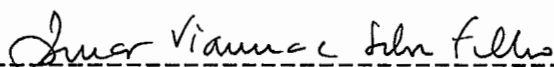


DEFINIÇÃO E IMPLEMENTAÇÃO DE UM SISTEMA ESPECIALISTA EM
MODULARIZAÇÃO DE PROJETO ESTRUTURADO DE SISTEMAS

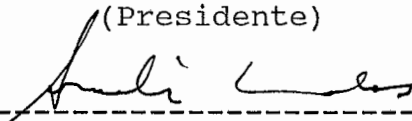
Claudia Lage Rebello da Motta

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.

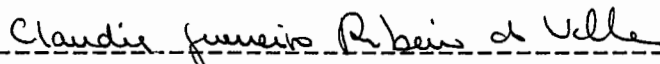
Aprovada por:



Prof. Ysmar Vianna e Silva Filho, Ph.D.
(Presidente)



Prof(a). Sueli Bandeira Teixeira Mendes, Ph.D.



Prof(a). Claudia Guerreiro Ribeiro do Valle, Doutora

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 1989

MOTTA, CLAUDIA LAGE REBELLO

Definição e Implementação de um Sistema Especialista em Modularização de Projeto Estruturado de Sistemas [Rio de Janeiro] 1989

IX, 121 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1989) Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Sistemas Especialistas 2. Inteligência Artificial I. COPPE/UFRJ II. Título (série).

Ao meu marido Ricardo,
ao meu filho Rodrigo
e a meus pais.

AGRADECIMENTOS

Ao meu marido Ricardo Gonçalves da Motta pelo incentivo, paciência e apoio.

Ao Prof. Ysmar Vianna e Silva Filho pela orientação, incentivo e apoio prestados no decorrer deste trabalho.

À Prof^ª Claudia Guerreiro Ribeiro do Valle pelas excelentes sugestões e enorme incentivo para a conclusão deste trabalho.

Ao amigo Antônio Anibal de Souza Teles pela sua contribuição, sem a qual não teria sido possível concluir este trabalho.

Enfim, aos membros da banca, que muito me honraram com sua participação e aos amigos do NCE pelo apoio e colaboração prestados.

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

DEFINIÇÃO E IMPLEMENTAÇÃO DE UM SISTEMA ESPECIALISTA EM
MODULARIZAÇÃO DE PROJETO ESTRUTURADO DE SISTEMAS

Claudia Lage Rebello da Motta

Dezembro, 1989

Orientador: Prof. Ysmar Vianna e Silva Filho, Ph.D.

Programa: Engenharia de Sistemas e Computação

Este trabalho apresenta uma análise das fases de desenvolvimento de um Sistema Especialista em Modularização de Projeto de Sistemas usando metodologias estruturadas.

Apresenta-se inicialmente uma visão geral da Inteligência Artificial e da construção de Sistemas Especialistas. Em seguida, aborda-se a Engenharia de Software e as aplicações da Inteligência Artificial à mesma, explicando como este trabalho pode ser útil ao projetista quando utilizado em conjunto com ferramentas CASE (Computer Aided Software Engineering).

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

DEFINITION AND IMPLEMENTATION OF AN EXPERT SYSTEM IN
MODULARIZATION OF STRUCTURED SYSTEMS DESIGN

Claudia Lage Rebello da Motta

December, 1989

Thesis Supervisor: Ysmar Vianna e Silva Filho, Ph.D.

Department: Systems and Computation Engineering

This work presents an analysis of the development of an Expert System in Modularization of Systems Design using structured methodologies.

Firstly it is shown a general overview of the Artificial Intelligence area and the construction of Expert Systems. Then, an introduction to Software Engineering and the applications of the Artificial Intelligence on that are presented, explaining how this work can be useful to the designer when it is used along with CASE (Computer Aided Software Engineering) tools.

índice

I. Introdução.....	1
I.1. Objetivos do trabalho.....	1
I.2. Composição do trabalho.....	2
I.3. Introdução.....	3
I.3.1. Inteligência Artificial.....	3
I.3.2. Sistemas Especialistas.....	8
I.3.2.1. Sistemas Especialistas x Sistemas Tradicionais.....	10
I.3.2.2. O Engenheiro do Conhecimento...	12
I.3.2.3. Fases de Desenvolvimento.....	13
I.3.2.4. Formas de Representar o Conheci- mento.....	17
I.3.2.5. Classificação.....	23
I.3.2.6. Linguagens e Ambientes de Desen- volvimento.....	24
I.3.3. Engenharia de Software.....	26
I.3.3.1. Inteligência Artificial aplicada à Engenharia de Software.....	28

II. Apresentação do Problema.....	32
II.1. Projeto Estruturado de Sistemas.....	32
II.1.1. Qualidades de um Bom Projeto.....	33
II.1.2. Diagrama Hierárquico de Módulos.....	34
II.1.3. Conceitos Básicos.....	37
II.1.3.1. Coesão.....	38
II.1.3.2. Acoplamento.....	46
II.1.3.3. Diretrizes Adicionais.....	52
II.1.4. Estações de Trabalho CASE.....	59
II.1.5 Explicação do Sistema Especialista WALK como assistente do projetista.....	65
III. Apresentação da Solução.....	68
III.1. Etapas de Desenvolvimento.....	68
III.2. Codificação das Heurísticas.....	71
III.3. Estratégias de Solução.....	79
III.4. Implementação da Solução.....	85
III.4.1. Expert System Environment (ESE/IBM). 85	
III.4.2. Representação no ESE.....	91
III.4.3. Interação com Usuário.....	97
III.4.3.1. Representação do DHM e do DD.....	97
III.4.4. Melhorando a Performance e Heurísticas para Soluções.....	99
III.4.5. Exemplo.....	100

