto VS FORTRAN. Produto que foi desenvolvido de acordo com o entendimento e interpretação da IBM para as seguintes normas: American National Standard Programming Language FORTRAN, ANSI X3.9-1978 (também conhecido como FORTRAN 77) e a International Organization for Standardization, ISO 1539-1980 Programming Languages - Fortran.

Valendo-se de uma base de dados de custo de desenvolvimento/manutenção (BDC) e outra de ocorrências de erros (BDE), o usuário pode avaliar a existência de correlações estatisticamente significativas entre valores de métricas e de custos ou erros. Uma vez determinadas quais métricas melhor descrevem o comportamento de um módulo (quantidade de erros, por exemplo), componente (conjunto de módulos) ou sistema (conjunto de componentes) o usuário pode ainda realizar regressões tendo em vista estimar comportamentos do software em questão. A ferramenta dá apoio à confecção e manutenção das bases de custos e erros. Na Figura 6 apresentamos uma visão esquemática da utilização da PIÁCATÚ. Desta forma, pode-se estabelecer um

controle da qualidade sobre o software produzido, utilizando-se métricas de complexidade.

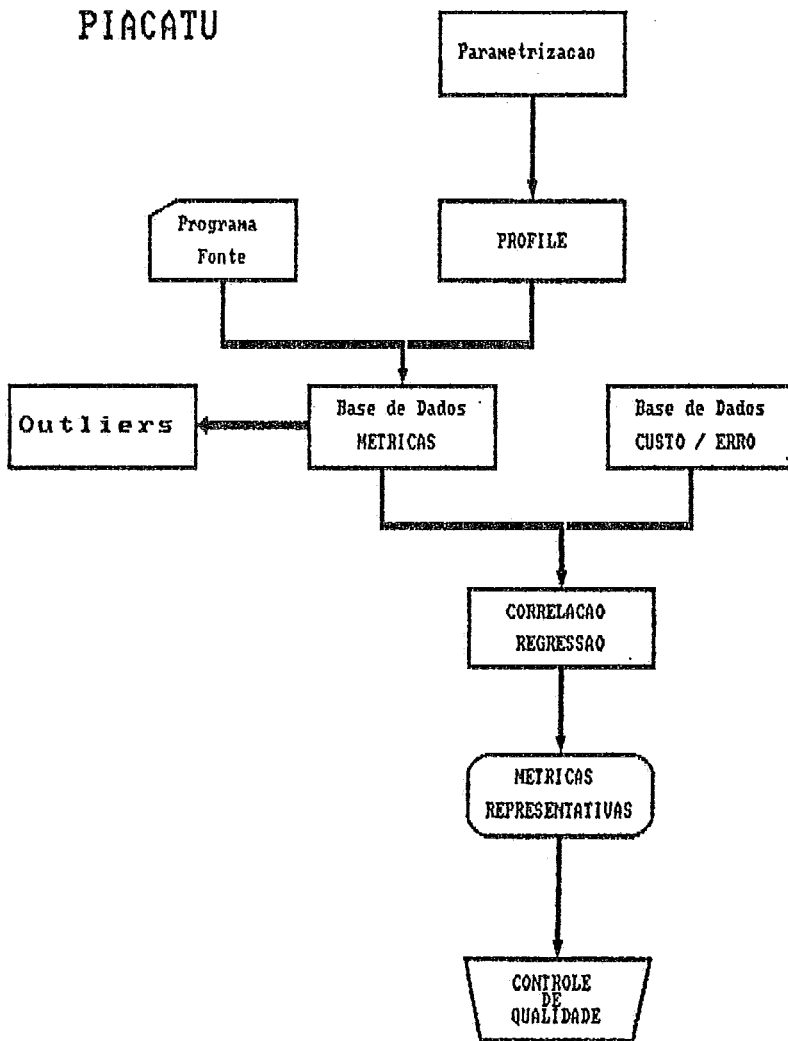


Figura 6. Visão Esquemática da PIACATÚ: Utilização da Piacátú para exercer controle da qualidade sobre o software produzido.

Para permitir um amplo leque de utilização da Piácatú, ela admite diversas alternativas de parametrização, que uma vez exercidas, constituem "*profiles*" que podem ser reutilizados e alterados. De modo a facilitar as ações de parametrização, elas foram agrupadas da seguinte forma: Sintaxe, Componentes, Métricas, Alerta, *Outliers*, Planilhas e Geral.

Cabe ressaltar que através da parametrização da sintaxe pode-se obter um analisador para pseudocódigo. Desta forma, é possível utilizar-se a ferramenta para ter-se valores de métrica ainda na fase de projeto. A inclusão de um analisador sintático foge ao escopo da ferramenta proposta. Assim sendo, é necessário que o código/pseudocódigo se apresente sintaticamente correto para que os cálculos sejam devidamente efetuados. O Apêndice B mostra de forma resumida os elementos da linguagem e a sintaxe requerida.

A ferramenta guarda um registro de controle indicando que opções foram exercidas na geração do arquivo de métricas. Desta maneira, impedirá que o usuário, futuramente, solicite operações com métricas não previamente calculadas. Os arquivos de métricas, custos e erros terão que ser compatíveis no que diz respeito à sua constituição em termos de módulos e componentes.

As métricas disponíveis podem ser apresentadas ao usuário para seleção, segundo duas alternativas. Na primeira as métricas estão agrupadas de acordo com o tipo de característica de complexidade que elas objetivam mensurar (Tamanho, Fluxo de Controle, Estrutura Modular, Estrutura de Dados, Entrada/Saída de Dados, Documentação e Agregadas). No segundo caso o usuário acessa grupos de métricas (Primário, Mediano e Extenso) que permitem obter níveis crescentes de caracterização da complexidade. O usuário, através de qualquer abordagem, customiza um novo grupo, denominado Usuário, que será armazenado em seu "*profile*". Existe para cada métrica uma descrição para auxiliar o usuário durante a fase de seleção.

Uma vez encerrada a etapa de parametrização o usuário passa à fase de cálculo. Tendo as métricas sido previamente selecionadas, ele fornece apenas o nome do "*profile*" a ser utilizado. Para cada métrica será calculado seu valor médio e o desvio padrão para cada componente e para o sistema como um todo. De posse destes valores e da definição de "*outliers*", estes podem ser devidamente identificados tendo como referência o componente ou sistema a que pertencem.

Uma vez calculados e armazenados, o usuário pode solicitar a apresentação dos dados contidos numa BDM via tela ou direcioná-los para impressão. Na apresentação os módulos podem ser dispostos hierarquicamente, segundo sua estrutura modular, ou em ordem alfabética. Os "*outliers*" serão devidamente destacados. Na tela o usuário pode expandir ou contrair os elementos mostrados de maneira a facilitar a visão do conjunto.

O programa fonte pode ser fornecido em letra minúscula ou maiúscula já que internamente toda linha é convertida e tratada como maiúscula. Durante os cálculos será impresso o fonte analisado, na forma como foi submetido, onde cada linha aparece numerada sequencialmente. Se no arquivo do programa fonte não for fornecido um programa principal (main) a estrutura modular não poderá ser determinada.

Para realizar o tratamento estatístico o usuário deve primeiro carregar a base de dados de erros (BDE) ou de custos (BDC). Para tanto, deve se valer das funções de definição das planilhas de erros/custos e posteriormente carregar os valores para cada módulo/componente de forma individual (apenas um módulo ou componente) ou de forma global (hierárquica ou alfabeticamente). A estrutura modular é obtida, via comando do usuário, após o processamento para análise das métricas.

Uma vez calculadas as métricas e dispondo de uma BDE ou BDC, o usuário pode solicitar a realização de correlações e/ou regressões sobre pares ou conjuntos de pares de valores de métricas e erros e/ou custos.

3.2 - Projeto da Interface com o Usuário

A interface foi projetada no ambiente X-Windows de acordo com o padrão OSF/MOTIF. Objetivando um melhor entendimento do funcionamento da PIÁCATÚ descrevemos, a seguir, as telas que permitem a iteração com o usuário.

Nas telas de diálogo, sempre que pertinentes, foram dispostos três botões com funções padronizadas: **C**onfirma , para aceitar uma ou mais das opções exercidas no diálogo e retornar o controle para a tela anterior; **C**ancela , para desconsiderar as alterações efetuadas e retornar o controle para a tela anterior; **A**juda , que fornecerá informações explicativas dentro do contexto específico.

Após a execução do comando:

> piacatu

Será exibida a tela inicial mostrada na Figura 7

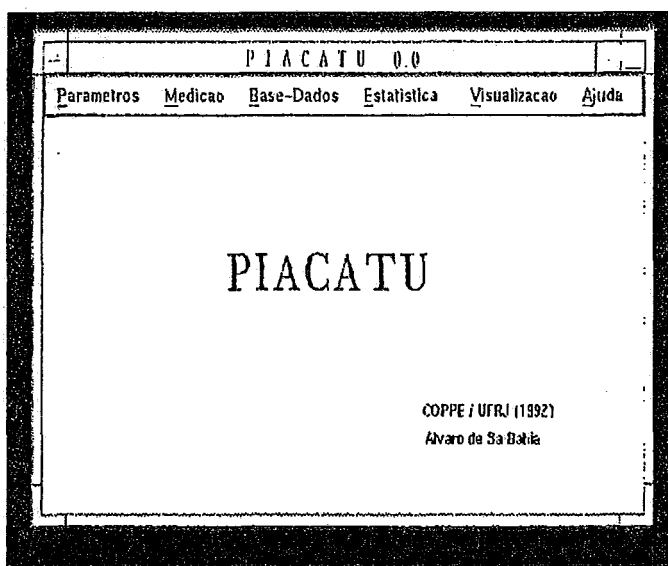


Figura 7. Tela Inicial: Tela apresentada após execução do comando: *piacatu*

3.2.1 - Opção Parâmetros

Para realizar as operações de parametrização o usuário deve selecionar a opção Parametros ² com o que são apresentadas as diversas alternativas disponíveis (ver Figura 8):

² Observação: Nas telas da interface as palavras não aparecem acentuadas. Para manter compatibilidade, todas as referências do texto a elementos específicos da tela aparecem indicadas em **negrito** ou em *itálico* e sem acentuação.

Parametros

Profile

Arquivos

Sintaxe

Componentes

Metricas

Alerta

Outliers

Planilhas

Geral

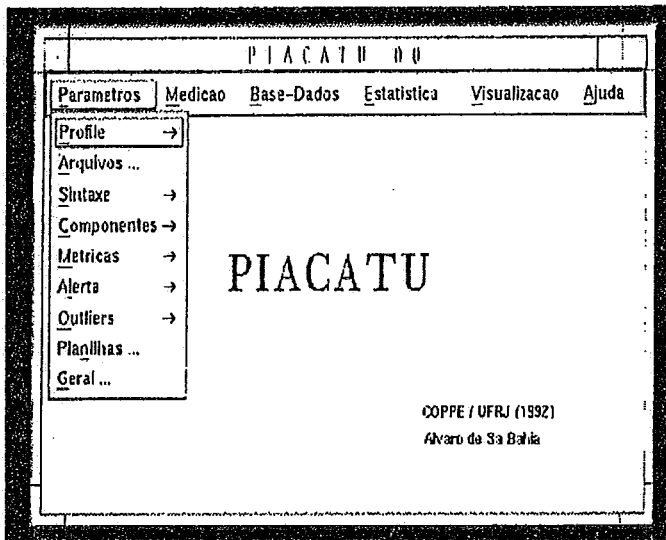


Figura 8. Opção Parâmetros: Tela obtida após a seleção de *Parametros*.

3.2.1.1 - Subopção Profile

Para realizar as operações de seleção ou armazenamento de "profiles" o usuário deve selecionar a opção Profile quando são então oferecidas as seguintes alternativas:

Profile

Novo

Abrir

Salva

Salva Como

Para indicar a criação de um novo "profile" o usuário deve selecionar Novo . É então mostrado o diálogo da Figura 9 para que o usuário forneça o nome do "profile" a ser utilizado.

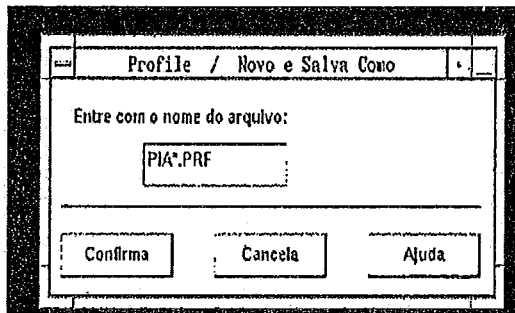


Figura 9. Diálogo Profile / Novo e Salva Como: Diálogo para fornecimento do nome do "profile".

Para indicar a utilização de um "profile" existente o usuário deve selecionar Abrir . É, então, mostrado o diálogo da Figura 10 para que o usuário escolha o nome do "profile" a ser utilizado. Na lista aparecerá, sempre, um "profile" padrão para pseudocódigo (PIAPSEUDO.PRF) e um para código FORTRAN (PIACODIGO.PRF) que podem

ser utilizados como modelos. Não é permitido salvar "profiles" com qualquer destes dois nomes.

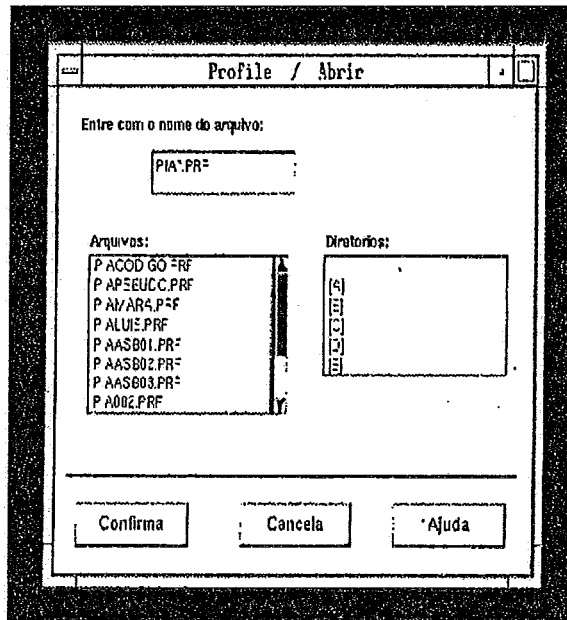


Figura 10. Diálogo Profile / Abrir: Diálogo para selecionar um "profile" já existente.

Ao terminar a confecção do "profile" o usuário deve salvá-lo. Para tanto, pode escolher a opção Salva ou Salva Como. No primeiro caso o "profile" é salvo com o nome corrente, no segundo é apresentado o diálogo da Figura 9 para que seja indicado um novo nome.

3.2.1.2 - Subopção Arquivos

Para indicar os nomes dos arquivos envolvidos nas diversas etapas do processamento o usuário deve selecionar a opção Arquivos quando é então mostrado o diálogo da Figura 11. Não é necessário que neste momento os arquivos já existam.

