AVALIAÇÃO DA ESTRUTURA MODULAR DE PROGRAMAS

NA FASE DE PROJETO

Maria Cristina João da Fonseca Passos

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:

Prof. Ana Regina C. da Rocha, Dsc
Presidente da Banca

Prof. Jano Moreira de Souza, Phd
COPPE/UFRJ

Prof. Regina Célia de Souza Pereira, Dsc
UFF

Rio de Janeiro, RJ - Brasil

Maio de 1991

PASSOS, Maria Cristina João da Fonseca

Avaliação da Estrutura Modular de Programas na Fase de Projeto [Rio de Janeiro] 1991

viii, 176 pg., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1991)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Qualidade de projeto I. COPPE/UFRJ

II. Título (série)

Ao Carlos Eduardo

e à Camila Maria

AGRADECIMENTOS

A Profa. Ana Regina C. da Rocha, que deixou a marca da orientação didática e eficiente, tornando este trabalho possível.

Ao Prof. Jano Moreira de Souza, cujas valiosas sugestões foram vitais para o armazenamento da ferramenta.

Ao Guilherme Travassos, ao Flávio de Araújo Mattos e ao Geraldo Xexéo, pelo apoio durante a fase de implementação.

A Vera, pela sagrada paciência de rever o texto, discuti-lo e apresentar sugestões.

A Adriana L. Q. Mattoso, pelo auxílio inestimável na utilização do método de especificação orientado a objetos.

A Norma e ao Marcelo, que me cederam o seu microcomputador, agilizando muito o meu trabalho.

Aos meus pais, que nunca mediram esforços para me proporcionar uma vida plena de estudos que, sem dúvida, contribuiu muito para superar mais esse desafio.

Aos colegas e funcionários do Programa de Sistemas que, certamente, propiciaram um ambiente favorável ao meu desenvolvimento acadêmico.

A CAPES e ao CNPq pelo apoio financeiro e incentivo à pesquisa.

RESUMO DA TESE APRESENTADA A COPPE - UFRJ COMO PARTE DOS REQUISITOS NECESSARIOS PARA A OBTENÇAO DO GRAU DE MESTRE EM CIENCIAS.

Avaliação da Estrutura Modular de Programas
na Fase de Projeto

Maria Cristina João da Fonseca Passos

MAIO DE 1991

Orientador: Prof. Ana Regina C. da Rocha
Programa: Engenharia de Sistemas e Computação

Este trabalho tem como objetivo identificar atributos de qualidade para especificações de projeto que possam ser automatizados para dar apoio à construção e à avaliação de Gráficos de Estrutura.

A princípio, foi feito um estudo das características de qualidade, presentes na literatura, para a fase de projeto, destacando-se as relacionadas ao atributo modularidade.

A partir deste estudo, foram identificados critérios e definidos processos de avaliação que podem ser usados para medir a modularidade de Gráficos de Estrutura, a nível de Projeto da Arquitetura, sendo implementada a ferramenta para edição de Gráficos de Estrutura e avaliação da Modularidade.

ABSTRACT OF THE THESIS PRESENTED TO COPPE/UFRJ AS PART OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE.

The Evaluation of Program Modularity

During Design Phase

Maria Cristina João da Fonseca Passos

MAY, 1991

Thesis Supervisor: Prof. Ana Regina C. da Rocha

Departament: Systems and Computer Engineering

This work intends to indentify the quality factors for software design specification that can be automated to support the development and validation of Structure Charts.

At first, were studied quality characteristics for the design phase with focus in the modularity attributes. As a result of this preliminary study, some metrics were indentified and the evaluation processes to measure the modularity of Structure Charts were defined. Finally, a software tool for Structure Chart editing and modularity evaluation, was constructed.

INDICE

CAPITULO I

INTRODUÇAO

I.1 O DESENVOLVIMENTO DE SOFTWARE

A crescente evolução dos sistemas computacionais tem provocado o desenvolvimento de produtos de software a serem utilizados em tarefas das mais variadas, que vão desde o controle de processos em centrais nucleares, tráfego aéreo, supervisão de tratamento médico, sistemas de informações governamentais e administrativos até jogos.

Essa diversificação do uso dos computadores, nas mais diferentes áreas de aplicação, aumentou a complexidade e o tamanho dos sistemas desenvolvidos, gerando uma série de problemas enfrentados por gerentes e equipe de desenvolvimento, entre eles (Hausen 84, Wasserman 82):

- caráter dinâmico e iterativo do sistema ao longo de todo o ciclo de vida decorrente da ausência de requisitos completos e precisos;

- dificuldade de término do sistema dentro de um tempo limitado pela organização;

- altos custos financeiros e humanos destinados à manutenção corretiva;

- aumento constante nos custos de desenvolvimento causados pela baixa produtividade;

- o gerenciamento de uma grande quantidade e variedade de recursos;

- documentação desatualizada, prejudicando os trabalhos de manutenção e uso do sistema;

- falta de procedimentos normalizados de avaliação da qualidade para a obtenção do grau de confiabilidade e desempenho desejados.

Fairley (Fairley 85) afirma que de 40 a 60% do tempo do ciclo de vida total de um software, é dedicado à manutenção, podendo em certos casos chegar a 90%. Estão aí computados os esforços necessários para corrigir erros, realizar adaptações a novas condições e, também, dotar o produto de funções, inicialmente, não previstas.

A qualidade de produtos de software, bem como a produtividade do seu processo de desenvolvimento, depende da qualidade e adequação do ambiente de desenvolvimento de software adotado.

O ambiente ideal deve provocar o aumento da produtividade e da qualidade dos produtos desenvolvidos, e a simplificação do trabalho através de ferramentas manuais ou automatizadas (Werneck90).

Um ambiente de desenvolvimento de software constitui-se de métodos, instrumentos (técnicas) e ferramentas que apoiem todo o ciclo de vida do software.

Estes métodos, instrumentos e ferramentas devem apoiar as três atividades envolvidas no processo de desenvolvimento de software: construção, avaliação e gerência.

Baseado nas afirmações acima, surgiu a proposta do projeto COPPE-SISTEMAS (Rocha 86), cujo objetivo era integrar um conjunto de ferramentas automatizadas de apoio ao desenvolvimento de software, a serem utilizadas na fase de Definição e Projeto. Estas ferramentas servem de apoio ao método de Análise e Projeto Estruturado de Sistemas, para uso em microcomputadores da linha IBM-PC.

Na primeira versão dessas ferramentas, foram desenvolvidas as ferramentas de edição do Diagrama de de Fluxo de Dados (EDIT-DFD), do Dicionário de Dados (EDIT-DD) e do Gráfico de Estrutura (EDIT-GE). Entretanto, sentiu-se a necessidade de integrá-las e optou-se por utilizar um sistema avançado de interface com o usuário (MS-Windows).

Assim, surgiu a segunda versão de ferramentas, donde emergiu o projeto FEGRES (Travassos 90) com o objetivo de integrar as ferramentas para a fase de Definição e desenvolver um editor de Entidades e Relacio-namentos (EDIT-ER).

Esta tese é parte desse projeto e visa desenvolver as ferramentas para construção e avaliação relacionadas a fase de Projeto da Arquitetura.

I.2 QUALIDADE DE SOFTWARE

Qualidade é um conceito multidimensional, que se realiza através de um conjunto de atributos associados a um determinado objeto.

Software, como qualquer produto que é colocado à disposição de seus usuários, necessita de um nível aceitável de qualidade. Entretanto, torna-se difícil estabelecer um padrão de aceitação, pois os atributos de qualidade do software diferem de projeto para projeto, uma vez que os critérios utilizados para medí-lo variam em função do domínio de aplicação, das características específicas do produto, das necessidades do usuário e da organização.

Um dos problemas na produção de software de alta qualidade é a falta de uma definição clara, precisa e universalmente aceita para qualidade de software. Ainda mais, é extremamente difícil identificar que atributos auxiliam a predizer a qualidade do produto final e devem ser avaliados ao longo do processo de desenvolvimento.

Mesmo tendo em conta esta dificuldade, o controle da qualidade deve ser realizado em paralelo com a construção do software. Para se atingir níveis satisfa-tórios de qualidade, é indispensável buscar o seu alcance desde o início do desenvolvimento. Cada resultado intermediário no processo de produção deve ser examinado, imediatamente após sua conclusão, procurando garantir que os erros e inadequações no produto sejam detectados o mais cedo possível.

Assim sendo, não basta identificar que atributos determinam a qualidade do produto final. É necessário, também, identificar que atributos determinam a qualidade e que procedimentos adotar, no sentido de controlar o pro-

cesso de desenvolvimento, de forma a atingir o nível de qualidade desejado em cada fase.

I.3 CONTROLE DA QUALIDADE DE SOFTWARE NA FASE DE PROJETO

O desenvolvimento de software é feito através da modelagem da realidade do sistema, até se obter uma representação executável em computador, seguindo uma série de etapas, de acordo com o ciclo de vida escolhido.

Essas etapas tem por finalidade disciplinar o processo de desenvolvimento. Em cada uma delas, utilizam-se os resultados da anterior, detalhando e/ou formalizando mais estes resultados. No modelo de ciclo de vida tradicional tem-se as fases de Definição, Projeto, Construção, Avaliação e Operação.

Durante a fase de projeto tem-se como objetivo identificar uma solução satisfatória para que o software atenda aos requisitos especificados na fase de definição.

O projeto é a representação do software mais próxima da realidade computacional onde, ainda, há uma relativa facilidade para modificações. Este produto intermediário, entre a especificação de requisitos e a programação, retrata uma ponte entre "o que" o usuário deseja e "como" será implementado (Staa 83).

Uma análise de 200 tipos de erros, encontrados durante o desenvolvimento de um grande produto de software, revelou que mais de 60% dos erros foram erros de projeto. O mais grave é que a maioria dos erros de desenvolvimento

ocorrem na fase de projeto e só são detectados, muito mais tarde, na fase de teste ou mesmo durante a fase de operação. Isso se torna extremamente grave ao se constatar que o custo de remoção de tais erros aumenta, exponencialmente, com o tempo (Cho 80). Erros introduzidos na fase de projeto e só detectados durante a fase de operação do produto podem chegar a custar várias vezes mais do que, caso descobertos e reparados durante a fase de projeto. Assim sendo, todo o esforço deve ser feito para garantir a qualidade da especificação de projeto.

As maiores dificuldades encontradas durante a fase de projeto são causadas por:

- requisitos definidos de forma inadequada;

- tendência em se detalhar o produto muito cedo devido aos curtos prazos impostos pela organização;

- inconsistência na especificação;

- inconsistência entre as especificações de requisitos e de projeto;

- falhas na consideração das alternativas estratégicas de projeto;

- falta de sistematização e treinamento de pessoal especializado.

Tornam-se, portanto, necessários métodos, que apoiem o desenvolvedor nas atividades da fase de projeto e que

forneçam ferramentas automatizadas de apoio tanto para a construção, quanto para avaliação do produto.

Um dos métodos mais usados, atualmente, no desenvolvimento de produtos de software é o Projeto Estruturado (Yourdon 79) que possui uma abordagem estruturada e fornece uma linguagem gráfica para construção do Gráfico de Estrutura. Outro aspecto importante deste método é o fato de propor normas e atributos de qualidade de uma boa especificação do projeto.

I.4 OBJETIVO DO TRABALHO

Este trabalho teve como objetivo estudar as características da especificação de projeto e, a partir da utilização do método para avaliação da qualidade proposto por Rocha (Rocha 83), identificar e definir critérios que possibilitem a avaliação dos atributos de qualidade relacionados à Confiabilidade da Representação desta especificação. Dentre estes atributos destaca-se, por sua importância nesta fase do processo de desenvolvimento o atributo modularidade. São, então, definidos critérios para medir o grau de modularidade da Especificação de Projeto.

A partir deste estudo é implementada a ferramenta para apoio a este processo: um editor de Gráficos de Estrutura (EDIT-GEII) e avaliador de Modularidade (Avalie-GE). Esta ferramenta, tem como objetivo, facilitar o processo de construção e avaliação do Gráfico de Estrutura.

EDIT-GEII é uma ferramenta de auxílio na construção de Gráficos de Estrutura segundo o método Projeto

Estruturado. Está baseada na ferramenta anteriormente desenvolvida por Nogueira (Nogueira 88). Esta nova versão tem a finalidade de ser desenvolvida num ambiente de programação que possibilite a integração com outras ferramentas existentes ou em desenvolvimento na COPPE, além de facilitar a interface com o usuário.

AVALIE-GE tem como objetivo medir o grau de modularidade de Gráficos de Estrutura, considerando os seguintes critérios: acoplamento, coesão, número de módulos superiores, número de módulos subordinados e complexidade total do sistema. Estes critérios foram escolhidos por referirem-se ao Gráfico de Estrutura (outros aspectos relativos à modularidade, tais como, tamanho do módulo, isolamento, simplicidade, complexidade do módulo, relato de erros, referem-se à especificação de módulos) e serem possíveis de avaliar através de uma ferramenta.

I.5 ORGANIZAÇÃO DO TRABALHO

Esta tese está organizada em cinco capítulos. O capítulo I contém a introdução do trabalho com a apresentação dos fatores, que motivaram sua elaboração, definindo os seus objetivos e organização.

O capítulo II apresenta um estudo sobre medidas de software, os principais enfoques e características de qualidade de projeto e descreve o método para avaliação da qualidade utilizado.

O capítulo III define os critérios e processos de avaliação para o objetivo Confiabilidade da Representação,

de acordo com o método de Rocha (Rocha 83) descrito no capítulo II.

No capítulo IV apresenta-se a especificação e o projeto da ferramenta, bem como uma definição de sua implementação.

No capítulo V são expostas as conclusões do trabalho e são sugeridas algumas possibilidades de futuras pesquisas.

O anexo I apresenta um exemplo de utilização da ferramenta.

CAPITULO II

CARACTERISTICAS DE QUALIDADE DE PROJETO

II.1 INTRODUÇÃO

Conforme descrito no primeiro capítulo, na fase de projeto são introduzidos erros e inadequações ao produto, que muitas vezes só são descobertos em fases posteriores. Além disso uma série de decisões fundamentais ao projeto, são tomadas nesta fase o que a torna uma etapa crítica no processo de desenvolvimento de software. Para se atingir a qualidade desejada no produto final, é necessário que a especificação de projeto seja elaborada considerando-se um conjunto de atributos.

Este capitulo tem como objetivo descrever os principais conceitos, enfoques e caracteristicas de qualidade encontrados na literatura relacionados a esta fase.

A seção II.2 trata das medidas de software, evidenciando a necessidade de se possuir um método para aplicá-las de forma eficaz.

A seção II.3 descreve o método para avaliação da qualidade utilizado neste trabalho.

Por fim, a seção II.4 ressalta os principais enfoques e caracteristicas de qualidade da especificação de projeto, sendo identificados os atributos, que podem ser aplicados para avaliação do Gráfico de Estrutura (GE).

II.2 MEDIDAS DE SOFTWARE

A finalidade de qualquer empresa é atingir a "qualidade essencial", ou seja, a adequação ao uso de seus produtos ou serviços, a um custo mínimo, competitivo, segundo os requisitos de mercado. Essa "qualidade essencial" pode ser definida como a qualidade que é necessária adicionar ao produto ou serviço para assegurar a satisfação do cliente (Novelino 89).

As indústrias preocupadas com a produção de produtos de alta qualidade estabelecem um nível aceitável e através de mecanismos específicos asseguram que este nível é mantido. Do mesmo modo, as organizações desenvolvedoras de software devem assumir a respon-sabilidade de determinar qual o nível de qualidade aceitável para seus produtos e de estabelecer mecanismos para determinar se ele é atingido (Rocha85).

Denicoff e Grafton (Denicoff81) enfatizam a necessidade de que a Engenharia de Software disponha de parâmetros de medidas precisas e bem entendidas. Para isto, é necessário que a abordagem dada às medidas de software seja marcada pelo paradigma científico tradicional de hipótese, avaliação, crítica e revisão.

Tal processo deve ser aplicado desde a fase de especificação de requisitos, utilizando-se medidas preditivas orientadas a cada fase do ciclo de desenvolvimento e não, apenas, ao produto final, tentando

com isso, predizer quão bem o produto operará em relação aos requisitos de qualidade desejados (McCall 78).

Quanto mais cedo no processo de desenvolvimento de software os problemas de qualidade são detectados, mais efetivas serão as medidas para sua solução (Basili 87). Assim, aplicando-se medidas de qualidade, desde o início do processo de desenvolvimento, é possível detectar os erros de cada fase antes do teste final ou entrega do produto, resultando em uma grande economia nos custos de manutenção e operação do sistema.

O principal objetivo das medidas de software não é apenas determinar as propriedades do produto, mas sim realizar uma predição da qualidade durante o ciclo de desenvolvimento. Curtis (Curtis81) aponta que, por causa do seu valor preditivo potencial, medidas de software podem ser usadas no mínimo de três modos, como:

- ferramentas de gerência de informação: para predizer o futuro, medidas podem ser desenvolvidas para calcular custo e duração do projeto, bem como para estimar a produtividade. Tais medidas permitem aos gerentes visualizar o progresso no desenvolvimento, os problemas futuros e a satisfação dos requisitos básicos;

- medidas de qualidade de software: para criar critérios quantificáveis a partir dos quais o produto possa ser julgado;

- "feedback" para o pessoal envolvido no projeto.

Basili e Rombach (Basili87) argumentam que o processo de medição é um mecanismo muito poderoso para definição e análise da qualidade do processo de desenvolvimento e do produto. O método por eles descrito utiliza medidas objetivas e subjetivas, bem como medidas diretas e indiretas.

Medidas objetivas são expressões numéricas ou representações gráficas destas expressões que possam ser computadas a partir de documentos de software tais como código fonte, projetos e dados de teste. Fazendo uso destas medidas dois avaliadores, atuando separadamente, devem chegar a valores idênticos.

Medidas subjetivas são relativas,pois baseiam-se em uma estimação individual ou em um compromisso dentro de um grupo. Exemplo típico, é a experiência do pessoal envolvido em uma certa aplicação.

Medidas diretas permitem quantificação de um dado fator de interesse. Uma medida indireta ajuda a predizer o valor de uma medida direta. Por exemplo, a medida direta "operacionalidade" pode ser definida como o número de falhas por semana de operação ou número de falhas por chamada de algum componente do software. Uma medida indireta relacionada a ela, poderá ser o número de falhas durante o teste de aceitação. Esta pode ser útil para predizer a operacionalidade já durante o desenvolvimento.

O método, definido por Basili e Rombach (Basili 87), envolve três fases, cada uma delas sugerindo um tipo de medida:

1a. fase: definição dos requisitos de qualidade em termos quantitativos, que consiste em selecionar características de interesse, definir prioridades e relacionamentos entre elas, definir cada característica através de uma ou mais medidas diretas e estabelecer os requisitos de qualidade quantitativamente, atribuindo valor a cada medida.

2a. fase: plano de controle da qualidade, que envolve o planejamento de ações adequadas para garantir o alcance dos requisitos de qualidade definidos, o controle de execução dessas ações e a avaliação dos resultados.

3a.fase: desempenhar o controle da qualidade através de medições, onde os métodos e técnicas especificados durante o planejamento das fases são aplicados para coletar valores para medidas diretas e indiretas; e **avaliação**, onde as medidas diretas são comparadas aos requisitos de qualidade e as medidas indiretas são interpretadas para predizer os valores das medidas diretas. Essa avaliação, também, verifica se os requisitos do projeto foram satisfeitos pelas características de qualidade.

Card e Glass (Card 90) basearam-se no método descrito acima e definiram que uma medida é uma escala de valores possíveis que correspondem às variações observadas em uma determinada característica. Na definição das

medidas, deve ser incluída a escala e os métodos para levantamento e validação dos dados.

Esta escala de medidas pode ser, assim, especificada:

- nominal: trata-se da classificação baseada em características simples tais como cor ou tipo;

- ordinal: envolve uma graduação, isto é, do pior ao melhor;

- intervalo: implica em distâncias iguais entre os valores, significando diferenças equivalentes em medidas de qualidade, como por exemplo, o intervalo de tempo;

- razão: tem a propriedade de intervalo acrescida de um ponto zero para indicar que nenhuma característica foi atingida , como por exemplo comprimento.

Métodos para mediçõe podem ser subjetivos (baseados no julgamento humano) ou objetivos (baseados em pesos e operações aritméticas). Apenas os tipos de medida nominal ou ordinal podem ser coletados por meio de questionários intensivamente elaborados ou através da experiência. Neste caso, o conteúdo exato da medida pode não ser reproduzido. Observadores diferentes, ou o mesmo observador em ocasiões diferentes, importam razões diversas. Proporcionando o desempenho de instruções mais detalhadas, a razão subjetiva aumenta a consistência e a objetividade dos resultados.

Pela própria natureza do projeto, certas características devem ser medidas subjetivamente. Por outro

lado, certos aspectos de qualidade podem ser medidos em meios quantificáveis mais objetivos com a utilização de um método. A representação do projeto deve ser medida objetivamente por várias razões. Uma delas é que algumas ferramentas podem automatizá-la.

De acordo com o método a ser utilizado não é importante quem desempenha a parte de medição do controle da qualidade, mas sim como este é planejado e avaliado preferencialmente por pessoal diferente daquele que participou do desenvolvimento (Basili87).

II.3 METODO PARA AVALIAÇÃO DA QUALIDADE DE SOFTWARE

O método para avaliação da qualidade de software adotado neste trabalho é o proposto por Rocha (Rocha83). Influenciado por trabalhos anteriores de Boehm (Boehm78) e Mc Call (McCall79), este método baseia-se nos seguintes conceitos (Rocha87):

- **Objetivos de qualidade:** são propriedades gerais que o produto deve possuir;

- **Fatores de qualidade do produto:** determinam a qualidade do ponto de vista dos diferentes usuários do produto (usuário final, mantenedores, etc.);

- **Critérios:** atributos primitivos possíveis de serem avaliados;

- **Processos de avaliação:** determinam o processo e os instrumentos a serem usados de forma a se medir o grau de presença, no produto, de um determinado critério;

- **Medidas:** são o resultado da avaliação do produto segundo os critérios;

- **Medidas agregadas:** são o resultado da agregação das medidas obtidas na avaliação dos critérios, quantificando os fatores.

A figura II.1 mostra a estrutura do método para avaliação da qualidade de software.

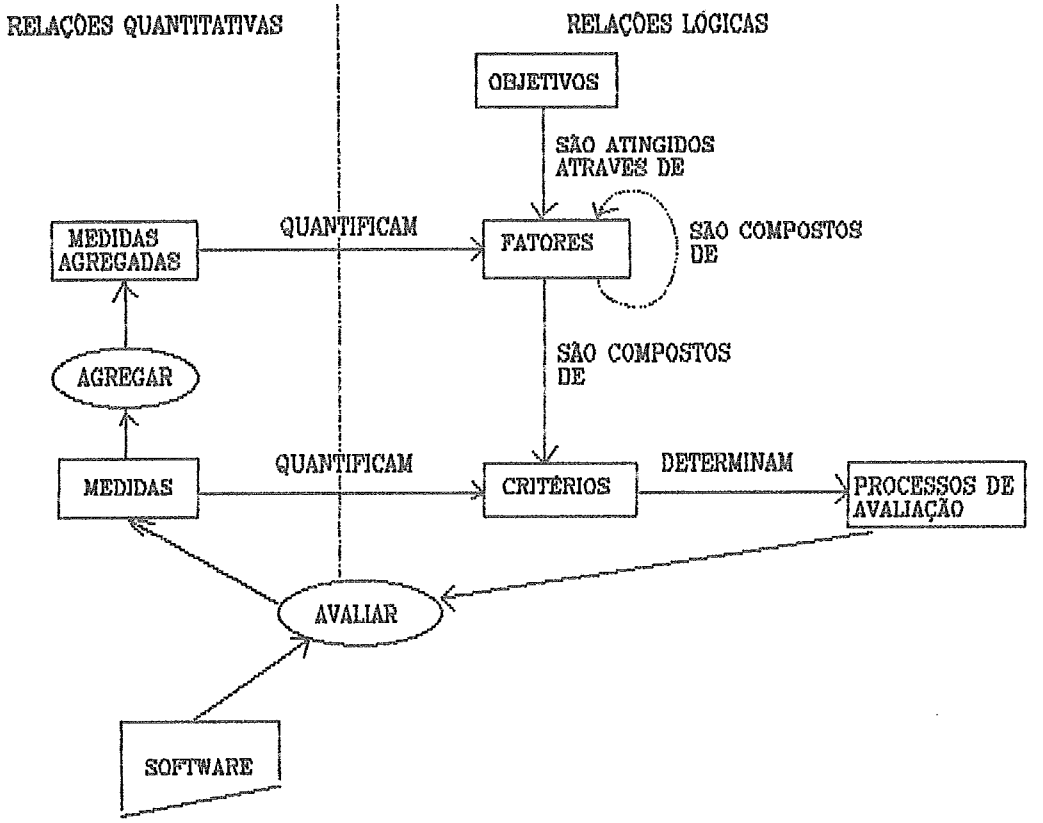


Figura II.1: Método para avaliação da qualidade de software
Fonte: Rocha 87

Os objetivos de qualidade são atingidos através dos fatores de qualidade, que podem ser compostos por outros

fatores e são avaliados através de critérios. Estes definem atributos de qualidade para os fatores.

Medidas são valores resultantes da avaliação de um produto segundo um critério específico.

Objetivos e fatores não são diretamente mensuráveis e só podem ser avaliados através de critérios. Um critério é um atributo primitivo, isto é, independente de todos os outros atributos. Nenhum critério isolado é uma descrição completa de um determinado fator ou subfator. Da mesma forma, nenhum fator define completamente um objetivo.

Esse método está sendo utilizado para definir objetivos, fatores e subfatores para todas as fases do desenvolvimento de software, bem como está sendo particularizado para atender as especificidades de diferentes áreas de aplicação (Clunie 87, Palermo 88, Nascimento 86, Stahl 88, Rocha 90).

Neste trabalho, o método foi utilizado para definir critérios e processos de avaliação para os atributos de qualidade relacionados à fase de projeto e que serão apresentados no próximo capítulo.

II.4 QUALIDADE DA ESPECIFICAÇÃO DE PROJETO

II.4.1 A FASE DE PROJETO DE SOFTWARE

O objetivo da fase de projeto é determinar uma solução viável para o sistema definido na Especificação de Requisitos, mantendo as características de qualidade já

especificadas, dentro das restrições econômicas e técnicas impostas. Os produtos desta fase são (Rocha87):

- o Projeto da Arquitetura, que é um refinamento da especificação requisitos do sistema com o objetivo de chegar a uma organização modular lógica (projeto lógico), capaz de tornar real o programa em algum ambiente de programação. Resulta na definição das interfaces entre elementos internos e externos ao software, descrição funcional de cada elemento do software, detalhamento da estrutura de dados utilizada e considerações especiais de empacotamento. Nesta etapa, é irrelevante o código que o módulo venha a possuir, sendo importante apenas a sua função.