IV.Considerações Finais e Conclusões.....	111
IV.1. Avaliação e Conclusões.....	111
IV.2. Sugestões para Extensão.....	114
V. Referências Bibliograficas.....	117

CAPÍTULO I

INTRODUÇÃO

I.1. Objetivos do trabalho

O objetivo principal deste trabalho é o desenvolvimento de um Sistema Especialista em Modularização de Projetos Estruturados de Sistemas. Entretanto, um esclarecimento sobre os obstáculos encontrados no desenvolvimento deste trabalho faz-se necessário. Primeiro, em 1986, quando essa pesquisa foi iniciada, poucos grupos de Inteligência Artificial dominavam as técnicas para desenvolvimento de Sistemas Especialistas. Segundo, raro era o grupo que já tivesse tido contato direto com um ambiente para desenvolvimento de Sistemas Especialistas (SHELL). E, por último, a Modularização de Projetos Estruturados estava começando a ser divulgada entre os profissionais da área. Com isso, ao invés de um, tinham-se três objetivos a serem alcançados: aprender a construir Sistemas Especialistas, estudar e aprender a utilizar um SHELL e formular heurísticas para tratar do problema escolhido.

Através de heurísticas obtidas da literatura especializada, o WALK (como foi chamado este Sistema Especialista) verifica, utilizando a descrição do Diagrama Hierárquico de Módulos e o Dicionário de Dados, se o

projeto do novo sistema está dentro dos padrões propostos pelas metodologias de projeto estruturado, se está consistente e, se pode ser melhorado, apresentando ao final da consulta um diagnóstico dos principais problemas encontrados, com suas respectivas sugestões de solução.

Por fim, é importante citar que este trabalho é inédito nesta área de Projeto Estruturado de Sistemas.

I.2. Composição do trabalho

Este trabalho apresenta as fases de desenvolvimento de um Sistema Especialista em Modularização de Projetos de Sistemas usando metodologias estruturadas. Descreve-se aqui, desde a fase de definição do problema até a implementação de um protótipo usando um ambiente próprio para desenvolvimento de Sistemas Especialistas.

Inicialmente, procura-se apresentar um breve histórico sobre o surgimento da Inteligência Artificial e suas aplicações. Em seguida, focaliza-se os Sistemas Especialistas descrevendo suas características, processos de desenvolvimento, utilização, etc. Ainda neste capítulo, fala-se sobre a Engenharia de Software e da utilização de técnicas de Inteligência Artificial aplicadas à mesma.

No capítulo II, detém-se no Projeto Estruturado de Sistemas, descrevendo seus conceitos básicos e características. Procura-se dar uma abordagem geral das estações de trabalho CASE (Computer Aid Software Engineering), e por fim, explica-se como este trabalho pode auxiliar um projetista na realização de sua tarefa.

No capítulo III, propõe-se uma solução utilizando-se um ambiente para desenvolvimento de Sistemas Especialistas (SHELL) da IBM, o Expert System Environment (ESE/VM). Descreve-se sumariamente a ferramenta e, em seguida detalha-se o processo de codificação das heurísticas, estratégia de solução adotada e da implementação em si.

No capítulo IV, por fim, avalia-se os resultados obtidos e sugere-se estudos para extensão do trabalho.

I.3. Introdução

I.3.1. Inteligência Artificial

"Desde que os gregos inventaram a lógica e a geometria, a idéia de que todo raciocínio poderia ser reduzido a algum tipo de cálculo - de modo que todos os argumentos pudessem ser expostos de uma vez por todas - tem fascinado a maioria dos pensadores radicais da tradição ocidental. Sócrates foi o primeiro a dar voz a

essa visão. A história da Inteligência Artificial bem poderia ter início em torno de 450 a.C. quando (segundo Platão) Sócrates exige de Euthyphro, um colega ateniense que, em nome da piedade, está para entregar o próprio pai, acusado de homicídio : - Quero saber que característica da piedade é essa que torna as ações pias ... para que eu possa voltar-me para ela e usá-la como um padrão pelo qual julgue suas ações e a de outros homens. Sócrates está indagando de Euthyphro o que os teóricos modernos da computação chamariam de "procedimento útil" - um conjunto de regras que nos diz exatamente, de momento a momento, como nos comportarmos".

Este fascínio pode ser observado, por exemplo, quando grupos de cientistas americanos e ingleses estavam trabalhando, ao final da II Guerra Mundial, para construir uma Máquina Eletrônica que pudesse resolver cálculos complexos comandada por programas armazenados. Alain Turing, um dos principais cientistas Britânicos da época, pensou em uma Máquina com proposições gerais, comandada através de operadores lógicos do tipo "NOT", "AND" e "OR". Para ele uma Máquina assim, além de poder usar tais operadores na construção de operadores numéricos mais específicos para cálculos aritméticos, seria capaz de manipular qualquer tipo de simbologia.