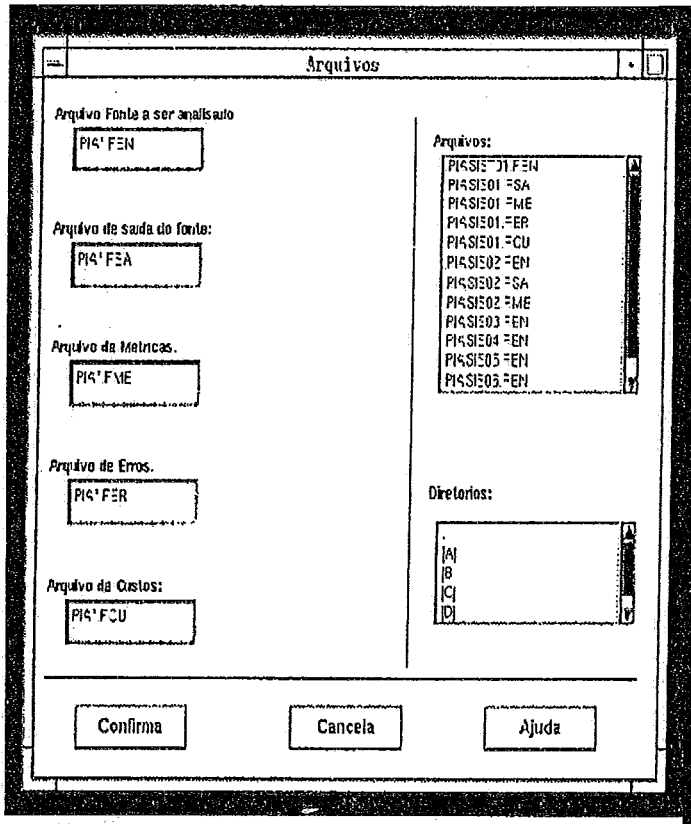


Figura 11. Diálogo Arquivos: Diálogo para informar nomes de arquivos.

Ao ser apresentado, o diálogo conterà para cada arquivo um nome do tipo PIA*.qualificador. Uma vez especificado o primeiro *, este valor será usado como "default" para os outros nomes, onde ainda não tiver sido alterado. Ao se estabelecer o "focus" para qualquer campo de nome de arquivo são apresentados à direita do campo dois botões: **OK** e **Cancela**. O usuário deve selecionar a primeira opção quando quiser confirmar o nome fornecido e a segunda caso contrário.

A lista de arquivos apresentada se referencia ao campo de arquivo que possuir o "focus". Uma vez designado um diretório para um determinado arquivo, o

mesmo diretório é usado para os outros, a menos que o usuário não o queira e decida modificá-lo.

3.2.1.3 - Subopção Sintaxe

Para realizar as parametrizações nos elementos da sintaxe o usuário deve selecionar a opção Sintaxe quando são então oferecidas as seguintes alternativas:

Sintaxe

Comandos

Operadores

Funcoes

Comentarios

Linha

A partir do conjunto de comandos e operadores relacionais existentes na linguagem FORTRAN, como definido em IBM (1987), o usuário pode alterar o nome pelo qual a ferramenta reconhece cada comando ou operador. Assim, por exemplo, um comando READ pode ser representado por GET ou Leia_Arquivo e o operador relacional .LE. (Less than or Equal) por .MenorIgual.

À exceção das palavras compostas já admitidas na sintaxe FORTRAN (BLOCK DATA, DOUBLE PRECISION, END DO, END IF, GO TO, TRACE OFF e TRACE ON), as designadas pelo usuário devem ser separadas por hífen ou travessão.

Ao selecionar a opção Comandos , Operadores ou Funcoes é apresentado o diálogo da Figura 12 com a lista de comandos, operadores relacionais ou funções,

respectivamente. Os operadores relacionais representados por: < , > , <= , >= , <> e = não são passíveis de alteração.

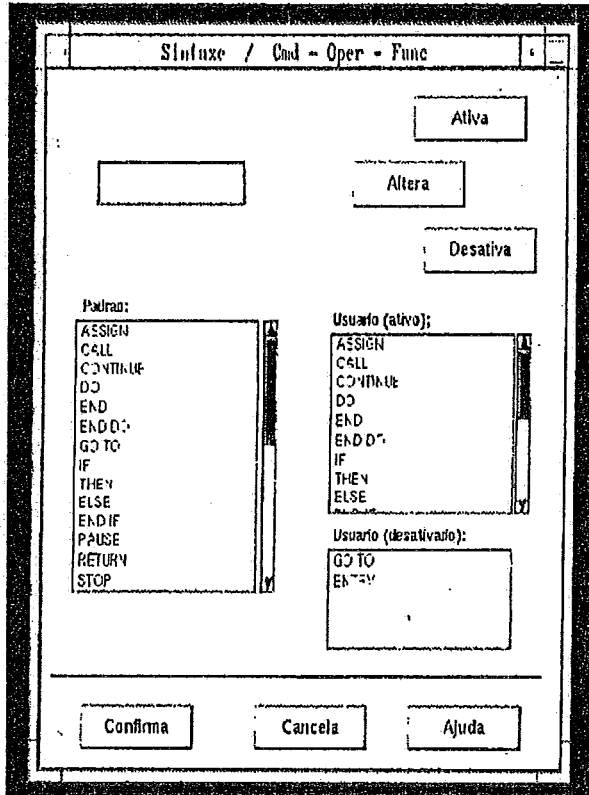


Figura 12. Diálogo Sintaxe / Cmd - Oper - Func: Diálogo para parametrizar nomes de comandos, operadores relacionais ou funções.

Na figura 12 o usuário pode alterar o nome do comando/operador/função e opcionalmente ativar ou desativar determinado comando/função. Quando determinado comando/função é desativado ele passa a ser ignorado em todas as linhas em que aparecer no programa fonte. Não é possível desativar operadores. Ao longo do processo o usuário pode acompanhar as listas de elementos ativos e daqueles desativados.

Ao selecionar **Comentários** ou **Linha** é apresentado o diálogo da Figura 13 para que seja definida a sintaxe do cartão de comentário e a estrutura da linha do programa fonte. O usuário deve informar o carácter e a coluna que indicam ser o cartão de comentário. Devem ser especificadas sempre duas definições, se o usuário desejar apenas uma deve torná-las iguais. Cabe ressaltar que, durante o processamento da linha, em primeiro lugar verifica-se se ela é ou não de comentário (em caso afirmativo ela é ignorada).

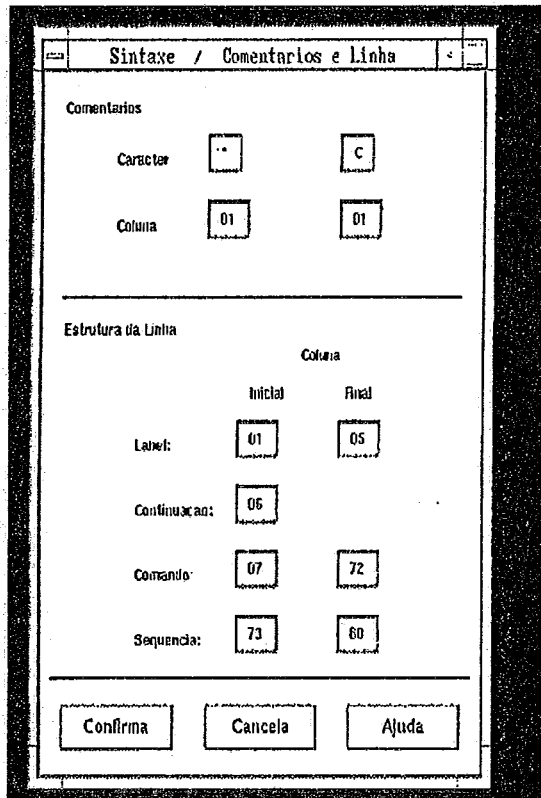


Figura 13. Diálogo Sintaxe / Comentários e Linha: Diálogo para parametrizar sintaxe de comentário e estrutura da linha.

Ao definir a estrutura da linha o usuário deve garantir que a coluna inicial é menor que a final, que não há superposição de colunas e que o intervalo contém pelo

menos uma coluna. É necessário, também, que as colunas de "label" antecedam a de continuação, esta a de comandos e por fim que estas sejam anteriores às de seqüência (isto é, "label" < continuação < comando < seqüência). A coluna final deve ser 80.

3.2.1.4 - Subopção Componentes

Se necessário, o usuário pode fornecer uma definição de componentes (conjunto de módulos que constituem uma subdivisão lógica ou funcional do sistema) de modo a agregar valores de métricas/custos/erros por componentes.

Para realizar a definição ou redefinição dos componentes o usuário deve selecionar a opção **C**omponentes quando são então oferecidas as seguintes alternativas:

Componentes

Definicao

Redefinicao

Ao selecionar **D**efinicao é apresentado o diálogo da Figura 14 com a lista dos módulos existentes no sistema. O usuário fornece o nome do componente a ser criado e adiciona ou apaga módulos até completar a definição. Pode então salvar a definição com o nome corrente (**S**alva) ou com outro nome (**S**alva **C**omo), neste caso é mostrado o diálogo da Figura 16.

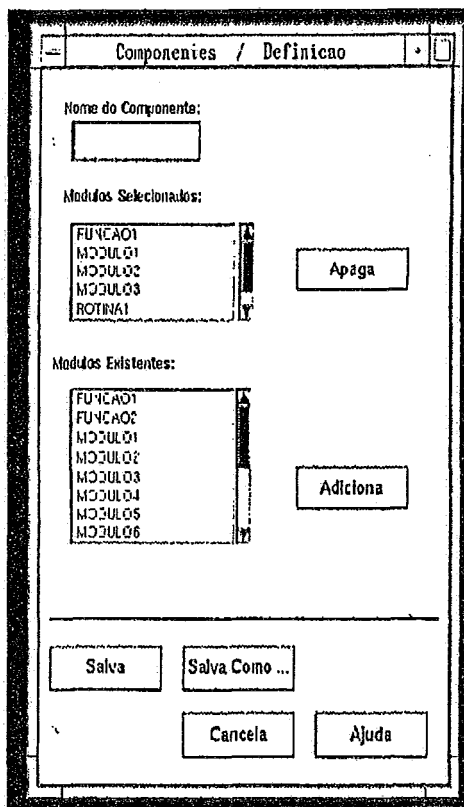


Figura 14. Diálogo Componentes / Definição: Diálogo para definir os módulos que constituem um componente.

Ao selecionar Redefinição é apresentado o diálogo da Figura 15 com a lista de componentes existentes. O usuário seleciona então um componente para ter sua definição alterada. Após a confirmação da seleção o controle é passado para o diálogo da Figura 14.

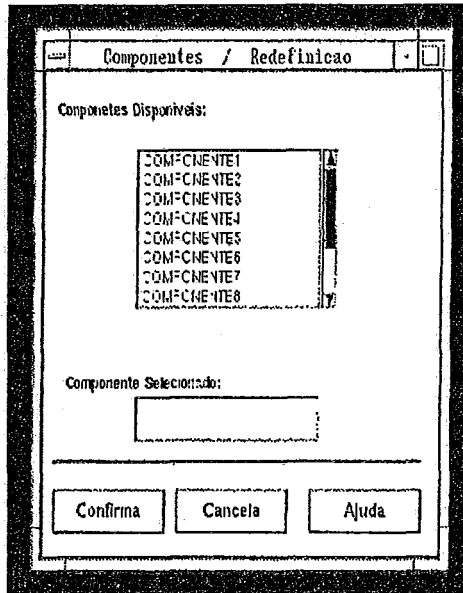


Figura 15. Diálogo Componentes / Redefinição: Diálogo para iniciar redefinição de componentes.

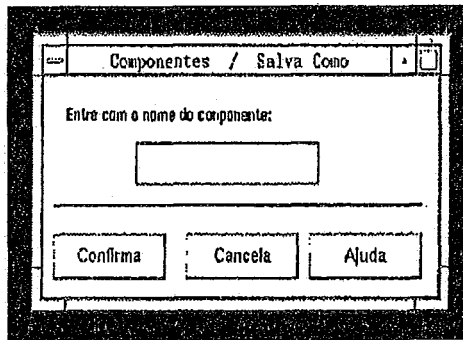


Figura 16. Diálogo Componentes / Salva Como: Diálogo para indicar o novo nome para salvar a definição do componente.

3.2.1.5 - Subopção Métricas

Para realizar a escolha do grupo de métricas a ser analisado, o usuário deve selecionar a opção Métricas quando são então oferecidas as seguintes alternativas:

Métricas

Modificacao

Ativacao / Desativacao

Ao selecionar qualquer das opções apresentadas, é mostrado o diálogo da Figura 17 onde o usuário pode customizar o conjunto de métricas que constituirão o grupo USUÁRIO e, dentro deste, quais métricas estarão ativas (serão efetivamente calculadas) e quais estarão desativadas (farão parte do "profile" mas não serão calculadas).

Ao ser ativado o diálogo da Figura 17 a lista **Métricas Disponíveis**, apresenta as métricas que compõem o grupo USUÁRIO, se houver, e caso contrário as do grupo PRIMÁRIO.

O usuário pode selecionar a partir de qual grupo ou tipo ele deseja selecionar métricas. Para tanto, deve informar o nome do grupo ou tipo desejado ou acionar a seta correspondente, sendo que neste caso é apresentada uma lista com as opções possíveis.

Ao selecionar uma métrica em qualquer das listas, uma descrição sucinta da métrica é exibida no campo correspondente.

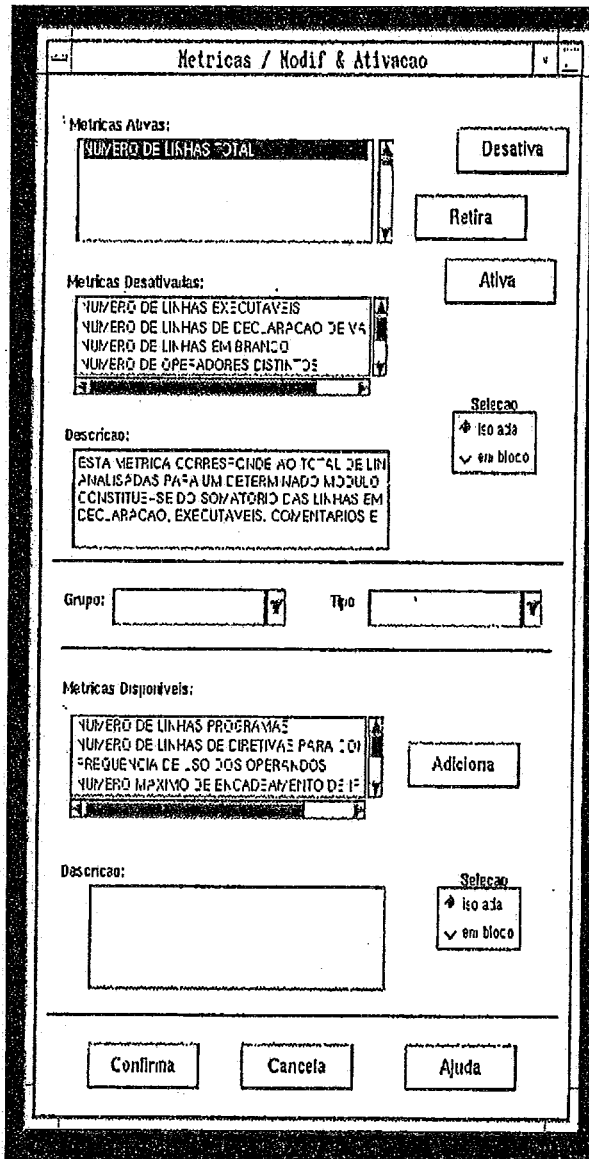


Figura 17. Diálogo Métricas / Modif & Ativação: Diálogo para customizar grupo de métricas a serem analisadas.