- o Projeto Detalhado, que é a especificação dos algoritmos que implementam as funções identificadas na organização modular (projeto físico). É um esboço para a codificação e como tal, deve proporcionar informação suficiente para que alguém, diferente do próprio projetista, possa desenvolver o código fonte resultante.

- o Projeto de Arquivos, que é feito em paralelo à produção do Projeto da Arquitetura e do Projeto Detalhado. Este produto define o conteúdo dos arquivos, bem como as características físicas de armazenamento e acesso aos dados.

Segundo Sommerville (Sommerville82), um bom projeto de software é a chave para a construção de um software eficiente. Por ser considerada a fase mais crítica do

processo de criação do software, nos últimos anos, houve uma grande preocupação em se formalizar alguns procedimentos e normas para elaboração da especificação do projeto. Na literatura técnica estão apresentadas várias propostas de métodos e linguagens de apoio a construção da especificação de projeto. Entre os métodos propostos destaca-se, pela difusão de seu uso, o método Projeto Estruturado (Stevens 74, Myers78, Yourdon 79, Page-Jones 80, Gane 83).

Projeto Estruturado propõe a construção da organização modular a partir do fluxo de dados. O escopo deste enfoque é projetar programas como estrutura de módulos de funções únicas, independentes, que facilitam o entendimento, tornando os sistemas manuteniveis. Visa atender aos seguintes objetivos:

- permitir que a forma do problema guie a forma da solução;

- contornar a complexidade de sistemas grandes, particionando o sistema em módulos de funções simples e organizando-os em hierarquias convenientes de modo que a união destas funções permita a solução de problemas complexos. As caracteristicas de uma caixa preta compreendem: conhecimento das entradas esperadas; conhecimento das saídas que devem ser devolvidas; conhecimento das funções internas (o que faz as suas entradas produzirem suas saídas); desconhecimento de como suas funções são executadas;

- utilizar linguagem gráfica, para tornar o projeto comunicável. Um Gráfico de Estrutura (GE) ilustra o particionamento de um sitema em módulos mostrando sua hierarquia, organização e comunicação, sem entretanto, mostrar o procedimento e os dados internos de cada módulo. O pseudocódigo é uma linguagem informal e muito flexível, usada para descrever o algoritmo dos módulos do GE e organizar os pensamentos do programador antes da codificação;

- fornecer um conjunto de estratégias para projetar;

- oferecer um conjunto de diretrizes para identificação de um bom projeto, baseado na funcionalidade e inter-relacionamento dos módulos.

Segundo Page-Jones (Page-Jones80), Projeto Estruturado produz sistemas que são fáceis de entender, confiáveis, flexíveis, eficientes, facilmente desenvolvidos e que funcionam.

II.4.2 QUALIDADE DA ESPECIFICAÇAO DE PROJETO

O projeto de software não é um processo mecânico, mas uma atividade humana que requer muita clareza de pensamento, trabalho, e revisão para ser bem sucedida. Por isso, é necessário manter o projeto dentro dos limites de entendimento humano, definindo a sua solução em termos de uma estrutura de entidades (módulos) que descrevem uma função simples dentro do contexto da estrutura total.

Jensen e Tonies (Jensen 79) consideram como atributos de um bom projeto:

- **necessidade:** apenas os requisitos de desempenho e as características de projeto necessários para alcançar os requisitos de utilizabilidade são incluídos no projeto. Características extras desejáveis, mas que não são realmente necessárias, devem ser eliminadas. A simplicidade é uma característica essencial.

- **completeza:** todos os módulos na estrutura são identificados e todas as interfaces especificadas. Todas as funções e tarefas a serem desempenhadas pelo software para atingir os requisitos do usuário, devem ser identificadas e definidas no nível de detalhe apropriado à fase de projeto.

- **consistência:** as incompatibilidades na filosofia do projeto devem ser identificadas e resolvidas.

- **rastreabilidade:** deve haver uma auditoria capaz de relatar se os requisitos estabelecidos pelo usuário foram satisfeitos na especificação de projeto.

- **viabilidade:** a especificação de projeto deve estar de acordo com as decisões tomadas nos estudos preliminares.

- **praticabilidade:** os pontos críticos do projeto devem ser demonstrados através da implementação de um protótipo, se necessário. Deve ser mostrado que o propósito do projeto é implementável dentro das restrições de custo,

cronograma e configuração de equipamentos, caso esta tenha sido especificada.

Evans e Marcianiak (Evans87) mostram que elaborar o projeto de software, a partir da definição de requisitos, é um processo iterativo, decompondo requisitos sucessivamente dentro de baixos níveis de abstração até o nível mais baixo, onde a codificação pode começar.

Os componentes do Projeto da Arquitetura devem possuir os seguintes atributos:

- **completeza:** o projeto deve conter uma definição completa do sistema alocada a subsistemas individuais ou componentes de software. A partir desse conjunto funcional detalhado, o projeto detalhado deve ser particionado e construído através de desenvolvimento, integração e teste.

- **avaliabilidade:** todos os requisitos funcionais devem ser verificados por meio de teste funcional, análise funcional ou de desempenho, ou observação. O projeto funcional, assim como os requisitos operacionais ou do usuário, deve ser avaliado para ser correto.

- **rastreabilidade:** todo projeto funcional deve ser diretamente mapeado para os requisitos a partir do qual foi derivado.

- **validade funcional:** toda a especificação funcional do projeto deve ser clara, concisa e funcionalmente válida na luz da realidade operacional, do sistema e do hardware.

- **validade de dados:** o projeto funcional deve prover uma definição clara dos relacionamentos entre os dados existentes no sistema, e os dados e os fluxos de controle previstos no projeto. A estrutura de dados é a base para um projeto de software eficiente.

- **realizabilidade:** o projeto funcional deve ser realizável à luz dos requisitos de desempenho, dos requisitos de desenvolvimento e da realidade implementacional do sistema. A melhor especificação de projeto é inútil se não corresponder à capacidade de trabalho do sistema.

Os componentes do Projeto Detalhado devem possuir os seguintes atributos:

- **avaliabilidade e rastreabilidade:** da mesma forma que no projeto da Arquitetura, a definição do projeto detalhado deve ser avaliada e mapeada aos requisitos específicos do usuário.

- **integridade:** o projeto deve ser desenvolvido de acordo com os requisitos do usuário.

- **implementabilidade:** o projeto deve estar elaborado de acordo com os recursos e experiências disponíveis e os requisitos técnicos, de cronograma e de orçamento.

- **características do projeto:** as heurísticas, o acoplamento e os atributos do software projetado devem estabelecer uma base sólida para o desenvolvimento de um sistema de qualidade.

- **tamanho e particionamento:** o projeto deve ser particionado em unidades pequenas, modulares e funcionais, para facilitar a manipulação de uma implementação grande e complexa.

- **integridade estrutural:** o projeto deve ser estruturado para proporcionar interfaces claras, bem definidas e preditíveis entre os vários componentes do projeto.

- **confiabilidade:** o projeto detalhado deve apresentar uma definição completa das condições de erro, dos métodos de recuperação e do tratamento de condições que requeiram processamento especial para garantir a integridade do sistema.

Parnas e Weiss (Parnas85) salientam que, de acordo com o objeto a ser projetado, é possível identificar propriedades desejáveis em um projeto, tais como:

- **estar bem estruturado**, isto é, o projeto deve ser consistente com os princípios escolhidos, tal como o princípio de isolamento.

- **ser o mais simples possível.**

- **ser eficiente** de forma que, as funções desempenhadas pelo projeto possam ser executadas dentro dos recursos de máquina disponível sem, entretanto, interferir nos requisitos de espaço e tempo.

- **ser adequado** satisfazendo os requisitos do usuário.

- **ser flexível** para atender com facilidade aos requisitos de mudança.

- **ser prático**, isto é, o módulo de interfaces deve ser suficiente para a realização da tarefa, de forma que o usuário não tenha que desempenhar funções extras para executá-lo.

- **ser implementável**,isto é, as funções oferecidas devem estar de acordo com as informações disponíveis.

- **estar padronizado** de acordo com o padrão de documentação definido pela organização para elaboração do projeto.

Fairley (Fairley85) considera os seguintes conceitos como fundamentais em um projeto:

- **abstração:** separação dos aspectos conceituais do sistema dos detalhes de implementação. A abstração permite organizar o pensamento adiando as considerações estruturais e detalhes algoritmicos até que características funcionais, de fluxo e de armazenamento de dados tenham sido definidas, reduzindo, com isso, a complexidade a um ponto específico do projeto.

- **isolamento:** quando um sistema de software é projetado usando o princípio do isolamento (Parnas72), cada módulo no sistema deve esconder os detalhes internos da sua atividade de processmanto e comunicar apenas interfaces bem definidas. Cada módulo deve ser projetado para esconder as decisões internas dos outros módulos da estrutura.

- **estrutura:** o uso de estruturação permite a decomposição de um sistema grande em sistemas menores (unidades facilmente gerenciáveis com relacionamentos bem definidos).

- **modularidade:** sistemas modulares são compostos de unidades bem definidas, gerenciáveis com interfaces bem definidas entre as unidades. Propriedades desejáveis de um sistema modular:

  - cada abstração processada é um subsistema bem definido que é potencialmente útil em outras aplicações;
  - cada função em cada abstração tem um propósito único e bem definido;
  - cada função manipula não mais do que uma estrutura de dados principal;
  - funções compartilham dados globais seletivamente. É fácil identificar todas as rotinas que compartilham uma estrutura de dados principal.

Modularidade torna o projeto claro, facilitando a implementação, depuração, teste, documentação e manutenção do sistema.

- **verificação:** um projeto é verificável se pode ser demonstrado que o projeto resultará numa implementação que satisfaça os requisitos do usuário.

- **estética:** é difícil listar critérios objetivos para avaliar fatores estéticos do software, mas um produto agradável, do ponto de vista estético, é facilmente

reconhecido. Simplicidade, elegância e clareza distinguem os produtos de qualidade de produtos medíocres.

Conte, Dunsmore e Shen (Conte86) recomendam a modularização de programas para se atingir a qualidade na fase de projeto. Algumas vantagens da modularização:

- **compreensibilidade:** ambos, programador e usuário, podem entender mais facilmente a lógica do programa;

- **gerenciabilidade:** os gerentes podem designar pessoal para cada módulo separadamente e a responsabilidade torna-se mais localizada;

- **eficiência:** os esforços de implementação são reduzidos;

- **redução de erros:** testar módulos independentes é mais fácil;

- **manutenção reduzida:** identificação dos módulos é facilitada já que diferentes funções são desempenhadas por módulos diferentes.

Pressman (Pressman 88) recomenda, como medidas qualitativas, que:

- um projeto deve exibir uma organização hierárquica entre elementos de software;

- um projeto deve ser modular, isto é, o software deve ser logicamente particionado em elementos que desempenham funções e subfunções específicas;

- um projeto deve conduzir a módulos (isto é, subrotinas e procedimentos) que exibem características funcionais independentes;

- um projeto deve ser derivado através de refinamentos sucessivos devendo retratar a informação obtida durante a análise de requisitos do software.

Segundo Dunn e Ullman (Dunn82), programas que apresentam problemas de controle e qualidade são aqueles que envolvem um grande número de pessoas (implicando em comunicação), pessoas de um meio distinto dos desenvolvedores (implicando em diferentes significados) e uma grande expectativa de uso.

Documentação pobre, programas complexos e baixo nível de modularidade podem tornar a manutenção quase impossível, certamente mais cara e mais sujeita a erros do que necessitaria ser.

Sob o aspecto técnico de desenvolvimento de software em fases, os autores argumentam que um projeto modular talvez tenha atingido o maior efeito encontrado na qualidade durante o ciclo de vida. Recomendam que a documentação de projeto seja revista tendo em mente: a independência entre os módulos, a especificação de cada módulo com detalhe suficiente e uma hierarquia de módulos bem definida.

Partindo do método descrito na seção 2.2, Rocha (Rocha87) definiu objetivos, fatores e subfatores para a fase de projeto.

Na fase de projeto foram identificados três objetivos: confiabilidade da representação, confiabilidade conceitual e utilizabilidade.

A **confiabilidade da representação** é necessária para facilitar a elaboração e a manutenção do software, tornando-o compreensível ao longo do seu desenvolvimento. A **confiabilidade conceitual** é o objetivo fundamental a ser atingido, pois refere-se à correção do projeto com relação ao seu conteúdo. Não é possível continuar o desenvolvimento e iniciar a construção baseando-se em um projeto não utilizável. Daí o objetivo, **utilizabilidade.**

A figura II.2 mostra os fatores e subfatores de qualidade de projeto relacionados a estes objetivos.

O objetivo confiabilidade da representação é atingido através de dois fatores: comunicabilidade e manipulabilidade.

Comunicabilidade refere-se a capacidade de comunicação da especificação do projeto. O fator comunicabilidade é atingido através de cinco subfatores:

- **correção no uso da linguagem de projeto:** é a característica de um projeto estar correto do ponto de vista do uso da linguagem de projeto, no que se refere à notação e regras de aplicação;

- **concisão:** é a característica de um projeto estar escrito com um mínimo de volume de texto, isto é, com uma

maximização da quantidade de informações por unidade de texto;

- **uniformidade de terminologia:** é a característica de um projeto estar escrito de maneira que não existam termos diferentes descrevendo o mesmo objeto do mundo real;

- **uniformidade no nível de abstração:** é a característica de um projeto estar descrito de modo que, a cada passo na sequência de refinamentos não sejam impostas restrições para os passos posteriores e que, em um determinado nível de abstração, todos os aspectos estejam descritos com o mesmo nível de detalhe;

- **modularidade:** é a característica de um projeto estar elaborado com uma estrutura modular adequada.

Manipulabilidade refere-se a facilidade de manipulação do projeto no seu uso, modificação, etc.. O fator manipulabilidade é atingido através dos seguintes subfatores:

- **disponibilidade:** é a característica que permite que o projeto possa ser consultado por desenvolvedores e mantenedores, na sua versão mais atualizada;

- **estrutura:** é a característica de um projeto estar elaborado dentro de um padrão definido de composição de suas partes de forma que facilite ao leitor encontrar a visão desejada;

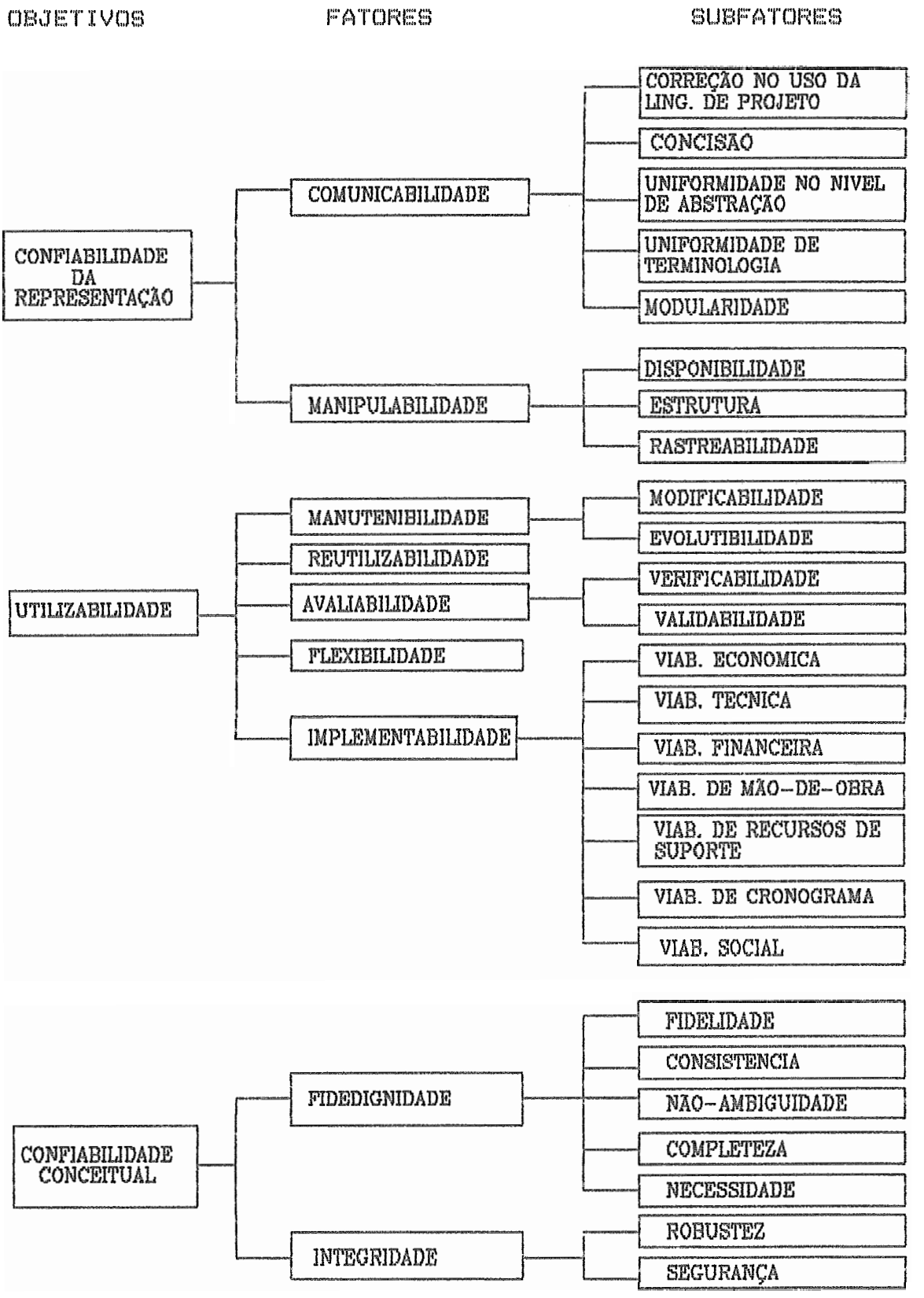


Figura II.2: Objetivos, fatores e subfatores da qualidade de projeto

Fonte: ROCHA87

- **rastreabilidade:** é a característica de um projeto permitir um acompanhamento de uma seqüência de detalhes relacionados a um determinado aspecto do problema, desde a sua visão mais geral até a mais detalhada e vice-versa.

O objetivo confiabilidade conceitual é atingido através dos fatores fidedignidade e integridade.

O fator **fidedignidade** é o atributo que o projeto deve ter de forma a permitir a confiança de que ele realmente retrata, a nível do projeto, o que foi especificado. Fidedignidade é atingida através dos seguintes subfatores:

- **fidelidade:** é a característica de um projeto estar escrito de acordo com a especificação de requisitos, a fim de manter fidelidade ao problema que resolve;

- **consistência:** é a característica de um projeto estar elaborado de modo que as partes que o compõe se complementem sem estabelecer contradições;

- **não ambiguidade:** é a característica de um projeto estar elaborado isento da possibilidade de falsas interpretações;

- **completeza:** é a característica de um projeto estar elaborado contendo todas as funções descritas na Especificação de Requisitos;

- **necessidade:** é a característica de um projeto estar elaborado sem conter outras funções além das descritas na Especificação de Requisitos.

O fator **integridade** é o atributo que o projeto deve ter de forma a permitir confiança em sua resposta a falhas e situações hostis. Integridade é atingida através dos seguintes atributos:

- **robustez**, que é a característica do projeto prever soluções para situações hostis ao software, prevendo o adequado tratamento das mesmas;

- **segurança**, que é a característica do projeto prever soluções para se evitar falhas que possam provocar um alto risco em termos de perdas de vidas humanas ou grandes prejuízos financeiros decorrentes do mau funcionamento do software, sempre que a natureza da aplicação o exigir.

Não tem sentido continuar o desenvolvimento e iniciar a construção baseados em um projeto não utilizável. A utilizabilidade de um projeto depende dos seguintes fatores: manutenibilidade, reutilizabilidade, avaliabilidade, flexibilidade e viabilidade.

O fator **manutenibilidade** é o atributo que o projeto deve ter de forma a poder ser facilmente modificado (subfator modificabilidade) e expandido (subfator evolutibilidade).

O fator **reutilizabilidade** é o atributo que o projeto deve ter de forma a poder ser reutilizado, totalmente ou em parte, ao se realizar o desenvolvimento de outro software com funções semelhantes.

Especificações de projeto devem ser avaliadas. O fator **avaliabilidade** é atingido através dos seguintes subfatores:

- **validabilidade**, de forma que seja possível realizar um processo finito e economicamente viável para avaliar se o projeto atende às especificações;

- **verificabilidade**, de forma que o projeto possa ser avaliado com relação à forma de sua representação.

O fator **flexibilidade** é o atributo do projeto estar elaborado sem definir com rigidez como o software deverá ser construído. Isto é, nesta fase não devem ser impostas restrições próprias da implementação, pois podem acarretar na perda de atributos de qualidade, tais como, eficiência, portabilidade, reutilizabilidade entre outros, nos programas a serem gerados.

Por fim, é fundamental que o projeto mantenha a **implementabilidade** já avaliada na especificação de requisitos, em todos os seus aspectos (viabilidade econômica, técnica, financeira, de mão-de-obra, de recursos de suporte, de cronograma, social).

II.4.3 MODULARIDADE DE PROJETO

Dentre os atributos de qualidade necessários de serem atingidos, durante a fase de projeto, destaca-se o atributo modularidade.

Modularidade é um atributo relacionado à qualidade da representação que afeta vários outros atributos, tais

como manutenibilidade, reutilizabilidade e avaliabilidade. A definição da estrutura modular é uma decisão de projeto e deve ser avaliada nesta fase. A constatação de um baixo grau de modularidade pode significar uma volta ao início da fase de projeto com sérias incidências no custo e no cronograma (Nascimento 86).

O objetivo de se buscar a modularidade é a necessidade de se vencer a complexidade, tanto a nível da estrutura modular do programa quanto a nível dos diferentes módulos. O que se quer é ter uma representação simples e legível que possibilite realizar verificações, manutenções e reutilizações de partes.

Preocupados com a manutenibilidade do software, Myers, Stevens, Yourdon e Constantine (Stevens 74, Stevens 86, Yourdon 79) pesquisaram aspectos relacionados à modularidade do projeto, estabelecendo como propriedades principais desejáveis para os módulos a coesão e o acoplamento. Posteriormente, outros estudos foram realizados e novas propriedades foram somadas àquelas e incluídas como características desejáveis para se ter um bom nível de qualidade no Projeto Estruturado.

As características, mais comumente, identificadas como determinantes da modularidade a nível da especificação de projeto, são:

ACOPLAMENTO (Yourdon 79, Stevens 86, Page-Jones 80, Nascimento 86, Staa 83)

Acoplamento é o grau de interdependência entre dois módulos. O objetivo é minimizar o acoplamento entre os módulos para que eles sejam tão independentes quanto possível. Quanto menor for o acoplamento, menos provável será que outros módulos tenham que ser considerados para entender, consertar ou modificar um módulo.

A complexidade pode ser reduzida através do projeto de programas onde se busca atingir o acoplamento mais fraco possível entre módulos. O acoplamento é medido não apenas pelo número de conexões, mas pelo tipo de conexão e o tipo de informação comunicada na conexão.

Baixo acoplamento é indicação de que um sistema está bem particionado. Os principais motivos pelos quais busca-se baixo acoplamento entre os módulos são:

- evitar a propagação de erros, pois quanto menos conexões existirem entre os módulos menor é a chance de que um erro em um determinado módulo repercuta em outro;

- facilitar a troca ou modificação um módulo com risco mínimo de se ter que alterar outros módulos;

- facilitar a manutenção de um módulo, pois não é necessário conhecer detalhes internos dos módulos com os quais este se relaciona.

Existem diferentes tipos de acoplamento que estão citados a seguir na ordem do mais baixo para o mais alto acoplamento:

. **Acoplamento por dados:** dois módulos são acoplados por dados se a comunicação entre eles é feita por parâmetros, sendo cada parâmetro um dado elementar ou uma tabela homogênea, ou seja, uma tabela onde cada entrada contem informações do mesmo tipo. O acoplamento de dados é a comunicação de dados necessária entre módulos.

. **Acoplamento por dados estruturados:** dois módulos são acoplados por dados estruturados se a comunicação entre eles é feita através de estruturas de dados. O problema deste tipo de acoplamento é a existência de dependência entre módulos não relacionados.

. **Acoplamento por controle:** dois módulos são acoplados por controle se na comunicação entre eles, um módulo passa informações para o controle interno do outro. Um problema com este tipo de acoplamento é a "inversão da autoridade", onde, um módulo subordinado passa um parâmetro que controla o módulo de nível superior.

. **Acoplamento por área de dados comum:** dois módulos são acoplados por área de dados comum se eles se referem à mesma área de dados global. Este tipo de acoplamento não é desejável, pois:

- pode ocorrer um erro em um módulo que usa área global de dados e este mesmo erro aparecer em outro módulo qualquer que utiliza a mesma área;

- é difícil descobrir qual o dado que deve ser modificado ao se alterar um módulo, assim como é difícil determinar quais os módulos a serem modificados quando uma parte do dado é alterada.

. **Acoplamento por conteúdo:** dois módulos exibem um acoplamento de conteúdo, se um dos módulos faz referência ao interior do outro, através de um desvio, uma referência aos dados ou uma alteração de comandos de outro módulo.

Este tipo de acoplamento deve ser evitado, pois afeta o princípio de unidade independente do módulo, uma vez que um módulo tem que ter conhecimento do conteúdo e da implementação do outro.

Para Staa (Staa 83) o fator acoplamento avalia as propriedades entre módulos que se comunicam diretamente, sem no entanto considerar a organização modular como um todo. Considera a comunicação entre módulos estabelecida através de conectores. Cada conector é capaz de comunicar um conjunto de dados e/ou sinais de controle. Um conector é um ponto, ou área, no código, ao qual podemos passar controle de execução ("call", "resume", "signal", "goto") ou um ponto, ou área de memória, onde colocamos dados a serem permutados de alguma forma entre os dois módulos.

Desta forma, argumenta que o grau de acoplamento deve ser avaliado através dos seguintes critérios:

Número de conectores: avalia o número de conectores. Quanto mais conectores existirem, mais alto será o acoplamento e mais difícil será o entendimento do módulo,

mais difícil será a substituição do módulo por outro equivalente e mais dificilmente o módulo poderá ser reutilizado em outro programa.

Tamanho do conector: avalia o número de elementos que figuram na lista que descreve o conector (lista de parâmetros, estrutura de dados, "records",etc.).

Coesão do conector: avalia o inter-relacionamento dos elementos de um conector. Desta forma, se forem definidos diversos conectores, e cada conector contiver todos os elementos relacionados a um único conceito (conector altamente coeso), o acoplamento é menor (melhor) do que se todos estes objetos fossem colocados em um único conector, tornado-o pouco coeso. A razão disso é a dificuladade do entendimento inerente a um grande volume de elemntos não agrupados logicamente.

Complexidade de um conector: avalia o esforço realizado pelo programador para estabelecer a conexão. A complexidade depende de requisitos de validação, de siginificado, posicionais e de delimitação.

Tipo de dado: avalia o efeito dos dados recebidos pelo módulo sobre o próprio módulo. Um dado puro é um objeto que não tem um significado especial quanto ao seu uso pelo módulo receptor, ao passo que um dado de controle ("flag", chave) determina o que o módulo receptor deve fazer.