Entretanto, a praticidade dos cientistas americanos mostrou que seria mais econômico e viável construir Máquinas que usassem operadores aritméticos, tais como :

"+", "-" e ">", pois, estavam seguros que tais Máquinas seriam construídas para processar somente cálculos aritméticos.

"...Assim, enquanto homens práticos como Eckert e Mauchly, na Universidade da Pensilvânia, projetavam a primeira Máquina eletrônica digital, teóricos tais como A.M. Turing, tentando apreender a essência e a capacidade de tais Máquinas, interessavam-se por uma área que, até então, fora por muito tempo a província dos filósofos : a natureza da própria razão" DREYFUS [1].

Essa decisão acabou sendo seguida também pela Inglaterra, pois essa idéia pareceu mais razoável para a maioria das pessoas envolvidas na questão. Contudo, um pequeno grupo de cientistas continuou a explorar a habilidade dos computadores em manipular símbolos não numéricos. Simultaneamente, psicólogos envolvidos com a solução de problemas humanos procuravam desenvolver programas que simulassem o comportamento humano.

"Com os computadores digitais a solucionar problemas do tipo como conseguir atravessar um rio com três canibais e três missionários sem que os canibais devorem os missionários, pareceu que, finalmente, a ambição filosófica tinha encontrado a necessária tecnologia: que o computador universal, de alta velocidade, tinha recebido as regras para converter o raciocínio em cálculo " DREYFUS [1].

Ao passar dos anos, pessoas interessadas tanto em processamento simbólico quanto em solução de problemas humanos, formaram um subcampo interdisciplinar na ciência da computação chamado Inteligência Artificial (IA). Pesquisadores em IA estão procurando desenvolver sistemas que produzam resultados que normalmente associaríamos à Inteligência humana. Aqui, Inteligência não significa especificamente as capacidades de inovação e criatividade dos seres humanos mas, neste caso principalmente, a aquisição, transformação e aplicação do conhecimento.

Várias são as áreas que compõem a Inteligência Artificial. Entre outras, temos: Sistemas Especialistas, Processamento em Linguagem Natural, Sistemas Conversacionais, Prova Automática de Teoremas, Robótica, Jogos, Visão por Máquina, Programação Automática, sendo que cada uma dessas áreas está em diferente estágio de desenvolvimento.

Atualmente, a Inteligência Artificial anda em evidência, embora nem sempre as perspectivas tenham sido tão otimistas. Logo no início, quando companhias formaram grupos de IA para desenvolver aplicações práticas, os esforços não foram bem sucedidos porque os programas eram custosos, muito lentos e não produziam resultados satisfatórios. Os programas de IA eram simplesmente muito complexos para serem executados nos computadores existentes na época. No entanto, o desenvolvimento da tecnologia da microeletrônica resultou numa nova geração de circuitos, mais rápidos, mais poderosos e em computadores relativamente baratos e a IA ressurgiu dos laboratórios. O hardware dos computadores atuais combinado com os significativos avanços teóricos da área, tem resultado numa tecnologia que vem se firmando ao longo do tempo. Outros fatores que contribuíram para a evolução e transformações destes últimos anos foram os Sistemas Especialistas e o projeto de "Computadores de Quinta Geração" anunciado pelos japoneses.

No Brasil, as pesquisas em IA vinham sendo feitas por uns poucos pesquisadores, mas, de uns tempos para cá estas pesquisas se intensificaram e os grupos interessados aumentaram razoavelmente. Debates, encontros, cursos e congressos vêm sendo organizados a fim de proporcionar a troca de experiência entre os grupos, divulgando pesquisas e concretizando convênios. Há quem diga que estamos defasados dos mais desenvolvidos nesta área de uns 3 a 5 anos. Independente deste fato, o

importante é saber que, de um modo ou de outro, estão todos começando e que esta é a hora de investir e acreditar em nossas pesquisas.

I.3.2. Sistemas Especialistas

Os Sistemas Especialistas (SE) são programas que procuram simular o raciocínio de um especialista humano na sua área de atuação. Eles atuam, preferencialmente, em uma área restrita focalizando um problema de cada vez. Seu domínio deve ter um tamanho e complexidade tal que seja possível sua implementação e, ao mesmo tempo, justifique seu uso [1,2,4,5,6,11].

Os SE são compostos basicamente por:

- .Base do Conhecimento;
- .Máquina de Inferência;
- .Interface de Explicação.

A Base de Conhecimento é formada por informações sobre um domínio específico (fatos e heurísticas) representados de uma forma estruturada (regras, frames, redes semânticas, etc). O desempenho de um SE está diretamente relacionado a extensão e qualidade da sua Base de Conhecimento.

A Máquina de Inferência informa de que maneira as regras deverão ser selecionadas e ativadas, através de inferências e estratégias de controle. Ela pode ser desenhada e implementada pelo engenheiro do conhecimento de acordo com a aplicação, ou pode estar embutida na própria linguagem utilizada na construção do SE. Neste caso, o desenvolvimento do mesmo demanda menos tempo e esforço mas, o engenheiro tem menos opções quanto à organização do conhecimento e à maneira de acessá-lo e nem todas as aplicações se adequam ao mecanismo de controle existente.