A seleção para adicionar, retirar, reativar ou desativar métricas pode ocorrer uma a uma ou em bloco, para tanto o usuário deve indicar o tipo de seleção desejada (isolada ou em bloco).

Ao selecionar métricas agregadas a interface garante que as métricas primitivas necessárias à sua determinação sejam também escolhidas. Por sua vez, as métricas primitivas só são retiradas do grupo se explicitamente selecionadas.

Uma vez selecionada para compor o grupo USUÁRIO a métrica deixa de pertencer aos grupos originais evitando que o usuário venha a escolhê-la uma segunda vez. No Apêndice C é apresentada a lista geral de métricas disponíveis bem como a classificação por tipo e a constituição dos grupos.

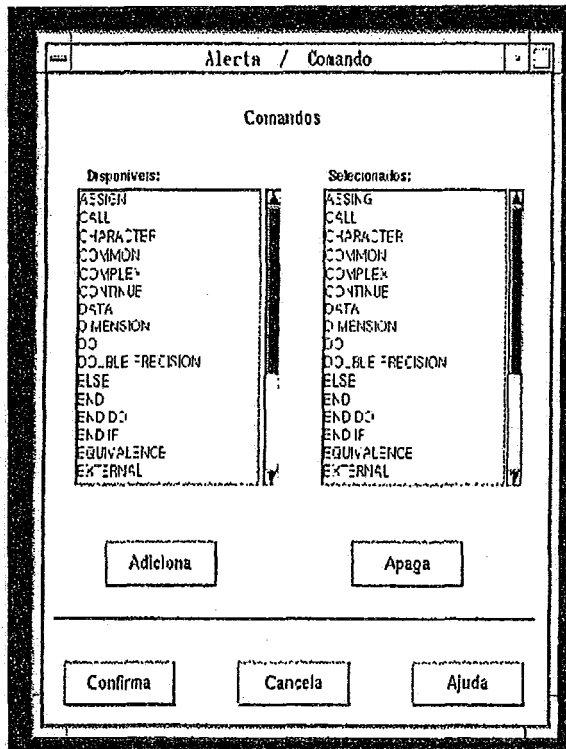


Figura 18. Diálogo Alerta / Comando: Diálogo para solicitar alerta para comandos específicos.

3.2.1.6 - Subopção Alerta

Para facilitar a identificação de estruturas indesejáveis o usuário deve selecionar a opção **A**lerta quando são então oferecidas as seguintes alternativas:

Alerta

Comandos

Labels

Ao selecionar Comandos é mostrado o diálogo da Figura 18 onde o usuário pode criar uma lista de comandos para os quais será destacada (alertada) sua presença no programa fonte.

Ao selecionar Labels o usuário indica, no diálogo da Figura 19, se deseja mensagem de alerta na ocorrência de "labels" fora de seqüência (ascendente ou descendente) no programa.

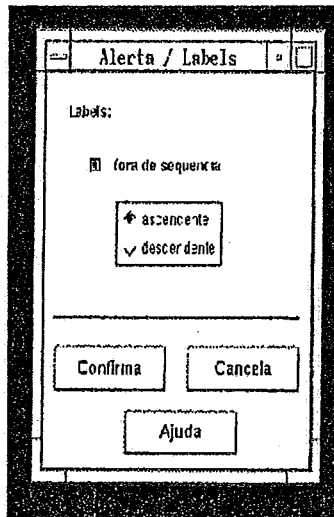


Figura 19. Diálogo Alerta / Labels: Diálogo para solicitar alerta para "labels" fora de seqüência.

3.2.1.7 - Subopção Outliers

Para permitir a customização do conceito de "outlier" e ativar/desativar o envio de mensagem de alerta na ocorrência de módulos "outliers", o usuário deve selecionar a opção Outliers quando são então oferecidas as seguintes alternativas:

Outliers

Definicao

Alerta

Ao selecionar qualquer das alternativas apresentadas é mostrado o diálogo da Figura 20. Lá o usuário pode definir quando um valor de métrica será considerado "outlier" (tendo por base o seu afastamento com relação à média do sistema). A condição de alerta também pode ser ativada/desativada neste diálogo.

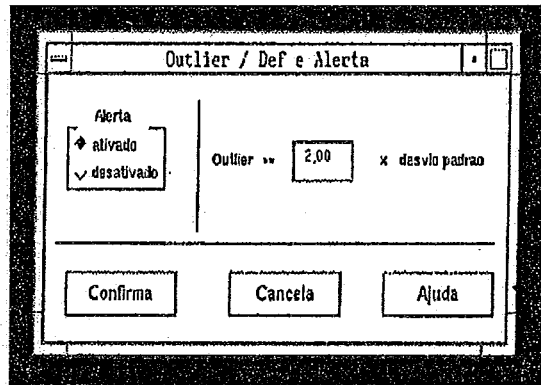


Figura 20. Diálogo Outlier / Def e Alerta: Diálogo para alerta e definição de "outliers".

3.2.1.8 - Subopção Planilhas

Para permitir a customização da disposição física dos arquivos que conterão

as bases de dados de custos e/ou de erros o usuário deve selecionar a opção **Planilhas** quando é então mostrado o diálogo da Figura 21.

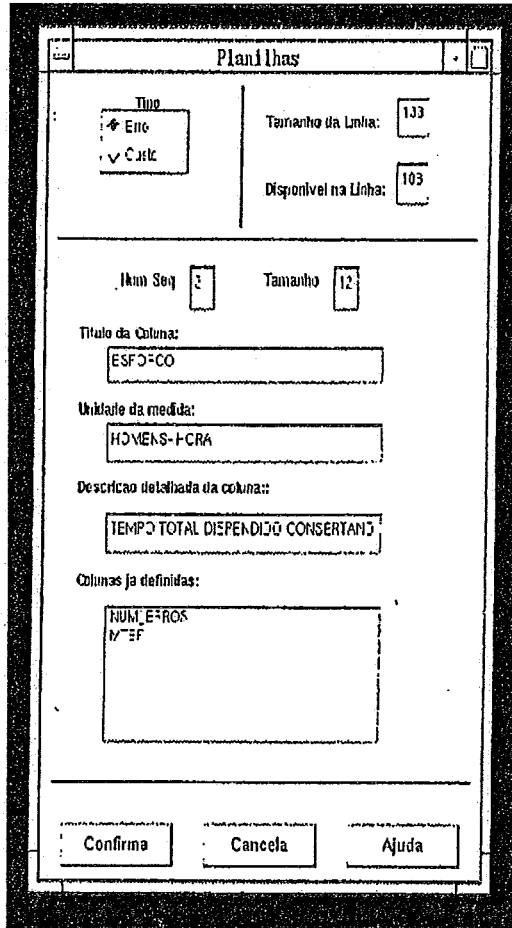


Figura 21. Diálogo Planilhas: Diálogo para definição de planilhas de erros e custos.

Inicialmente o usuário deve indicar o tipo da planilha (erro / custo). Havendo uma definição anterior, as colunas já definidas são apresentadas. O procedimento de customização, dos arquivos de erros e de custos, envolve a atribuição de um número de seqüência para a coluna, a especificação do seu tamanho (suficiente ao menos para armazenar o título da coluna), o título da coluna, sua unidade de medida e uma descrição mais detalhada do significado da coluna. No canto superior

direito do diálogo são mostrados o tamanho máximo especificado para a linha e a quantidade de colunas ainda disponível.

3.2.1.9 - Subopção Geral

Neste diálogo (ver Figura 22) foram agrupadas diversas opções que podem ser exercidas ao submeter-se um "profile" para o processamento das métricas.

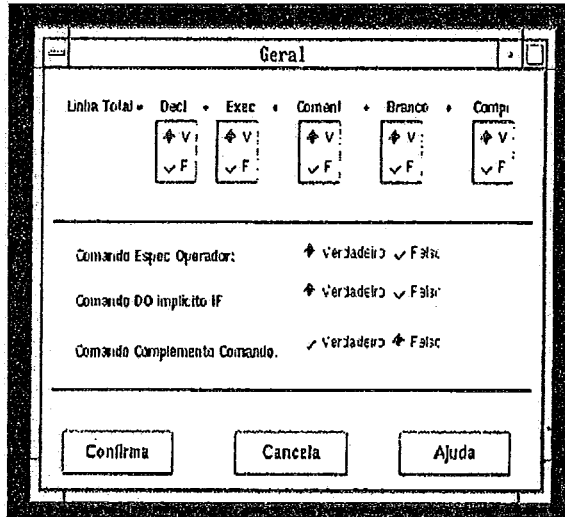


Figura 22. Diálogo Geral: Diálogo para customizações diversas.

3.2.2 - Opção Medição

Uma vez tendo sido construído um "profile", o usuário ao selecionar a opção Medicao iniciará o cálculo das métricas. Enquanto aguarda o fim do processamento, o usuário pode monitorar o andamento dos trabalhos através de um diálogo que mostra o tempo decorrido, nome do módulo sendo analisado e quantidade de linhas já processadas.

3.2.3 - Opção *Base-Dados*

Para carregar a BDE ou BDC, o usuário deve selecionar a opção **Base-Dados** quando são então oferecidas as seguintes alternativas (a Figura 23 mostra a tela correspondente):

Base-Dados

Monta-Estrutura

Alfabetica

Hierarquica

Individual

Global

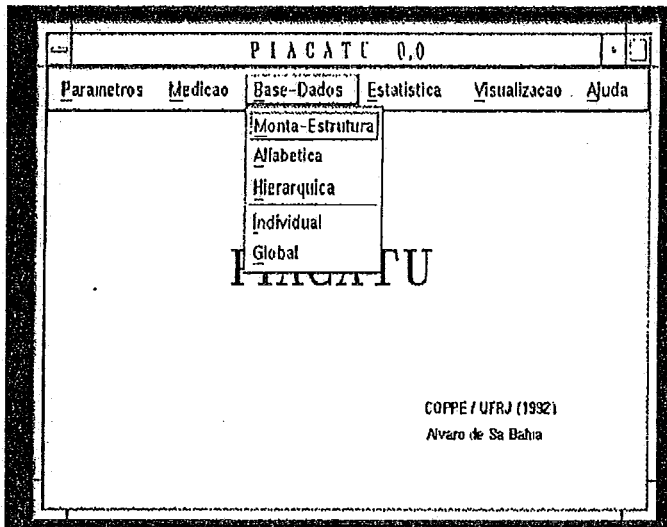


Figura 23. Tela Inicial - Base-Dados: Tela após seleção de *Base-Dados*.

3.2.4 - Opção Estatística

Para realizar correlações e regressões estatísticas o usuário deve selecionar a opção Estatística quando são então oferecidas as seguintes alternativas (ver também a Figura 24)

Estatística

Matriz

Correlacao

Regressao

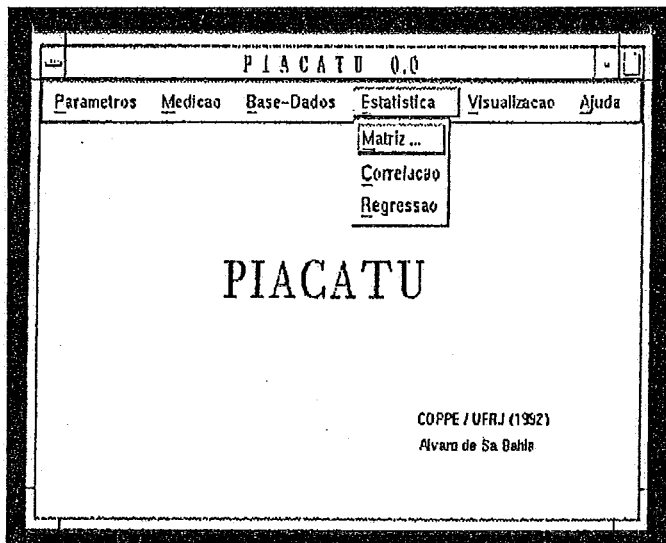


Figura 24. Tela Inicial - Estatística: Tela após seleção de *Estatística*.

Na opção **Matriz** o usuário indica com quais valores de métricas e custos/erros deseja efetuar a correlação ou regressão. As opções seguintes, ao serem selecionadas, iniciam o processamento indicado.

3.2.5 - Opção *Visualização*

Para analisar os resultados do processamento o usuário deve selecionar a opção **Visualizacao** quando são então oferecidas as seguintes alternativas (ver também a Figura 25)

Visualizacao

Alfabetica

Hierarquica

Filtro

Destaque

Tela

Impressora

As duas primeiras opções indicam a forma de visualização pretendida. Ao selecionar a opção **Filtro** o usuário estabelece condições para que determinado elemento seja apresentado. Pode-se estabelecer filtros para: nome de módulo, módulos associados a um componente, valores de métricas e módulos "outliers". A opção **Destaque** determina que os módulos "outliers" sejam destacados na apresentação. As duas últimas opções direcionam os elementos selecionados para a tela ou para a impressora.

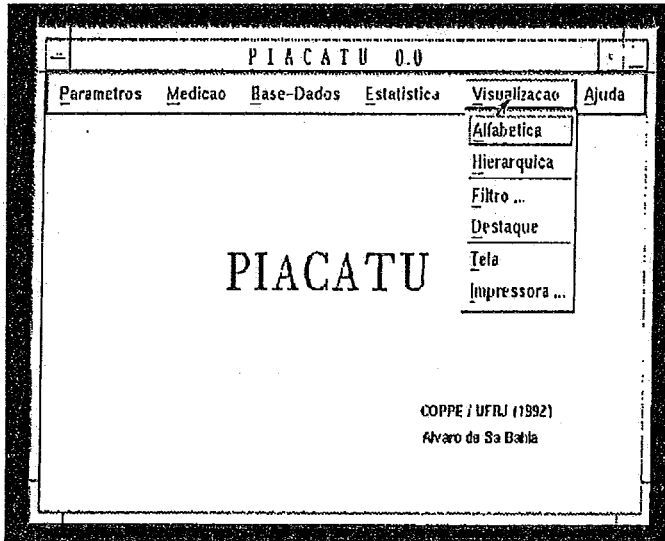


Figura 25. Tela Inicial - Visualização: Tela após seleção de *Visualizacao*.

3.2.6 - Opção Ajuda

Para auxiliar o usuário em caso de dúvidas ele pode requerer Ajuda . Desta maneira, ele dispõe de auxílios contextuais que o ajudam na execução das diversas ações necessárias para a plena utilização da Piácatú.

3.3 - Exemplo de Utilização

Para ilustrar o uso da Piácatú foram avaliados um programa (MUTMETA) e duas versões de pseudocódigo que o antecederam.

O objetivo do programa é determinar os números múltiplos de 3 menores ou iguais à metade de um determinado número fornecido pelo usuário. Ele deve

especificar, também, se deseja a impressão da lista de números ou se apenas da quantidade de números determinada.