Complexidade do dado: avalia a dificuldade de se entender o signicado do dado em questão. Um tipo abstrato

(uma estrutura, um arquivo uma lista, etc.) é mais complexo do que um dado elementar (um número inteiro, por exemplo), pois possui propriedades que contribuem para dificultar o seu entendimento.

Epoca de conexão: avalia o instante em que é estabelecida a conexão, isto é, em tempo de programação, compilação, carga ou execução. Quanto mais tarde for estabelecida a conexão, mais modificável será o programa.

Memorização: avalia o espaço de memória para manter o estado das variáveis e/ou do algoritmo, entre ativações sucessivas. Sempre que o estado dos elementos manipulados pelo módulo deve ser mantido de uma ativação para outra, tem-se memorização.

COESÃO (Page-Jones 80, Stevens 86, Nascimento 86, Staa 87)

É a medida do tipo de relacionamento que existe entre elementos de um mesmo módulo. Quanto mais forte é este relacionamento, melhor o módulo será visto como uma unidade independente.

Certificando-se que todos os módulos tem boa coesão é o melhor modo de minimizar o acoplamento entre eles.

Existem diferentes níveis de coesão que estão descritos, a seguir,na ordem da mais alta para a mais baixa coesão (Page-Jones 80).

. **Coesão funcional:** um módulo tem coesão funcional quando contem elementos que contribuem para a execução de uma e somente uma tarefa relacionada ao problema. Caso se

possa identificar tudo o que é executado por um módulo e obter-se como resultado uma única função relacionada ao problema, então este módulo tem coesão funcional.

. **Coesão sequencial:** um módulo tem coesão sequencial se os seus elementos constituintes estão envolvidos em atividades onde o dado de saída de uma atividade serve como entrada para a atividade seguinte.

Um módulo com coesão sequencial normalmente tem um bom nível de acoplamento e é de fácil manutenção. A única desvantagem do módulo com coesão sequencial é que não é tão facilmente reutilizável quanto o módulo com coesão funcional. Isto porque, um módulo com coesão sequencial contém atividades que unidas perdem a utilidade no geral.

. **Coesão comunicacional:** um módulo tem coesão comunicacional se os seus elementos contribuem para atividades que utilizam o mesmo dado de entrada ou saída.

Módulos com coesão comunicacional e módulos com coesão sequencial são similares, pois ambos contém atividades organizadas sobre os dados do problema original. A principal diferença é que nos módulos com coesão sequencial as atividades individuais são executadas em uma ordem determinada e nos módulos com coesão comunicacional a ordem de execução não é importante. Estes módulos têm, também, um baixo nível de acoplamento, porque poucos dos seus elementos são relacionados com os elementos de outros módulos.

. **Coesão procedural:** um módulo tem coesão procedural se os seus elementos constituintes estão envolvidos em atividades diferentes e possivelmente não relacionadas, onde o controle passa de uma atividade para outra. Módulos com coesão procedural apresentam nível médio de coesão e as manutenções já não são tão fáceis.

. **Coesão temporal:** um módulo tem coesão temporal se os seus elementos estão envolvidos em atividades relacionadas apenas pelo momento em que ocorrem. Módulos com coesão procedural e módulos com coesão temporal são similares. A diferença entre estes dois tipos de coesão é que nos módulos com coesão procedural a ordem de execução é mais importante.

. **Coesão lógica:** um módulo tem coesão lógica se os seus elementos pertencem à mesma classe lógica de funções, como por exemplo, um módulo que executa todas as entradas. Entretanto, os elementos são grosseiramente colocados na mesma classe, com algumas semelhanças e também diferenças.

. **Coesão coincidental:** um módulo tem coesão coincidental se os seus elementos estão envolvidos em atividades cujos relacionamentos não tem nenhum siginificado, apenas aconteceu, coincidentalmente, de estarem juntos.

A coesão coincidental é similar à coesão lógica, pois suas atividades não estão relacionadas nem pelo fluxo de dados, nem pelo fluxo de controle. Como os módulos não possuem nenhuma função bem definida, precisam ser ativados

através de um "flag", completamente artificial, para dizer a eles o que fazer. A diferença entre estes dois tipos de coesão é que na coesão lógica existe um critério, apesar de sua pouca eficácia, enquanto na coesão coincidental não existem critérios.

Segundo Staa (Staa 87), apesar do conceito coesão já ter surgido há muito tempo, a sua definição, ainda, não possui características, que permitam utilizá-lo, sistematicamente, como um instrumento eficaz de controle da qualidade. Assim, torna-se necessário uma definição do conceito de projeto modular, de modo que se possa utilizar o fator coesão para avaliar a estruturação proposta nas diversas classes de abstrações.

Ao especificar, projetar e/ou implementar sistemas de programação, trabalha-se com abstrações e com várias soluções modulares denominadas por Staa de conjuntos de solução destas abstrações. Esses conjuntos surgem como consequência de uma etapa do projeto, podendo ser de decomposição ("top-down") ou de agregação ("bottom-up").

Cada conjunto de solução representa uma solução para a abstração raiz e deve corresponder a uma solução exata, de modo que o trabalho possa ser continuado, sem necessitar de correções nas decisões tomadas anteriormente. A qualidade do conjunto de solução é avaliada pela coesão.

Staa (Staa 87) define coesão como o fator de qualidade composto pelos seguintes critérios de avaliação:

- **necessidade:** não existe elemento pertencente ao conjunto de solução, que possa ser retirado sem comprometer a capacidade do conjunto de solução ser uma solução completa da abstração raiz.

- **suficiência:** o conjunto de elementos do conjunto de solução resolve completamente o problema abstrato, ou seja, não existe elemento que possa ser adicionado a este conjunto de solução de modo que este conjunto de solução aumentado passe a ser uma solução mais completa da abstração raiz.

- **integrabilidade:** as interfaces entre os elementos do conjunto solução estão definidas completa e consistentemente. A integrabilidade viabiliza a integração das soluções das abstrações contidas no conjunto de solução.

- **não agregabilidade:** não é possível encontrar um subconjunto de elementos pertencentes ao conjunto de solução que possa ser agregado para resolver uma abstração raiz própria.

- **independência:** qualquer que venha a ser a solução obtida para uma determinada abstração contida no conjunto de solução, esta não interfere nas soluções dadas às demais abstrações do conjunto de solução.

- **viabilidade:** cada elemento do conjunto de solução ou é uma solução primitiva ou, então, admite uma solução de modo que os objetivos e requisitos da abstração raiz possam ser alcançados.

Para o autor, é evidente que esta definição ainda possui subjetividade. Entretanto, a variação de opinião decorrente desta subjetividade é, significativamente, menor do que nas definições originais dadas à coesão.

NÚMERO DE MÓDULOS SUBORDINADOS ("Fan-out") (Page-Jones 80, Stevens 86)

O número de módulos subordinados de um módulo ("fan-out") é o número de módulos imediatamente inferiores aquele módulo.

Um número muito alto ou muito baixo de módulos imediatamente subordinados são indicadores de um projeto pobre, sendo que um alto "fan-out" é mais perigoso, pois torna o módulo mais difícil de ser reutilizado.

Page-Jones (Page-Jones 80) considera que sete é um número muito importante na psicologia humana. Alguém programando um módulo com sete subordinados, deve ter sete problemas em sua mente, estando, portanto, próximo do limite de sua capacidade de processar informações com segurança.

Assim, sendo, sugere que o número de subordinados a um determinado módulo deve estar limitado a sete. A partir daí, o módulo coordenador passa a ser complexo e de difícil implementação.

Entretanto, Card (Card 88) considera fan-out igual a três o número que minimiza a complexidade da estrutura total do sistema e o valor usualmente aceito de 7 mais ou menos 2 é excessivo. Justifica aqui que, Constantine

(Stevens 74) observou que a maioria dos programas podem ser decompostos efetivamente em uma estrutura comum de três pontos: entrada, processamento e saída.

NÚMERO DE MÓDULOS SUPERIORES ("Fan-in") (Page-Jones 80, Stevens 86)

O número de módulos superiores de um módulo é o número de módulos que chamam um mesmo módulo subordinado. Neste caso, está se evitando duplicação de código.

Se um módulo tem uma função útil, ele pode ser usado em qualquer lugar no sistema. Assim, os módulos superiores podem ser de mesmo nível ou de níveis diferentes. Deve-se, entretanto, tomar cuidado para não criar módulos com baixa coesão e forte acoplamento.

Há duas regras básicas para restringir o uso de "fan-in":

- módulos com "fan-in" devem ter boa coesão, funcional ou pelo menos toleravelmente comunicacional ou sequencial;

- a interface para cada módulo que o chama deve ter o mesmo número e tipos de parâmetros.

TAMANHO DO MÓDULO (Arthur 85, Card 85, Conte 86, Yourdon 79, Nascimento 86)

O tamanho de um módulo não é uma característica governante, mas deve ser considerado uma vez que todos os

outros atributos mais importantes já tenham sido satisfeitos.

Um bom tamanho para um módulo é cerca de meia página de listagem, aproximadamente 30 comandos, codifi-cados em uma linguagem de alto nível. Mas, certamente, todo o código de um módulo deve estar em uma página de listagem o que limita em 60 o número máximo de comandos ou em duas páginas (aproximadamente 120 comandos) (Nascimento86).

Yourdon (Yourdon 79) considera 50 comandos e Boehm (Boehm78) 100 comandos o tamanho adequado. Conte (Conte86) considera de 50 a 200 comandos o tamanho adequado para se ter um bom entendimento do módulo e minimizar a ocorrência de erros. Arthur (Arthur 85) considera que o adequado é se ter 1 ou 2 páqinas de listaqem, e ressalta que quanto maior for a descrição de uma função simples, maior a chance de que a função não seia uma única função (módulo não funcional).

Card (Card 85) descreveu os resultados de uma pesquisa realizada com 453 módulos escritos em FORTRAN onde concluiu que bons programadores não tem preferência por um tamanho específico para os módulos, que módulos grandes custam menos e que a incidência de erros não está relacionada ao tamanho do módulo e sim à habilidade dos programadores.

Estes estudos, descritos acima, sugerem que o tamanho de um módulo não deve estar limitado por nenhuma norma arbitrária. Entretanto, pode-se afirmar que:

. módulos maiores do que uma ou duas páginas devem ser verificados quanto à possibilidade de serem divididos em mais de um módulo, sempre que isto não comprometa a funcionalidade.

. módulos com menos do que cinco linhas devem ser examinados para ver se podem ser fundidos com seus superiores.

RELATO DE ERROS (Page-Jones 80)

Erros devem ser relatados pelo módulo que os detectou e os identificou, o que torna o sistema mais simples e mais legível uma vez que os módulos devem ser projetados para evitar a necessidade de variáveis de erro.

MEMORIA DE ESTADO (Page-Jones 80)

Memória de estado é um dado interno a um módulo que sobrevive imutável de uma ativação para outra.

Um módulo com memória de estado é geralmente imprevisível. Embora chamado com entradas idênticas, o módulo age diferente e/ou produz saída diferente a cada chamada. Módulos com memória de estado devem ser evitados por serem imprevisíveis, acarretarendo problemas de operação e de manutenção.

ABRANGENCIA (Page-Jones 80)

O Projeto Estruturado objetiva construir módulos que possam ser reutilizados por outros sistemas, ou mesmo, dentro da própria estrutura.

No entanto, é necessário uma dosagem na generalidade do módulo. O maior problema de módulos restritos é a dificuldade de sua reutilização, pois para isso, os módulos devem ser flexíveis.

A generalização para permitir maior flexibilidade dos dados é normalmente menos perigosa do que a generalização para permitir maior flexibilidade de função, pois esta leva a mais código e, consequentemente, pior acoplamento e pior coesão.

SIMPLICIDADE (Staa 83)

Um módulo é simples quando não possui mais controle do que o necessário, não manipula dados de forma específica (o que o torna restrito), todos os dados estão envolvidos com a função executada pelo módulo e esta função é única e clara.

ISOLAMENTO (Staa 83, Nascimento 86)

Este atributo avalia tudo o que é preciso saber com relação a implementação do módulo para que se possa utilizá-lo corretamente. O ideal é que baste conhecer a definição do módulo, onde esta não contenha referência a quaisquer restrições de implementação.

DOCUMENTAÇÃO (Staa 83, Nascimento 86, Pressman 87)

Avalia a qualidade e a disponibilidade da documentação relativa ao módulo. Aplica-se explicitamente aos módulos físicos, uma vez que estes correspondem a unidades reutilizáveis. Módulos físicos são componentes

identificáveis de um programa e devem em princípio, poder existir desvinculados deste programa, sendo, possivelmente, reutilizados em outro programa. Para cada módulo físico deve ser produzido um documento específico contendo:

- a especificação do módulo (objetivo, requisitos, assertivas de entrada e saída, comentário),

- o projeto físico do módulo,

- listagem do código fonte,

- roteiro de teste, dados de teste,

- laudos de teste, verificação e validação, e

- alterações efetuadas (controle de versão ou modificação).

COMPLEXIDADE DO MODULO (McCabe 76)

Depende de suas estruturas de decisão e, como consequência, do número de caminhos que contém. Usando teoria dos grafos, a complexidade é avaliada pelo número de caminhos básicos do programa (como um grafo de controle com uma única entrada e uma única saída):

V(G)=e-n+2p, onde:

e= número de ligações no grafo

n=número de nós no grafo

p=número de componentes conexos no grafo (cada módulo tem um único componente)

A complexidade afeta a testabilidade e manutenibilidade. Segundo McCabe, V(G) menor do que 10 é adequado.

MODULO INICIAL/FINAL (Page-Jones 80)

No desenvolvimento tradicional existe uma grande tendência a agrupar todas as inicializações de dados, tabelas, áreas e outras em um mesmo módulo.

As inicializações devem ser feitas no momento de uso da informação e de preferência nos níveis mais baixos da estrutura. As finalizações devem ser procedidas imediatamente após o término de uso.

Um erro decorrente da má inicialização/finalização é a introdução de memória de estado.

Além das propriedades aplicadas aos módulos, a organização modular deve apresentar as seguintes características:

BALANCEAMENTO (Page-Jones 80)

Uma estrutura é balanceada se os módulos de nível mais alto tratam os dados de uma forma lógica de modo que estes não sejam dependentes das características físicas. Estas características referentes a um dado de entrada ou saída podem ser relegadas aos níveis mais baixos. Sistemas balanceados devem parecer e, frequentemente o são, fisicamente assimétricos.

Estruturas balanceadas são mais fáceis de implementar e de adaptar a mudanças nas especificações, especialmente se estas são nos dispositivos físicos ou formatos de entrada ou saída.

EQUILIBRIO (Staa 83)

Uma organização modular é equilibrada quando funções correlacionadas (como por exemplo, abrir e fechar arquivo) são ativadas a partir de um mesmo módulo. A ativação de tais funções, a partir de módulos diferentes e, pior ainda, em níveis diferentes, torna difícil o entendimento e a correção do programa, pois para saber o que se passa no programa, temos que entender pelo menos toda a porção da organização modular à qual pertençam tais funções.

Além de dificultar o entendimento, os testes e a manutenção, organizações desiquilibradas tendem a deteriorar as outras propriedades relativas à modularidade, em especial a coesão.

FATORAÇÃO (Page-Jones 80)

Ao fazer-se uma avaliação da organização modular, nota-se, muitas vezes, que existem falhas, tais como:

- módulos muito grandes (há duplicidade de código)
- um mesmo módulo executa (cálculos e edições) e gerencia (decisões e chamadas)
- a implementação do código não é simples

A fatoração é a decomposição de módulos visando corrigir uma ou mais dessas falhas. Permite a construção de módulos mais gerais e, consequentemente, elimina a duplicidade de código, simplifica o módulo e melhora a qualidade da estrutura.

COMPLEXIDADE TOTAL DO SISTEMA (Card 90, Conte 86, Henry 90, Staa 83)

O uso de medidas para complexidade tem se tornado muito comum na comunidade de informática, entretanto, o seu uso tem sido restringido às medidas do produto final ou do código fonte. Henry e Salig (Henry 90) realizaram estudos no sentido de chegar a métricas quantitativas que possam ser automatizadas e sirvam para predizer a complexidade do código já na fase de projeto. Tais métricas recaem em três categorias:

- métricas para código que incluem as versões mais populares sobre linhas de código (LOC), parte do estudo sobre ciência de software de Halstead e a complexidade ciclomática de McCabe;

- métricas estruturais que tentam capturar a interconectividade entre os compontentes do programa. Um exemplo destas, é a métrica estrutural proposta por Henry e Kafura que baseia-se nas conexões de fluxos de informação (chamadas "fan-in" e "fan-out") entre um procedimento e seu ambiente. "Fan-in" é o número de fluxos locais entrando num procedimento mais o número de estruturas de dados globais das quais um procedimento recupera informação. "Fan-out" é o número de fluxos locais a um procedimento mais o número de estruturas globais de dados que são atualizadas por este. Assim, definem a complexidade de um procedimento (Cp) como:

$$Cp = (\text{fan-in} \times \text{fan-out})^2.$$

Os resultados preliminares indicaram que as métricas de código dependem do nível de refinamento da especificação enquanto métricas estruturais são independentes do nível de refinamento. Entretanto, as métricas do código e estrutura mediram aspectos diferentes do código fonte.

- métricas hibridas que consistem de uma ou mais métricas de código combinadas com as estruturais, pois as últimas são estáticas e não consideram a dinâmica dos programa. Pelas razões já apresentadas no parágrafo anterior, os autores acharam rasoável propor uma métrica composta por estes dois tipos. A complexidade híbrida do procedimento (HCp) é calculada por:

HCp = Cip x Cp, onde Cip é qualquer métrica usada para complexidade do código.

Concluiram que as métricas estruturais e híbridas são extremamente úteis na fase de projeto. Com elas, um projetista pode determinar o nível de refinamento da especificação do projeto detalhado e o resultado da complexidade do código fonte.

Conte (Conte 86) coloca que as métricas relacionadas ao número de caixas no gráfico de estrutura e a profundidade do gráfico podem ser usadas para quantificar complexidade. Alta complexidade é evidenciada por um grande número destas caixas.

Staa (Staa 83) considera que a complexidade da organização modular depende do número de módulos, do nível deste módulos (número de arestas de ativação), da

distribuição de ativação ("fan-out") e da concentração de ativação ("fan-in"). Quanto maior for a concentração de ativação, maior é o grau de reutilização do módulo e menor é a complexidade da organização modular (Staa 83).

O mais completo trabalho relacionado à complexidade total do sistema é o de Card e Agresti (Card 90). O objetivo deste estudo foi demonstrar uma medida de complexidade que possa ser aplicada a uma vasta classificação de produtos, sem dar prioridade à forma como são produzidos. Práticas corretas de projeto são essenciais para o alcance de bons projetos.

Os autores consideram que a complexidade total de um sistema (Ct) é composta de duas partes: a complexidade contida dentro de cada parte ou módulo (chamada complexidade local (Lo), ou intramódulo), e a complexidade dos relacionamentos entre as partes ou módulos (chamada complexidade estrutural (Es), ou intermódulo). Assim, tem-se que:

Ct = Lo + Es, onde:

Lo = [Somatório (vi / (fi+1))] / n, e

Es = [Somatório (fi^2)] / n

n = número de módulos no sistema;

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

fi = fan-out do módulo i

Este modelo foi concebido para ser usado na fase de projeto da arquitetura e depende, basicamente, do "fan-

out" (número de módulos subordinados) de cada módulo. Baixo "fan-out" indica baixo acoplamento no sentido de que há poucas conexões, sem obviamente, levar em conta o tipo ou a coesão.

O modelo foi utilizado no estudo de oito sistemas similares em termos de ambiente, aplicação e pessoal, o cálculo da complexidade e a análise subjetiva coincidiram.

II.4.4 QUALIDADE DO GRAFICO DE ESTRUTURA

As medidas mais comumente usadas para avaliação do Gráfico de Estrutura (GE) são as de coesão e acoplamento que são normalmente determinadas através da técnica de inspeção. Segundo Brandi (Brandi 90), muitas das regras utilizadas na inspeção podem ser medidas diretamente. Um exemplo disto, é a lista seguinte que pode ser inclusive automatizada para determinar o nível de coesão e acoplamento, controlando, assim, as mudanças efetuadas no Gráfico de Estrutura:

- número de parâmetros entrando no módulo como medida de acoplamento (um número grande de parâmetros é um sinal de fraqueza do projeto, enquanto que uma baixa média de parâmetros indica acoplamento comum abusivo);

- verbos usados em flags, indicam acoplamento de controle;

- número total de referências para áreas comuns indicam número inaceitável de acoplamento comum;

- número médio de módulos chamados por módulo, que não deve ser muito pequeno, nem muito grande. Se pequeno pode indicar um nível pobre de coesão nos módulos. Se for grande, indica grande acoplamento ou baixa coesão nos submódulos.

Entretanto, torna-se difícil julgar o projeto baseando-se em tais medidas, sem um banco de dados com exemplos bons e ruins que possam servir como comparação. Neste caso, o autor sugere que se analize uma parte do projeto em comparação com o restante. Se esta não estiver dentro de um nível aceitável, deve-se apontar as áreas afetadas.

O ideal é ter esse conjunto de medidas para avaliação de projeto que possa ser aplicado tanto a estrutura modular do projeto (Projeto da Arquitetura) quanto ao pseudocódigo (Projeto Detalhado).

Na seção anterior foram apresentadas as características de qualidade relacionadas ao atributo modularidade. Essas características podem ser utilizados na fase de projeto para avaliar a qualidade do Gráfico de Estrutura (GE) e do pseudocódigo que complementa a descrição dos módulos componentes do GE.

Os critérios acoplamento, coesão, fan-in, fan-out, balanceamento, equilíbrio, complexidade total, podem ser aplicados ao GE e os demais (tamanho do módulo, relato de erros, memória de estado, abrangência, módulo

inicial/final, simplicidade, isolamento, documentação e complexidade do módulo) ao pseudocódigo.

II.5 CONCLUSÃO

No estudo realizado neste capítulo foram apresentados os atributos para avaliação da qualidade de projeto, encontrados na literatura técnica atual, onde foram destacados os fatores relacionados à qualidade da representação do projeto.

Assim sendo, partindo-se do método para avaliação da qualidade apresentado na seção II.2, detalhou-se o objetivo Confiabilidade da Representação a fim de se chegar a critérios, que pudessem ser medidos com a utilização de uma ferramenta para construção e avaliação de Gráficos de Estrutura.

CAPITULO III

AVALIAÇÃO DO OBJETIVO CONFIABILIDADE DA REPRESENTAÇÃO

III.1 INTRODUÇÃO

No capítulo anterior foram apresentadas as características de qualidade da Especificação de Projeto, destacando-se as relacionadas à modularidade.

Apesar do objetivo deste trabalho ser a identificação de critérios para a avaliação da modularidade do Gráfico de Estrutura (GE) a nível de Projeto da Arquitetura, no presente capítulo serão apresentados todos os atributos pertinentes à Confiabilidade da Representação.

De acordo com o método de Rocha (Rocha 87), apresentado no capítulo anterior (seção II.3.2), detalha-se o objetivo Confiabilidade da Representação, em seus fatores, subfatores e critérios, a fim de avaliar-se a especificação do projeto.

Na seção III.1, introduz-se o objetivo Confiabilidade da Representação, ressaltando sua importância na fase de projeto e identificando os fatores de qualidade a ele relacionados.

Na seção III.2, descreve-se o fator comunicabilidade e são expostos os seus subfatores e critérios de avaliação. Na seção III.3, o mesmo é feito para o fator manipulabilidade.

III.2 A CONFIABILIDADE DA REPRESENTAÇÃO DA ESPECIFICAÇÃO DE PROJETO

A confiabilidade da representação refere-se à aspectos relacionados à compreensão e manipulação da especificação de projeto. Isto é, avalia a qualidade da especificação de projeto quanto a sua forma. A especificação de projeto deve ter uma forma de representação adequada para que possa ser lido e manipulado pelas diferentes pessoas (desenvolvedores, projetistas, manutenedores) que necessitam realizar suas atividades com facilidade (Menezes 86).

Seguindo o enfoque de Rocha, detalha-se o objetivo Confiabilidade da Representação, seus fatores, subfatores e critérios de forma a definir processos que permitam avaliar a especificação do projeto.

O objetivo confiabilidade da representação é atingido através de dois fatores: comunicabilidade e manipulabilidade.

A seguir detalham-se cada um destes fatores.

III.2.1 COMUNICABILIDADE

Comunicabilidade é a fidelidade com que o especificador é capaz de transmitir as informações desejadas ao leitor através, exclusivamente, da especificação (Staa 83). Especificações são documentos lidos e consultados por diversas pessoas não necessariamente conhecidas entre si (desenvolvedores, projetistas, manu-

tenedores) (Rocha 87) e são, muitas vezes, o único meio de comunicação entre essas pessoas.

Segundo Boehm (Boehm 78), um produto de software possui a característica comunicabilidade caso as intenções do produto sejam claras para o leitor. Pode-se dizer que um produto possui este atributo se é claro e escrito com simplicidade, livre de gíria, termos ou símbolos sem definição adequada, e contendo referências a outros documentos disponíveis e que complementem seu conteúdo.

Assim sendo, a especificação do projeto deve utilizar uma notação uniforme, ser concisa e não repetir desnecessariamente informações pois isto pode prejudicar o seu entendimento. Além disso, deve ser produzida de forma modular de modo a que se possa ter acesso à descrição de aspectos individuais sem necessidade de, para isto, consultar toda a especificação (Rocha 83). A comunicabilidade é, portanto, atingida através dos seguintes subfatores:

- correção no uso da linguagem de projeto;
- concisão;
- uniformidade no nível de abstração;
- uniformidade de terminologia, e,
- modularidade.

III.2.1.1 CORREÇÃO NO USO DA LINGUAGEM DE PROJETO

Homens comunicam conceitos abstratos a outros homens e a máquinas através de uma linguagem, com uma sintaxe e

semântica pré-definida e um conjunto de representações e convenções (Clunie 87).

Especificações de projeto são escritas utilizan-do-se linguagens, desenvolvidas para este fim. Assim sendo, a qualidade da especificação de projeto é, também, dependente da qualidade destas linguagens e do uso que se faça das mesmas.

As linguagens de projeto devem possibilitar a descrição do projeto de maneira clara e precisa. O objetivo de tais linguagens é auxiliar na comunicação entre projetistas e entre estes e os implementadores capturando as decisões de projeto de forma que este possa ser implementado (Jensen 79).

O sub-fator correção no uso da linguagem de projeto avalia se a especificação de projeto está correta do ponto de vista do uso da linguagem escolhida para sua elaboração, isto é, se esta linguagem foi empregada corretamente no que se refere a sua simbologia e regras de aplicação. Ao se avaliar uma especificação de projeto segundo o subfator correção no uso da linguagem de projeto, deve-se levar em conta os seguintes critérios (Clunie 87):

- correção no uso do formato de documentação;
- correção da notação;
- correção sintática, e,
- correção semântica.

Correção no uso do formato de documentação

A linguagem de especificação do projeto, muitas vezes, tem definido um formato específico de documentação, sempre que isto acontecer, este formato deve ser obedecido com o intuito de facilitar a leitura e compreensão dos documentos gerados nesta fase, oferecendo assim uma padronização na documentação.