As estratégias de controle mais utilizadas pela Máquina de Inferência são:

- Encadeamento Reverso ou Dirigido por Hipóteses (Backward Chaining): Parte das soluções que se deseja provar, pesquisando fatos envolvidos na conclusão das mesmas. É mais adequada para problemas cujas soluções possíveis são poucas e conhecidas;

- Encadeamento Direto ou Dirigido por Dados (Forward Chaining): Busca soluções a partir dos dados iniciais do problema. É mais adequada para problemas que possuem muitas soluções possíveis.

Através da interface de explanação o SE pergunta ao usuário informações pendentes, explica a razão de tê-las perguntado e justifica suas conclusões. Como os sistemas são construídos adotando a terminologia relacionada com a área em questão, a interação com o usuário fica

facilitada. Esta interface é muito útil para a depuração do sistema pois permite que se acompanhe o raciocínio utilizado durante a execução.

Além de serem altamente interativos, os Sistemas Especialistas muitas vezes trabalham com o raciocínio aproximado utilizando cálculos probalísticos. Deste modo, funcionam mesmo quando seus dados de entrada estão incompletos ou incertos. Isto se dá associando-se fatores de certeza a seu conhecimento e manipulando-os durante o processo de raciocínio. Como resultado pode-se obter uma ou mais possíveis soluções com suas respectivas probabilidades.

I.3.2.1. Sistemas Especialistas X Sistemas Tradicionais

Os Sistemas Especialistas oferecem novas possibilidades no uso de computadores para a resolução de problemas, entretanto, não pretendem substituir os Sistemas Convencionais e sim complementá-los uma vez que suas formas e campos de atuação são bem distintos.

Os Sistemas Convencionais são aplicáveis a problemas bem definidos e, por isto, podem se valer de algoritmos para a obtenção de soluções precisas, de forma eficaz. São orientados para o processamento numérico e para a manipulação de dados.

Os Sistemas Especialistas tentam resolver problemas complexos e inexatos cuja solução algorítmica seria praticamente inviável. Ao invés de algoritmos, eles se valem de heurísticas que não obrigatoriamente conduzem a uma resposta exata; na verdade, podem conduzir a diversas respostas, ou até mesmo a nenhuma. São sistemas orientados para o processamento simbólico e para a manipulação do conhecimento.

Sistemas Especialistas		Sistemas de Informação
processamento	altamente interativo e simbólico	sequencial por lotes e numérico
explicação a meio da execução	fácil	difícil
solução	heurística	algorítmica
armazenamento da informação	Banco de Conhecimento em uma memória de trabalho global	Banco de Dados numericamente endereçados
Dados a tempo de execução	poderão ser imprecisos ou desconhecidos	Todos devem ser conhecidos

Tabela I.1

Dois aspectos deverão ser avaliados antes de se empregar um Sistema Especialista : sua viabilidade e sua necessidade. Para que seja possível a construção de um SE, é necessário que se disponha de uma fonte de conhecimento significativo e consistente, ou seja, é necessário que existam especialistas - pessoas de um alto nível de conhecimento e experiência em uma determinada área - e que estes sejam capazes de explicar os métodos utilizados

na resolução dos problemas, permitindo que o conhecimento seja adquirido e armazenado no computador. Sendo possível, analisa-se sua necessidade por implicar em grande esforço e alto custo de desenvolvimento. Sua construção será mais justificada quando :

.os especialistas são insuficientes ou caros;

.é necessário garantir o acesso ao conhecimento em diferentes localidades;

.a saída de pessoal especializado leva à perda de conhecimento importante;

.o ambiente é hostil para o especialista, tornando conveniente a utilização de uma máquina.

Embora a tecnologia empregada na construção de sistemas especialistas seja ainda bastante limitada, eles vêm conquistando seu lugar no mercado, pois tornam acessível ao usuário um valioso conhecimento especializado.

I.3.2.2. Engenheiro do Conhecimento

Com o intuito de aproveitar a disponibilidade do profissional que apesar de experiente e altamente capacitado, não possuía conhecimento de técnicas computacionais, surgiu o engenheiro do conhecimento, responsável pela tarefa de aquisição e representação do conhecimento. Para tanto, ele leva em consideração vários

fatores, como: verificar se a base de conhecimento está consistente e completa, estabelecer heurísticas que retratem de algum modo o conhecimento adquirido do especialista, escolher a ferramenta que será utilizada para a implementação e verificar sua adequação, minimizar o número de interfaces com usuário e verificar a confiabilidade dos resultados.

O especialista é, sem dúvida, a fonte mais importante para a obtenção do conhecimento. O engenheiro do conhecimento procura apreender o seu modo de atuação apresentando-lhe problemas reais, pertinentes ao domínio em estudo. Esta forma de extração, embora ainda rudimentar, é mais eficiente que o simples questionamento sobre os métodos usados para a resolução de problemas, uma vez que o conhecimento do especialista se encontra principalmente a nível subconsciente.

I.3.2.3. Fases de Desenvolvimento

Ainda não se tem um método formal para o desenvolvimento de um SE. Entretanto, pode-se identificar algumas etapas de seu desenvolvimento com suas respectivas atividades, embora estas não obedeçam a uma ordem rígida e possam se repetir ou se sobrepor. São elas:

1. Identificação do Problema

Esta é a primeira fase, onde o engenheiro do conhecimento e o especialista escolhem o problema que irão trabalhar, delimitando-o e determinando seus aspectos relevantes. Feito isto, avaliam os recursos necessários, tais como : facilidades computacionais, disponibilidade de especialistas, softwares e tempo disponíveis. Procuram determinar quais os principais objetivos e intenções na construção de tal sistema.