Os códigos apresentados não têm a pretensão de ser a melhor forma de resolver o problema descrito mas apenas servir de meio para mostrar a funcionalidade da ferramenta proposta. O que se busca é mostrar a possibilidade de acompanhar-se o crescimento da complexidade, na medida em que o programa evolui.

Na Versão 1 temos apenas um esboço da solução do problema com chamadas a sub-rotinas e a presença das variáveis globais necessárias (ver Figura 26).

```
          Inicia Main
*-----*
*
* Versao 1
*
* Dado um numero N, determinar os numeros multiplos de 3
* menores ou igual metade de N.
* O usuario deve especificar se deseja a impressao da
* lista de numeros ou se apenas da quantidade total
*
          Declara_Variavel NUM, N, OPT, LISTA
          Chama_Rotina Le_Parametros (NUM, OPT)
          Chama_Rotina Calcula_Numeros (NUM, N, LISTA)
          Chama_Rotina Imprime_Resultados (OPT, LISTA)
          Fim
```

Figura 26. Versão 1 - Programa MUTMETA: Pseudocódigo inicial

Na versão mostrada na Figura 27 temos o pseudocódigo praticamente concluído.

```

      Inicia Main
*-----
*
* Versao 2
*
* Dado um numero N, determina os numeros multiplos de 3 menores ou
* igual a metade de N.
* O usuario deve especificar se deseja a impressao da
* lista de numeros ou se apenas da quantidade total
*
      Caracter OPT
      Inteiro N, QPRI, LISTA(10)
      Area_Global /Areal/ LISTA

      Chama_Rotina Le_Parametros (NUM, OPT)
      Chama_Rotina Calcula_Numeros (NUM, N)
      Chama_Rotina Imprime_Resultados (NUM, N, OPT)

      Para
      Fim

      Rotina Le_Parametros (N, OPT)
*-----
      Caracter OPT
      Inteiro N

      Leia (arqin) N, OPT

      Retorna_Controlo
      Fim

      Rotina Calcula_Numeros(NUM, I)
*-----
      Inteiro I, NUM
      Area_Global /Areal/ LISTA

      Limite = ( NUM / 2 ) + 1
      I = 0

      Faca Labell II=1, Limite
      Resto = Resto_da_Divisao(II,3)
      Se ( Resto = Zero )
      Se ( I > 10 )
      Chama_Rotina Erro(NUM)
      senao
      I = I + 1
      LISTA(II) = I
      Fim Se
      Fim Se

      Labell Fim Faca

      Retorna_Controlo
      Fim

      Rotina Imprime_Resultados (NUM, N, OPT)
*-----
      Caracter OPT
      Inteiro N, NUM, LISTA(1)
      Area_Global /Areal/ LISTA

      Faca Enquanto (I .MenorIgual. N)
      Escreva (arqout) I, LISTA(I)
      I = I + 1

      Fim Faca

      Retorna_Controlo
      Fim

```

Figura 27. Versão 2 - Programa MUTMETA: Pseudocódigo praticamente concluído.

Na Figura 28 apresentamos o código FORTRAN que implementa uma solução para o problema.

```
PROGRAM MAIN
*-----
CHARACTER*3 OPT
INTEGER LISTA(10), N, NUM
COMMON /AREA1/ LISTA
CALL LEPARA (NUM, OPT)
CALL CALNUM (NUM, N)
CALL IMPRES (NUM, N, OPT)
STOP
END
SUBROUTINE LEPARA (NUM, OPT)
*-----
* Esta rotina le o numero para o qual se quer determinar os
* multiplos de 3 (NUM) e a opcao de impressao (OPT).
CHARACTER*3 OPT
INTEGER NUM
10 READ (5,10) NUM, OPT
FORMAT(I6, A3)
RETURN
END
SUBROUTINE CALNUM (NUM, I)
*-----
* Esta rotina recebe NUM e retorna em LISTA os numeros que
* sao multiplos de 3 e menores que a metade do numero lido.
* Em I eh retornado a quantidade de multiplos encontrada.
INTEGER LISTA, I, NUM, RESTO
COMMON /AREA1/ LISTA(1)
LIMITE = ( NUM / 2 ) + 1
I = 0
DO 100 II=1, LIMITE
  RESTO = MOD(II,3)
  IF ( RESTO .EQ. 0 ) THEN
    IF ( I .GT. 10 ) THEN
C      se for maior que 10 estoura o array LISTA
      CALL ERRO(NUM)
    ELSE
      I = I + 1
      LISTA(I) = II
    END IF
  END DO
END DO
100 RETURN
END
SUBROUTINE IMPRES (NUM, N, OPT)
*-----
* Esta rotina imprime o resultado de acordo com a opcao (OPT)
CHARACTER*3 OPT
INTEGER N, NUM, LISTA
COMMON /AREA1/ LISTA(1)
I = 1
WRITE(6, 20) NUM
IF (OPT .NE. 'NA' ) THEN
  WRITE(6, 21) N
  DO WHILE ( I .LE. N)
    WRITE(6, 22) I, LISTA(I)
    I = I + 1
  END DO
ELSE
  WRITE(6, 21) N
END IF
RETURN
20 FORMAT(//' NUMERO ESCOLHIDO ..', I6)
21 FORMAT(//' MULTIPLOS DE TRES MENOR QUE A METADE DO ',
1 ' NUMERO LID , i6)
22 FORMAT(2I4)
END
SUBROUTINE ERRO (NUM)
*-----
INTEGER NUM
20 WRITE(6,20) NUM
1 FORMAT(' MEMORIA ALOCADA NAO PERMITE CALCULO PARA NUMERO PEDID ,
1 I6)
STOP 1
END
```

Figura 28. Versão 3 - Programa MUTMETA: Programa FORTRAN implementado.

As Figuras 29 a 33 apresentam, para cada uma das três versões, a descrição dos operandos e operadores obtida pela Piácatú para cada um dos módulos que

constituem o programa MUTMETA (MAIN, LEPARA, CALNUM, IMPRES e ERRO).

O P E R A N D O S											O P E R A D O R E S		
ver	nome	tp	td	dc	ct	es	fg	in	fm	io	ver	nome	ct
V1	N	1	0	0	1	1	2	16	16	0	V1	CHAMA_ROTINA	3
V1	NUM	1	0	0	2	1	2	14	16	0			
V1	OPT	1	0	0	2	1	2	14	18	0			
V1	LISTA	1	0	0	2	1	2	16	18	0			
V1	LE_PARAMETROS	6	0	0	1	1	0	14	14	0			
V1	CALCULA_NUMEROS	6	0	0	1	1	0	16	16	0			
V1	IMPRIME_RESULTADOS	6	0	0	1	1	0	18	18	0			
V2	N	1	1	1	3	1	2	13	20	0	V2	CHAMA_ROTINA	3
V2	NUM	1	0	0	3	1	2	16	20	0			
V2	OPT	1	4	1	3	1	2	12	20	0			
V2	QPRI	1	1	1	1	0	0	13	13	0			
V2	AREAL	4	0	1	1	1	1	14	14	0			
V2	LISTA	1	1	1	1	1	1	13	13	0			
V2	LE_PARAMETROS	6	0	0	1	1	0	16	16	0			
V2	CALCULA_NUMEROS	6	0	0	1	1	0	18	18	0			
V2	IMPRIME_RESULTADOS	6	0	0	1	1	0	20	20	0			
V3	N	1	1	1	3	1	2	5	12	0	V3	CALL	3
V3	NUM	1	1	1	4	1	2	5	12	0	V3	STOP	1
V3	OPT	1	4	1	3	1	2	4	12	0			
V3	AREAL	4	0	1	1	1	1	6	6	0			
V3	LISTA	1	1	1	1	1	1	5	5	0			
V3	LEPARA	6	0	0	1	1	0	8	8	0			
V3	CALNUM	6	0	0	1	1	0	10	10	0			
V3	IMPRES	6	0	0	1	1	0	12	12	0			

onde:

- ver - numero da versao
- nome - nome do operando ou operador
- tp - tipo do operando (1=variavel, 2=palavra-chave, 3=label
4=operando, 5=constante, 6=nome-rotina e
7=arquivo)
- td - tipo do dado (1=inteiro, 2=real, 3=complexo, 4=char,
5=logico e 6=double)
- dc - variavel declarada antes de usada
- ct - contador de ocorrencias
- es - escopo da variavel (0=local e 1=global)
- fg - forma global(1=common e 2=lista-parametros)
- in - linha da referencia inicial
- fm - linha da referencia final
- io - (1=read, 2=write e 3=read/write)

Figura 29. Programa principal (MAIN): Operandos e operadores nas três versões

O P E R A N D O S											O P E R A D O R E S		
ver	nome	tp	td	dc	ct	es	fg	in	fm	io	ver	nome	ct
V2	N	1	1	1	3	1	2	6	8	1	V2	LEIA	1
V2	OPT	1	4	1	3	1	2	5	8	1	V2	RETORNA_CONTROLE	1
V3	5%A	7	0	0	1	0	0	11	11	0	V3	READ	1
V3	10%L	3	0	1	2	0	0	11	12	0	V3	RETURN	1
V3	NUM	1	1	1	3	1	2	9	11	1			
V3	OPT	1	4	1	3	1	2	8	11	1			

Figura 30. Sub-rotina LEPARA (Le-Parametros): Operandos e operadores nas versões 2 e 3

O P E R A N D O S											O P E R A D O R E S			
ver	nome	tp	td	dc	ct	es	fg	in	fm	io	ver	nome	ct	
V2		10	1	0	0	1	0	0	14	14	0	V2	CHAMA_ROTINA	1
V2		0	1	0	0	1	0	0	9	9	0	V2	+	2
V2		1	5	0	0	3	0	0	8	17	0	V2	/	1
V2		2	1	1	0	1	0	0	8	8	0	V2	=	5
V2		I	1	1	1	7	1	2	5	18	0	V2	>	1
V2		10	5	0	0	1	0	0	14	14	0	V2	SE	2
V2		II	1	0	0	4	0	0	11	18	0	V2	FACA	1
V2		NUM	1	1	1	4	1	2	5	15	0	V2	RETORNA_CONTROLE	1
V2		ERRO	6	0	0	1	1	0	15	15	0			
V2		AREAL	4	0	1	1	1	1	6	6	0			
V2		LISTA	1	0	1	1	1	1	18	18	0			
V2		LIMITE	1	0	0	2	0	0	8	11	0			
V2		LABEL1*L	3	0	0	1	0	0	11	11	0			
V3		0	1	0	0	2	0	0	13	19	0	V3	+	2
V3		1	5	0	0	3	0	0	12	26	0	V3	/	1
V3		2	1	1	0	1	1	0	12	12	0	V3	*	2
V3		I	1	1	1	1	1	2	9	27	0	V3	IF	6
V3		10	5	0	0	1	0	0	20	20	0	V3	DO	1
V3		II	1	0	0	4	1	2	15	27	0	V3	MOD	1
V3		NUM	1	1	1	4	1	2	9	24	0	V3	.EQ.	1
V3		100*L	3	0	0	2	0	0	15	31	0	V3	.GT.	1
V3		ERRO	6	0	0	1	1	0	24	24	0	V3	CALL	1
V3		AREAL	4	0	1	1	1	1	10	10	0	V3	RETURN	1
V3		LISTA	1	1	1	1	1	1	9	27	0			
V3		RESTO	1	1	1	1	0	0	9	19	0			
V3		LIMITE	1	0	0	2	0	0	12	15	0			

Figura 31. Sub-rotina CALNUM (Calcula-Numeros): Operandos e operadores nas versões 2 e 3

O P E R A N D O S											O P E R A D O R E S			
ver	nome	tp	td	dc	ct	es	fg	in	fm	io	ver	nome	ct	
V2		1	1	0	0	1	0	0	13	13	0	V2	+	1
V2		I	1	0	0	5	0	0	9	13	3	V2	=	1
V2		N	1	1	1	3	1	2	6	9	0	V2	FACA	1
V2		NUM	1	1	1	2	1	2	6	6	0	V2	RETORNA_CONTROLE	1
V2		OPT	1	4	1	2	1	2	5	5	0	V2	ESCREVA	1
V2		AREAL	4	0	1	1	1	1	7	7	0	V2	.MENORIGUAL.	1
V2		LISTA	1	1	1	2	1	1	6	11	2			
V3		1	1	0	0	2	0	0	11	17	0	V3	+	1
V3		I	1	0	0	6	0	0	11	17	3	V3	=	2
V3		N	1	1	1	5	1	2	8	20	3	V3	IF	1
V3		6*A	7	0	0	4	0	0	12	20	0	V3	DO	1
V3		20*L	3	0	1	2	0	0	12	24	0	V3	.LE.	1
V3		21*L	3	0	1	3	0	0	14	25	0	V3	.NE.	1
V3		22*L	3	0	1	2	0	0	16	27	0	V3	WRITE	4
V3		NUM	1	1	1	3	1	2	8	12	2	V3	RETURN	1
V3		OPT	1	4	1	3	1	2	7	13	0			
V3		AREAL	4	0	1	1	1	1	9	9	0			
V3		LISTA	1	1	1	2	1	1	8	16	2			
V3		'cte com pliq'	5	0	0	1	0	0	13	13	0			

Figura 32. Sub-rotina IMPRES (Imprime-Resultados): Operandos e operadores nas versões 2 e 3.

O P E R A N D O S											O P E R A D O R E S		
ver	nome	tp	td	dc	ct	es	fg	in	fm	io	ver	nome	ct
V3		1	4	0	0	1	0	0	0	0	V3	STOP	1
V3	6*A	7	0	0	1	0	0	6	6	0	V3	WRITE	1
V3	20*L	3	0	1	2	0	0	6	7	0			
V3	NUM	1	1	1	3	1	2	4	6	2			

Figura 33. Sub-rotina ERRO: Operandos e operadores na versão 3.

Uma vez apresentados os elementos básicos passamos à análise de algumas das métricas. Nas Figuras 34 a 38 vemos uma tabulação de métricas para cada um dos módulos nas diversas versões.

	Versao 1	Versao 2	Versao3
decl	2	5	5
exec	3	3	4
codg	5	8	9
come	9	9	1
prog	14	17	10
cont	0	0	0
fora	0	0	0
bran	5	7	5
compi	0	0	0
tot	19	24	15
if's	0	0	0
con-log	0	0	0
opo-uni	7	9	8
opo-tot	10	15	15
opr-uni	1	1	2
opr-tot	3	3	4
var-glb	4	4	4

onde:

- decl - linha de declaracao de dados
- exec - linha com comando executavel
- codg - linhas de codigo (decl + exec)
- come - linhas de comentario
- prog - linhas de programas (codg + come)
- cont - linhas de continuacao
- fora - linhas desprezadas
- bran - linhas em branco
- compi - linhas com diretivas para o compilador
- tot - linhas total (prog + cont + fora + bran + compi)
- if's - numero de if's
- con-log - numero de conectores logicos (and or not)
- opo-uni - numero de operandos distintos
- opo-tot - numero total de operandos
- opr-uni - numero de operadores distintos
- opr-tot - numero total de operadores
- var-glb - numero de variaveis globais

Figura 34. Programa principal (MAIN): Algumas métricas para o módulo analisado.