O formato de documentação, por sua influência na utilizabilidade da especificação ao facilitar o acesso às informações e à revisão de documentos, é um fator significativo no custo do ciclo de vida não somente no que se refere à documentação, mas também à manutenção do sistema.

O uso correto do formato de documentação ajuda na produção de uma documentação organizada que possa servir de base para as atividades posteriores.

Correção da notação

Uma linguagem de especificação de projeto deve ter uma notação definida com precisão. Para que se possa entender um documento escrito utilizando a linguagem, é necessário que sua notação seja usada de forma correta observando sua definição (Clunie 87).

Correção sintática

A sintaxe de uma linguagem é descrita a partir de um conjunto de regras, que permitem utilizar os signos e combiná-los. É necessário conhecer a sintaxe de uma

linguagem para se poder compreender o sentido de uma determinada sentença utilizando a linguagem. Dada uma sentença definida, sua sintaxe resulta em uma espécie de mapa que pode ser usado para investigar o seu significado. Isto evidencia a importância de considerar-se a correção sintática como um atributo fundamental para a compreensão da especificação (Clunie 87), pois erros sintáticos comprometem, fortemente, o entendimento do documento.

Correção semântica

Regras sintáticas não são suficientes para caracterizar uma linguagem. E necessário introduzir-se um outro grupo de regras chamado de regras semânticas para tratar o significado contido nestas sentenças sintaticamente corretas. Duas pessoas podem compartilhar a mesma estrutura linguistica com regras semânticas diferentes sendo, então, incapazes de se comunicar (Clunie 87).

A semântica da linguagem de especificação é definida como a relação entre o sistema de signos (fonemas, gráficos ou simbolos matemáticos) e os objetos que estes denotam.

A importância da correção semântica torna possível a compreensão da especificação, pois a semântica dá significado a linguagem utilizada.

III.2.1.2 CONCISAO

Concisão é o volume de informação produzido por unidade de texto (página, diagrama, etc.). O que se deseja é uma notação ou estilo de redação que assegure a

comunicabilidade e ao mesmo tempo minimize o volume de texto (papel, arquivo) a ser manipulado.

A objetividade da especificação de projeto influi positivamente na compreensão e na facilidade de realizar possíveis modificações a serem executadas por desenvolvedores ou por futuros mantenedores (Freitas 85a).

Em geral, equipes desenvolvedoras encontram dificuldade em produzir especificações concisas. A causa fundamental desta dificuldade, pode estar no fato de que, ao escrever um documento qualquer (relatório, memorando, etc.), existe uma preocupação maior com o aspecto criatividade do que com a tentativa de se dizer o que se deseja em poucas palavras (Boehm 78).

Para que se alcance um elevado grau de concisão, deve-se evitar:

- repetição do assunto em capítulos ou seções subsequentes, como resultado do processo de refinamento;

- ao se utilizar linguagens gráficas, que os diagramas gerados estejam divididos, arbitrariamente e sem necessidade, em um número excessivo de páginas;

- as explicações e/ou definições repetitivas e desnecessárias ao longo da especificação.

Deve-se, no entanto, evitar um excesso de concisão, que ao invés de auxiliar na comunicabilidade da especificação, a prejudique (Rocha 83).

A avaliação deste critério só pode ser feita de modo informal e depende do ponto de vista do avaliador.

Ao avaliar uma especificação segundo o subfator concisão, devem ser considerados os seguintes critérios (Clunie 87):

- não redundância;
- complementabilidade;
- formalidade, e,
- tamanho.

Não redundância

O atributo não redundância refere-se ao fato de que um mesmo aspecto não deve aparecer em mais de um lugar na especificação.

A presença de redundância pode tornar a especificação, de início, mais compreensível, mas, também, pode ocasionar problemas nos processos de manutenção da especificação. Nestas circunstâncias, um requisito deve ser alterado em mais de um lugar, o que pode gerar inconsistências. Caso sejam necessárias redundâncias, devem ser incluídas referências cruzadas explícitas cujo objetivo é permitir que as modificações sejam realizadas de forma confiável (Clunie 87).

Redundâncias são incluídas na especificação durante o desenvolvimento do documento. Assim, a cada refinamento deve-se ter em mente o acréscimo de definições novas, visões novas ou maior detalhamento dos assuntos tratados,

sem entretanto, repetir porções de texto já descritos anteriormente (Staa 83).

Complementabilidade

Muitas vezes incluem-se explicações adicionais com o intuito de definir os vocabulários empregados pelos usuários e que possam ser desconhecidos aos desenvolvedores. Estas explicações, mesmo facilitando a compreensão do documento, geram redundâncias e aumentam o texto. Para solucionar este problema, deve-se fazer referências a documentos prévios ou complementares, assim como, utilizar um glossário para termos, simbologias e notações, que sirva de complemento para a compreensão do documento (Clunie 87).

Formalidade

Uma especificação deve ser escrita utilizando linguagens de especificação. Estas linguagens podem ter diversos graus de formalidade. Portanto, o grau de alcance deste atributo depende da linguagem de especificação utilizada (Clunie 87).

Especificações escritas em linguagem natural (especificações informais), não requerem nenhum treinamento para sua leitura ou, escrita, mas podem, facilmente, causar problemas por sua ambiguidade e falta de organização, gerando más interpretações entre os vários grupos de pessoas que a manipulam (usuários, desenvolvedores, manutenedores, etc.) (Boehm 81).

Para tentar solucionar estes problemas chegou-se ao desenvolvimento de linguagens que tornam possivel gerar uma especificação formal. Este tipo de especificação, expressa numa forma próxima à notação matemática, requer para seu entendimento e uso um longo tempo de treinamento. Permitem, no entanto, uma verificação matemática, possibilitando a construção correta de sistemas.

As linguagens semi-formais ou formatadas tentam solucionar os problemas que normalmente estão associados ao uso de especificações informais e formais. Estas linguagens levam a uma especificação expressa numa sintaxe padronizada e provêem uma estrutura para organização da especificação e execução de avaliações. Geralmente requerem, para seu entendimento, um nível moderado de treinamento. Sua natureza semi-formal impossibilita certas ambiguidades, mas sua semântica imprecisa permite que certos erros não sejam detectados.

O grau de formalidade necessário à especificação depende, também, da natureza e grau de complexidade do software. A medida em que aumenta a complexidade e é necessária ausência de erros, aumenta a necessidade de formalidade.

Tamanho

O tamanho dos capítulos ou seções não é visto como uma caracteristica governante, mas deve ser considerado pois tem influência na facilidade de entendimento do documento.

É difícil apresentar regras que minimizem o tamanho dos capítulos e/ou seções. Cita-se, a seguir, um conjunto de diretrizes que podem ajudar a escrever especificações mais concisas:

- sentenças muito longas devem ser evitadas;

- um volume de texto excessivamente grande deve ser evitado;

- quando possível, apresentar em itens;

- parágrafos curtos devem ser mantidos (evitar que os parágrafos contenham mais de sete sentenças) (Schneiderman 80).

- ao utilizar linguagem gráfica, os diagramas gerados não devem estar divididos arbitrariamente e sem necessidade, em um número excessivo de páginas (Clunie 87).

III.2.1.3 UNIFORMIDADE NO NIVEL DE ABSTRAÇÃO

A construção de um sistema por refinamentos sucessivos faz com que a cada passo durante o ciclo de desenvolvimento, tenha-se um detalhamento maior do sistema do que no passo anterior. Entretanto é importante que em um determinado nível, todos os elementos estejam descritos com o mesmo grau de detalhe. Para tanto, deve-se decidir sobre o conteúdo, de forma que este contenha apenas as informações necessárias no nível de detalhamento adequado ao documento.

O conceito de abstração é uma técnica essencial para o tratamento da complexidade, permitindo separar aspectos conceituais do sistema que está sendo projetado, dos detalhes de implementação.

Ao se avaliar, segundo o subfator uniformidade no nível de abstração, devem ser considerados os critérios:

- uniformidade de detalhes, e,
- independência de detalhes de implementação.

Uniformidade de detalhes

Uma especificação deve ser clara, sem conter ambiguidades e suficientemente detalhada para permitir seu entendimento sem necessidade de consulta à equipe geradora da mesma. O grau de detalhamento da especificação varia, uma vez que cada pessoa tem idéias diferentes sobre o nível de detalhe necessário.

Uniformidade de detalhes é a característica da especificação abordar todos os aspectos com o mesmo grau de detalhe. Possuir este atributo significa que no processo de refinamentos sucessivos todos os aspectos, descritos no nível sob consideração, foram especificados com o mesmo grau de detalhe (Rocha 83).

Independência de detalhes de implementação

Os aspectos do projeto não devem estar restritos a detalhes de implementação.

A idéia é esconder a informação onde esta não é necessária naquele nível do projeto (ou programa ou

módulo). Isto é, deve-se manter o projetista no nível apropriado de detalhe (Parnas 72).

Um dos problemas mais comuns é se especificar cedo demais os detalhes, restringindo desta forma possíveis alternativas de solução ou tornando pouco flexíveis as soluções encontradas. Outro problema frequente é a validade da solução ser dependente de condições não explicitamente estabelecidas até o momento. Neste caso, a especificação pode estar condicionada a detalhes de implementação sem que o leitor se perceba disto (Staa 83).

III.2.1.4 UNIFORMIDADE DE TERMINOLOGIA

O conteúdo da especificação deve ter uma única interpretação. Para tanto é necessária a padronização da terminologia utilizada, principalmente no que se refere a termos técnicos. Sempre houver a possibilidade de dúvidas, estes termos devem ser incluídos em um glossário.

Uma padronização pré-estabelecida deve ser proveniente de normas da empresa e da própria especificação do projeto. Cabe ao projetista mantê-la dentro dos padrões pré-definidos, compatibilizando a terminologia do projeto com a utilizada na especificação de requisitos.

O objetivo deste atributo é de não existirem termos diferentes descrevendo o mesmo objeto do mundo real.

Ao se avaliar uma especificação segundo o subfator uniformidade de terminologia, devem ser considerados os critérios:

- padronização de termos técnicos, e,
- uniformidade de notação.

Padronização de termos técnicos

Em uma especificação são utilizados diversos termos que possuem significado próprio e preciso, mas não são necessariamente de conhecimento geral (Rocha 83). Para se evitar incompreensões devido ao desconhecimento de tais termos, estes devem estar definidos em um glossário e devem ser utilizados de maneira uniforme ao longo dos documentos.

Uniformidade de notação

Mesmo quando se utilizam metodologias e linguagens de especificação, existem partes de uma especificação que são escritas em linguagem natural.

Além disso linguagens gráficas utilizam, em geral, uma notação gráfico-linguística, onde a linguagem natural é usada para complementar as informações nos diagramas.

O emprego de notações apropriadas e seu uso uniforme, garantem a comunicação entre usuários e desenvolvedores (Clunie 87).

III.2.1.5 MODULARIDADE

A modularidade é um atributo de qualidade da representação que se refere ao projeto de software estar elaborado com uma estrutura modular adequada.

A modularização é uma atividade de projeto lógico, tendo por finalidade a definição de uma organização modular

executável em algum ambiente de programação não especificado (Staa 83).

Yourdon e Constantine (Yourdon79) tiveram como ponto de partida para seu trabalho sobre modularização, a observação de módulos onde alguns demonstraram ser mais manuteníveis do que outros.

O objetivo da modularização de um projeto é quebrar uma tarefa (sistema) complexa em sub-tarefas (sub-sistemas) menores e mais simples, facilitando a realização de verificações, manutenções e reutilização das partes (módulos) do sistema. Os objetivos específicos da decomposição em módulos são:

- a estrutura modular deve ser, suficientemente, simples tal que possa ser entendida facilmente;

- a troca da implementação de um módulo deve ser possível sem o conhecimento da implementação de outros e sem afetar o comportamento dos demais;

- com exceção da troca de interface, cada módulo deve ser alterado individualmente, possibilitando o teste do sistema com qualquer combinação das versões do módulo velho e novo.

A organização modular ideal é aquela onde os relacionamentos entre módulos são mínimos e os relacionamentos entre elementos de um mesmo módulo são máximos.

Para atingir tais objetivos, deve-se começar com uma organização modular composta de um único módulo. Avalia-se

a modularidade deste. Se for satisfatória o processo termina. Caso contrário, decompõe-se tal módulo em um módulo raiz e um ou mais submódulos subordinados a ele (Staa 83).

A avaliação do grau de modularidade deve ser diferenciada para os documentos de Projeto da Arquitetura e de Projeto Detalhado. Isto porque, no primeiro têm-se uma representação da estrutura modular de cada programa, descrição funcional e das interfaces de cada módulo que a compõem. No segundo, representa-se detalhadamente cada módulo através de seus algoritmos.

Projeto da Arquitetura

Na fase de Projeto da Arquitetura, a preocupação é com a qualidade da estrutura modular do programa. Na realidade a definição da estrutura modular é uma decisão de projeto e deve ser avaliada nesta fase. Estruturas modulares não adequadas levam a programas não mantenìveis.

Ao se avaliar o Projeto da Arquitetura segundo o subfator modularidade, devem ser considerados os seguintes critérios:

- acoplamento;
- coesão;
- número de módulos subordinados ("fan-out");
- número de módulos superiores ("fan-in");
- balanceamento, e,
- complexidade total do sistema.

Acoplamento

O acoplamento avalia o grau de interdependência entre os módulos, isto é, a possibilidade de que a modificação de um módulo acarrete alterações em outro módulo.

Os tipos de acoplamento na ordem do mais baixo para o mais alto são:

. **Acoplamento de dados:** a comunicação entre os módulos é feita por parâmetros, sendo cada parâmetro um dado elementar ou uma tabela homogênea.

. **Acoplamento por dados estruturados:** a comunicação entre os módulos é feita através de estruturas de dados.

. **Acoplamento por controle:** envolve a passagem de variável de controle para a lógica interna de outro módulo.

. **Acoplamento por área de dados comum:** os módulos se referem a mesma área de dados.

. **Acoplamento por conteúdo:** um módulo faz referência ao interior de outro módulo.

Dois módulos podem ser acoplados por mais de um tipo. Neste caso, diz-se que eles estão acoplados pelo tipo mais forte.

Segundo Page-Jones (Page-Jones 80), tem-se a seguinte tabela mostrando o impacto do tipo de acoplamento (figura III.1):

Tipo de acoplamento	Modificabilidade	Inteligibilidade	Reutilizabilidade
dados	bom	bom	bom
est. de dados	médio	médio	médio
controle	pobre	pobre	pobre
área de dados			
comum	médio	ruim	ruim
conteúdo	ruim	ruim	ruim

Figura III.1: Comparação entre os tipos de acoplamento

Fonte: Page-Jones 80

Coesão

A coesão avalia a força que mantém unidos os elementos de um módulo. O que se deseja é ter um alto nível de coesão.

Um módulo altamente coeso é uma coleção de sentenças e itens de dados que estão fortemente relacionados. De fato, a certeza de que todos os módulos tem boa coesão é o melhor modo para minimizar o acoplamento entre eles (Page-Jones 80).

Os tipos de coesão do nível mais baixo para o mais alto são:

. **coesão por coincidência:** os elementos de um módulo não tem razão aparente para estarem juntos. Este tipo de módulo tem normalmente nome sem sentido e necessita de uma variável de controle para saber o que executar.

. **coesão lógica:** existe alguma relação entre os elementos do módulo como por exemplo, um módulo que executa todas as entradas. Este tipo de módulo necessita de uma variável de controle para saber o que usar.

. **coesão temporal:** o relacionamento mais forte entre os elementos do módulo é que eles são executados ao mesmo tempo como por exemplo, um módulo que executa a inicialização de um programa.

. **coesão procedural:** os elementos estão envolvidos em atividades diferentes e possivelmente não relacionadas, (por exemplo, ler e imprimir) mas a ordem de execução é importante.

. **coesão comunicacional:** os elementos referenciam os mesmos dados de entrada e/ou saída como por exemplo, imprimir e perfurar. A ordem de execução não é importante.

. **coesão sequencial:** a saída de um elemento é entrada para o próximo como por exemplo, ler a próxima transação e atualizar mestre. A desvantagem deste tipo de coesão é que módulos com coesão sequencial não são prontamente reutilizáveis como os módulos funcionais, porque contem atividades que não são, necessariamente, úteis juntas.

. **coesão funcional:** todos os elementos estão relacionados ao desempenho de uma única função como por exemplo, calcular raiz quadrada.

Segundo Page-Jones (Page-Jones 80) tem-se a seguinte tabela mostrando o impacto do tipo de coesão em comparação com outros atributos de qualidade (figura III.2):

Nível de coesão	Acoplamento	Simplicidade	Reutilizabilidade	Modificabilidade	Intelibilidade
Funcional	bom	bom	bom	bom	bom
Sequencial	bom	bom	bom	bom	bom
Comunicacional	médio	bom	pobre	médio	médio
Procedural	variável	médio	pobre	variável	pobre
Temporal	pobre	médio	ruim	médio	pobre
Lógica	ruim	ruim	ruim	ruim	ruim
Coincidental	ruim	pobre	ruim	ruim	ruim

Figura III.2: Comparação entre níveis de coesão

Fonte: Page-Jones 80

Se um módulo possui mais de um tipo de coesão, considera-se para efeito de avaliação, que este tem o nível de coesão mais fraco.

É desejável que todos os módulos tenham coesão funcional. Este nível de coesão favorece ao princípio de caixa preta. Entretanto, é reconhecido que por limitações de projeto, isto nem sempre é possível. Nestes casos recomenda-se que sejam, pelo menos, sequenciais ou comunicacionais.

Número de módulos subordinados ("fan-out"):

Este critério avalia o número de módulos chamados por um determinado módulo. Um número muito alto ou muito baixo de módulos subordinados são indicadores de um projeto pobre, embora um número muito alto é mais perigoso do que um número baixo (quanto mais alto mais difícil de reutilizar).

De acordo com os estudos de psicologia computacional, não deve haver mais do que nove módulos subordinados imediatos de um módulo, pois sete mais ou menos dois está dentro do limite da capacidade da mente humana para resolver problemas distintos (Page-Jones 80). A partir desse número, o módulo pai passa a ser complexo e de difícil implementação. Caso haja a necessidade de mais de nove subordinados, deve-se fazer a fatoração do módulo pai em dois ou mais módulos (Nogueira 88).

Quanto menor o "fan-out", maior a complexidade do módulo (indicando uma maior funcionalidade). Por conseqüência, menor será a contribuição deste módulo para a complexidade total do sistema.

Card (Card 88) sugere que o valor de fan-out que minimiza a complexidade total do sistema é três e que o valor usualmente aceito de nove é excessivo. Justifica que, Constantine (Stevens 86) observou que a maioria dos programas podem ser decompostos efetivamente em uma estrutura comum de três pontos: entrada, processamento e saída.

Número de módulos superiores ("fan-in")

Avalia o número de módulos que chamam um mesmo módulo. Quando mais de um módulo superior chama um mesmo módulo subordinado, está se evitando duplicação de código. Se um módulo tem uma função útil, ele pode ser usado em qualquer lugar no sistema. Todavia deve-se tomar cuidado para não criar módulos de baixa coesão e forte acoplamento.

Há duas regras para restringir o uso de "fan-in":

- módulos com "fan-in" devem ter boa coesão, funcional ou pelo menos (toleravelmente) comunicacional ou sequencial;

- a interface para cada módulo que o chama deve ter o mesmo número e tipos de parâmetros.

Balanceamento

Quando se introduz um dado no sistema ele deve ser tratado imediatamente. Se, ao invés, os dados chegam aos módulos do topo sem sofrer qualquer crítica, diz-se que o sistema é fisicamente dirigido para entrada. Neste caso, qualquer alteração feita pelo usuário, sobre os dados, afetará todos os módulos que o utilizam.

Da mesma forma, os dados ao serem enviados para saída devem sofrer as devidas transformações, tornando o sistema pouco vulnerável a mudanças. Neste caso, diz-se que o sistema não é fisicamente dirigido para saída.

Uma estrutura é balanceada se os módulos de nível mais alto tratam os dados de uma forma lógica de modo que estes não sejam dependentes das características físicas (que são relegadas aos níveis mais baixos). Característica principal é o tratamento dos dados nos níveis mais baixos da estrutura.

Estruturas balanceadas são mais fáceis de implementar e de adaptar a mudanças nas especificações (especialmente se estas são nos dispositivos físicos ou formatos de entrada/saída).

Complexidade total do sistema

O Projeto da Arquitetura é o processo de partionamento de um sistema em partes com funções e dados que trabalham juntos para atingir a missão total do sistema. Assim, a complexidade do Projeto da Arquitetura pode ser vista como tendo dois componentes: a complexidade contida dentro de cada parte ou módulo definido pelo projeto e a complexidade dos relacionamentos entre essas partes ou módulos (Card 88).

O objetivo é atingir uma medida de complexidade que possa ser aplicada a uma vasta classificação de produtos, independentemente de como estes são produzidos. Assim, define-se que a complexidade total de um sistema (Ct) é composta de duas partes: a complexidade contida dentro de cada parte ou módulo (a complexidade local (Lo), ou intramódulo), e a complexidade dos relacionamentos entre as

partes ou módulos (complexidade estrutural (Es), ou intermódulo).

Ct = Lo + Es, onde:

Lo = [Somatório (vi / (fi+1))] / n, e

Es = [Somatório (fi^2)] / n

n = número de módulos no sistema;

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

fi = fan-out do módulo i

Projeto	Es	Lo	Ct	Avaliação Subjetiva	Qualidade
A	24,6	8,2	32,8	5	pobre
B	15,8	9,5	25,3	2	boa
C	11,8	12,1	23,9	3	boa
D	18,4	4,9	23,3	1	boa
E	12,6	10,0	22,6	4	boa
F	22,3	7,3	29,6	6	pobre
G	18,3	10,8	29,1	8	pobre
H	19,2	7,3	26,5	7	pobre

Figura III.3: Complexidade total do sistema e qualidade

Fonte: Card 90

Este modelo foi utilizado no estudo de oito sistemas similares em termos de ambiente, aplicação e pessoal, o cálculo da complexidade e a análise subjetiva coincidiram, conforme apresentado na tabela acima (figura III.3).

Projeto Detalhado

Na fase de Projeto Detalhado, preocupa-se com a qualidade de cada módulo especificado na organização

modular. O que se pretende é construir módulos simples, manuteníveis e reutilizáveis

Ao se avaliar o Projeto Detalhado segundo o subfator modularidade devem-se considerar os seguintes critérios:

- complexidade do módulo;
- abrangência;
- isolamento;
- memória de estado;
- estrutura de dados;
- tamanho do módulo;
- simplicidade;
- relato de erros, e,
- documentação do módulo.

Complexidade do Módulo

Depende de suas estruturas de decisão e, como consequência, do número de caminhos que contém. Usando teoria dos grafos, a complexidade é avaliada pelo número de caminhos básicos do programa (como um grafo de controle com uma única entrada e uma única saída):

V(G)=e-n+2p, onde:

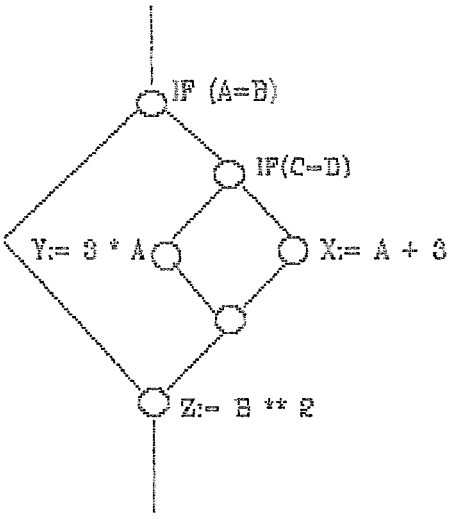
e= número de ligações no grafo;

n=número de nós no grafo;

p=número de componentes conexos no grafo (cada módulo tem um único componente).

Na figura III.4, mostra-se um exemplo de avaliação segundo esse critério.

```
IF (A=B)
THEN BEGIN
     IF (C=D)
     THEN X:= A + 3
     ELSE Y:= 3 * A
     END;
Z:= B ** 2;
```



V(G) = e − n + 2p
V(G) = 7 − 6 + 2(1)
V(G) = 3

Figura III.4: Exemplo da aplicação do critério complexidade do módulo

Fonte: Nascimento 86

A complexidade do módulo (excessivo número de caminhos) afeta sua avaliabilidade e manutenibilidade. Segundo McCabe (McCabe 76), V(G) menor do que 10 é adequado podendo ser um pouco maior para módulos que contenham instruções do tipo CASE com muitas opções. Módulos com estrutura lógica simplificada facilitam o seu entendimento.

Abrangência

Os módulos não devem ser nem muito restritos nem muito genéricos. O maior problema de módulos restritos é a dificuldade de sua reutilização (os módulos devem ser flexíveis). A generalização para permitir maior flexibilidade dos dados é normalmente menos perigosa do que a generalização para permitir maior flexibilidade de função, pois esta leva a mais código e, consequentemente, pior acoplamento e pior coesão.

Um módulo muito especifico ou muito geral, perde o sentido, na medida que leva ao não cumprimento de algumas normas já preconizadas, tais como (Nogueira 88):

- duplicidade de código (na medida em que os módulos são restritos),

- parâmetros desnecessários passados para o módulo,

- dificuldade de implementação e manutenção, e,

- memória de estado (na medida em que são generalistas).

Isolamento

Avalia tudo que se precisa saber com relação à implementação do módulo para que seja utilizado corretamente. O ideal seria que fosse necessário conhecer apenas a definição do módulo e esta não fizesse referência a restrições de implementação.

Memória de estado

Avalia se existe a guarda de parte de uma informação para ser utilizada na próxima chamada do módulo.

Quando um módulo retorna o controle para o seu superior ele "morre". Quando é invocado novamente é executado como se fosse pela primeira vez, não tendo memória de sua existência prévia. Mas, existe um tipo de módulo que está ciente do seu passado através de sua memória de estado, o que o torna imprevisível. Este tipo de

módulo deve ser evitado por conduzir a problemas na manutenção.

Estrutura de dados

Sempre que possível deve-se compatibilizar a estrutura do programa com a estrutura de dados porque, além de facilitar a programação dos módulos, permite que a forma da solução retrate a forma do problema.

Tamanho do módulo

O tamanho do módulo não é característica governante mas deve ser considerada.

Alguns autores (Yourdon 79, Conte 86, Boehm 78, Arthur 85) sugerem que este tamanho deve ser entre 50 a 200 comandos contidos entre uma ou duas páginas de listagem. Entre-tanto, uma pesquisa realizada com 453 módulos escritos em FORTRAN (Card 85) conclui que bons programadores não tem preferência por um tamanho específico para os módulos, que módulos grandes custam menos e que a incidência de erros não está relacionada ao tamanho do módulo.

Isto sugere que o tamanho de um módulo não deve estar limitado por nenhuma norma arbitrária. Entretanto, pode-se dizer que:

. Módulos maiores do que uma ou duas páginas devem ser verificados quanto à possibilidade de serem separados, sempre que isto não comprometa a funcionalidade.