A dificuldade maior que surge nesta fase é justamente a de identificar e delimitar o problema. Na maioria das vezes o primeiro problema delimitado é muito grande ou complexo demais. Para resolver esta questão focaliza-se num pequeno mas interessante subproblema manipulável pelo SE e implementa-se rotinas para resolvê-lo.

2. Conceituação ou Análise da Natureza do Conhecimento

Nesta fase o engenheiro do conhecimento e o especialista decidem quais conceitos, relações e mecanismos de controle serão necessários para descrever o domínio do problema a ser selecionado, e verifica-se se o conhecimento envolvido permite a utilização de um SE.

A maior dificuldade nesta etapa é saber até que nível de detalhe deve-se ir para que os conceitos chaves sejam adequadamente discriminados. Esta dúvida será esclarecida

quando a implementação for iniciada, pois ela dará forma e direcionará o processo conceitual.

Nas duas etapas acima descritas (Identificação e Análise) o engenheiro do conhecimento procura obter o "conhecimento bruto", através da literatura especializada, exemplos ou entrevistas com o especialista, onde a preocupação com detalhes, consistência e completude do domínio em questão é menor.

3. Formalização e/ou Representação do Conhecimento

Nesta fase, procura-se expressar os conceitos e relações chaves de alguma maneira formal e escolher uma estrutura para sua representação. A esta altura, o engenheiro do conhecimento já deverá ter uma idéia de qual ferramenta será utilizada.

Através da formalização, o "conhecimento bruto", um conjunto ainda confuso e complexo de informações toma forma e se torna mais claro. Os detalhes, a consistência e a completude do domínio passam a ter considerável relevância.

4. Prototipagem (implementação)

Esta fase depende diretamente da ferramenta escolhida, sendo que, ao transformar o conhecimento formalizado em uma representação computacional, verifica-se

se o desenvolvimento do sistema está sendo conduzido de maneira adequada, isto é, se as escolhas estão sendo convenientes.

5. Testes e Validação do Sistema

Diversos fatores sobre o modelo implementado são analisados, neste momento, tanto pelo engenheiro do conhecimento e o especialista como pelo usuário final. Alguns deles são : a consistência da base de conhecimento, a naturalidade na ordem das perguntas, a eficiência da interface de explanação, o tempo de resposta do sistema, a interação homem\máquina e a performance do sistema como um todo.

O protótipo a partir deste ponto será expandido até que se obtenha um sistema final com uma performance satisfatória.

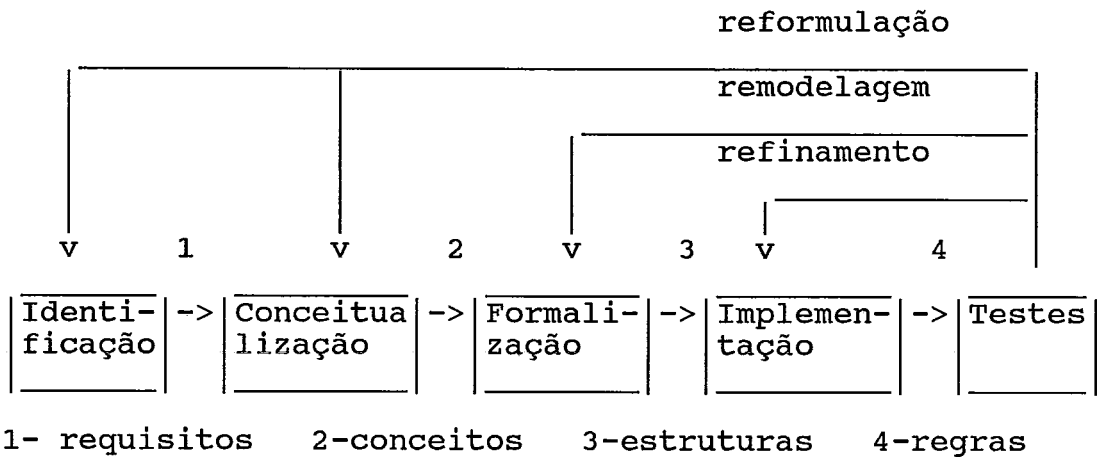


figura I.1 - Fases de desenvolvimento de um Sistema Especialista, WATERMAN [3]

A aquisição do conhecimento é sem dúvida , dentre as etapas descritas acima, um dos maiores "gargalos" no processo de construção de um Sistema Especialista [3,8,9]. Este problema aumenta quando se trata de aplicações onde o conhecimento se concentra em especialistas que (como ocorre na maioria das vezes) têm dificuldade em verbalizar sua forma de raciocínio de uma maneira estruturada e formal. Dentre outros motivos para tornar esta fase crítica estão também: a necessidade do engenheiro do conhecimento de se familiarizar com o domínio em questão, e os problemas de comunicação entre o engenheiro do conhecimento e o especialista.

I.3.2.4. Formas de Representação do Conhecimento

A representação é um conjunto de convenções sintáticas e semânticas que torna possível descrever as informações que obtemos sobre o mundo real WINSTON [10].