	Versao 2	Versao 3
decl	4	4
exec	2	3
codg	6	7
come	1	5
prog	7	12
cont	0	0
fora	0	0
bran	5	4
compi	0	0
tot	12	16
if's	0	0
con-log	0	0
opo-uni	2	4
opo-tot	6	9
opr-uni	2	2
opr-tot	2	2
var-glb	2	2

Figura 35. Sub-rotina LEPARA (Le-Parametros): Algumas métricas para o módulo analisado.

	Versao 2	Versao 3
decl	4	4
exec	11	10
codg	15	14
come	1	9
prog	16	23
cont	0	0
fora	3	4
bran	8	7
compi	0	0
tot	26	34
if's	2	2
con-log	0	0
opo-uni	13	13
opo-tot	28	33
opr-uni	8	10
opr-tot	14	17
var-glb	3	4

Figura 36. Sub-rotina CALNUM (Calcula-Numeros): Algumas métricas para o módulo analisado.

	Versao 2	Versao 3
decl	5	5
exec	4	12
codg	9	17
come	1	4
prog	10	21
cont	0	1
fora	0	3
bran	7	3
compi	0	0
tot	19	28
if's	0	1
con-log	0	0
opo-uni	7	12
opo-tot	16	34
opr-uni	6	8
opr-tot	6	12
var-glb	4	4

Figura 37. Sub-rotina IMPRES (Imprime-Resultados): Algumas métricas para o módulo analisado.

Versao 3	
decl	3
exec	3
codg	6
come	1
prog	7
cont	1
fora	0
bran	3
compi	0
tot	11
if's	0
con-log	0
opo-uni	4
opo-tot	7
opr-uni	2
opr-tot	2
var-glb	1

Figura 38. Sub-rotina ERRO: Algumas métricas para o módulo analisado.

Na Figura 39 consolidamos as métricas para cada versão somando os valores apresentados para cada módulo em cada versão.

	Versao 1	Versao 2	Versao3
decl	2	18	21
exec	3	20	32
codg	5	38	53
come	9	12	20
prog	14	50	73
cont	-	-	-
fora	0	3	7
bran	5	27	22
compi	0	0	0
tot	19	80	103
if's	0	2	3
con-log	0	0	0
opo-uni	7	31	41
opo-tot	10	65	108
opr-uni	1	17	24
opr-tot	3	25	37
var-glb	4	13	15

Figura 39. Resumo das métricas por programa: Métricas consolidadas por versão.

Como se pode observar, pela análise das métricas é possível acompanharmos a evolução da complexidade ao longo do desenvolvimento de um programa. Os elementos fornecidos pela ferramenta permitem, também, a determinação do valor médio e do desvio padrão para cada métrica. Isto nos capacita a identificar a existência de "outliers" (o que não ocorreu no exemplo) e decidir qual a postura mais adequada

para cada caso: projetar o módulo novamente? codificá-lo com mais atenção? recodificá-lo? testá-lo mais exaustivamente?

Por fim, dispondo de uma base de dados de métricas e outra de custos de desenvolvimento ou de ocorrência de erros, podemos procurar estabelecer relações de causa e efeito.

Acreditamos, firmemente, que módulos excessivamente complexos levem a um aumento exagerado no esforço de desenvolvimento, pois são necessariamente mais difíceis de testar e, portanto, mais propensos a apresentarem mais erros tanto na fase de testes quanto na fase em que são manipulados pelos usuários.

4. CONCLUSÕES

4.1 - Estratégias para a Utilização da PIÁCATÚ

De acordo com o alcance dos objetivos definidos para o programa de controle da qualidade do software a ser desenvolvido por determinado ambiente, a ferramenta PIÁCATÚ pode ser utilizada de duas formas.

Numa estratégia de curto prazo ela pode ser usada apenas para identificar os módulos "outliers" (aqueles que possuem valores de métricas que se situam acima da média por um valor delta especificado, usualmente duas vezes o desvio padrão). Esta identificação pode ocorrer tanto na fase de projeto (através da análise do pseudocódigo) quanto na fase de codificação. Para estes módulos pode-se reservar maior atenção para diminuir-lhes a complexidade, principalmente se isto for detectado na fase de projeto, ou dedicar-lhes mais tempo e cuidado durante as etapas de codificação e testes. Nesta situação o controle da qualidade é exercido apenas de forma reativa. A Figura 40 esquematiza esta estratégia.

Numa estratégia de médio/longo prazo, além de usar-se a PIÁCATÚ como descrito anteriormente, passa-se a utilizá-la também de uma forma pró-ativa, isto é, impedindo a ocorrência de módulos com complexidade excessiva. Para tanto, é necessário que, paralelamente à determinação das métricas, seja feito um

acompanhamento do comportamento futuro do software produzido tendo em vista a ocorrência de erros.

Estrategia I

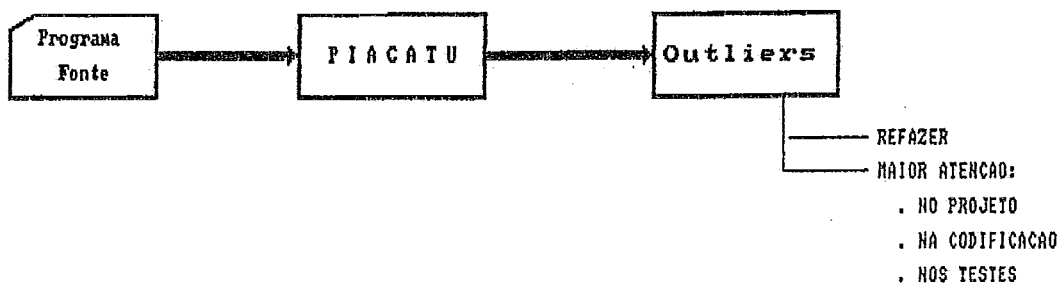


Figura 40. Uma estratégia de utilização da PIACATU: Utilização da Piacatú para controlar a qualidade a partir da identificação dos "outliers".

Desta forma, numa primeira fase os valores obtidos para cada módulo seriam armazenados numa base de dados. Os valores unitários para cada módulo seriam agregados obtendo-se os valores para cada componente do sistema e por fim estes seriam, também, agregados para obter-se a quantificação relativa ao sistema. Nesta fase estaríamos apenas quantificando os elementos que traduzem a complexidade existente no software produzido. Não sabemos ainda se a complexidade é excessiva, em que medida é responsável pela quantidade de erros existentes e de que forma influenciará o esforço de manutenção para aquele sistema.

De posse dos valores de métricas obtidos durante o desenvolvimento e através do acompanhamento do comportamento do software (no tocante a erros encontrados) uma análise estatística permite identificar quais as métricas são mais

relevantes para aquele ambiente e para aquele tipo de software. É necessário que sejam acompanhados e quantificados para cada módulo valores como: número total de erros em todo processo de desenvolvimento; número de erros encontrados na fase de testes; número de erros encontrados pelo usuário; tempo médio entre as falhas, esforço necessário para corrigir os erros encontrados, etc.

Estrategia II

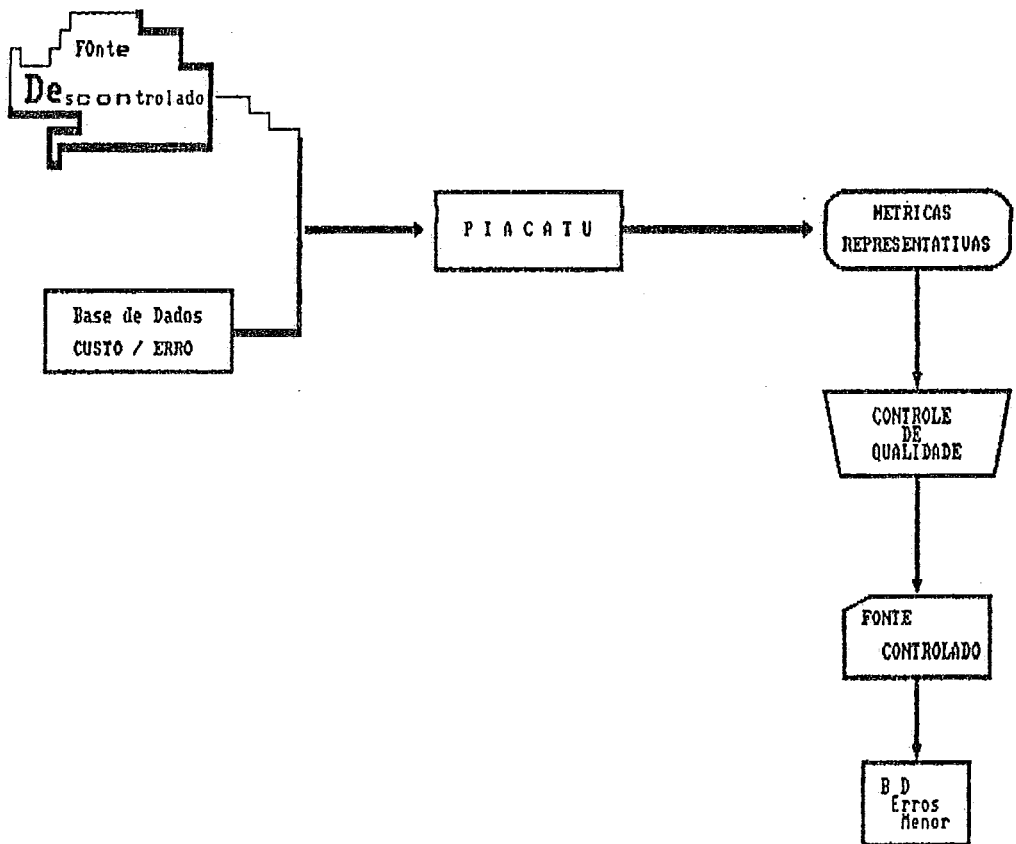


Figura 41. Outra estratégia para a utilização da PIÁCATÚ: Utilização da Piácatú para controlar a qualidade a partir do conhecimento das métricas representativas para o ambiente.

Numa terceira etapa, tendo as métricas sido determinadas e os valores calibrados para aquele ambiente, todo processo de desenvolvimento pode ser balizado por tais valores de métrica. Espera-se que, após uma análise criteriosa, apenas um subconjunto das métricas coletadas seja suficiente para quantificar a qualidade a ser buscada pelo ambiente. Todo este processo pode ser representado pelo esquema da Figura 41.

Como objetivo secundário, esperamos que a utilização da PIÁCATÚ possa incentivar a coleta de informações a respeito do software produzido, principalmente na área de custos de desenvolvimento e de erros. Inclusive, por que a ferramenta dá algum apoio à manutenção destas bases de dados. Não há como avaliar ganhos de qualidade se não sabemos quantificar adequadamente a qualidade atual atingida por nosso ambiente de desenvolvimento.

Para finalizar gostaríamos de citar Grady e Caswell (1986) " ... o primeiro passo para a melhoria de um processo é entender como as coisas vem sendo feitas. Com tal conhecimento, muitas vezes, mudanças podem então ser efetuadas e levarão a melhoramentos do processo e a um maior controle ... ". Desta forma, as medições irão nos levar a uma melhor compreensão e maior previsibilidade de todo o processo. Poderemos, então, acompanhar e medir a qualidade presente no software desenvolvido a partir da fase de projeto e prosseguindo por todo o ciclo de vida do produto.

4.2 - Sugestões para Futuros Trabalhos

A ferramenta PIÁCATÚ foi desenvolvida tendo em vista a obtenção de um protótipo e, portanto, apresenta um nível de qualidade compatível com o objetivo proposto. É preciso, ainda, submetê-la a programas de testes mais completos.

Entendemos ser importante, para que a ferramenta atinja seus objetivos, que ela seja capaz de adaptar-se às necessidades e objetivos de cada ambiente. Para isto contribuem muito as funções de parametrização, que para se tornarem práticas, precisam que a interface projetada seja implementada. Um analisador sintático que valide o pseudocódigo seria de muita valia ao usuário que utilize a PIÁCATÚ já na fase de projeto.

Outras linguagens, também, podem ser objeto de estudo, principalmente o COBOL que possui ampla utilização nas aplicações não científicas.

Por fim, sugere-se a realização de estudos de casos que venham avaliar a proposta formulada nesta Tese.

REFERÊNCIAS

- ANDRADE-1991** C. J. ANDRADE; Avaliação da Qualidade de Programas ; Tese de M.Sc em Engenharia de Sistemas e Computação; COPPE-UFRJ; 1991.
- ARTHUR-1985** L. J. ARTHUR; Measuring Programmer Productivity and Software Quality ; John Wiley & Sons; 1985.
- BAHIA-1988a** A. S. BAHIA; Métricas de Complexidade de Programas: Algumas Considerações ; Relatório Técnico, COPPE-UFRJ, ES-171/88; outubro 1988.
- BAHIA-1988b** A. S. BAHIA; Controle de Qualidade do Software para a Área Científica ; II Simpósio Brasileiro de Engenharia de Software; pp 01-06; Canela/RS; outubro 1988.
- BASILI-1983** V. R. BASILI, R. W. SELBY Jr e T. PHILLIPS; Metrics Analysis and Data Validation Across FORTRAN Projects ; IEEE Transaction on Software Engineering, vol. SE-9(6), pp 652-603; novembro 1983.

- BASILI-1984** V. R. BASILI e B. T. PERRICONE; Software Errors and Complexity: An Empirical Investigation ; Comm ACM, 27(1), pp 42-52; janeiro 1984.
- BASILI-1985** V. R. BASILI e R. SELBY Jr; Calculation and Use of an Environments's Characteristics Software Metrics Set ; 8th ICSE, pp 386-391; 1985.
- BELL-1986** K. BELL; Some Thoughts on Design, Development and Maintenance of Engineering Software ; Adv. Eng. Software, n. 2, vol. 8; 1986.
- BERNS-1984a** G. M. BERNS; Assessing Software Maintainability ; Comm ACM, vol. 27(1), pp 14-23; janeiro 1984.
- BERNS-1984b** G. M. BERNS; New Life for FORTRAN ; Datamation, pp 166-174; setembro 1984.
- BLAINE-1985** J. D. BLAINE e R. A. KEMMERER; Complexity Measures for Assembly Language Programs ; The Journal of Systems and Software, 5, pp 229-245; 1985.
- BOEHM-1978** B. W. BOEHM et alli; Characteristics of Software Quality ; North Holand; 1978.
- BOEHM-1987** B. W. BOEHM et alli; Improving Software Productivity ; Computer, vol. 20, #9, pp 43-57; setembro 1987.