. Módulos com menos do que uma a cinco linhas devem ser examinados para ver se podem se fundidos com seus superiores.

Deve-se, entretanto, cuidar para que isto não acarrete perda de coesão ou acoplamento, que são atributos fundamentais para modularidade (Freitas 85b).

Simplicidade

Avalia se o módulo atinge o seu objetivo da forma mais simples possível, satisfazendo todos os requisitos de qualidade.

Um módulo é simples quando não possui mais controle do que o necessário, não manipula dados de forma específica (o que o torna restrito), todos os dados estão envolvidos com a função executada pelo módulo e esta função é única e clara.

Um módulo simples tem coesão funcional, acoplamento de dados, não é restrito nem geral, é de fácil implementação e manutenção e possue um fluxo de dados entrando e/ou outro saindo.

Segundo Stevens (Stevens 74), simplicidade é a primeira medida recomendada para avaliar projetos relativo a redução do tempo de depuração e modificação. Observou-se que programas que foram os mais fáceis para implementar e trocar foram aqueles compostos por módulos simples e independentes. A razão disso é que a solução do problema é mais rápida e mais fácil quando o problema pode ser

subdividido em pedaços que podem ser considerados separadamente. A solução do problema é difícil quando todos os aspectos do problema devem ser considerados simultaneamente.

Relato de erros

Deve ser feito pelo módulo que detectou e identificou os erros, o que torna o módulo mais legível. Caso contrário, separa-se o quando do o que, forçosamente os módulos estarão acoplados através de "flags" que viajam pela estrutura.

Documentação do módulo

Avalia a qualidade e a disponibilidade da documentação relativa ao módulo. Aplica-se explicitamente aos módulos físicos, uma vez que estes correspondem a unidades reutilizáveis (Pressman 87). Deve conter:

- objetivo;
- descrição do processamento;
- descrição das entradas e saídas;
- descrição da linguagem de projeto;
- módulos referenciados;
- organização dos dados, e,
- comentários.

III.2.2 MANIPULABILIDADE

É um atributo que torna possível a facilidade de acesso e manipulação a diferentes versões da especificação e a partes da mesma. É o fator que retrata a facilidade de

manipulação do projeto no seu uso, modificação e teste (Freitas 85a).

Possuir este atributo significa que existem cópias atualizadas da especificação e que é possível o acesso às mesmas. Significa, também que existe facilidade de acesso a todos os elementos da especificação relacionados com determinado aspecto. Significa, ainda, que há facilidade para se localizar uma determinada visão, no nível de detalhe desejado, e facilidade para se percorrer a seqüência de agregação de detalhes (Rocha 83).

Ao se avaliar uma especificação segundo o fator manipulabilidade, devem ser considerados os subfatores:

- disponibilidade;
- estrutura, e,
- rastreabilidade.

III.2.2.1 DISPONIBILIDADE

A fim de se desenvolver um produto que atinja seus objetivos e ao mesmo tempo seja fácil de ser usado, certas informações devem estar disponíveis.

A disponibilidade é a característica do projeto poder ser consultado por desenvolvedores e mantenedores, na sua versão mais atualizada, pois se este não estiver disponível possivelmente não acompanhará as modificações do programa (Freitas 85a).

A documentação do projeto deve estar pronta para ser utilizada sempre que isto for necessário. Por ser

dinâmico, ou seja, por ser capaz de acompanhar as alterações que surgirem no mundo real um projeto deve (Staa 83):

- estar atualizado;
- ser reproduzível de forma parcial ou total;
- poder ser exibido por meios mecânicos (terminal, listagem);
- estar distribuído a todas as pessoas envolvidas no desenvolvimento ou manutenção do produto.

Ao se avaliar uma especificação segundo o subfator disponibilidade, devem ser considerados os seguintes critérios:

- acessibilidade, e,
- estar atualizada.

Acessibilidade

A especificação deverá ser utilizada por diversas pessoas ao longo do desenvolvimento, devendo, também, atender às necessidades da fase de operação (uso e manutenção). Desta forma, deve poder ser facilmente acessada.

Assim sendo, é vantajoso manter toda a documentação em meio automatizado, pois isto facilita sua consulta e permite que:

- a especificação esteja sempre disponível;
- a especificação possa ser facilmente reproduzida após cada alteração, e,
- os usuários possam ter uma cópia sempre que necessitarem.

Estar atualizada

A especificação do projeto deve conter as informações mais recentes do problema, decorrentes da necessidade do usuário fazer mudanças ou acertos na especificação, tanto durante o processo de desenvolvimento como depois na fase de manutenção/operação.

A especificação deve sempre permitir que as atualizações sejam feitas com facilidade. Não possuir este atributo pode produzir diversos problemas, pela dificuldade resultante em se manter um perfeito controle sobre o software que está sendo construído (Clunie 87).

III.2.2.2 ESTRUTURA

Um projeto de software possue essa característica quando está elaborado dentro de um padrão definido de composição de suas partes, formando uma organização hierárquica. A estrutura permite aos desenvolvedores e manutenedores uma rápida compreensão do projeto, no nível de detalhe desejado, de acordo com as necessidades do momento (Freitas 85a).

Durante o processo de desenvolvimento, é gerado um conjunto de especificações, com grau variável de detalhe. É virtualmente impossível conseguir-se manipular e entender este conjunto de especificações sem que se disponha de uma documentação que esteja bem organizada. Com uma documentação organizada, permite-se ao leitor da especificação percorrê-la, através dos documentos gerados nos diversos passos de refinamentos do primeiro ao último e vice-versa,

e encontrar o nível de detalhe mais adequado às suas necessidades momentâneas.

Boa documentação é especialmente importante para a atividade de manutenção.

A estrutura de um artigo, papel, ou livro é muito importante para comunicar idéias claras. Possuir este atributo significa que é possível percorrer os documentos resultantes do processo de refinamento, do primeiro ao último e vice-versa. Com este atributo se facilita ao leitor da especificação encontrar uma descrição no nível de detalhe mais adequado a suas necessidades momentâneas. Para que isto seja possível os documentos devem estar organizados hierarquicamente (Rocha 83).

A organização hierárquica da documentação torna possível manusear conjuntos de especificações. Além disso, torna possível remover uma especificação do conjunto e, ainda assim, manter o resto do conjunto utilizável. Este é um atributo poderoso quando se deseja criar especificações que sejam fáceis de manter e de reutilizar (Clunie 87).

A disposição hierárquica nos diferentes níveis de abstração facilita o desenvolvimento de sistemas modulares.

III.2.2.3 RASTREABILIDADE

É a característica de um projeto estar elaborado de forma a permitir navegar através de uma sequência de detalhes vinculados a um determinado aspecto do problema, desde

a sua visão mais geral até a mais detalhada e vice-versa (Rocha 87).

A rastreabilidade facilita muito o trabalho dos desenvolvedores e, principalmente, dos mantenedores, ajudando-os a localizar os aspectos de interesse. Permite acompanhar os diversos requisitos desde sua primeira formulação até sua incorporação no programa. Permite ainda acompanhar a sequência de tomadas de decisão que culminaram na forma final da incorporação do aspecto ao programa (Staa 83).

Uma especificação é rastreável se é clara a origem de cada um de seus requisitos, se ela facilita a referência de cada requisito nos documentos resultantes do desenvolvimento, ou crescimento do software. Para que isto seja possível, é necessária a existência de índices remissivos. A rastreabilidade é facilitada pela existência de especificações modulares (Rocha 83).

Ao se avaliar uma especificação segundo o subfator rastreabilidade, devem ser considerados os seguintes critérios:

- localizabilidade interna, e,

- localizabilidade externa.

Localizabilidade Interna

A especificação é um documento que deve proporcionar caminhos, facilmente percorríveis, que permitam, a partir de certos requisitos, alcançar outros requisitos. Essa facilidade de percorrer itens dentro da especificação está

relacionada à existência de mecanismos que facilitem o acesso a todos os elementos da especificação que estão relacionados. Para isto, a especificação deve conter sumários, índices remissivos e tabelas de referências cruzadas (Staa 83).

Localizabilidade externa

Assim como desejamos localizar os itens relacionados a um determinado aspecto ou assunto numa mesma especificação, também desejamos poder localizar todas as especificações e demais documentos que tratem de um determinado assunto e, dentro destes, localizar onde é tratado o assunto procurado. Finalmente, é necessário saber a localização de todos os itens que foram afetados por modificações à especificação (Clunie 87).

Além disso, a especificação do projeto deve ser rastreável com a especificação de requisitos.

III.3 CONCLUSÃO

Neste capítulo, detalhou-se o objetivo, Confiabilidade da Representação, em seus subfatores e critérios de avaliação, dando ênfase ao subfator modularidade.

A partir desse estudo, foi possível determinar os processos de avaliação e a interpretação dos resultados referentes aos critérios de modularidade, que serão utilizados na ferramenta.

CAPITULO IV

UMA FERRAMENTA PARA AVALIAÇÃO DA QUALIDADE NA FASE DE PROJETO

IV.1 INTRODUÇAO

Este capitulo apresenta a ferramenta desenvolvida nesta tese, através de sua documentação. Trata-se de um analisador estático para avaliação da estrutura modular de programas na fase de projeto (AVALIE-GE). Faz parte da nova versão do Editor de Gráficos de Estrutura permitindo, desta forma, uma avaliação imediata dos gráficos que representam os programas. Para sua implementação utilizou-se o ambiente de programação Actor.

IV.2 A FERRAMENTA PARA AVALIAÇAO DA ESTRUTURA MODULAR DE PROGRAMAS NA FASE DE PROJETO

Esta ferramenta dá apoio à construção e à avaliação na fase de Projeto da Arquitetura, segundo o método Projeto Estruturado, proposto por Yourdon (Yourdon 79). Consiste em dois módulos:

- o Editor de Gráficos de Estrutura (EDIT-GEII); e
- o Avaliador de Modularidade (Avalie-GE).

O EDIT-GEII auxilia a construção de Gráficos de Estrutura (GE).

O AVALIE-GE permite a avaliação automática do GE construído com o EDIT-GEII através do controle dos seguintes critérios relacionados ao subfator modularidade na fase de Projeto da Arquitetura:

- acoplamento;
- coesão;
- número de módulos superiores;
- número de módulos subordinados e
- complexidade total do sistema.

No capitulo III, detalhou-se o método para avaliação da qualidade proposto por Rocha (Rocha 83) para o objetivo Confiabilidade da Representação na fase de projeto.

Apesar da ferramenta implementada nesta tese ter-se detido no subfator modularidade, outros atributos da Confiabilidade da Representação são automaticamente atingidos com a utilização do EDIT-GE, como por exemplo, correção da notação, acessibilidade e estrutura.

Na ferramenta AVALIE-GE avaliam-se quase todos os critérios de modularidade relacionados ao Projeto da Arquitetura que são identificados no GE, com exceção, do balanceamento que não foi possivel automatizar e que deve ser avaliado através de uma Inspeção (Fagan 76) no GE.

Os critérios relacionados à modularidade do Projeto Detalhado não podem ser avaliados nesta ferramenta, pois é necessário criar-se um editor para lógica de módulos, onde alguns critérios podem ser automaticamente avaliados (como por exemplo, complexidade do módulo, memória de estado, tamanho do módulo e relato de erros).

A justificativa para o desenvolvimento desta ferramenta, assim como seus objetivos e principais funções, são apresentados na Definição do Sistema. O roteiro de

documentação utilizado foi o proposto por Mattoso (Mattoso 90) que, por se tratar de um método de apoio ao desenvolvimento de software com orientação a objetos, é o que melhor se enquadra às características do ambiente de programação Actor.

A identificação das principais classes do sistema e seus requisitos são vistos na Especificação de Requisitos e o detalhamento destes e o projeto da interface com o usuário são mostrados na Especificação do Projeto.

A seguir, apresentam-se esses documentos.

IV.2.1 DEFINIÇÃO DO SISTEMA

1. Introdução

1.1 Propósito do Documento

Este documento descreve os objetivos e as características gerais de uma ferramenta automatizada de software para construção (EDIT GEII) e avaliação da qualidade (AVALIE-GE) de gráficos de estrutura. Esta ferramenta apóia a fase de Projeto da Arquitetura.

EDIT-GEII permite a construção de Gráficos de Estrutura através da descrição dos módulos que o compõe e das interfaces entre estes. Foi construído tendo como base a versão desenvolvida, anteriormente, por Nogueira (Nogueira 88) onde foram acrescentadas novas funções e feita nova implementação, desta vez no ambiente de programação Actor (Actor 90).

AVALIE-GE é um avaliador da qualidade cujo objetivo é medir o grau de modularidade em Gráficos de Estrutura. Está integrado ao EDIT-GEII, permitindo uma avaliação imediata dos gráficos de estrutura. AVALIE-GE emite relatórios que permitem ao projetista de sistemas, verificar a modularidade de seu projeto, de forma a controlar a qualidade, simultaneamente, ao processo de criação do GE. O AVALIE-GE permite o controle dos seguintes critérios relacionados ao subfator modularidade:

- acoplamento;
- coesão;
- número de módulos superiores;
- número de módulos subordinados, e,
- complexidade total do sistema.

Os critérios acima, processos de avaliação e interpretação dos resultados estão descritos no item 3.3 deste documento.

1.2 Sumário

No capítulo 2 são apresentadas as justificativas para o desenvolvimento desta ferramenta, traçando-se seus objetivos globais. O capítulo 3 contém a descrição geral do sistema, as principais funções do produto, o detalhamento dos critérios de avaliação utilizados no AVALIE-GE, as características dos usuários, as hipóteses e restrições ao desenvolvimento do sistema.

2. Definição do Problema

2.1 Justificativa do Produto

Existem vários métodos e linguagens de apoio à construção da especificação de projeto. Entre estes, os métodos estruturados ganharam importância por proporem linguagens gráficas, que facilitam o entendimento do sistema e a comunicação com o usuário. Além disso, permitem a construção de especificações de projeto de boa qualidade, tanto no que se refere ao conteúdo, quanto à forma de representação.

O método Projeto Estruturado, proposto por Yourdon e Constantine (Yourdon 79), embora seja amplamente utilizado, necessita de um apoio automatizado, que facilite a manutenção dos documentos gerados.

Com esse intuito, surgiu EDIT-GE como ferramenta de apoio à elaboração do Gráfico de Estrutura através da descrição dos módulos que o compõe e das interfaces entre eles. Com a utilização da ferramenta garante-se o uso correto da notação. Baseado em critérios pré-estabelecidos, o GE é avaliado pelo AVALIE-GE e os erros encontrados são registrados e impressos, para que possam ser corrigidos pelo usuário e posteriormente reavaliado até a eliminação dos erros.

O uso do AVALIE GE indicará as melhorias a serem feitas ainda na fase de projeto da arquitetura, evitando a propagação de erros para as fases posteriores do desenvolvimento.

2.2 Objetivos Globais

A motivação para o desenvolvimento desta ferramenta é fornecer apoio automatizado para a criação e avaliação de Gráficos de Estrutura.

2.3 Area de Aplicação

O produto, a ser construído, classifica-se como uma ferramenta de desenvolvimento de software para ser utilizada na fase de Projeto da Arquitetura em um ambiente de suporte ao método estruturado de sistemas.

3. Descrição Geral

3.1 Ambiente de Uso

Esta ferramenta faz parte de um ambiente destinado a suportar os métodos estruturados de desenvolvimento de software.

3.2 Funções do Produto

A ferramenta realiza as seguintes funções:

a) criação e alteração de gráficos de estrutura (GE): o usuário terá o controle sobre a disposição dos módulos no GE, bem como, poderá alterar sua descrição e interfaces. Poderá, ainda, removê-los e copiá-los;

b) impressão do GE e da descrição de cada módulo incluindo suas interfaces;

c) avaliação do GE de acordo com a necessidade do usuário: ao solicitar uma avaliação, o projetista deve

informar se deseja avaliar todo o gráfico (opção Global) ou apenas um módulo (opção Local). Além disso, deve informar que critérios deseja avaliar, de acordo com as opções disponíveis.

d) impressão das avaliações;

e) gerência de arquivamento do GE e das avaliações;

f) gerência do relacionamento entre os módulos que compõe o GE;

g) garantia de integridade: o usuário não poderá criar definições inadequadas de módulos e relacionamentos.

3.3 Descrição dos critérios avaliados na ferramenta AVALIE-GE

O método para avaliação da qualidade de software utilizado é o proposto por (Rocha 83) e apresentado no capítulo II (seção II.2).

A ferramenta AVALIE-GE avaliará a modularidade na fase de Projeto da Arquitetura, levando em consideração os critérios acoplamento, coesão, "fan-in", "fan-out" e complexidade total, já definidos no capítulo anterior (seção III.2.1.5).

A seguir apresentam-se os processos de avaliação, interpretação dos resultados e processos de correção de cada um desses critérios.

3.3.1 Acoplamento

Processo de Avaliação:

Identificar os tipos de acoplamento entre cada módulo e o(s) módulo(s) que o chama(m).

Interpretação dos resultados:

Os módulos com acoplamento de dados e estrutura de dados são aceitos como bons. Os com acoplamento por controle, área de dados comum e conteúdo devem ser evitados ao máximo, por afetarem a manutenibilidade.

Procedimento de Correção:

Rever os módulos com acoplamento por controle, área de dados comum e conteúdo e verificar a possibilidade de serem modificados.

3.3.2 Coesão

Processo de Avaliação:

Identificar o tipo de coesão de cada módulo.

Interpretação dos resultados:

Módulo com coesão funcional, sequencial ou comunicacional são considerados de boa coesão. Módulos com coesão procedural ou temporal são considerados com média coesão. Módulos com coesão lógica ou coincidental são considerados com péssima coesão e devem ser evitados.

Procedimento de Correção:

Rever os módulos com média e péssima coesão e verificar a possibilidade de corrigi-los.

3.3.3 Número de módulos subordinados ("fan-out")

Processo de Avaliação:

Conta-se o número de módulos subordinados a cada módulo.

Interpretação dos resultados:

Deve-se evitar módulos com mais de nove subordinados.

Procedimento de Correção:

Havendo mais de 9 subordinados imediatos a um módulo deve-se fazer, quando possível, a fatoração de módulos de médio gerenciamento com forte coesão e baixo acoplamento.

3.3.4 Número de Módulos Superiores ("fan-in")

Processo de Avaliação:

Contar o número de módulos superiores a cada módulo.

Interpretação dos resultados:

Um módulo, que é chamado por mais de três módulos, deve ser verificado, pois pode conter mais de uma função com diferentes chamadores.

Procedimento de Correção:

Neste caso, avalia-se a possibilidade de fundi-lo ao seu superior ou fatorá-lo, de acordo com a função que realiza.

3.3.5 Complexidade total do sistema

Processo de Avaliação:

Aplicar a fórmula de Card e Agresti, descrita na seção III.2.1.5, página 83, considerando todos os módulos da estrutura.

Interpretação dos resultados:

Rever a estrutura, caso a complexidade total do sistema seja maior do que 26, identificando os módulos com alto "fan-out" e alto acoplamento.

Procedimento de Correção:

Verificar nos módulos com "fan-out" maior do que três, a possibilidade de serem fatorados para minimizar a complexidade total da estrutura. Da mesma forma, os módulos com alto acoplamento (controle, área de dados comum e conteúdo) devem ser revistos.

3.4 Características dos Usuários

O usuário da ferramenta deverá estar habituado com as técnicas de projeto estruturado propostas por (Yourdon79).

Isso é necessário, pois as medidas apresentadas pelo avaliador têm aspecto altamente técnico, exigindo conhecimentos específicos na área.

3.5 Restrições, Hipóteses e Dependências

Por uma questão de integração de ferramentas, optou-se por implementá-las em um mesmo ambiente, o ambiente Actor (Actor 90). Este ambiente, por ser orientado a objetos, facilita a reutilização e manutenção das ferramentas existentes, bem como a integração de novas ferramentas.

O Actor utiliza o sistema de interface com o usuário do MS-Windows (Windows 87). Desta forma, todas as ferramentas poderão ter o mesmo padrão de interface.

Esta versão será implementada em microcomputadores compatíveis com a linha PC-AT. É recomendável que o microcomputador utilizado tenha memória de, ao menos, 640 Kbytes, possua um "mouse" e possa acessar um disco rígido (Winchester). É necessário, também, uma impressora.

3.6 Prioridades de Implementação

A versão inicial do EDTI-GEII tem como prioridade de implementação as funções básicas para criação, alteração e avaliação do GE, armazenando-o e imprimindo-o, além da interface com o usuário proporcionada pelo MS-Windows que permite visualizar todo o gráfico e mover objetos na janela. O AVALIE-GE avaliará os critérios descritos no item 3.3 deste documento.

IV.2.2 ESPECIFICAÇAO DE REQUISITOS

1. Introdução

1.1 Propósito

Este documento tem como objetivo, identificar os requisitos de software do editor e avaliador de Gráfico de Estrutura (EDIT-GEII), que se desejam encontrar implementados no produto final, bem como identificar as interfaces necessárias.

1.2 Escopo

O EDIT-GEII, conforme descrito na Definição do Sistema, está acoplado ao AVALIE-GE, para servir de apoio ao desenvolvimento estruturado de software.

1.3 Visão Geral da Especificação

A descrição geral da ferramenta já foi feita na Definição do Sistema, sendo apresentados seus objetivos globais, suas funções, caracteristicas do usuário e restrições existentes.

A Especificação de Requisitos irá apresentar no capitulo 2, os requisitos especificos do EDIT-GEII, através da especificação de suas classes principais, identificando seus atributos, métodos e subclasses. No capítulo 3, são definidas as prioridades de implementação e o capitulo 4 descreve melhoramentos e modificações previstas para versões futuras. O Anexo A contém o diagrama das classes decritas neste documento.

2. Requisitos Especificos

2.1 Requisitos da Classe

2.1.1 Especificação da Classe Sessão

Especificação de Sessão

Superclasse:

Descrição da classe: Uma sessão inicia quando o usuário ativa o EditGEII e termina quando o usuário encerra a sessão.

Subclasses:

Contém:

E componente de:

Atributos:

Mapeamentos:

SistemaCorrente com (0:1) Projeto

ExplicaSessão com (0:1) Explicação

Serviços:

IniciaSessão: cria uma sessão, permitindo a edição do GE.

EncerraSessão: termina a sessão. Neste caso, existe a opção de salvar o GE e a Avaliação, se ainda não tiver sido feita.

2.1.2 Requisitos de interface com o Usuário

2.1.2.1 Representação da classe

Uma sessão é representada através de uma janela da classe GEWindow.

2.1.2.2 Ações da Interface com o Usuário

2.2 Requisitos de Classe

2.2.1 Especificação da Classe Projeto

Especificação de Projeto

Superclasse:

Descrição da classe: Esta classe contém o projeto do sistema a ser construído, através de seus módulos e do relacionamento entre estes.

Subclasses:

Contém:

Módulos: um Projeto é um conjunto de (0:N) módulos e seus relacionamentos.

É componente de:

Atributos:

NomeProjeto: string

Mapeamentos:

SessãoAberta com (0:1) Sessão

ExplicaProjeto com (0:1) Explicação

Armazenado com (0:N) Arquivo

Serviços:

CriaProjeto

NomeiaProjeto (N: string): define N como o nome do projeto

IncluiMódulo (M: módulo escolhido)

ExcluiMódulo (M: módulo escolhido)

2.2.2 Requisitos de interface com o Usuário

2.2.2.1 Representação da classe

Um projeto é representado por um Gráfico de Estrutura (GE) que é composto por módulos e seus relacionamentos.

2.2.2.2 Ações da Interface com o Usuário

MoverMódulo: move símbolo de módulo para outra posição

ImprimirGE: imprime o gráfico corrente

ImprimirES: imprime o sumário das informações do módulo

2.3 Requisitos de Classe

2.3.1 Especificação da Classe Arquivo

Especificação de Arquivo

Superclasse:

Descrição da classe: Faz o gerenciamento do armazenamento do gráfico de estrutura e da avaliação.

Subclasses:

Contém:

É componente de:

Atributos:

NomeArquivo: string

Status (char): diz se o arquivo está fechado ou aberto, etc.

Mapeamentos:

EmEdição com (0:1) GE

EmAvaliação com (0:1) Avaliação

Serviços:

CriaArquivo

NomeiaArquivo (N:string)

SalvaArqGE

SalvaArqAvaliação

CarregaArqGE (N:string): carrega o arquivo N contendo um GE na sessão ativa pelo EDITGEII.

CarregaArqAvaliação (N:string): carrega o arquivo N contendo uma Avaliação na sessão ativa pelo EDTIGEII.

2.3.2 Requisitos de interface com o Usuário

Não apresenta.

2.4 Requisitos de Classe

2.4.1 Especificação da Classe Módulo

Especificação de **Módulo**

Superclasse:

Descrição da classe: é responsável pela criação dos módulos, que compõe o Projeto e o relacionamento entre eles.

Subclasses:

MóduloPreDefinido

MóduloReferência

MóduloGlobal

Contém:

E componente de: Projeto

Atributos:

Nome: strino

Número: strino

descrição: texto

entrada: string

saída: string

recursivo: char

iteração1: char

iteração2: char

embutido: char

transação: char

filhoEsquerda: ponteiro

irmãoEsquerda: ponteiro

irmãoDireita: ponteiro

pai: ponteiro

posiçãoModulo: ponto

Mapeamentos:

Serviços:

CriaFilho (M: um módulo, P: posição módulo): cria módulo M' subordinado a M na posição P determinada pelo usuário.

CriaIrmãoDireita (M: um módulo, P: posição módulo): cria módulo M' a direita de M na posição P determinada pelo usuário.

CriaIrmãoEsquerda (M: um módulo, P: posição módulo): cria módulo M' a esquerda de M na posição P determinada pelo usuário.

EditaMódulo: apresenta a ficha de descrição do módulo a ser criado.

DesenhaMódulo (P: posição do módulo): desenha o símbolo de módulo na posição P determinada pelo usuário.

AlteraDescrição (M: um módulo): apresenta a ficha de descrição do módulo escolhido e permite a sua alteração.

2.4.2 Requisitos de interface com o Usuário

2.4.2.1 Representação da classe

O símbolo de módulo é um retângulo representado como um elemento do GE e como uma ficha de Edição do Módulo, que contém o sumário das informações do módulo.

2.4.2.2 Ações da Interface com o Usuário

Não apresenta.

2.5 Requisitos de Classe

2.5.1 Especificação da Classe MóduloPréDefinido

Especificação de **ModuloPréDefinido**

Superclasse: Módulo

Descrição da classe: representa um tipo de módulo chamado de Módulo Pré-definido.

Subclasses:

Contém:

É componente de:

Atributos: herda os atributos da classe Módulo

Mapeamentos:

Serviços:

DesenhaMódulo (P: posição do módulo): desenha o símbolo de módulo pré-definido na posição P determinada pelo usuário.

2.5.2 Requisitos de interface com o Usuário

2.5.2.1 Representação da classe

O símbolo de módulo pré-definido é um retângulo com uma barra no lado esquerdo e outra no lado direito unindo os lados superior e inferior.

2.5.2.2 Ações da Interface com o Usuário

Não apresenta.