Os métodos de representação do conhecimento são necessários para modelar eficientemente o conhecimento e prepará-lo para ser acessado. A prática tem mostrado que projetar uma boa representação muitas vezes é a chave para transformar um difícil problema em algo claro e fácil de ser solucionado.

O ponto importante, ao se escolher uma representação para um problema em particular, é optar por aquela que

permita que todo o conhecimento necessário seja representado e que facilite a sua utilização de modo a resolver o problema apresentado.

Um dos maiores desafios na Inteligência Artificial vem sendo descobrir uma maneira de representar o raciocínio de senso comum.

LÓGICA DE PREDICADOS DE 1a. ORDEM

A unidade básica para a representação de fatos na Lógica de Predicados de 1a. Ordem é a fórmula lógica. Uma base de conhecimento então ficará sendo vista como um conjunto de fórmulas lógicas. Esta metodologia emprega noções de constantes, variáveis, funções, predicados, conectores lógicos e quantificadores.

Possuir regras de inferência com as quais se podem definir procedimentos de prova, ter uma semântica formal bem definida, ter uma notação simples e clara, e gerar regras compactas são as principais vantagens de se empregar este tipo de representação do conhecimento. Contudo, existem algumas desvantagens que devem ser levadas em consideração: a falta de princípios organizacionais para os fatos da base de conhecimento e a dificuldade na representação de conhecimentos heurísticos ou incertos.

REGRAS DE PRODUÇÃO

As Regras de Produção são expressas através de condições do tipo :

SE <condição>

ENTÃO <ação>

onde, caso a <condição> for satisfeita então a <ação> será realizada.

Este tipo de representação além de ser fácil de ser entendida e modificada, também é vista como uma das formas mais convenientes de se representar heurísticas e informações subjetivas.

As regras permitem uma construção modular da base de conhecimento sendo que cada regra expressa uma parte independente de conhecimento e poderá ou não ter associado a ela um determinado fator de certeza.

A representação de conhecimento através de Regras é a forma mais difundida, especialmente apropriada quando o domínio do conhecimento é empírico, ou seja, resultante de experiências na área.

REDES SEMÂNTICAS

Além de ser a forma de representação mais geral, a rede semântica é também a mais antiga na Inteligência

Artificial, tendo sido, originalmente, desenvolvida para simular o funcionamento da memória humana a partir de modelos psicológicos.

Os elementos básicos de formação das redes semânticas são os nós e as arestas. Os nós simbolizam objetos físicos ou conceituais, e as arestas relacionam os objetos e suas descrições.

A flexibilidade é a maior vantagem desse esquema representacional onde novos nós e arestas podem ser definidos sempre que necessário. A hereditariedade é outra vantagem, onde um nó pode herdar uma característica de outro nó com o qual esteja relacionado. Sendo assim, instâncias de classes herdam propriedades de classes mais gerais das quais elas são membros. Isto facilita a representação de informações redundantes, economizando espaço. Por outro lado, dificulta a manipulação de exceções. É especialmente útil em domínios que se utilizam de métodos classificatórios para simplificar a resolução de problemas.

FRAMES

Os frames foram derivados da rede semântica. Cada frame define um objeto como uma coleção de atributos e seus valores. Os atributos são chamados "slots" sendo que cada um deles pode conter: valores atribuídos durante a execução, valores default, ponteiros para outros frames,

ponteiros para procedimentos ou conjunto de regras que calcule o valor do slot. Os frames podem ser vistos como uma estrutura de dados que contém informações declarativas e procedimentais com relações internas definidas.

Os frames fornecem métodos organizacionais que permitem facilitar a recuperação das informações e o processo de inferência, entretanto são inadequados para a representação de conhecimento lógico e heurístico HSU[11].

Frames são mais complexos, mas permitem representar o conhecimento de uma maneira mais completa. São especialmente úteis em domínios onde a forma e o conteúdo dos dados são significativos na resolução de problemas como, por exemplo, a interpretação visual.

OBJETO-ATRIBUTO-VALOR

As triplas O-A-V, assim como os Frames, são um caso especial da Rede Semântica. Este tipo de representação baseia-se em triplas formadas por objetos, atributos e valores. Os objetos são entidades físicas ou conceituais. Os atributos são geralmente características ou propriedades associadas aos objetos. Os valores definem o valor de um atributo em determinada situação.

A dupla Objeto-Atributo representa o conhecimento estático enquanto que o Valor representa o conhecimento dinâmico.

Os objetos podem ser representados sob a forma de árvore, definindo uma ordenação e um relacionamento entre eles. A raiz (objeto que está no topo da árvore) será usada como ponto de partida do processo de resolução do problema, e, conseqüentemente, da aquisição de informações.

As triplas possuem a característica da hereditariedade e podem estar associadas a fatores de certeza.

CÁLCULO DE PREDICADOS

O Cálculo de Predicados se vale da lógica para representar o conhecimento.

Predicados são declarações sobre objetos e só podem receber valores falsos ou verdadeiros. Um predicado pode envolver mais de um objeto e pode se ligar a outros predicados através de conectivos (e, ou, não, implica, equivalente) para formar expressões maiores.

A Base do Conhecimento neste tipo de representação é formada pelo conjunto dos predicados verdadeiros.