- BURRIL-1986** C. W. BURRIL e L. W. ELLSWORTH; Quality Data Processing: The Potential for the 80's ; Burril-Ellsworth Associates, Inc; 1986.
- CARD-1985** D. N. CARD, G. T. PAGE e F. E. McGARRY; Criteria for Software Modularization ; 8th ICSE, pp 372-377; 1985.
- CARD-1987** D. N. CARD e W. W. AGRESTI; Resolving the Software Science Anomaly ; The Journal of Systems and Software, 7, pp 29-35; 1987.
- CARD-1988** D. N. CARD e W. W. AGRESTI; Measuring Software Design Complexity ; The Journal of Systems and Software, 8, pp 185-197; 1988.
- CHIVERS-1985** I. D. CHIVERS e M. W. CLARK; History and Future of FORTRAN ; Systems, vol. 27, n. 1, pp 39-41; Butterworth & Co (Publishers) Ltd.; janeiro/fevereiro 1985.
- CO/ACARD-1986** Cabinet Office - Advisory Council for Applied Research Development (ACARD Report); Software: A Vital Key to UK Competitiveness ; pp 1-87; março 1986.
- CONTE-1986** S. D. CONTE, H. E. DUNSMORE e V. Y. SHEN; Software Engineering Metrics and Models ; Benjanim/Cummings Pub Company; 1986.
- COTE-1988** V. COTE, P. BOURQE, S. OLIGNY e N. RIVARD; Software Metrics: An Overview of Recents Results ; The Journal of Systems and Software, 8; 1988.

- CRAWFORD-1985** S. G. CRAWFORD, A. A. McINTOSH e D. PREGIBON; An Analysis of Static Metrics and Fault in C Software ; The Journal of Systems and Software, 5 , pp 37-48; 1985.
- CROSS-1986** M. CROSS, A. MOSCARDINI e B. LEWIS; Software Engineering Methodologies for Scientific and Engineering Computation ; Appl. Math. Modeling, vol. 10, pp 376-385; outubro 1986.
- DAVIS-1987** S. G. DAVIS; FORTRAN at 30: Formula for Success ; Datamation, pp 46-56; abril 1987.
- DUCAN-1988** A. S. DUCAN; Software Development Productivity Tools and Metrics ; 10th ICSE, pp 41-48; abril 1988.
- FARLEY-1985** R. FAIRLEY; Software Engineering Concepts ; McGraw-Hill,1985.
- FITZSIMMONS-1978** A. FITZSIMMONS e T. LOVE; A Review and Evaluation of Software Science ; Computer Survey, vol. 10, 1; março 1978.
- GIBSON-1989** V. R. GIBSON e J. A. SENN; System Structure and Software Maintenance Performance ; Comm ACM, vol. 32(3), pp 347-358; março 1989.
- GILL-1991** G. K. GILL e C. F. KEMERER; Cyclomatic Complexity Density and Software Maintenance Productivity ; IEEE TSE, vol. 17(12), pp 1284-1288; dezembro 1991.

- GRADY-1986** R. GRADY e D. CASWELL; Software Metrics: Estabilishing a Company-wide Program ; Prentice-Hall, Inc; 1986.
- GREMILLION-1984** L. L. GREMILLION; Determinants of Program Repair Maintenance Requirements ; Comm ACM, vol. 27(8), pp 826-832; agosto 1984.
- HALL-1984** N. HALL e S. PREISER; Combined Network Complexity Measures ; IBM Journal of Research and Development; pp 15-27; janeiro 1984.
- HALSTEAD-1977** M. H. HALSTEAD; Elements of Software Science ; Elsevier Computer Science Library; 1977.
- HARRISON-1986** W. HARRISON e C. COOK; Are Deeply Nested Conditionals Less Readable? ; The Journal of Systems and Software, 6, pp 335-341; 1986.
- HARRISON-1987** W. HARRISON e C. COOK; A Micro/Macro Measure of Software Complexity ; The Journal of Systems and Software, 7, pp 213-219; 1987.
- HARRISON-1988** W. HARRISON; MAE: A Syntatic Metric Analysis Environment ; The Journal of Systems and Software, 8, pp 57-62; 1988.
- HENRY-1981** S. HENRY e D. KAFURA; Software Structure Metrics Based on Information Flow ; ISS TSE, vol. SE-7(5), pp 510-518; setembro 1981.

- HENRY-1984** S. HENRY e D. KAFURA; The Evaluation of Software System's Structure Using Quantitative Software Metrics ; SPE, vol. 14(6), pp 561-573; junho 1984.
- HENRY-1989** S. HENRY e R. GOFF; Complexity Measurement of Graphical Programming Language ; Software-Practice and Experience, vol. 19(11), pp 1065-1088; novembro 1989.
- IBM-1987** International Business Machines Corporation; VS FORTRAN Version 2 Language and Library Reference (SC26-4221-2) ; IBM Corporation Programming Publishing; junho 1987.
- IBM-1989** International Business Machines Corporation; Systems Application Architecture, Common Programming Interface, C Reference - Level 2 (SC09-1308-1) ; dezembro 1989.
- IBM-1991** International Business Machines Corporation; AIX Version 3 for RISC System/6000, Commands Reference Volume 1, 2 and 3 (GC23-2376-00, GC23-2366-00 and GC23-2367-00) ; março 1991.
- KAFURA-1981** D. KAFURA e S. HENRY; Software Quality Metrics Based on Interconnectivity ; The Journal of Systems and Software, 3, pp 121-131; junho 1981.
- KAFURA-1985** D. KAFURA e J. CANNING; A Validation of Software Metrics Using Many Metrics and Two Resources ; 8th International Conference on Software Engineering (ICSE), pp 378-385; 1985.

- KAFURA-1987** D. KAFURA e G. R. REDDY; The Use of Software Complexity Metrics in Software Maintenance ; IEEE TSE, SE-13(3), pp 335-343; março 1987.
- KEARNEY-1986** J. K. KEARNEY et alli; Software Complexity Measurement ; Comm ACM, vol. 29(11), pp 1044-1050; novembro 1986.
- KERNIGHAN-1986** B. KERNIGHAN e D. RITCHIE; C: A Linguagem de Programação ; Edisa, Editora Campus;1986.
- LAKSHMANAN-1991** K. B. LAKSHMANAN, S. JAYAPRAKASH e P. K. SINHA; Properties of Control-Flow Complexity Measures ; IEEE TSE, vol. 17(12), pp 1289-1295, dezembro 1991.
- LI-1987** H. F. LI e W. K. CHEUNG; An Empirical Study of Software Maintenance ; IEEE TSE, vol. SE-13(6), pp 697-708; junho 1987.
- LIENTZ-1981** B. P. LIENTZ e B. E. SWANSON; Problems in Application Software Maintenance ; Comm ACM, vol. 24(11), pp 763-769; novembro 1981.
- LIND-1989** R. K. LIND e K. VAIRAVAN; An Experimental Investigation of Software Metrics and Their Relationship to Software Development Effort ; IEEE TSE, vol. 15(5), pp 649-653; maio 1989.
- McCABE-1976** T. J. McCABE; Complexity Measure ; IEEE TSE, SE-2(4), pp 308-320; dezembro 1976.

- McCABE-1989** T. J. McCABE e C. W. BUTLER; Design Complexity Measurements and Testing ; Comm ACM, vol. 32(12), pp 1415-1425; dezembro 1989.
- McCALL-1979** J. A. McCALL; An Introduction to Software Quality Metrics, Software Quality Management ; J. D. Cooper, M. J. Fischer (eds) Petrocelli, pp 127-142; 1979.
- MEDES-1987** J. MEDES, P. SMITH e M. NEWTON; A Survey of the Methods Used for Development of Engineering Software in the United Kingdom ; 1st International Conference on Reliability & Robustness of Engineering Software, pp 19-32; setembro 1987.
- MASUCCI-1979** O. MASUCCI; Dicionário Tupi-Português e Vice-Versa ; Brasiliavros - Editora Distribuidora Ltda; 1979.
- NASCIMENTO-1986** E. M. NASCIMENTO e A. R. C. ROCHA; Manual para Avaliação da Qualidade de Programas: Sub-fator Modularidade ; Relatório Técnico, COPPE/UFRJ, ES-112/86; junho 1986.
- NOMURA-1987** T. NOMURA; Use of Software Engineering Tools in Japan ; 9th ICSE, pp 263-269; 1987.
- OSF/MOTIF-1990a** Open Software Foundation / Motif; Programmer's Guide ; Prentice-Hall, Inc.; 1990.
- OSF/MOTIF-1990b** Open Software Foundation / Motif; Programmer's Reference ; Prentice-Hall, Inc.; 1990.

- OSF/MOTIF-1990c** Open Software Foundation / Motif; Style Guide ; Prentice-Hall, Inc.; 1990.
- OSF/MOTIF-1990d** Open Software Foundation / Motif; User's Guide ; Prentice-Hall, Inc.; 1990.
- OCBE-1984** Office of Computer and Business Equipment / Science and Eletronics in the U S Departament of Commerce; A Competitive Assesment of the United States Software Industry ; pp 1-77; dezembro 1984.
- PALERMO-1988** S. PALERMO e A. R. C. ROCHA; A Proposal to Evaluate Program Quality for the Delphi Off-line Environment ; Relatório Técnico, COPPE-UFRJ, ES-180-88; pp 01-62; dezembro 1988.
- POORE-1988** J. H. POORE; Derivation of Local Software Quality Metrics (Software Quality Circle) ; Software Practice and Experience, vol. 18(11), pp 1017-1027; novembro 1988.
- REDISH-1986** K. A. REDISH e W. F. SMYTH; Program Style Analysis: A Natural By-Product of Program Complexity ; Comm ACM, Vol. 9(2), pp 120-133; fevereiro 1986.
- REYNOLDS-1989** R. G. REYNOLDS; The Partial Metrics System: A Tool to Support the Metrics Driven Design of Pseudocode Programs ; The Journal of Systems and Software, 9, pp 287-295; 1989.

- ROCHA-1987** A. R. C. ROCHA; Análise e Projeto Estruturado de Sistemas ; Editora Campus; 1987.
- SAGRI-1989** M. M. SAGRI; Rated and Operating Complexity of Program - An Extension to McCabe's Theory of Complexity Measure ; SIGPLAN Notices, vol. 24(8), pp 8-12; 1989.
- SOAT-1991** J. SOAT; Software Factories ; InformationWeek, pp 24-25; julho 1991.
- SCHNEIDEWIND-1987** N. F. SCHNEIDEWIND; The State of Software Maintenance ; IEEE Transaction on Software Engineering (TSE), SE-13(3), pp 303-310; março 1987.
- WEYUKER-1988** E. J. WEYUKER; Evaluating Software Complexity Measures ; IEEE TSE, vol. 14(9), pp 1357-1365; setembro 1988.

APÊNDICE A - Métricas de Complexidade

Neste apêndice descrevem-se as métricas de complexidade propostas na literatura. Uma descrição mais detalhada pode, ainda, ser encontrada em Bahia (1988a).

A.1 - Métricas de Tamanho

Existem várias propostas para a determinação do tamanho de um módulo. Crawford et alli (1985), por exemplo, ao construírem um analisador estático para software em C elaboraram quatro alternativas:

- **LCE** - Linhas de Código Entregues, que é a quantidade total de linhas entregues ao usuário e necessárias à confecção do sistema.
- **LCNC** - Linhas de Código Não Comentário, que é igual à LCE excluindo-se as linhas em branco e de comentários.
- **NPV** - Número de caracteres Ponto-e-Vírgula, que é usado como delimitador de comandos em C.

- **TCO** - Tamanho do Código Objeto, obtido a partir do tamanho do código objeto do programa compilado (excluindo-se a área de dados).

Conte et alli (1986) apresentam ainda um outro conceito:

- **LCX** - Linhas de Código eXecutáveis, que excluiria de LCNC as linhas referentes à definição de variáveis e as de inicialização não executáveis.

Assim como não há consenso a respeito da forma de se medir, não há também a respeito do tamanho adequado. Nascimento e Rocha (1986) citam as recomendações de Yourdon (50 comandos), Boehm (100 comandos) e os resultados de uma pesquisa de Card que concluiu que: a) bons programadores não tem preferência por um tamanho específico para os módulos; b) módulos grandes custam menos que módulos pequenos; c) a incidência de erros não está relacionada ao tamanho do módulo.

A.2 - Métricas de Estrutura Lógica

É largamente aceito que o nível de complexidade de um módulo é afetado por sua estrutura lógica. McCabe (1976), apoiado na teoria dos grafos, sugere que uma medida desta estrutura pode ser obtida pelo número ciclomático. Neste caso associa-se a um programa, com pontos únicos de entrada e saída, um grafo. Desta forma seu número de componentes (p) será 1, cada bloco onde o processamento é sequencial corresponderá a um nó (n) e cada ramo de um desvio corresponderá a uma borda (e), o valor do número ciclomático de complexidade $V(G)$, deste programa será dado por:

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

Complicações à parte, este mesmo valor pode ser obtido pelo número de predicados (ou decisões) existentes no módulo mais um:

$$V(G) = \text{IF's} + 1$$

Um refinamento desta medida propõe que se considere o número total de condições, já que uma estrutura do tipo:

IF (c1 and c2) then ..

é na verdade equivalente a:

IF (c1) then IF (c2) then ..

Desta maneira teríamos o número ciclomático modificado $V_m(G)$ igual a:

$$V_m(G) = \text{IF's} + \text{AND's} + \text{OR's} + \text{NOT's} + 1$$

McCabe sugere como limite superior para esta métrica o valor 10, podendo ser um pouco maior para módulos que contenham instruções do tipo CASE com muitas opções.

Outro fator que pode ser considerado é o nível de entendimento dos comandos, resultante das decisões e laços existentes no módulo. Parece-nos intuitivo, e Conte e alli (1986) afirmam que quanto maior a profundidade ou nível de encadeamento, mais difícil é perceber-se as condições de acesso à determinada linha de comando. Entretanto um estudo realizado por Harrison e Cook (1986) não conseguiu perceber diferenças significativas no entendimento de módulos com 10 condições, apresentadas em versões encadeadas ou não, sugerindo inclusive que muito do folclore a respeito da influência deste fator no estilo de programação seja reexaminado".

A.3 - Métricas de Estrutura de Dados

Conte et alli (1986) sugerem que para avaliar-se o uso de estruturas de dados de um módulo pode-se utilizar o critério das variáveis ativas. Mede-se aí a quantidade de comandos existentes entre a primeira e a última referência a uma dada variável.

Outro fator que pode ser considerado é a relação entre a quantidade de variáveis locais e globais, tendo em vista que as variáveis globais requerem mais atenção, pois alterações nelas efetuadas poderão produzir interfeirências em outros módulos.

A.4 - Métricas de Halstead

Halstead (1977), baseado em aspectos da teoria de informação e da psicologia cognitiva e apoiado em resultados empíricos significativos, concluiu que a implementação de um algoritmo em qualquer linguagem apresenta características próprias. Isto é, existe um conjunto de operadores e operandos que medidos e agregados de forma adequada permitem caracterizar o seu tamanho, esforço para desenvolvê-lo e até estimar o número de erros que aquele módulo irá conter quando pronto. Estudos posteriores, dentre eles o de Fitzsimmons e Love (1978) validaram a hipótese de que a métrica de esforço (E) está relacionada com o número de erros presentes num determinado módulo.

A seguir descrevemos o Modelo de Halstead.

n_1	- número de operadores distintos;
n_2	- número de operandos distintos;
N_1	- número total de operadores;
N_2	- número total de operandos;
$n = n_1 + n_2$	- vocabulário;
$N = N_1 + N_2$	- tamanho;
$V = N \times \log_2 n$	- volume;
$E = V^2 / V^m$	- esforço;
$V^m = (2 + n_1^2) \log_2(2 + n_1^2)$	volume
$V^m = n^m \times \ln n^m$	- volume potencial (volume mínimo);
$n^m = n_1^m + n_2^m$	- vocabulário mínimo em V^m

$B = E / 3000$ - número estimado de erros.