2.6 Requisitos de Classe

2.6.1 Especificação da Classe MóduloReferência

Especificação de **MóduloReferência**

Superclasse: Módulo

Descrição da classe: representa um tipo de módulo chamado de módulo referência.

Subclasses:

Contém:

E componente de:

Atributos: herda os atributos da classe módulo

Mapeamentos:

Serviços:

DesenhaMódulo (P: posição do módulo): desenha o símbolo de módulo referência na posição P determinada pelo usuário.

2.6.2 Requisitos de interface com o Usuário

2.6.2.1 Representação da classe

O símbolo de MóduloReferência é um pentágono com um dos vértices voltado para baixo e ligeiramente achatado no lado superior.

2.6.2.2 Ações da Interface com o Usuário

Não apresenta.

2.7 Requisitos de Classe

2.7.1 Especificação da Classe MóduloGlobal

Especificação de **MóduloGlobal**

Superclasse:

Descrição da classe: representa um tipo de módulo que é uma área de dados comum.

Subclasses:

Contém:

É componente de:

Atributos: herda os atributos da classe módulo

Mapeamentos:

Serviços:

DesenhaMódulo (P: posição do módulo): desenha o símbolo de módulo com área global de dados na posição P determinada pelo usuário.

2.7.2 Requisitos de interface com o Usuário

2.7.2.1 Representação da classe

O símbolo de módulo com área global de dados é um retângulo com os lados direito e esquerdo arredondados.

2.7.2.2 Ações da Interface com o Usuário

Não apresenta.

2.8 Requisitos de Classe

2.8.1 Especificação da Classe Avaliação

Especificação de Avaliação

Superclasse:

Descrição da classe: fornece a avaliação de um GE, de acordo com as opções escolhidas pelo usuário. Para tal, são consultados os valores limites para cada um dos critérios de avaliação na tabela AVALIACAO.

Subclasses:

Contém:

E componente de:

Atributos:

tipo: string, representa se a avaliação é local ou global.

critério: string, representa o critério de avaliação escolhido.

Mapeamentos:

Avaliação com (1:1) GE

Armazenado com (0:N) Arquivo

ExplicaAvaliação com (0:1) Explicação

Serviços:

AvaliaCoesão

AvaliaAcoplamento

AvaliaFanin

AvaliaFanout

AvaliaCt

2.8.2 Requisitos de interface com o Usuário

2.8.2.1 Representação da classe

É representada por uma janela de edição, onde é apresentada a avaliação de acordo com o serviço executado.

2.8.2.2 Ações da Interface com o Usuário

FornecerAvaliação: fornece na janela de edição a avaliação pedida pelo usuário.

ImprimirAvaliação

2.9 Requisitos de Classe

2.9.1 Especificação da Classe Explicação

Especificação de Explicacão

Superclasse:

Descrição da classe: fornece uma explicação a um termo, escolhido pelo usuário, relacionado a utilização da ferramenta e no uso do método.

Subclasses:

Contém:

É componente de:

Atributos:

termo: string

Mapeamentos:

ExplicaSessão com (1:1) Sessão

ExplicaProjeto com (1:1) Projeto

ExplicaAvaliação com (1:1) Avaliação

Serviços:

MostraExplicação (T: termo)

MostraExplAnterior (T: termo)

MostraExplPosterior (T: termo)

2.9.2 Requisitos de interface com o Usuário

2.9.2.1 Representação da classe

É representada por uma caixa de diálogo contendo uma lista de termos que podem ser escolhidos pelo usuário.

2.9.2.2 Ações da Interface com o Usuário

Não apresenta.

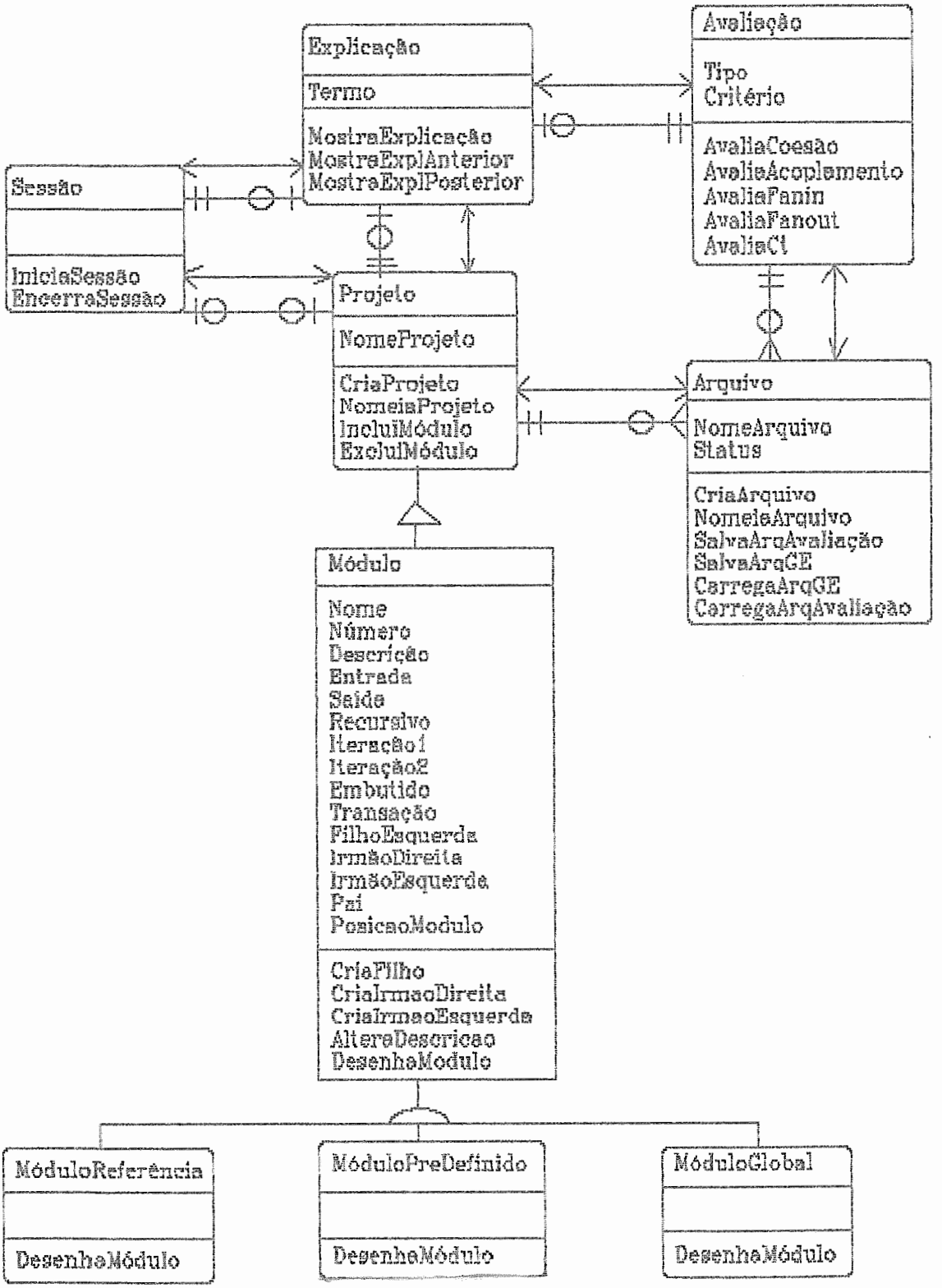
3. Prioridade de Implementação

A prioridade de implementação foi dada à edição de Gráficos de Estrutura (GE) e à sua avaliação segundo os critérios apresentados na definição, ao armazenamento das informações e à impressão dos gráficos de estrutura (GE) e de suas respectivas avaliações.

4. Modificações e Melhoramentos Previstos

Permitir copiar um módulo ou subgráfico para uma posição determinada pelo usuário, cortar (excluir) um subgráfico, reformatar o GE caso o usuário necessite, e avaliação global dos critérios acoplamento e coesão.

Anexo A - Diagrama de Classes



IV.2.3 ESPECIFICAÇÃO DE PROJETO

Seguindo o roteiro de documentação para a fase de projeto proposto por Mattoso (Mattoso 90), será apresentada a seguir a especificação de projeto do EDIT-GEII.

1. Introdução

1.1 Propósito

Baseado no Diagrama de Classes apresentado no documento de Especificação de Requisitos, este documento tem como objetivo apresentar a Especificação de Projeto do EDIT-GEII acoplado ao AVALIE-GE.

1.2 Descrição Geral

O EDIT-GEII é uma ferramenta para edição de Gráficos de Estrutura (GE) que está integrada a uma outra ferramenta, AVALIE-GE, que é um analisador estático para avaliação da modularidade do GE.

O EDIT-GEII, quando ativo, mostra uma janela (da classe GEWindow). Nesta, o usuário pode editar o GE referente ao projeto em questão ou fornecer ao sistema o GE construído com o EDIT-GEII e selecionar as opções de avaliação.

O sistema fornece ao usuário a avaliação de qualidade de um GE , de acordo com as opções selecionadas. A avaliação do gráfico baseia-se na comparação do valor obtido por cada tipo de medida com os valores limites.

Os tipos de medida possíveis são: Acoplamento, Coesão, Número de módulos subordinados, Número de módulos superiores, Complexidade total da estrutura. A definição, processo de avaliação e interpretação dos resultados de tais medidas foram apresentados no documento de Definição do Sistema (item 3.3).

O sistema fornece ajuda na utilização da ferramenta e no uso do método.

Toda a interface com o usuário segue o padrão das aplicações do MS-Windows.

1.3 Visão Geral da Especificação

A partir da Especificação de Requisitos este documento apresenta no capítulo 2, as classes, métodos, atributos, subclasses e detalha os serviços. No capítulo 3, apresenta-se as tabelas para armazenamento e manipulação dos dados.

2. Especificação dos objetos da interface com o usuário

2.1 Especificação da Classe

Especificação de **GEWindow**

Representa: Sessão

Superclasse: Janela

Descrição da classe: uma janela de documento padrão MS-Windows com rolamento ("scrolling"), redimensionamento, "zoom", e demais controles. A GEWindow é responsável pela representação do Gráfico de Estrutura (GE) através de uma hierarquia de módulos e seus relacionamentos. Sua função é

exibir o GE, um de cada vez, e permitir a sua edição. A figura IV.1 mostra a GEWINDOW, o seu ícone e o cardápio com as opções.

Subclasses:

Contém:

Cardápio: uma barra de cardápio

Conteúdo: uma vista retangular do GE corrente

Ícone: representação de um GE

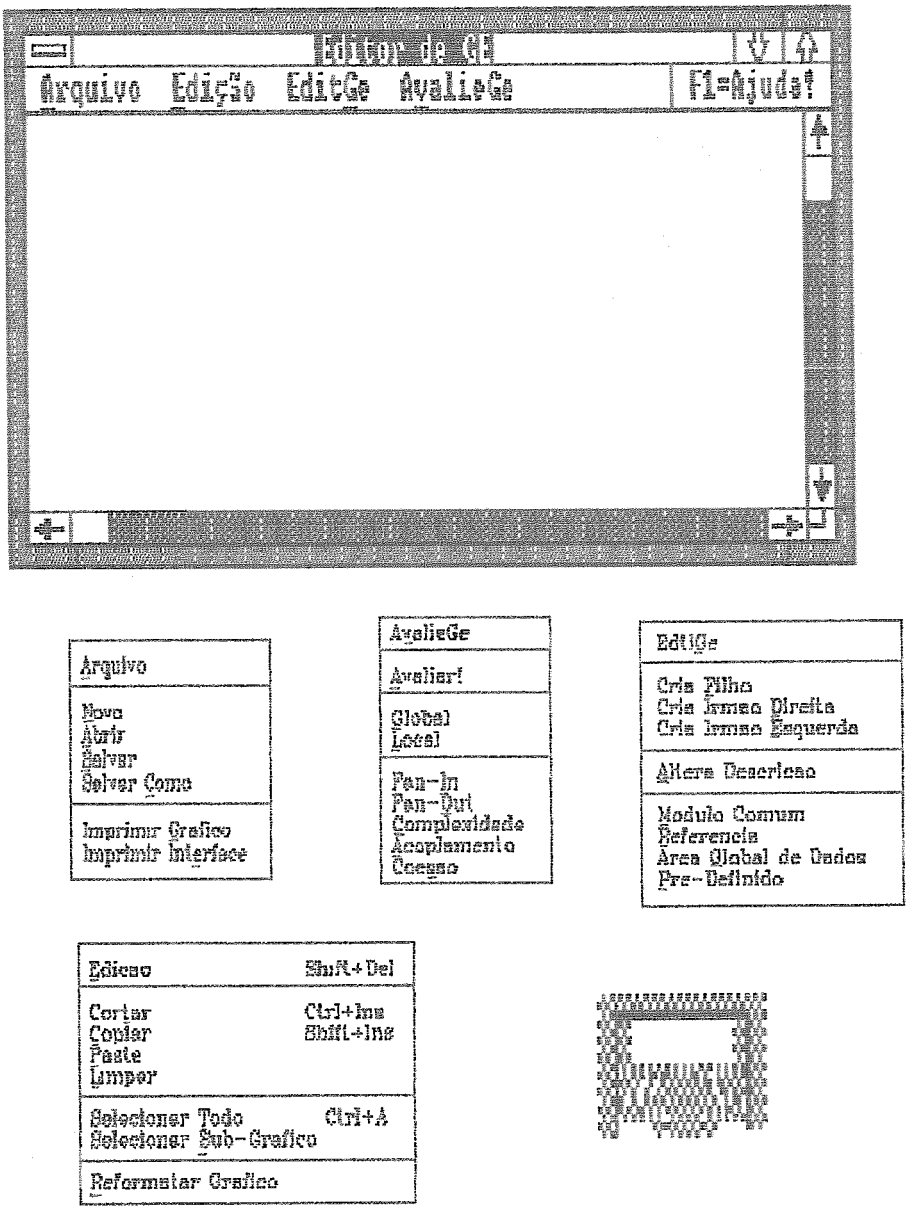


Figura IV.1: A janela do Editor de Grafico de Estrutura

E componente:

Atributos:

Vista: retângulo que representa o que está sendo exibido na janela ("client's" area do MS-Windows).

Mapeamentos:

Ações:

IniciaSessão

entrada:

saída: uma GEWindow

descrição: realizado externamente através do Ms-Windows, com o usuário ativando o EDTIGEII.

pré-condições:

pós-condições: a GEWindow é aberta.

EncerraSessão:

entrada: uma GEWindow

saída:

descrição: realizada através do fechamento da GEWindow pelo MS-Windows ou da opção Fim no menu de Arquivo.

pré-condições:

pós-condições:

funcionamento: se o GE ou a avaliação foi modificado após a última operação SalvarArqGE ou SalvarArqAvaliação, então será iniciado um diálogo perguntando se o usuário deseja salvar as alterações feitas (padrão MS-Windows).

CancelaSessão

2.2 Especificação da Classe

Especificação de GE

Representa: Projeto

Superclasse:

Descrição da classe: A classe GE é responsável pela representação do GE do projeto de um sistema, segundo o método Projeto Estruturado. Contém módulos e as ligações que representam o relacionamento entre eles. O GE é representado por uma coleção ordenada de módulos que mostra a hierarquia entre eles (figura IV.2).

Subclasses:

Contém:

módulos: uma coleção de módulos e seus relacionamentos.

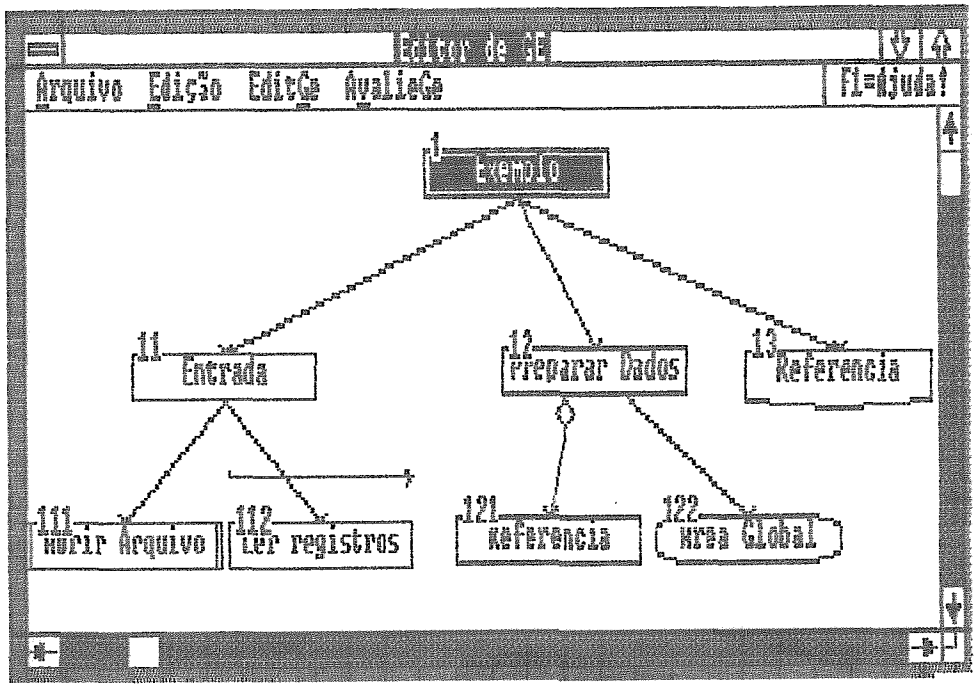


Figura IV.2: Um exemplo de GE com os diferentes tipos de módulos.

E componente:

Atributos:

NomeProjeto: string

Pai: ponteiro (módulo de topo no GE)

Posição: ponto (ponto de inserção de um módulo no GE)

Ativo: ponteiro (é o módulo escolhido)

Mapeamentos:

Ações:

CriaGE

entrada:

saída:

descrição: cria-se um GE através da escolha da opção NOVO no menu de Arquivo seguindo o padrão MS-Windows.

pré-condições:

pós-condições:

funcionamento: executa a ação IncluiMódulo

DuplicaGE

entrada:

saída:

descrição: através da opção SALVAR COMO... do menu de Arquivo, seguindo o padrão do MS-Windows.

pré-condições:

pós-condições:

funcionamento:

NomeiaProjeto

SalvaArqGE (Caso tenha se alterado o GE após a última operação de Salvar, as alterações só serão gravadas no novo arquivo).

IncluiMódulo

entrada:

saída: um novo módulo no GE

descrição: no menu Módulo, através da escolha de uma das opções CriaFilho, CriaIrmãoEsquerda, CriaIrmãoDireita e de uma das opções de tipo (módulo comum, módulo referência, módulo pré-definido).

pré-condições: se primeira inclusão de módulo ou módulo Ativo não tem filho, é obrigatório escolher a opção CriaFilho.

pós-condições:

funcionamento: o usuário "clica" com o mouse no módulo que deseja que tenha subordinado ou irmão. Em seguinda, escolhe ("clica" com o mouse) a posição onde deseja que seja desenhado o símbolo de módulo e escolhe a opção no menu Módulo. Em seguida, é aberto o diálogo de Ediçao do Módulo, pedindo informações sobre o módulo, como nome, descrição, interfaces, tipo de cada interface e tipo do módulo. Após preenchido este diálogo, os dados são armazenados nas tabelas de manipulação (SalvaMódulo) e um símbolo referente ao tipo de módulo escolhido (com nome e número) é desenhado na tela na posição indicada.

ExcluiMódulo

entrada: módulo Ativo

saída:

descrição: o usuário "clica" no módulo a ser escolhido, tornando-o Ativo, e solicita a opção de Cortar no menu Edição.

pré-condições: o módulo Ativo não ter subordinados.

pós-condições:

funcionamento: ao selecionar essa opção, o sistema verifica se o módulo possui subordinados. Neste caso, é enviada uma mensagem de erro. Caso contrário, o módulo é excluído do GE e das tabelas de manipulação e a GEWindow é redesenhada sem este módulo.

MoveMódulo

entrada: módulo Ativo

saída:

descrição: move a posição de Ativo pelo GE

pré-condições:

pós-condições:

funcionamento: o usuário seleciona o módulo e arrasta-o pela janela até a posição desejada. O símbolo anterior é apagado e desenhado na nova posição que é atualizada na tabela MODULO.

DesenhaGE

entrada:

saída:

descrição: após cada alteração na GEWindow o gráfico é redesenhado.

pré-condições:

funcionamento: de acordo com a estrutura de dados utilizada para construção do GE (seção 3), a GEWindow conhece o módulo de topo (Pai). A partir do Pai, a árvore é percorrida em end-ordem. Para cada módulo visitado, chama DesenhaMódulo (Módulo ou MóduloPréDefinido ou MóduloGlobal ou MóduloReferência). Caso módulo igual a Ativo, então chama DesenhaMóduloAtivo. Desenha ligação entre o módulo superior e subordinado.

2.3 Especificação da Classe

Especificação de **Módulo**

Representa: Módulo

Superclasse: Objeto

Descrição da classe: responsável pelo gerenciamento da edição e especificação dos módulos que compõe o GE. É representada por uma ficha que o usuário pode editar contendo informações sobre o módulo, como: nome, descrição, interfaces e tipo de módulo. O símbolo de módulo é um retângulo que contém no seu interior o nome do módulo e no canto superior esquerdo o número do módulo (gerado automaticamente e representa a posição do módulo no GE). Se o usuário desejar, ele pode consultar a ficha do módulo para verificação e alteração.

Subclasses:

MóduloPréDefinido

MóduloReferência

MóduloGlobal

Contém:

É componente: GE

Atributos:

Nome: string

Número: string

descrição: conjunto de strings

entrada: string

saída: string

recursivo: char

iteração1: char

iteração2: char

embutido: char

transação: char

filhoEsquerda: ponteiro

irmãoEsquerda: ponteiro

irmãoDireita: ponteiro

pai: ponteiro

posiçãoModulo: ponto

Mapeamentos:

Ações:

CriaFilho:

entrada: módulo escolhido

saída:

descrição: o usuário escolhe o módulo que receberá um subordinado através da opção CriaFilho no menu Módulo

pré-condições: não existir módulo subordinado ao módulo escolhido

pós-condições:

funcionamento: o usuário "clica" com o mouse no módulo que deseja que tenha subordinado ou irmão. Em seguinda,

escolhe ("clica" com o mouse) a posição onde ele deseja que seja desenhado o símbolo de módulo. Caso já exista subordinado ao módulo escolhido é enviada uma mensagem de erro. Caso contrário, é aberto o diálogo para Edição do Módulo onde o usuário fornece os dados sobre o módulo (EditaMódulo). O módulo é desenhado na posição indicada (DesenhaMódulo).

CriaIrmãoDireita

entrada: módulo escolhido

saída:

descrição: o usuário escolhe o módulo que receberá um módulo à sua direita através da opção CriaIrmãoDireita no menu Módulo

pré-condições: o módulo escolhido não pode ser o de topo (Pai) do GE.

pós-condições:

funcionamento: o usuário "clica" com o mouse no módulo que deseja que tenha subordinado ou irmão. Em seguinda, escolhe ("clica" com o mouse) a posição onde ele deseja que seja desenhado o símbolo do novo módulo. Caso o módulo escolhido seja o de topo (Pai) é enviada uma mensagem de erro. Caso contrário, chama EditaMódulo. O módulo é desenhado na posição indicada (DesenhaMódulo).

CriaIrmãoEsquerda

entrada: módulo escolhido

saída:

descrição: o usuário escolhe o módulo que receberá um módulo à sua esquerda através da opção CriaIrmão-Esquerda no menu Módulo.

pré-condições: o módulo escolhido não pode ser o de topo (Pai) do GE.

pós-condições:

funcionamento: o usuário "clica" com o mouse no módulo que deseja que tenha subordinado ou irmão. Em seguinda, escolhe ("clica" com o mouse) a posição onde ele deseja que seja desenhado o símbolo do novo módulo. Caso o módulo escolhido seja o de topo (Pai) é enviada uma mensagem de erro. Caso contrário, chama EditaMódulo. O módulo é desenhado na posição indicada (DesenhaMódulo).

DesenhaMódulo

entrada: descrição do módulo e a posição onde deverá ser desenhado no GE.

saída: o símbolo de módulo

descrição: desenha o símbolo de módulo com nome e número na posição indicada.

pré-condições:

pós-condições:

funcionamento: desenha o símbolo de módulo na posição determinada contendo o nome e o número do módulo. O símbolo é representado por um retângulo. Aqui, também são desenhados os símbolos de embutido e recursivo, caso o usuário tenha especificado no diálogo de edição do módulo.

DesenhaMóduloAtivo

entrada: descrição do módulo e a posição onde deverá ser desenhado no GE.

saída: o símbolo de módulo Ativo

descrição: desenha o símbolo de módulo Ativo com nome e número na posição indicada.

pré-condições:

pós-condições:

funcionamento: desenha o símbolo de módulo na posição determinada contendo o nome e o número do módulo. O símbolo é representado por um retângulo em vídeo reverso. Aqui, também são desenhados os símbolos de embutido e recursivo, caso o usuário tenha especificado no diálogo de edição do módulo.

AlteraDescrição

entrada: módulo selecionado

saída: o módulo alterado

descrição: o usuário "clica" no módulo escolhido e solicita a opção AlteraDescrição no menu Módulo.

pré-condições:

pós-condições:

funcionamento: Ao selecionar esta opção, o sistema apresenta o diálogo de Edição do Módulo com as informações correntes, permitindo sua alteração.

EditaMódulo

entrada: um módulo a ser criado

saída: a descrição do módulo contendo nome, descrição, tipo, interfaces e tipo de cada uma das interfaces.

descrição: responsável pelo diálogo de Edição do Módulo.

pré-condições:

pós-condições:

funcionamento: o diálogo de Edição do Módulo (figura IV.3) é uma ficha de descrição do módulo que deve ser preenchida pelo usuário com as seguintes informações: nome, descrição, interfaces, tipo de cada interface e tipo de módulo.