Uma vez que as únicas respostas possíveis são verdadeiro ou falso, as consultas não podem ser genéricas. Para se obter as características de um objeto,

por exemplo, deve-se perguntar para cada característica possível, se a mesma é verdadeira ou falsa.

O Cálculo de Predicados é mais útil na representação formal de um domínio do conhecimento, uma vez que é capaz de traduzir em sentenças de sintaxe e semântica bem definidas, situações da vida cotidiana.

I.3.2.5. Classificação

Os SEs podem ser classificados de acordo com a área que se destinam ou através da atividade básica que desempenham. Agrupando os SEs de acordo com suas tarefas obtém-se as seguintes categorias, WATERMAN [3]:

Interpretação : inferem descrições de situações a partir de dados de sensores. Trabalham diretamente com dados reais ao invés de representações simbólicas do problema.

Predição ou Prognóstico : inferem possibilidades de situações dadas, isto é, prováveis conseqüências para uma dada situação, permitindo uma visão teórica do desfecho das mesmas.

Diagnose ou Diagnóstico : inferem mau funcionamento de sistemas através de observações, isto é, usam descrições de situações características de comportamento ou conhecimento sobre o projeto para inferir quais as possíveis causas do mau funcionamento de um sistema.

Projeto : configuram objetos e restrições, isto é, desenvolvem configurações de objetos baseados num conjunto de restrições do problema.

Planejamento : Projetam ações passo a passo.

Monitoração : Comparam o comportamento atual do sistema com o comportamento esperado.

Depuração : prescrevem soluções para mau funcionamento.

Conserto : executam um plano para aplicar uma solução prescrita.

Instrução ou Ensino : diagnosticam, depuram e corrigem o comportamento do estudante. (Uma vez que possuem conhecimento embutido, este pode ser usado para formar novos especialistas)

Controle : interpretam, predizem, corrigem e monitoram o comportamento de sistemas. Gerenciam o comportamento geral de um sistema.

I.3.2.6. Linguagens e Ambientes de Desenvolvimento

As ferramentas utilizadas no desenvolvimento de sistemas especialistas vão desde linguagens de programação de alto nível até ambientes de desenvolvimento.

Embora alguns sistemas tenham sido desenvolvidos através de linguagens como FORTRAN, C e PASCAL, as linguagens mais adequadas para aplicações em Inteligência Artificial são aquelas voltadas para a manipulação

simbólica. As mais difundidas e utilizadas são o LISP (List Processing) que contém mecanismos para manipular símbolos em forma de estruturas de listas e o PROLOG (Programming Language for Logic) que manipula facilmente expressões lógicas. Essas linguagens oferecem uma enorme flexibilidade, contudo, não são adequadas quando se pretende construir um protótipo rapidamente.

Os ambientes de desenvolvimento podem ser vistos como uma "estrutura básica" constituída pelo Motor de Inferência, pelos arquivos destinados a receber a Base de Conhecimento e pelos procedimentos de interface. Eles facilitam a estruturação e a construção de sistemas pois direcionam a representação do conhecimento e os mecanismos de inferência, tornando-se mais rápidos e mais fáceis de manipular. Por outro lado, por usarem os esquemas de controle pré-definidos pela máquina de inferência, eles carecem de flexibilidade e generalidade, ficando assim, restritos a certas classes de problemas.

Os ambientes de desenvolvimento devem oferecer algumas facilidades de suporte para tornar seu uso mais eficiente, fácil e amigável. Abaixo, alguns deles:

- facilidades de depuração;
- procedimentos de entrada/saída;
- mecanismos de explanação;
- programas que auxiliam na aquisição de conhecimento;

- programas que auxiliam no projeto do SE;
- editores da base de conhecimento;

A escolha da ferramenta mais adequada não é uma tarefa trivial, já que o engenheiro do conhecimento nem sempre consegue classificar seu sistema dentro de uma das classes de problemas tratadas pelas ferramentas disponíveis no mercado. Geralmente, seleciona-se a ferramenta de acordo com o equipamento disponível e pela familiaridade do construtor com a mesma. Outra opção prática para avaliar a ferramenta é através da implementação de protótipos já que nem sempre a ferramenta escolhida para o desenvolvimento se mostra a melhor escolha para a versão final do sistema.

I.3.3. Engenharia de Software

Devido a crescente utilização de sistemas computacionais para a solução de problemas, a tecnologia relacionada à este desenvolvimento tem sido encarada com mais seriedade, principalmente após a chamada "crise do software" na década passada. Com o aumento da complexidade destes sistemas tornou-se imprescindível um planejamento detalhado e coordenado desde a idealização até a concepção e manutenção dos mesmos. A Engenharia de Software, através de suas metodologias, objetiva melhorar a qualidade dos programas e aumentar a produtividade e satisfação do pessoal envolvido no desenvolvimento dos mesmos, STEWARD [12].

Gerenciando a tecnologia computacional, a equipe atuante com suas distintas tarefas, os custos, os recursos e o tempo disponível, a Engenharia de Software procura atingir as expectativas do cliente através do produto final com um retorno satisfatório aos seus produtores.

Para chegar ao produto final dentro dos prazos e custos estabelecidos, um número cada vez maior e mais variado de métodos vem sendo utilizado nas fases de especificação, planejamento, projeto, construção, validação e manutenção.