Halstead não fornece valores limites para suas métricas, sugerindo que 260 seria um valor adequado para o tamanho de um módulo (soma total de operandos e operadores).

Embora amplamente utilizadas, tais métricas também têm seus críticos. Card e Agresti (1987) afirmam peremptoriamente que a formulação apresentada não possui uma fundamentação sólida nem no aspecto teórico nem no empírico. Segundo eles "as altas correlações verificadas entre o tamanho estimado e o real são consequência do fato de uma ser matematicamente dependente da outra". Afirmam também que formulações alternativas de igual ou superior validade estatística podem ser apresentadas, mas que implicariam em profundas mudanças na base teórica utilizada

na elaboração das métricas. E, finalmente, que os dados apresentados, aparentemente, suportam um leque amplo de teorias de forma igualmente satisfatórias.

A.5 - Métrica de Harrison e Cook

Esta métrica, denominada de Complexidade Micro/Macro (CMM) busca avaliar a complexidade total de um sistema, incluindo em sua elaboração um componente micro e outro macro. A parcela micro é obtida pelo Número Ciclométrico de McCabe e a macro leva em conta os efeitos decorrentes da quantidade de variáveis globais existentes, da composição das listas de parâmetros, da quantidade de módulos e da maneira como uma determinada rotina documenta o uso das variáveis ali utilizadas (Harrison, 1987).

A medida de complexidade CMM é dada por:

$$CMM = \sum_{i=1}^k (CS_i \times CM_i), \text{ onde:}$$

k - número de subprogramas;

CS_i - contribuição da rotina i à complexidade do sistema;

$$CS_i = [(G_i \times (k - 1)) + Parm_i] \times (1 - ID_i)$$

G_i = número de vezes que variáveis globais são utilizadas em i ;

$Parm_i$ = número de vezes que as variáveis da lista de parâmetros são usadas em i ;

ID_i = índice de documentação do módulo i , dado por:

$$ID_i = (LCE_i - LCNC_i) / LCE_i \text{ (ver item A.1)}$$

CM_i = complexidade da rotina;

$$CM_i = V(G_i) \text{ (ver item A.2)}$$

Os autores acreditam que a métrica apresentada possui uma boa correlação com o número de erros encontrados no módulo e que ela pode ser utilizada na identificação dos módulos mais propensos a erros. Uma vez identificados eles poderão ser reescritos ou então tratados de forma mais controlada na fase de testes de maneira a detectar melhor a presença de erros e assegurar-lhes uma melhor qualidade.

A.6 - Métrica de Card e Agresti

Estes autores entendem que a complexidade total de um sistema, ou a complexidade relativa de cada módulo (Ct), é composta de duas partes. Uma delas traduz a complexidade local (Lo), ou intermódulo, e a outra a complexidade estrutural (Es), ou intermódulo (Card, 1988). Desta maneira:

$$Ct = Lo + Es$$

$$Lo = [\sum (v_i / (f_i + 1))] / n$$

$$Es = [\sum (f_i^2)] / n$$

n = número de módulos no sistema;

v_i = número de variáveis de entrada/saída no módulo i ;

f_i = *fan-out* do módulo i (quantidade de módulos referenciados pelo módulo i).

Este modelo foi concebido para ser aplicado já na fase de projeto, sendo o *fan-out* seu componente básico. Temos então que: quanto menor o *fan-out*, maior a complexidade do **módulo** (indicando uma maior funcionalidade). Como consequência este módulo deverá contribuir em menor escala para complexidade total do **sistema**. Os autores sugerem que o valor de *fan-out* que minimiza a complexidade total do

sistema é três e que o valor usualmente aceito de nove (sete mais ou menos dois) é excessivo.

A.7 - Métrica de Hall e Preiser

Este estudo, que é uma extensão do trabalho de McCabe, combina a complexidade do módulo com o da rede, de forma a permitir que já na fase do projeto de arquitetura se possa decidir entre alternativas de projeto. Pretende também considerar a complexidade existente na confecção de módulos que administram a alocação de recursos em ambientes de multiprocessamento (Hall, 1984).

A complexidade é, então, dada por:

$$Ct = (p_1 \times CR) + (p_2 \times \sum_{i=1}^n C_i), \text{ onde:}$$

n = número de módulos do sistema;

$p_1 p_2$ = fatores de ponderação atribuídos pelo usuário;

C_i = complexidade micro do módulo que leva em consideração o número de decisões e quantidade média de comandos em cada ramo;

$$CR_i = \sum_{j=1}^k (e_i + \sum_{j=1}^n (d_j \times r_{ij}))$$

k = número de recursos a serem controlados pela rede;

r_{ij} = 0, se o recurso não for requerido pelo módulo i ,

= 1, se o recurso for requerido pelo módulo i ;

d_j = complexidade associada à alocação do recurso j ;

e_i = complexidade da interface do módulo, caracteriza o modo de invocar-se e retornar-se do módulo.

Os autores afirmam que esta métrica permite: a) identificar módulos extremamente complexos antes que decisões de maior monta sejam efetivamente tomadas; b) auxiliar

na determinação do número ótimo de módulos para o sistema; c) medir a complexidade associada à aquisição de recursos; d) ajudar a decidir sobre a conveniência de ter-se módulos de serviço em unidade separadas ou agrupadas nos módulos da aplicação.

A.8 - Métrica de Henry e Kafura

Estes autores entendem que a complexidade total de um sistema pode ser melhor avaliada através da análise do fluxo de informação existente entre os módulos. São então identificados fluxos globais, onde estruturas de dados são compartilhadas entre módulos e os fluxos locais estabelecidos através de chamadas diretas entre módulos, ou aqueles indiretos, onde a comunicação entre dois módulos se dá via parâmetros (Henry, 1981; Kafura, 1981). Esta métrica é então obtida por:

$$C = Tamanho \times (FanIn \times FanOut)^2$$

Tamanho = LCE ou LCNC (ver item A.1)

FanIn = número de fluxos que chegam no módulo mais o número de estruturas de dados usadas como entrada;

FanOut = similar ao *fan-in*, só que considera os fluxos que saem e as estruturas que são usadas como saída.

Os autores alegam que esta técnica de aferição da complexidade é adequada por possuir as seguintes características:

- a) pode ser determinada a partir de elementos obtidos ainda na fase de projeto;
- b) revela mais aspectos das conexões do sistema do que relações do tipo *chama módulos, usa a rotina ou depende de*;

c) permite definir medidas para a complexidade do módulo, acoplamento dos módulos, iteração entre níveis de abstração e pontos de estrangulamento no sistema.

APÊNDICE B - Sintaxe Resumida

B.1 - Descrição da Sintaxe

Neste apêndice apresenta-se de forma resumida a sintaxe entendida pelo analisador PIÁCATÚ. Com isto pretende-se oferecer ao leitor um acesso rápido aos elementos da linguagem. Sua utilização deve ser feita de forma criteriosa. Para uma leitura mais detalhada sugere-se o manual VS FORTRAN Language and Library Reference (IBM, 1987).

Convenções adotadas:

- os elementos **fora** de chaves { } são obrigatórios;
- os abre e fecha parênteses () , onde aparecem, são obrigatórios;
- os elementos **entre** chaves { } são opcionais;
- os caracteres -> precedendo o comando indicam que aquele comando ou aquela forma de utilização do comando é uma extensão da IBM que foge ao padrão ANSI;

- as palavras em MAIÚSCULA são palavras chaves e quando especificadas devem ser usadas da forma indicada;

- significado de alguns valores:
 - stl - label de comando (statement number);
 - un - variável ou inteiro especificando arquivo;
 - name - nome;
 - arg1, arg2 - variável ou lista de variáveis;
 - len1, len2 - especificação de tamanho de variável;
 - dim - especificação de dimensão de variável;
 - list1 - lista de variáveis;
 - clist1 - lista de inicialização de variáveis;
 - a1, a2 - definição de array;
 - e1, e2, e3 - variáveis ou constante do comando DO;
 - f1, f2 - comandos de especificação de FORMAT;
 - logexp - expressão lógica válida;
 - fmt - número de um comando FORMAT.

```
operadores ARITMETICOS ** * / + -
operadores CARACTERES //
operadores LOGICOS .NOT. .AND. .OR. .EQV. .NEQV.
operadores RELACIONAIS > >= < <= = <>
operadores RELACIONAIS .GT. .GE. .LT. .LE. .EQ. .NE.
operadores ATRIBUICAO =
-> @PROCESS
  ASSIGN st1 TO i
-> AT st1
  BACKSPACE un
  BACKSPACE ( {UNIT=} un {,IOSTAT=ios} {,ERR=st1} )
  BLOCK-DATA {name}
  CALL name { ( { arg1 {,arg2} .. } ) }
  CHARACTER {*len1} name1 {,name2 ...} name = a { (dim) } {*len2}
  CLOSE ( {UNIT=}un {,ERR=st1} {,STATUS=sta} {,IOSTAT=ios} )
  COMMON { / {name1} / } list1 {,} /{name2} / list2 ... }
  COMPLEX {*len1} name1 {,name2 ...} name = a { (dim) } {*len2}
  CONTINUE
  DATA list1 /clist1/ { {,} list2 /clist2/ ...}
-> DEBUG ( { UNIT() SUBCHK() TRACE INIT() SUBTRACE } ) ao menos uma
-> DELETE un
-> DELETE ( {UNIT=}un {,IOSTAT=ios} {,ERR=st1} )
  DIMENSION a1(dim1) {, a2(dim2) ...}
-> DISPLAY list
  DO (implied list) ( dlsit, i=m1, m2 {,m3} )
  DO { st1 {,} } i = e1, e2, {,e3} { END_DO }
-> DO { st1 {,} } WHILE (logexp) { END_DO }
  DOUBLE_PRECISION igua a COMPLEX
-> EJECT
  END
-> END_DEBUG
  ENDFILE un
  ENDFILE (un IOSTAT= ERR=)
  ENDFILE ( {UNIT=}un {,IOSTAT=ios} {,ERR=st1} )
  ENTRY name { ( {arg1, arg2 ...} ) }
  EQUIVALENCE (list1) { , (list2) ...}
  EXTERNAL name1 {,name2 ...}
```

```
FORMAT      (f1 {,f2 ...})
FUNCTION name ( { arg1 {,arg2 ...} } )
GO TO i { {,} {(st11 {st12} ...)} }
GO TO (st11 {,st12} ...) {,} i
GO TO st11
IF (logexp) THEN {ELSE} {ELSE_IF} END_IF
IF (logexp) stmt
IF (logexp) st11 , st12 , st13
IMPLICIT type (letter ...)
```

-> IMPLICIT NONE

-> INCLUDE (name) {n}

-> INCLUDE char-cte

```
INQUIRE (FILE=fn {,ACCESS=acc} {,BLANK=blk} {,DIRECT=dir}
INQUIRE      {,ERR=st1} {,EXIST=exs} {,FORMATTED=fmt}
INQUIRE      {,FORM=frm} {,IOSTAT=ios} {,NAME=nam} {,NAMED=nmd}
INQUIRE      {,NEXTREC=nxr} {,NUMBER=num} {,OPENED=opn}
INQUIRE      {,RECL=rcl} {,SEQUENTIAL=seq} {,UNFORMATTED=un} )
INQUIRE ({UNIT=}un {,ACCESS=acc} {,BLANK=blk} {,DIRECT=dir}
INQUIRE      {,ERR=st1} {,EXIST=exs} {,FORMATTED=fmt}
INQUIRE      {,FORM=frm} {,IOSTAT=ios}{,NAME=nam}{,NAMED=nmd}
INQUIRE      {,NEXTREC=nxr} {,NUMBER=num} {,OPENED=opn}
INQUIRE      {,RECL=rcl}{,SEQUENTIAL=seq}{,UNFORMATTED=unf} )
```

-> INQUIRE ({UNIT=}un FILE=fn {,ACTION=act} {,CHAR=chr} {,KEYED=kyd}

-> INQUIRE (FILE=fn {,ACTION=act} {,CHAR=chr} {,KEYED=kyd}

-> INQUIRE ({UNIT=}un FILE=fn {,KEYEND=ken} {,KEYID=kid}

-> INQUIRE (FILE=fn {,KEYEND=ken} {,KEYID=kid}

-> INQUIRE ({UNIT=}un FILE=fn {,KEYLENGTH=k1e} {,KEYSTART=kst}

-> INQUIRE {,LASTKEY=lky} {,LASTRECL=lr1}

-> INQUIRE {,PASSWORD=pwd} {,READ=ron}

-> INQUIRE {,READWRITE=rwr} {,WRITE=wri})