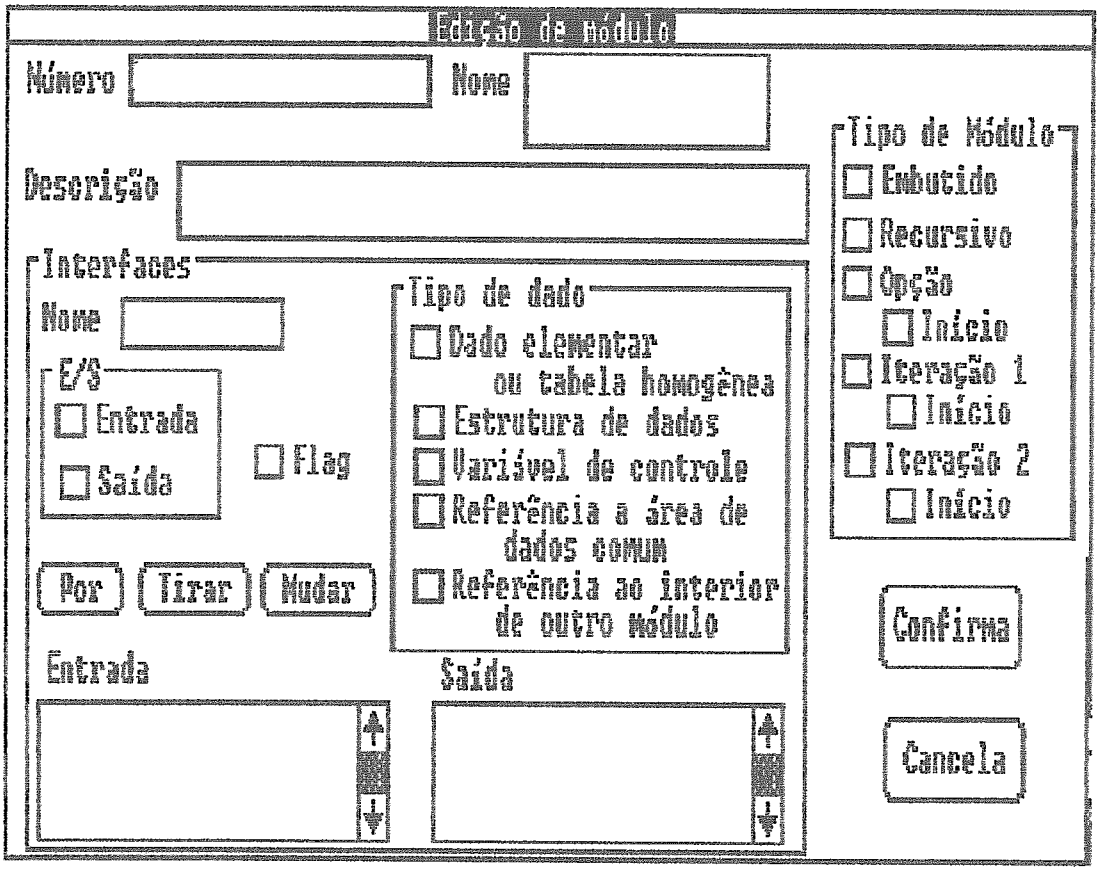


Figura IV.3: Diálogo de edição de um módulo

2.4 Especificacão da Classe

Especificação de MóduloPréDefinido

Representa: MóduloPréDefinido

Superclasse: Módulo

Descrição da classe: representa um tipo de módulo chamado de Módulo Pré-definido.

Subclasses:

Contém:

E componente:

Atributos: herda os atibutos da classe Módulo

Mapeamentos:

Ações:

DesenhaMódulo

entrada: o módulo a ser desenhado e a posição onde deverá ser desenhado no GE.

saída: o símbolo de módulo pré-definido.

descrição: desenha o símbolo de módulo com nome e número na posição indicada.

pré-condições:

pós-condições:

funcionamento: desenha o símbolo de módulo pré-definido na posição determinada pelo usuário contendo no seu interior o nome e o número do módulo. O símbolo de módulo pré-definido é representado por um retângulo com uma barra no lado esquerdo e outra no lado direito unindo o lados superior e inferior.

2.5 Especificação da Classe

Especificação de **MóduloReferência**

Representa: MóduloReferência

Superclasse: Módulo

Descrição da classe: representa um tipo de módulo chamado de módulo referência.

Subclasses:

Contém:

E componente:

Atributos: herda os atributos da classe Módulo

Mapeamentos:

Ações:

DesenhaMódulo

entrada: o módulo a ser desenhado e a posição onde deverá ser desenhado no GE.

saída: o símbolo de módulo referência

descrição: desenha o símbolo de módulo na posição determinada pelo usuário contendo no seu interior nome e número.

pré-condições:

pós-condições:

funcionamento: desenha o símbolo de módulo de módulo referência contendo no seu interior o nome e o número do módulo na posição indicada. O símbolo de módulo referência é um pentágono com um dos vértices virado para baixo.

2.6 Especificação da Classe

Especificação de **MóduloGlobal**

Representa: MóduloGlobal

Superclasse: Módulo

Descrição da classe: representa um tipo de módulo que contém uma área global de dados

Subclasses:

Contém:

E componente:

Atributos: herda os atributos da classe Módulo

Mapeamentos:

Ações:

DesenhaMódulo

entrada: o módulo a ser desenhado e a posição onde deverá ser desenhado no GE.

saída: o símbolo de módulo com área global de dados

descrição: desenha o símbolo de módulo com nome e número na posição indicada.

pré-condições:

pós-condições:

funcionamento: desenha o símbolo de módulo com área global de dados na posição determinada pelo usuário contendo no seu interior o nome e o número do módulo. O símbolo deste tipo de módulo é um retângulo com os lados direito e esquerdo arredondados.

2.7 Especificação da Classe

Especificação de **AvalieWindow**

Representa: Avaliação

Superclasse: JanelaDeTexto

Descrição da classe: uma janela de documento padrão MS-Windows com rolamento ("scrolling"), redimensionamento, "zoom", e demais controles. A AvalieWindow (Avaliação) é responsável pela avaliação do gráfico de estrutura apresentado na GEWindow. Sua função é exibir a avaliação de um GE, uma de cada vez, de acordo com as opções escolhidas pelo usuário. Para tal, são consultados os valores limites para cada uma das medidas de avaliação na

tabela AVALIAÇÃO. A figura IV.4 mostra a janela Avaliação, seu cardápio com as opções e o seu ícone.

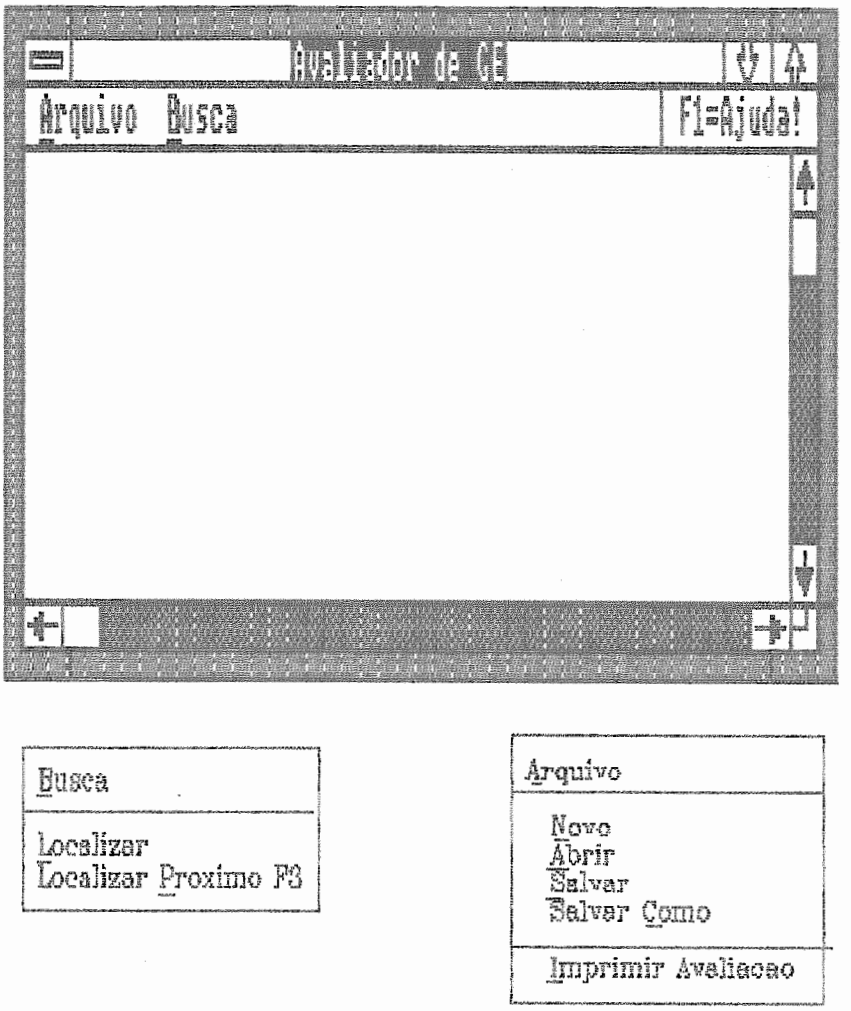


Figura IV.4: A janela de Avaliação

Descrição da classe:

Subclasses:

Contém:

Cardápio: uma barra de cardápio

Conteúdo: um retângulo para fornecer a avaliação solicitada.

Icone: é o símbolo do projeto TABA com uma interrogação.

É componente:

Atributos:

tipo: string, representa se a avaliação é local ou global.

critério: string, representa o critério de avaliação escolhido.

Mapeamentos:

Ações:

AvaliaCoesão

entrada: módulo escolhido e tipo

saída:

descrição: o usuário seleciona a opção Coesão no menu Avaliação, uma das opções de tipo (Local ou Global) e aciona o avaliador através da opção "Avalia!".

pré-condições: existir a descrição do módulo escolhido.

pós-condições:

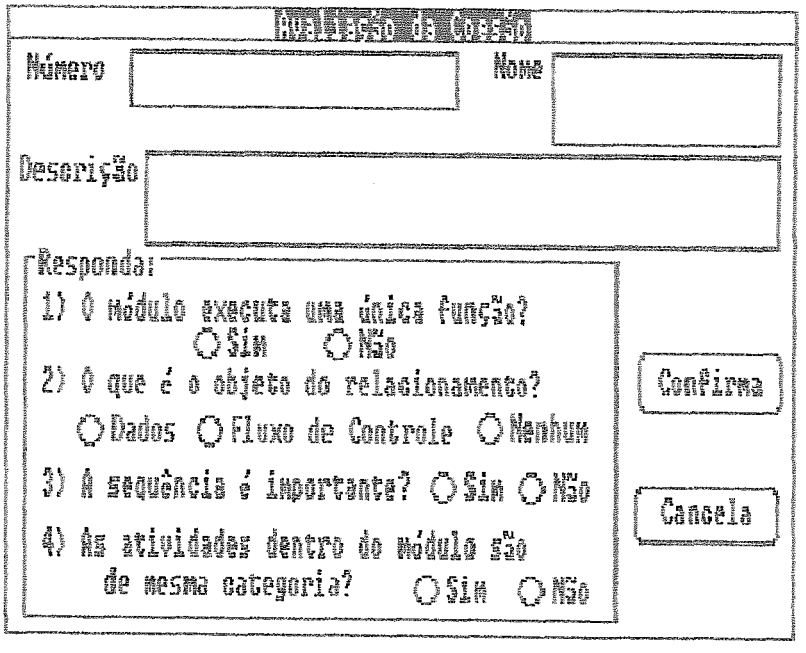


Figura IV.5: Diálogo para avaliação do critério coesão

funcionamento: se o usuário optou pela avaliação Local deve informar o módulo a ser avaliado ("clicando" no módulo). Caso tenha sido a opção Global, o sistema

avaliará todos os módulos do gráfico. A avaliação é feita através da descrição do módulo e é apresentado o diálogo para tratamento iterativo da coesão (figura IV.5), onde o usuário preenche os campos solicitados e, a partir destes, é feita uma análise que identifica o tipo de coesão, de acordo com o processo de avaliação descrito na seção 3.3.2 do documento de Definição do Sistema.

AvaliaAcoplamento

entrada: módulo escolhido e tipo.

saída:

descrição: o usuário seleciona a opção Acoplamento no menu Avaliação, uma das opções de tipo (Local ou Global) e aciona o avaliador através da opção "Avalia!".

pré-condições: as interfaces do módulo devem estar descritas.

pós-condições:

funcionamento: se o usuário optou pela avaliação Local deve informar o módulo a ser avaliado ("clicando no módulo"). Caso tenha sido a opção Global, o sistema avaliará todos os módulos do gráfico. A avaliação é feita através da análise da interface do módulo de acordo com o processo de avaliação descrito na seção 3.3.1 do documento de Definição do Sistema.

AvaliaFanin

entrada: módulo escolhido

saída:

descrição: o usuário seleciona a opção Fan-in no menu Avaliação, uma das opções de tipo (Local ou Global). Em seguida, aciona o avaliador através da opção "Avalia!".

pré-condições:

pós-condições:

funcionamento: se o usuário optou pela avaliação Local deve informar o módulo a ser avaliado: Caso tenha sido a opção Global, o sistema avaliará todos os módulos do gráfico. A avaliação é feita de acordo com o processo de avaliação descrito na seção 3.3.4 do documento de Definição do Sistema.

AvaliaFanout

entrada: módulo escolhido

saída:

descrição: o usuário seleciona a opção Fan-out no menu Avaliação, uma das opções de tipo (Local ou Global) e aciona o avaliador através da opção "Avalia!".

pré-condições:

funcionamento: se o usuário optou pela avaliação Local deve informar o módulo a ser avaliado. Caso tenha sido a opção Global, o sistema avaliará todos os módulos do gráfico. A avaliação é feita de acordo com o processo de avaliação descrito na seção 3.3.3 do documento de Definição do Sistema.

AvaliaCt

entrada:

saída:

descrição: o usuário seleciona a opção Complexidade no menu Avaliação e aciona o avaliador através da opção "Avalia!".

pré-condições: só pode ser utilizado para avaliação global.

pós-condições:

funcionamento: Se a opção "Fan-out" não tiver sido escolhida, Então AvaliaFanout.

Enquanto houver módulo faça:

Acumula o número de interfaces por módulo através da tabela MODULO;

Divide o valor acumulado pelo número de fan-out para aquele módulo mais 1;

Acumula esse valor;

Acumula o valor de fan-out ao quadrado de cada módulo;

Divide ambos os acumuladores pelo número de módulos;

Soma esses quocientes.

Fim Enquanto.

DuplicarAvaliação

entrada:

saída:

descrição: através da opção SALVAR COMO... do menu de Arquivo da AvalieWindow, seguindo o padrão do MS-Windows.

pré-condições:

pós-condições:

funcionamento:

NomeiaProjeto

SalvaArqAvaliação: Caso tenha se alterado a avaliação após a última operação de Salvar, as alterações só serão gravadas no novo arquivo.

2.8 Especificação da Classe

Especificação de **Explicação**

Representa: Explicação

Superclasse: Diálogo

Descrição da classe: uma caixa de diálogo que fornece ajuda ("help") na utilização da ferramenta e o uso do método. Consiste de uma lista de termos que podem ser escolhidos pelo usuário. Uma vez identificado o termo, abre-se uma outra caixa com a explicação (figura IV.6).

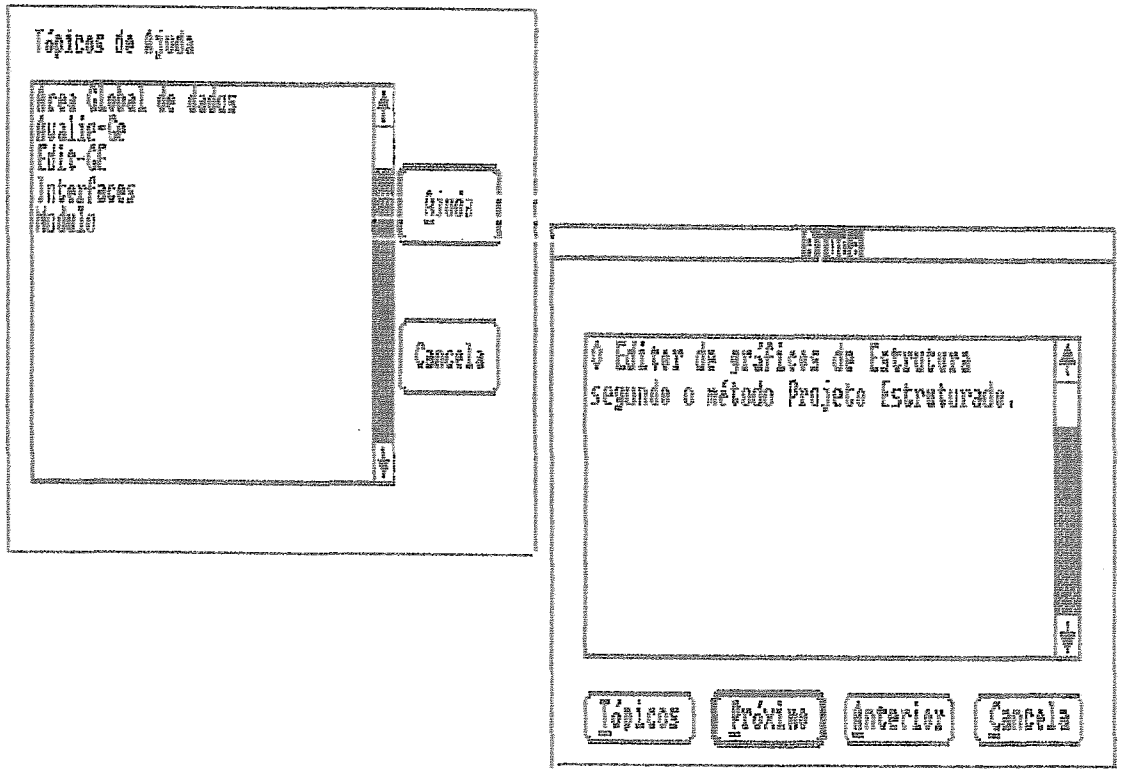


Figura IV.6: Diálogo de ajuda

Subclasses:

Contém:

E componente:

Atributos: termo: string

Mapeamentos:

ExplicaGEWindow (1:1) GEWindow

ExplicaProjeto (1:1) Projeto

ExplicaAvaliação (1:1) Avaliação

Ações:

MostraExplicação

entrada: termo

saída:

descrição: a partir de um termo selecionado pelo usuário, abre-se uma caixa de diálogo com a explicação.

pré-condições:

pós-condições:

funcionamento: o usuário seleciona o menu de Ajuda e uma caixa de diálogo é aberta com os ítens de consulta. O usuário "clica" com o mouse no termo escolhido e uma outra caixa de diálogo é aberta com a explicação.

MostraExplicaçãoAnterior:

entrada: explicação corrente

saída:

descrição: a partir de uma explicação exibida pelo diálogo de ajuda, o usuário tem a opção de voltar novamente à explicação anterior.

pré-condições:

funcionamento: mostra a explicação anterior à corrente através do diálogo de ajuda, bastando que o usuário "clique" com o mouse no botão Anterior.

MostraExplicaçãoPosterior:

entrada: explicação corrente

saída:

descrição: a partir de uma explicação corrente, o usuário pode passar a uma próxima explicação.

pré-condições:

pós-condições:

funcionamento: mostra a explicação posterior à corrente através do diálogo de ajuda, bastando que o usuário "clique" com o mouse no botão Próxima.

2.9 Especificação da Classe

Especificação de Arquivo

Representa: Arquivo

Superclasse:

Descrição da classe: responsável pelo gerenciamento do armazenamento do GE e da avaliação.

Subclasses:

Contém:

E componente:

Atributos:

NomeArquivo: string

Status (char): diz se o arquivo está fechado ou aberto, etc...

Mapeamentos:

EmEdição com (0:1) GE

EmAvaliação com (0:1) Avaliação

Ações:

NomeiaArquivo

entrada: nome do arquivo DOS

saída:

descrição: através da opção SALVAR ou SALVAR COMO ... do menu Arquivo, seguindo o padrão MS-Windows.

pré-condições:

pós-condições:

funcionamento: o usuário seleciona a opção e é aberto um diálogo para dar nome ao arquivo (MS-Windows).

SalvaArqGE

entrada: GE

saída:

descrição: através da opção SALVAR ou SALVAR COMO ... do menu Arquivo da GEWindow, seguindo o padrão MS-Windows.

pré-condições: existir o GE

pós-condições:

funcionamento: o usuário seleciona a opção e o GE é armazenado, em formato texto, no arquivo DOS, de acordo com o descrito para DesenhaGe da classe Ge.

SalvaArqAvaliação

entrada: avaliação

saída:

descrição: através da opção SALVAR ou SALVAR COMO ... do menu Arquivo da AvalieWindow, seguindo o padrão MS-Windows.

pré-condições: existir avaliação

pós-condições:

funcionamento: o usuário seleciona a opção e a avaliação é atualizada, em formato texto, no arquivo DOS.

CarregaArqGE

entrada: arquivo DOS contendo o GE

saída:

descrição: através da opção Abrir no menu de Arquivo da GEWindow, seguindo o padrão MS-Windows

pré-condições:

pós-condições:

funcionamento: o usuário seleciona a opção AbrirGE e é aberto um diálogo contendo uma lista de arquivos DOS que possuem extensão ".GE". O usuário seleciona o arquivo desejado, e o GE é desenhado na GEWindow conforme o especificado em DesenhaGe da classe Ge.

CarregaArqAvaliação

entrada: arquivo DOS com avaliação.

saída:

descrição: através da opção AbrirArqAvaliação no menu Arquivo da AvalieWindow, seguindo o padrão MS-Windows.

pré-condições: existir o arquivo DOS com a avaliação

pós-condições:

funcionamento: após selecionar a opção, é aberto um diálogo contendo os arquivos DOS que possuem extensão ".AV". O usuário seleciona o arquivo desejado e a avaliação é mostrada na janela AvalieWindow.

3. Estrutura de dados

A estrutura de dados utilizada é a de árvore n-ária composta por listas lineares duplamente encadeadas por ser a mais próxima da estrutura do GE (Nogueira 88).

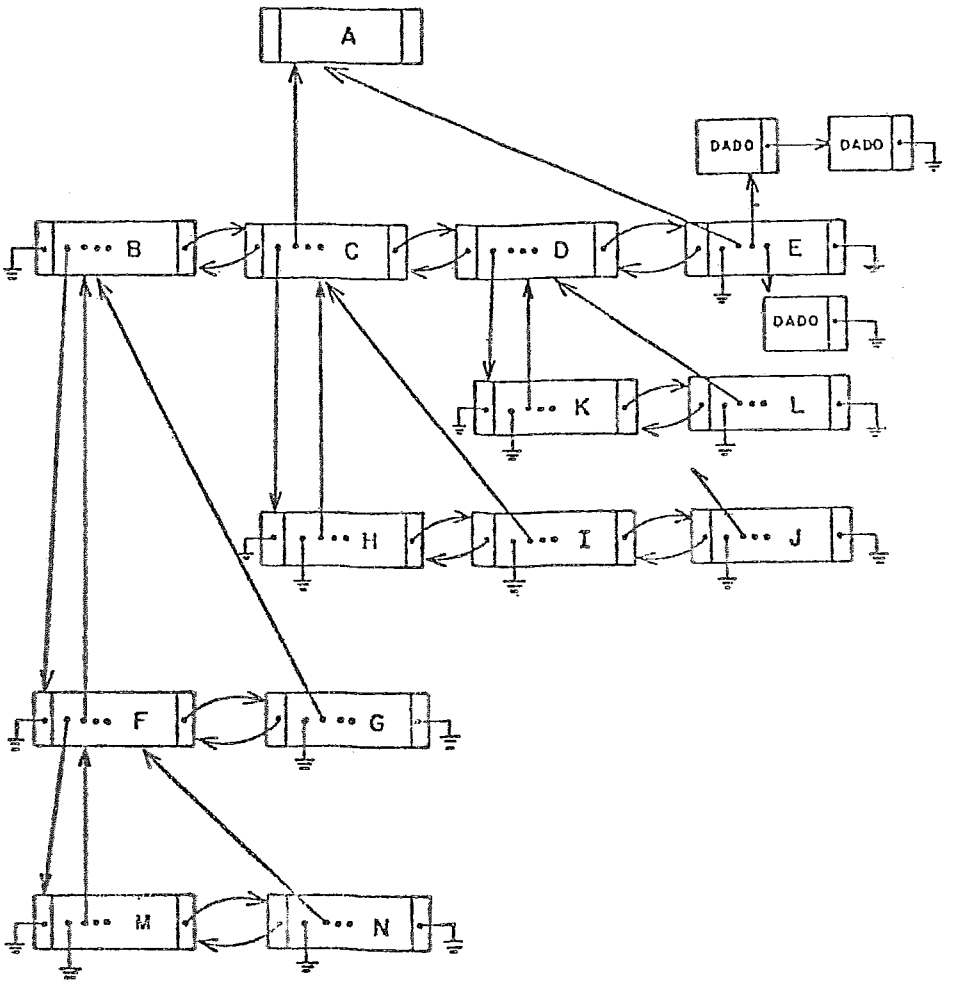


Figura IV.7: Estrutura de Dados do Editor de GE

Fonte: Nogueira 88

As figuras IV.7 e IV.8 apresentam, respectivamente, um exemplo da estrutura de árvore e sua representação gráfica.

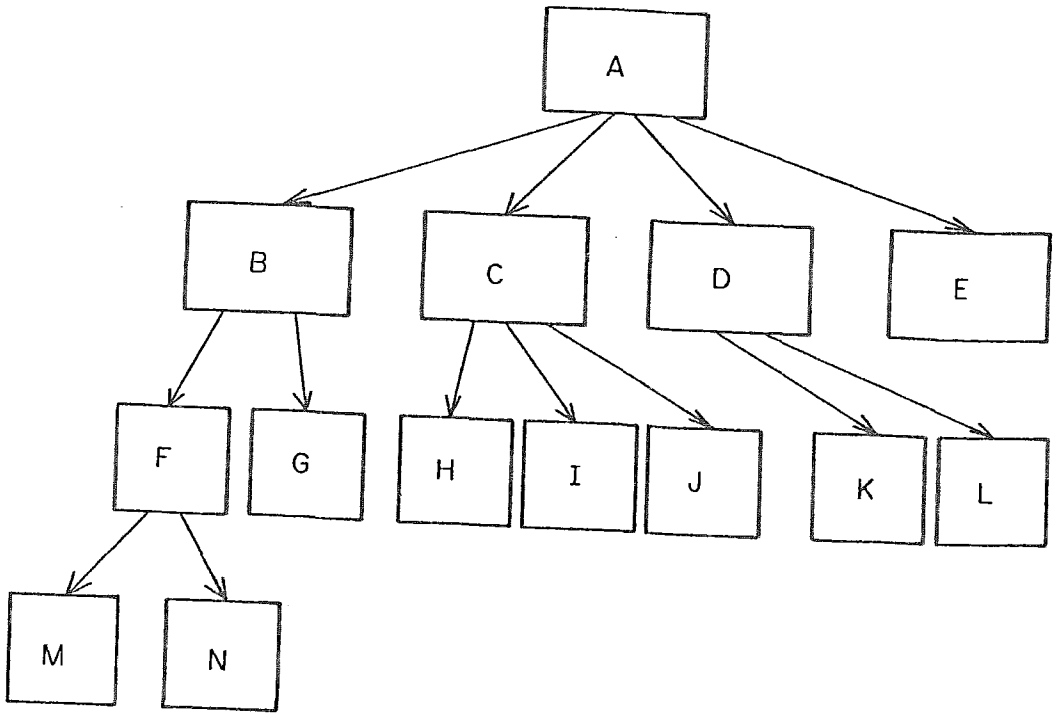


Figura IV.8: GE derivado da estrutura de dados

Fonte: Nogueira 88

IV.3 CONCLUSÃO

Neste capítulo, apresentou-se a ferramenta desenvolvida nesta tese através dos documentos de Definição do Sistema, Especificação de Requisitos e Especificação de Projetos.

No anexo I, tem-se um exemplo de utilização da ferramenta.

CAPITULO V

CONCLUSAO

Este trabalho teve como objetivo pesquisar aspectos relacionados a qualidade da especificação do projeto, com a finalidade de partindo-se de um método para avaliação de qualidade de software, identificar um conjunto de características que possam ser medidas de forma automatizada para avaliação de Gráficos de Estrutura a nível de Projeto da Arquitetura.

De acordo com o estudo da literatura feito nesta tese, constatou-se que na fase de projeto muitas vezes são introduzidos erros que só são detectados na fase de construção e operação, gerando um acréscimo no custo do produto e um decréscimo na produtividade.

Como uma tentativa de oferecer auxílio a esta fase no desenvolvimento, decidiu-se implementar uma ferramenta que auxilie na elaboração da especificação de projeto e que forneça apoio para a avaliação desta especificação.

O ponto de partida foi a proposta do projeto COPPE-SISTEMAS (Rocha 86), cujo objetivo era integrar um conjunto de ferramentas automatizadas de apoio ao desenvolvimento de software, a serem utilizadas na fase de Definição e Projeto. Estas ferramentas servem de apoio ao método de Análise e Projeto Estruturado de Sistemas, para uso em microcomputadores da linha IBM-PC.

Na primeira versão dessas ferramentas, foram desenvolvidas as ferramentas de edição do Diagrama de

Fluxo de Dados (EDIT-DFD), do Dicionário de Dados (EDIT-DD) e do Gráfico de Estrutura (EDIT-GE). Entretanto, sentiu-se a necessidade de integrá-las e optou-se por utilizar um sistema avançado de interface com o usuário (MS-Windows).

Assim, surgiu a segunda versão de ferramentas, donde emergiu o projeto FEGRES (Travassos 90) com o objetivo de integrar as ferramentas para a fase de Definição e desenvolver um editor de Entidades e Relacionamentos (EDIT-ER).

Dando continuidade a esse projeto, esta tese desenvolveu o editor de Gráfico de Estrutura (EDIT-GEII) integrado ao avaliador de modularidade (AVALIE-GE) para apoiar a fase de Projeto da Arquitetura.

O ambiente de programação escolhido para implementação das ferramentas foi o ambiente ACTOR. Deste modo, como o Actor utiliza-se do MS-Windows, todas as ferramentas poderão ter o mesmo padrão de interface, o padrão do MS-Windows.

Para a especificação das ferramentas desenvolvidas nesta tese, utilizou-se o método de especificação orientado a objetos, TABA-OBJ, proposto por Mattoso (Mattoso 90), que mostrou ser de grande valia para a documentação de produtos de software desenvolvidos segundo o paradigma da orientação a objetos.

SUGESTÕES PARA FUTURAS PESQUISAS

Uma ferramenta que pode ser facilmente integrada às aqui definidas, é um editor e avaliador para especificação

de módulos que permita avaliar a modularidade a nível de Projeto Detalhado.

O ideal seria que todas as ferramentas construtivas ao longo do processo de desenvolvimento de software possuissem uma ferramenta de avaliação, do tipo regras ou critérios, para atingir a qualidade do produto final. Isso torna mais fácil tanto o processo de construção como de avaliação, possibilitando um controle de qualidade ao longo de todo o processo de desenvolvimento. Espera-se que este trabalho tenha contribuído para esse fim.

REFERENCIAS BIBLIOGRAFICAS

ACTOR (1990), Actor Language Manual, The Whitewater Group, Inc., Version 2.0.

ARTHUR, J. (1985), "Measuring Programmer Productivity and Sotware Quality", John Wiley.

BAHIA, A.S. (1988), "Métricas de Complexidade de Programas: algumas considerações", Relatório Técnico do Programa de Engenharia de Sistemas e Computação, ES-171/88, COPPE/UFRJ.

BASILI, V.R.; ROMBACH, D.H. (1987), "Implementing Quantitative SQA: A Practical Model" in IEEE sotware, pages 6-8.

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

BOEHM, B.W. (1981), "Software Engineering Economics", Prentice Hall, Inc., Englewoodcliffs, New Jersey, USA.

BRANDI, D.L. (1990), "Quality Measures in Design: finding problems before coding" in Software Engineering Notes, Vol. 15 No. 1, page 68.

CARD, D.W.; PAIGE, G.T. (1985), "Criteria for Software Modularization", in 8th.

International Conference on Software Engineering, London, UK.

CARD, D.N.; AGRESTI, W.W. (1988), "Measuring Software Design Complexity", in The Journal of Systems and Software, n.8, pages 185-197.

CARD, D.N.; GLASS, R.L. (1990), "Measuring Software Desing Quality", Prentice-Hall.

CHO, C.K. (1980), "An Introduction to Software Quality Control", Joan Wiley & Sons Inc., New York.

CLUNIE BEAUFOND, C.E. (1987), "Verificação e Validação de Software na fase de Especificação de Requisitos", Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro.

CONTE, S.D.; DUNSMORE, H.E.; SHEN, V.Y. (1986), "Software Engineering Metrics and Models", The Benjamim/Cummings Publishing Company, Inc. California, USA.

CURTIS, B. (1981), "The Measurement of software Quality and Complexity", in Software Metrics: An Analysis and Evaluation, Cambridge, MIT PRESS.

DENICOFF, M.; GRAFTON, R. (1981), "Software Metrics: A Research Iniciative", in

Software Metrics: An Analysis and Evaluation, Cambridge, Mit Press.

DUNN, R.H.; ULLMAN, R. (1982), "Quality Assurance for Computer Software", Mc Graw-Hill, New York.

EVANS, M.W.; MARCIANIAK, J.J. (1987), "Software Quality Assurance and Management", A Wiley-Interscience Publication.

FAGAN, M.E. (1976), " Design and code inspections to reduce errors in program development", in IBM Systems Journal, vol.15, n.3, pages 219-248.

FAIRLEY, Richard (1985), "Software Engineering Concepts", McGraw-Hill Inc, USA.

FREITAS, A.C.C.T. (1985a), "Software Empresarial: Qualidade ao longo do desenvolvimento", Tese de Mestrado, IME, Rio de Janeiro.

FREITAS, A.C.C.T., et allii (1985b), "Characteisticas de qualidade de Programas", Relatório Técnico do Programa de Engenharia de Sistemas, ES-83/85, COPPE/UFRJ.

GANE, C.; SARSON,T (1983), "Análise Estrutuada de Sistemas", Livros Técnicos e Cientificos Editora.

HAUSEN, H.L.; MULLERBURG, M. (1984), "An introduction to quality assurance and control of software", in Software Validation, Elsevier Science Publishers B. V., North-Holland.

HENRY, S.; Selig, C. (1990), "Predicting Source-Code Complexity at the Design Stage" in IEEE Software, pp 33-44.

JENSEN, R.W.; TONIES, C.C., (1979) "Software Engineering", Prentice-Hall, New Jersey.

MATTOSO, A.L.Q. (1990), "TABA_OBJ: um ambiente de desenvolvimento de software com orientação a objetos", Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro.

MC CALL, J.; CAVANO, J. P. (1978), " A Framework for the Measurement of Software Quality", in Proceedings of the ACM Software Quality Assurance Workshop, San Diego.

MC CALL, J. A. (1979), "An introduction to Software Quality", in Software Quality Management, J.D. Cooper e M.J. Fisher (Ed.), Petrocelli.

MC CABE, T.J. (1976), "A Complexity Measure", in IEEE Transactions on Software Engineering, vol. SE 2, n.4.

MENEZES, I.M.G.; Rocha, A.R.C. da (1986), "Como produzir documentação de boa qualidade", Relatório Técnico do Programa de Engenharia de Sistemas, COPPE/UFRJ, Rio de Janeiro.

MYERS, G.J. (1978) "Composite/Structured Design", Van Nostrand, 1978.

NASCIMENTO, E.M.; ROCHA, A.R.C. da (1986), "Manual para Controle da Qualidade de Programa: sub-fator modularidade", Relatório Técnico do Programa de Engenharia de Sistemas, ES-112/86, COPPE/UFRJ.

NOGUEIRA, D.L. (1988), "Ferramentas Automatizadas para apoio ao projeto estruturado", Tese de Mestrado, COPPE/UFRJ.

NOVELINO, C. (1989), "Qualidade: uma questão em foco", in Boletim do CREA-RJ, abril/89.

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

PALERMO, S.; ROCHA, A. R . C. da (1988) "A Proposal to Evaluate Program Quality For The Delphi Off-Line Environment", Relatório Técnico do Programa de Engenharia de Sistemas e Computação, ES-180/88, COPPE/UFRJ.

PARNAS, D.L. (1972), "On the Criteria to be Used on Decomposing Systems into Modules", in CACM, vol.5, n.12.

PARNAS, D.L.; WEISS, D.M. (1985), "Active Design Reviews: Principle and Practices", in 8th. International Conference on Software Engineering; London, UK.

PRESSMAN, Roger S. (1988), "Software Engineering: a Practioner's Approach", McGraw-Hill.

ROCHA, A.R.C. da (1983), "Um Modelo para Avaliação da Qualidade de Especificações", Tese de Doutorado, PUC/RJ.

ROCHA, A.R.C.; Nogueira, D; Menezes, M.G.; Blaschek, J.R.S.; Matoso, M.L.Q. e Aguiar, T.C. (1986), "Um Conjunto de Ferramentas Automatizadas de Apoio ao Software", in VI SEMICRO, Rio de Janeiro.

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

ROCHA, A.R.C.; Passos, M.C.J.F. (1990), "Critérios para avaliação de software para pecuária do leite", Relatório Técnico do Programa de Engenharia de Sistemas e Computação, ES-223/90, COPPE/UFRJ.

SCHNEIDERMAN, B. (1980), "Software Psychology", Winthrop Publishers.

SOMMERVILLE, I. (1982), Software Engineering, London, Addison-Wesley.

STAA, A.v. (1983), Engenharia de Programas, LCT.

STAA, A. v. (1987), "Em busca de uma definição de coesão", in VII Congresso da Sociedade Brasileira de Computação, Salvador/BA.

STAHL, M.M. (1988), "Qualidade de Programas: Avaliação do Estilo de Prgramação", Relatório Técnico do Programa de Engenharia de Sistemas e Computação, ES-163/88, COPPE/UFRJ.

STEVENS, W.P.; MYERS, G.J.; CONSTANTINE, L.L. (1974), "Structured Design", in Software Design Strategies; IEEE Computer Society.

STEVENS, M.P. (1986), "Projeto Estruturado de Sistemas", Editora Campus, Rio de Janeiro.

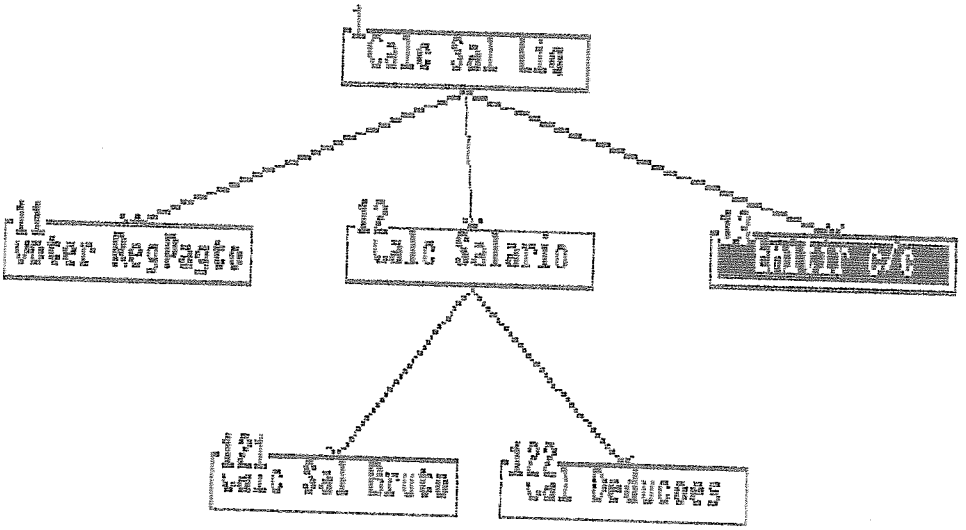
WASSERMAN, A. (1982), "Automated Tools in the Information System Development Environment", IFIP.

WERNECK, V.M.B. (1990), "Taxonomia de Domínios de Alicação", Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro.

WINDOWS (1987), Microsoft Windows Software Development Kit: Application Style Guide. Microsoft Inc., Version 2.0.

YOURDON, E.; CONSTANTINE, L.L. (1979), "Structured Design", Prentice-Hall.

ANEXO I


1
Calc Sal Liq
11
Veter RegPagto
12
Calc Salario
13
Salida CA
121
Calc Sal Bruto
122
Cal Deducoes

RELATORIO DE DESCRICAO DO MODULO

13/5/1991

Numero: 1

Modulo: Calc Sal Liq

Descricao: Calcular o salario liquido dos funcionarios.

Interfaces de entrada:
reg-func
s-liq-hora
s-liq-mens

Interfaces de saida:
reg-func

Numero: 11

Modulo: Obter RegPagto

Descricao: Obter o registro de pagamento dos funcionarios.

Interfaces de entrada:

Interfaces de saida:
reg-func

Numero: 12

Modulo: Calc Salario

Descricao: Calcular o salario liquido dos funcionarios horistas e mensalistas.

Interfaces de entrada:
reg-func

Interfaces de saida:
s-liq-hora
s-liq-mens

Numero: 121

Modulo: Calc Sal Bruto

Descricao: Calcular salario bruto de horistas e mensalistas.

Interfaces de entrada:
horas-tra
p-bonific
p-mens

Interfaces de saida:
p-bto-hora
p-bto-mens
*tipo-func

Numero: 122

Modulo: Cal Deducoes

Descricao: Calcular deducoes normais.

Interfaces de entrada:
p-bto-hora
p-bto-mens
*tipo-func

Interfaces de saida:
deducoes

Numero: 13

Modulo: Emitir c/c

Descricao: Emitir contra-cheques para funcionarios horistas e mensalistas.

Interfaces de entrada:
deducoes
p-bto-hora
p-bto-mens
reg-func
s-liq-hora
s-liq-mens
*tipo-func

Interfaces de saida:

RELATORIO DE AVALIACAO DA MODULARIDADE

13/5/1991

Nome do Projeto: Folha de Pagamento de Funcionarios Horistas e Mensalistas

Numero de Modulos: 6
Opcao de Avaliacao: local
Criterios Avaliados: Acoplamento
Coesao
Fan-in
Fan-out

Resultados da Avaliacao

Modulo: Emitir c/c Numero: 13 Tipo: Modulo

Criterio ACOPLAMENTO

Processo de Avaliacao: identifica-se o tipo de acoplamento de cada modulo.

Resultado:

Entrada: deducoes
p-bto-hora
p-bto-mens
reg-func
s-liq-hora
s-liq-mens
*tipo-func

-> O modulo nao possui saida

Acoplamento por controle.
Este tipo de acoplamento deve ser evitado ao maximo,pois destina-se a controlar a logica interna do modulo subordinado. Pior se o controle e passado do subordinado ao superior, acarretando inversao de autoridade.

Criterio COESAO

Processo de Avaliacao: identifica-se o tipo de coesao do modulo.

Resultado:

Descricao do modulo: Emitir contra-cheques para funcionarios horistas e mens

Coesao logica.
Modulos com coesao logica devem ser evitados ao maximo, pois as atividades a serem executadas pelo modulo sao selecionadas fora deste, por intermedio de variaveis de controle.

Criterio FAN-IN
==========

Processo de Avaliacao: conta-se o numero de modulos superiores de cada modulo. (limite maximo 3 superiores)

Resultado: 1 modulo(s) superior(es): Calc Sal Liq

Criterio satisfeito

Criterio FAN-OUT
==========

Processo de Avaliacao: conta-se o numero de modulos subordinados de cada modulo. (limite maximo 9 subordinados)

Resultado: nao tem modulos subordinados

Criterio satisfeito.

RELATORIO DE AVALIACAO DA MODULARIDADE

13/5/1991

Nome do Projeto: Folha de Pagamento de Funcionarios Horistas e Mensalistas

Numero de Modulos: 6
Opcao de Avaliacao: global
Criterios Avaliados: Acoplamento

Resultados da Avaliacao

Modulo: Calc Sal Liq Numero: 1 Tipo: Modulo

Criterio ACOPLAMENTO

Processo de Avaliacao: identifica-se o tipo de acoplamento de cada modulo.

Resultado:

Entrada: reg-func, s-liq-hora, s-liq-mens Saida: reg-func

Acoplamento por estrutura de dados.
Criterio satisfeito.

Modulo: Obter RegPagto Numero: 11 Tipo: Modulo

Criterio ACOPLAMENTO

Processo de Avaliacao: identifica-se o tipo de acoplamento de cada modulo.

Resultado:

Saida: reg-func

-> O modulo nao possui entrada

Acoplamento por estrutura de dados.
Criterio satisfeito.

Modulo: Calc Salario Numero: 12 Tipo: Modulo

Criterio ACOPLAMENTO

Processo de Avaliacao: identifica-se o tipo de acoplamento de cada modulo.

Resultado:

Entrada: reg-func Saida: s-liq-hora
s-liq-mens

Acoplamento por estrutura de dados.
Criterio satisfeito.

Modulo: Calc Sal Bruto Numero: 121 Tipo: Modulo

Criterio ACOPLAMENTO

Processo de Avaliacao: identifica-se o tipo de acoplamento de cada modulo.

Resultado:

Entrada: horas-tra
p-bonific
p-mens

Saida: p-bto-hora
p-bto-mens
*tipo-func

Acoplamento por controle.
Este tipo de acoplamento deve ser evitado ao maximo,pois destina-se a controlar a logica interna do modulo subordinado. Pior se o controle e passado do subordinado ao superior, acarretando inversao de autoridade.

Modulo: Cal Deducoes Numero: 122 Tipo: Modulo

Criterio ACOPLAMENTO

Processo de Avaliacao: identifica-se o tipo de acoplamento de cada modulo.

Resultado:

Entrada: p-bto-hora
p-bto-mens
*tipo-func

Saida: deducoes

Acoplamento por controle.
Este tipo de acoplamento deve ser evitado ao maximo,pois destina-se a controlar a logica interna do modulo subordinado. Pior se o controle e passado do subordinado ao superior, acarretando inversao de autoridade.

Modulo: Emitir c/c Numero: 13 Tipo: Modulo

Criterio ACOPLAMENTO

Processo de Avaliacao: identifica-se o tipo de acoplamento de cada modulo.

Resultado:

Entrada: deducoes
p-bto-hora
p-bto-mens
reg-func
s-liq-hora
s-liq-mens
*tipo-func

-> O modulo nao possui saida

Acoplamento por controle.
Este tipo de acoplamento deve ser evitado ao maximo,pois destina-se a controlar a logica interna do modulo subordinado. Pior se o controle e passado do subordinado ao superior, acarretando inversao de autoridade.

RELATORIO DE AVALIACAO DA MODULARIDADE

13/5/1991

Nome do Projeto: Folha de Pagamento de Funcionarios Horistas e Mensalistas

Numero de Modulos: 6
Opcao de Avaliacao: global
Criterios Avaliados: Coesao

Resultados da Avaliacao

Modulo: Calc Sal Liq Numero: 1 Tipo: Modulo

Criterio COESAO

Processo de Avaliacao: identifica-se o tipo de coesao do modulo.

Resultado:

Descricao do modulo: Calcular o salario liquido dos funcionarios.

Coesao funcional.
Criterio satisfeito.

Modulo: Obter RegPagto Numero: 11 Tipo: Modulo

Criterio COESAO

Processo de Avaliacao: identifica-se o tipo de coesao do modulo.

Resultado:

Descricao do modulo: Obter o registro de pagamento dos funcionarios.

Coesao funcional.
Criterio satisfeito.

==========
Modulo: Calc Salario Numero: 12 Tipo: Modulo
==========

Criterio COESAO
==========

Processo de Avaliacao: identifica-se o tipo de coesao do modulo.

Resultado:

Descricao do modulo: Calcular o salario liquido dos funcionarios horistas e mens

Coesao procedural,nivel medio de coesao.
As manutencoes nao sao tao simples, pois os elementos do modulo estao envolvidos em atividades diferentes e possivelmente nao relacionadas, nas quais o controle flui de uma atividade para a outra. Este tipo de modulo tende a fazer apenas parte de uma tarefa.

==========
Modulo: Calc Sal Bruto Numero: 121 Tipo: Modulo
==========

Criterio COESAO
==========

Processo de Avaliacao: identifica-se o tipo de coesao do modulo.

Resultado:

Descricao do modulo: Calcular salario bruto de horistas e mensalistas.

Coesao procedural,nivel medio de coesao.
As manutencoes nao sao tao simples, pois os elementos do modulo estao envolvidos em atividades diferentes e possivelmente nao relacionadas, nas quais o controle flui de uma atividade para a outra. Este tipo de modulo tende a fazer apenas parte de uma tarefa.

Modulo: Cal Deducoes Numero: 122 Tipo: Modulo

Criterio COESAO

Processo de Avaliacao: identifica-se o tipo de coesao do modulo.

Resultado:

Descricao do modulo: Calcular deducoes normais.

Coesao logica.
Modulos com coesao logica devem ser evitados ao maximo, pois as atividades a serem executadas pelo modulo sao selecionadas fora deste, por intermedio de variaveis de controle.

Modulo: Emitir c/c Numero: 13 Tipo: Modulo

Criterio COESAO

Processo de Avaliacao: identifica-se o tipo de coesao do modulo.

Resultado:

Descricao do modulo: Emitir contra-cheques para funcionarios horistas e mens

Coesao logica.
Modulos com coesao logica devem ser evitados ao maximo, pois as atividades a serem executadas pelo modulo sao selecionadas fora deste, por intermedio de variaveis de controle.

RELATORIO DE AVALIACAO DA MODULARIDADE

13/5/1991

Nome do Projeto: Folha de Pagamento de Funcionarios Horistas e Mensalistas

Numero de Modulos: 6
Opcao de Avaliacao: global
Criterios Avaliados: Fan-in

Resultados da Avaliacao

Modulo: Calc Sal Liq Numero: 1 Tipo: Modulo

Criterio FAN-IN

Processo de Avaliacao: conta-se o numero de modulos superiores de cada modulo. (limite maximo 3 superiores)

Resultado: nao tem modulos superiores

Criterio satisfeito

Modulo: Obter RegPagto Numero: 11 Tipo: Modulo

Criterio FAN-IN

Processo de Avaliacao: conta-se o numero de modulos superiores de cada modulo. (limite maximo 3 superiores)

Resultado: 1 modulo(s) superior(es): Calc Sal Liq

Criterio satisfeito

Modulo: Calc Salario Numero: 12 Tipo: Modulo

Criterio FAN-IN

Processo de Avaliacao: conta-se o numero de modulos superiores de cada modulo. (limite maximo 3 superiores)

Resultado: 1 modulo(s) superior(es): Calc Sal Liq

Criterio satisfeito

Modulo: Calc Sal Bruto Numero: 121 Tipo: Modulo

Criterio FAN-IN

Processo de Avaliacao: conta-se o numero de modulos superiores de cada modulo. (limite maximo 3 superiores)

Resultado: 1 modulo(s) superior(es): Calc Salario

Criterio satisfeito

Modulo: Cal Deducoes Numero: 122 Tipo: Modulo

Criterio FAN-IN

Processo de Avaliacao: conta-se o numero de modulos superiores de cada modulo. (limite maximo 3 superiores)

Resultado: 1 modulo(s) superior(es): Calc Salario

Criterio satisfeito

Modulo: Emitir c/c Numero: 13 Tipo: Modulo

Criterio FAN-IN

Processo de Avaliacao: conta-se o numero de modulos superiores de cada modulo. (limite maximo 3 superiores)

Resultado: 1 modulo(s) superior(es): Calc Sal Liq

Criterio satisfeito

RELATORIO DE AVALIACAO DA MODULARIDADE

13/5/1991

Nome do Projeto: Folha de Pagamento de Funcionarios Horistas e Mensalistas

Numero de Modulos: 6
Opcao de Avaliacao: global
Criterios Avaliados: Fan-out

Resultados da Avaliacao

Modulo: Calc Sal Liq Numero: 1 Tipo: Modulo

Criterio FAN-OUT

Processo de Avaliacao: conta-se o numero de modulos subordinados de cada modulo. (limite maximo 9 subordinados)

Resultado: 3 modulo(s) subordinado(s): Obter RegPagto
Calc Salario
Emitir c/c

Criterio satisfeito.

Modulo: Obter RegPagto Numero: 11 Tipo: Modulo

Criterio FAN-OUT

Processo de Avaliacao: conta-se o numero de modulos subordinados de cada modulo. (limite maximo 9 subordinados)

Resultado: nao tem modulos subordinados

Criterio satisfeito.

Modulo: Calc Salario Numero: 12 Tipo: Modulo

Criterio FAN-OUT

Processo de Avaliacao: conta-se o numero de modulos subordinados de cada modulo. (limite maximo 9 subordinados)

Resultado: 2 modulo(s) subordinado(s): Calc Sal Bruto
Cal Deducoes

Criterio satisfeito.

Modulo: Calc Sal Bruto Numero: 121 Tipo: Modulo

Criterio FAN-OUT

Processo de Avaliacao: conta-se o numero de modulos subordinados de cada modulo. (limite maximo 9 subordinados)

Resultado: nao tem modulos subordinados

Criterio satisfeito.

Modulo: Cal Deducoes Numero: 122 Tipo: Modulo

Criterio FAN-OUT

Processo de Avaliacao: conta-se o numero de modulos subordinados de cada modulo. (limite maximo 9 subordinados)

Resultado: nao tem modulos subordinados

Criterio satisfeito.

Modulo: Emitir c/c Numero: 13 Tipo: Modulo

Criterio FAN-OUT

Processo de Avaliacao: conta-se o numero de modulos subordinados de cada modulo. (limite maximo 9 subordinados)

Resultado: nao tem modulos subordinados

Criterio satisfeito.

RELATORIO DE AVALIACAO DA MODULARIDADE

13/5/1991

Nome do Projeto: Folha de Pagamento de Funcionarios Horistas e Mensalistas

Numero de Modulos: 6
Opcao de Avaliacao: global
Criterios Avaliados: Complexidade Total

Resultados da Avaliacao

Criterio COMPLEXIDADE TOTAL

Processo de Avaliacao:

Complexidade Total=(C. Local/n)+(C. Estrutural/n)

onde:
C. Local= somatorio[(no. de entradas e saidas/fanout+1) de cada modulo]
C. Estrutural= somatorio[(fanout*fanout) de cada modulo]
n= numero de modulos no Ge
(Complexidade total < 26 e o indicado)

Resultado:

Modulo	Numero	E/S	Acoplamento	Fan-out	CLoc.	CEstr.
Calc Sal Liq	1	4	Estrutura	3	1	9
Obter RegPagto	11	1	Estrutura	0	1	0
Calc Salario	12	3	Estrutura	2	1	4
Calc Sal Bruto	121	6	Conteudo	0	6	0
Cal Deducoes	122	4	Conteudo	0	4	0
Emitir c/c	13	7	Conteudo	0	7	0

C. Local= 3.33
C. Estrutural= 2.17
Complexidade Total= 5.5

Criterio satisfeito.