-> INQUIRE (FILE=fn {,KEYLENGTH=k1e} {,KEYSTART=kst}

-> INQUIRE {,LASTKEY=lky} {,LASTRECL=lr1}

-> INQUIRE {,PASSWORD=pwd} {,READ=ron}

-> INQUIRE {,READWRITE=rwr} {,WRITE=wri})

```
INTEGER  {*len1} name1 {,name2 ...} name = a { (dim) } {*len2}
INTRINSIC name1 {,name2 ...}
LOGICAL  {*len1} name1 {,name2 ...} name = a { (dim) } {*len2}
```

```
NAMELIST /name1/ list1 {/name2/ list2 ...}
OPEN ({UNIT=}un {,ERR=st1} {,STATUS=sta} {,FILE=fn} {,ACCESS=acc}
OPEN ({UNIT=}un {,BLANK=Bblk {,FORM=frm}
OPEN ({UNIT=}un {,IOSTAT=ios} {,RECL=rc1} )
-> OPEN ({UNIT=}un {,CHAR=chr} {,ACTION=act} {,PASSWORD=pwd}
-> OPEN ({UNIT=}un {,KEYS=} )
PARAMETER (name1=cte1 {,name2=cte2 ...})
PAUSE {n}
PRINT fmt {,list}
PRINT * {,list}
PRINT namelist
PROGRAM name
-> READ ({UNIT=}un ID=id) {list}
READ ({UNIT=}un,{FMT=}fmt, REC=rec {,ERR=st1} {IOSTAT=ios}) {list}
-> READ ({UNIT=}un, {FMT=}fmt, REC=rec {,ERR=st1} {IOSTAT=ios}
-> READ ({UNIT=}un, {,KEY=key} {,KEYGE=kge} {,KEYGT=kgt}
-> READ ({UNIT=}un, {,KEYID=kid} {,NOTFOUND=st1} ) {list}
-> READ ({UNIT=}un, {FMT=}fmt, {,ERR=st1} {IOSTAT=ios}
-> READ ({UNIT=}un, {,NOTFOUND=st1 | ,END=st1} ) {list}
READ ({UNIT=}un,{FMT=}fmt, {,ERR=st1}{,END=st1}{IOSTAT=ios}) {list}
READ fmt {,list}
READ * {,list}
READ ({UNIT=}un,{FMT=}*, {,ERR=st1}{,END=st1}{IOSTAT=ios}) {list}
-> READ name
-> READ ({UNIT=}un,{FMT=}name, {,ERR=st1}{,END=st1}{IOSTAT=ios})
READ ({UNIT=}un, REC=rec {,ERR=st1} {IOSTAT=ios}) {list}
-> READ ({UNIT=}un, REC=rec {,ERR=st1} {IOSTAT=ios} {,NUM=n}) {list}
-> READ ({UNIT=}un, {,ERR=st1} {IOSTAT=ios}
-> READ ({UNIT=}un, {,KEY=key} {,KEYGE=kge} {,KEYGT=kgt}
-> READ ({UNIT=}un, {,NOTFOUND=st1} {,NUM=} ) {list}
-> READ ({UNIT=}un, {,ERR=st1} {IOSTAT=ios}
-> READ ({UNIT=}un, {,NOTFOUND=st1 | END=st1} {,NUM=} ) {list}
READ ({UNIT=}un, {,ERR=st1} {IOSTAT=ios}) {list}
-> READ ({UNIT=}un, {,ERR=st1} {IOSTAT=ios} {,NUM=n}) {list}
REAL      {*len1} name1 {,name2 ...} name = a { (dim) } {*len2}
RETURN    (function)
RETURN {i} (subroutine)
```

```
REWIND un
REWIND ({UNIT=}un {,ERR=st1} {,IOSTAT=ios})
-> REWRITE ({UNIT=}un {FMT=}fmt {,ERR=st1}
-> REWRITE ({UNIT=}un {,IOSTAT=ios} {,DUPKEY=st1}) list
-> REWRITE ({UNIT=}un {,ERR=st1}{IOSTAT=ios}{,DUPKEY=}{,NUM=n}) list
SAVE {name1 {,name2 ...}}
STOP {n}
STOP {' message '}
SUBROUTINE name ({arg1, arg2 ...})
-> TRACE OFF
-> TRACE ON
-> WAIT ({UNIT=}un, ID=id {,COND=i1} {,NUM=i2}) {list}
-> WRITE ({UNIT=}un ID=id) {list}
WRITE ({UNIT=}un,{FMT=}fmt, REC=rec {,ERR=st1}{IOSTAT=ios}) {list}
-> WRITE ({UNIT=}un, {FMT=}fmt, {,ERR=st1} {IOSTAT=ios}
-> WRITE ({UNIT=}un, {,DUPKEY=st1} ) list
WRITE ({UNIT=}un, {FMT=}fmt, {,ERR=st1} {IOSTAT=ios} ) {list}
WRITE ({UNIT=}un, {FMT=}*, {,ERR=st1} {IOSTAT=ios}) {list}
-> WRITE ({UNIT=}un, {FMT=}name, {,ERR=st1} {IOSTAT=ios})
WRITE ({UNIT=}un, REC=rec {,ERR=st1} {IOSTAT=ios}) {list}
-> WRITE ({UNIT=}un, REC=rec {,ERR=st1} {IOSTAT=ios} {,NUM=n}) {list}
-> WRITE ({UNIT=}un, {,ERR=st1}{IOSTAT=ios}{,DUPKEY=st1}{,NUM=n}) list
WRITE ({UNIT=}un, {,ERR=st1} {IOSTAT=ios}) {list}
WRITE ({UNIT=}un, {,ERR=st1} {IOSTAT=ios} {list}
```

APÊNDICE C - Relação de Métricas

C.1 - Apresentação

Neste apêndice apresentamos as diversas métricas passíveis de serem selecionadas para análise. Uma descrição mais detalhada pode ser encontrada no Apêndice A e nas referências lá citadas.

As métricas, num total de 85, foram agrupadas segundo dois critérios. No primeiro elas estão organizadas de acordo com o tipo de característica de complexidade que objetivam medir. Desta forma, temos os grupos com métricas de Tamanho (19), Fluxo de Controle (10), Estrutura Modular (7), Estrutura de Dados (28), Entrada/Saída de Dados (13), Documentação (2) e Agregadas (6). No segundo caso, as métricas foram agrupadas de maneira a que se pudesse obter níveis crescentes de caracterização da complexidade e temos, então, os grupos Primário, Mediano e Extenso. As duas letras, antes de cada métrica, indicam o TIPO e o GRUPO a que cada uma delas pertence. Algumas das métricas listadas não foram implementadas no analisador PIÁCATÚ e aparecem assinaladas pelos caracteres ->

C.2 - Caracterização por Tipo de Complexidade

C.2.1 - Métricas de Tamanho (T)

T.P	Numero de linhas DECLARACAO DE DADOS (decl)
T.P	Numero de linhas EXECUTAVEIS (exec)
T.P	Numero de linhas CODIGO (codi = decl + exec)
T.P	Numero de linhas COMENTARIOS (come)
T.P	Numero de linhas de PROGRAMA (prog = codi + come)
T.P	Numero de linhas de CONTINUACAO (cont)
T.P	Numero de linhas FORA (desprezadas)
T.P	Numero de linhas EM BRANCO (bran)
T.P	Numero de linhas DIRETIVAS para COMPILADOR (comp)
T.P	Numero de linhas TOTAL (prog + cont + fora + bran + comp)
T.M	Numero de operadores DISTINTOS
T.M	Numero de operadores TOTAL
T.M	Numero de operandos DISTINTOS
T.M	Numero de operandos TOTAL
-> T.M	Numero MAXIMO de OPERADORES no FORMAT
-> T.M	Numero MEDIO de OPERADORES no FORMAT
-> T.E	Frequência de uso dos OPERANDOS
T.E	Lista de operadores DISTINTOS
T.E	Lista de operandos DISTINTOS

C.2.2 - Métricas de Fluxo de Controle (F)

F.P	Numero de GOTO
F.P	Numero TOTAL de CONECTORES LOGICOS (and, or not)
F.P	Numero TOTAL de IF
F.M	Numero MAXIMO de encadeamento de IF
F.M	Numero MEDIO de encadeamento de IF
F.M	Numero TOTAL de DO
F.M	Numero TOTAL de DOWHILE
F.M	Numero TOTAL COMANDOS CASE
-> F.E	Numero MEDIO de clausulas POR CASE
-> F.E	Numero TOTAL de clausulas CASE

C.2.3 - Métricas de Estrutura Modular (M)

M.P	Numero de rotinas chamadas (FAN-OUT)
M.P	Numero de rotinas chamantes (FAN-IN)
M.M	Lista das rotinas chamadas (FAN-OUT)
M.M	Lista de rotinas chamantes (FAN-IN)
-> M.M	Numero de ENTRY point DISTINTOS
-> M.M	Numero de EXIT e RETURN
-> M.M	Numero total de modulos por COMPONENTE

C.2.4 - Métricas de Estrutura de Dados (D)

D.P Numero de varGLOBAL NAO USADAS
D.P Numero de varGLOBAL TOTAL que CHEGAM (COMMON e PARAMETROS)
D.P Numero de varGLOBAL USADAS
D.M Extensao MAXIMA das variaveis
D.M Extensao MEDIA das variaveis
D.M Lista de COMMON USADOS POR MODULO
D.M Lista de MODULOS que USAM DETERMINADO COMMON
D.M Numero de varGLOBAL que CHEGAM via COMMON
D.M Numero de varGLOBAL que CHEGAM via LISTA DE PARAMETROS
D.M Numero de varGLOBAL usada APENAS UMA VEZ
D.M Numero de varGLOBAL SEM DECLARACAO DE TIPO
D.M Numero de varLOCAL usada APENAS UMA VEZ
D.M Numero de varLOCAL DECLARADAS
D.M Numero de varLOCAL DECLARADAS e NAO USADAS
D.M Numero de varLOCAL USADAS
D.M Numero de varLOCAL USADAS e NAO DECLARADAS
D.E Lista de varGLOBAL que CHEGAM via COMMON
D.E Lista de varGLOBAL que CHEGAM via LISTA DE PARAMETROS
D.E Lista de varGLOBAL usada APENAS UMA VEZ
D.E Lista de varGLOBAL NAO USADAS
D.E Lista de varGLOBAL SEM DECLARACAO DE TIPO
D.E Lista de varGLOBAL TOTAL que CHEGAM (COMMON e PARAMETROS)
D.E Lista de varGLOBAL USADAS
D.E Lista de varLOCAL usada APENAS UMA VEZ
D.E Lista de varLOCAL DECLARADAS
D.E Lista de varLOCAL DECLARADAS e NAO USADAS
D.E Lista de varLOCAL USADAS
D.E Lista de varLOCAL USADAS e NAO DECLARADAS

C.2.5 - Métricas de Entrada/Saída de Dados (S)

S.P Numero de variaveis de I/O
S.P Numero de ARQUIVOS ACESSADOS TOTAL
S.M Numero de comandos FORMAT
-> S.M Numero de variaveis de I/O NAO UTILIZADAS
-> S.M Numero de ARQUIVOS ACESSADOS READ
-> S.M Numero de ARQUIVOS ACESSADOS UPDATE
-> S.M Numero de ARQUIVOS ACESSADOS WRITE
S.E Lista de variaveis de I/O
S.E Lista de variaveis de I/O NAO UTILIZADAS
-> S.E Lista de ARQUIVOS ACESSADOS READ
-> S.E Lista de ARQUIVOS ACESSADOS TOTAL
-> S.E Lista de ARQUIVOS ACESSADOS UPDATE
-> S.E Lista de ARQUIVOS ACESSADOS WRITE

C.2.6 - Métricas de Documentação (O)

O.P Percentual LINHAS COMENTARIO
O.P Percentual LINHAS EM BRANCO

C.2.7 - Métricas Agregadas (A)

A.P	McCabe (if's)
A.P	McCabe Modificado (if's e conectores logicos)
A.M	Henry e Kafura (var in e out, fan-in e fan-out)
-> A.M	Card e Agresti (var in e out, fan-out)
A.M	Halstead (operandos e operadores)
-> A.M	Harrinson e Cook (varGlobal usadas, tamanho, mccabe)

C.3 - Caracterização por Nível de Complexidade

C.3.1 - Grupo Primário (.P)

A.P	McCabe (if's)
A.P	McCabe Modificado (if's e conectores logicos)
D.P	Numero de varGLOBAL NAO USADAS
D.P	Numero de varGLOBAL TOTAL que CHEGAM (COMMON e PARAMETROS)
D.P	Numero de varGLOBAL USADAS
F.P	Numero de GOTO
F.P	Numero TOTAL de CONECTORES LOGICOS (and, or not)
F.P	Numero TOTAL de IF
M.P	Numero de rotinas chamadas (FAN-OUT)
M.P	Numero de rotinas chamantes (FAN-IN)
O.P	Percentual LINHAS COMENTARIO
O.P	Percentual LINHAS EM BRANCO
S.P	Numero de variaveis de I/O
S.P	Numero de ARQUIVOS ACESSADOS TOTAL
T.P	Numero de linhas CODIGO (exec + decl)
T.P	Numero de linhas COMENTARIOS (come)
T.P	Numero de linhas DECLARACAO DE DADOS (decl)
T.P	Numero de linhas DIRETIVAS para COMPILADOR (comp)
T.P	Numero de linhas EM BRANCO (bran)
T.P	Numero de linhas ENTREGUES (exec + decl + come)
T.P	Numero de linhas EXECUTAVEIS (exec)
T.P	Numero de linhas TOTAL (exec + decl + come + comp + bran)

C.3.2 - Grupo Mediano (.M)

inclui todas do grupo anterior e mais:

- > A.M Card e Agresti (var in e out, fan-out)
- A.M Halstead (operandos e operadores)
- > A.M Harrinson e Cook (varGlobal usadas, tamanho, mccabe)
- A.M Henry e Kafura (var in e out, fan-in e fan-out)
- D.M Extensao MAXIMA das variaveis
- D.M Extensao MEDIA das variaveis
- D.M Lista de COMMON USADOS POR MODULO
- D.M Lista de MODULOS que USAM DETERMINADO COMMON
- D.M Numero de varGLOBAL que CHEGAM via COMMON
- D.M Numero de varGLOBAL que CHEGAM via LISTA DE PARAMETROS
- D.M Numero de varGLOBAL usada APENAS UMA VEZ
- D.M Numero de varGLOBAL SEM DECLARACAO DE TIPO
- D.M Numero de varLOCAL usada APENAS UMA VEZ
- D.M Numero de varLOCAL DECLARADAS
- D.M Numero de varLOCAL DECLARADAS e NAO USADAS
- D.M Numero de varLOCAL USADAS
- D.M Numero de varLOCAL USADAS e NAO DECLARADAS
- F.M Numero MAXIMO de encadeamento de IF
- F.M Numero MEDIO de encadeamento de IF
- F.M Numero TOTAL de DO
- F.M Numero TOTAL de DOWHILE
- F.M Numero TOTAL COMANDOS CASE
- M.M Lista das rotinas chamadas (FAN-OUT)
- M.M Lista de rotinas chamantes (FAN-IN)
- > M.M Numero de ENTRY point DISTINTOS
- > M.M Numero de EXIT e RETURN
- > M.M Numero total de modulos por COMPONENTE
- S.M Numero de comandos FORMAT
- > S.M Numero de variaveis de I/O NAO UTILIZADAS
- > S.M Numero de ARQUIVOS ACESSADOS READ
- > S.M Numero de ARQUIVOS ACESSADOS UPDATE
- > S.M Numero de ARQUIVOS ACESSADOS WRITE
- T.M Numero de OPERADORES DISTINTOS
- T.M Numero de OPERADORES TOTAL
- T.M Numero de operandos DISTINTOS
- T.M Numero de operandos TOTAL
- > T.M Numero MAXIMO de OPERADORES no FORMAT
- > T.M Numero MEDIO de OPERADORES no FORMAT

C.3.3 - Grupo Extenso (.E)

inclui todas do grupo anterior e mais:

- D.E Lista de varGLOBAL que CHEGAM via COMMON
- D.E Lista de varGLOBAL que CHEGAM via LISTA DE PARAMETROS
- D.E Lista de varGLOBAL usada APENAS UMA VEZ
- D.E Lista de varGLOBAL NAO USADAS
- D.E Lista de varGLOBAL SEM DECLARACAO DE TIPO
- D.E Lista de varGLOBAL TOTAL que CHEGAM (COMMON e PARAMETROS)
- D.E Lista de varGLOBAL USADAS
- D.E Lista de varLOCAL usada APENAS UMA VEZ
- D.E Lista de varLOCAL DECLARADAS
- D.E Lista de varLOCAL DECLARADAS e NAO USADAS
- D.E Lista de varLOCAL USADAS
- D.E Lista de varLOCAL USADAS e NAO DECLARADAS
- > F.E Numero MEDIO de clausulas POR CASE
- > F.E Numero TOTAL de clausulas CASE
- S.E Lista de variaveis de I/O
- S.E Lista de variaveis de I/O NAO UTILIZADAS
- > S.E Lista de ARQUIVOS ACESSADOS READ
- > S.E Lista de ARQUIVOS ACESSADOS TOTAL
- > S.E Lista de ARQUIVOS ACESSADOS UPDATE
- > S.E Lista de ARQUIVOS ACESSADOS WRITE
- > T.E Frequencia de uso dos OPERANDOS
- T.E Lista de operADORES DISTINTOS
- T.E Lista de operADORES TOTAL
- T.E Lista de operANDOS DISTINTOS
- T.E Lista de operANDOS TOTAL