A preocupação em buscar melhores métodos se justifica quando se tem em mente os crescentes custos necessários para o desenvolvimento dos programas já que, uma vez concluídos, modificá-los ou corrigi-los, além de não ser uma tarefa trivial, torna-se extremamente dispendioso. Planejar criteriosamente um sistema e desenvolvê-lo corretamente é mais econômico do que primeiro construí-lo cuidadosamente e depois tentar modificá-lo até ficar correto. A prevenção de erros custa menos que a detecção.

I.3.3.1. Inteligência Artificial aplicada à Engenharia de Software

A necessidade de aumentar a produtividade no desenvolvimento de software e a qualidade do produto fez a Engenharia de Software recorrer à automação de processos de desenvolvimento. Embora não esteja resolvido o problema de definição de modelos de processo adequados para a aplicação de métodos automatizados, muita coisa já foi feita tomando-se ainda como referência o modelo em fases, isto é, o ciclo de vida convencional do software.

O que se observa é que a simples automação não vem produzindo os resultados esperados. Os casos que envolvem aspectos mais subjetivos do conhecimento, como por exemplo, decisões sobre o projeto, requerem uma maior participação do ambiente, de forma a aliviar a carga do analista desta atividade, que ainda é altamente dependente de sua experiência no uso da própria metodologia. Além disso, as metodologias usadas na Engenharia de Software não garantem a produção de softwares absolutamente confiáveis, já que permanecem susceptíveis a erro.

Com tais problemas em foco, a Engenharia de Software vem buscando, atualmente, soluções em, principalmente, duas frentes : formalização de métodos e a incorporação de técnicas Inteligência Artificial aos processos de desenvolvimento de software.

No que diz respeito aos trabalhos de IA aplicados à Engenharia de Software, eles se expandiram em duas frentes principais: uma, no sentido da psicologia cognitiva, onde vem sendo investigado um meta-modelo de como especialistas realizam algumas tarefas para implementar um modelo num programa; uma outra, aplicando as técnicas da IA nos métodos da Engenharia de Software, também conhecidos como "softwares assistentes baseados no conhecimento".

Algumas aplicações da IA na Engenharia de Software vistas em LUCENA [13] são citadas abaixo:

- IA aplicada às fases do ciclo de vida e às metodologias
 - .fases de requisitos e especificação
 - .fase de projeto
- IA aplicada à programação
- IA aplicada à interface homem-máquina
- IA aplicada à gerência de Banco de Dados
- IA aplicada à geração automática de ambientes de desenvolvimento de software

Abaixo, pode-se ver alguns Sistemas Especialistas aplicados à Engenharia de Software relacionados na tabela I.2.

Nome do Sistema Especialista	Origem	País	Nível de Desenvolv.	Descrição Sumária
ADA Tutor [42]	Computer Thought	EUA	C	formação de programação em ADA
Programmer's Apprentice[42]	MIT	EUA	D	programação automática
PROUST [42]	Yale University	EUA	D	busca de erros não sintáticos em programas escritos em PASCAL
SPORA [42]	-	União Soviética	D	síntese de programas
Mixer [2]	Tokyo University	Japão	DA	auxilia na micro-programação de integrados TI990 VLSI da Texas Instrument

D - ainda em estado de desenvolvimento

DA- desenvolvimento avançado

C - comercial

Tabela I.2

FASE DE PROJETO

As atividades de IA associadas ao Projeto de software estão basicamente voltadas para dois pontos: o raciocínio por trás das decisões do Projeto e o desenvolvimento de sistemas inteligentes, baseados em metodologias já desenvolvidas que assistem ao projetista no desenvolvimento do software.

O primeiro ponto está diretamente relacionado com a busca por novas metodologias, ferramentas e , mesmo, novos modelos da atividade de projetar. A IA está particularmente interessada nos caminhos traçados pelo raciocínio de um projetista especialista em comparação com um projetista junior, tentando responder as questões relevantes: como pensa o projetista ? O que acontece quando ele se depara com um domínio não conhecido ? Quais as heurísticas usadas para a seleção de um método ? Bons trabalhos neste ramo de pesquisa são descritos em ADELSON [14], KANT [15] e STEIER [16].

Os assistentes especialistas em metodologias são ferramentas (que incluem, em geral, SE associados) que ensinam um projetista a usar as regras de uma metodologia. São duas as formas de realizar esta tarefa: uma, onde o projetista trabalha diretamente sob a supervisão da ferramenta, na outra, ele submete seu projeto para apreciação da mesma. De ambas as formas o projetista pode obter orientação sobre a aplicação do método, alternativas para a solução de problemas ou até mesmo planejamento e controle da solução. LUCENA [13] e GANE [17] apresentam ferramentas com as características citadas. O WALK, Sistema Especialista em Modularização de Projetos Estruturados, descrito nesta tese, também se enquadra nesta categoria.

CAPÍTULO II

APRESENTAÇÃO DO PROBLEMA

II.1. Projeto Estruturado de Sistemas

O Projeto Estruturado está baseado na pesquisa empírica feita por Larry Constantine (YOURDON [18]) que observou ser mais fácil desenvolver e manter os programas tipicamente modulares do que os programas convencionais produzidos sem este cuidado.

O Projeto Estruturado é um conjunto de conceitos, medidas e diretrizes cujo propósito é reduzir o custo do desenvolvimento e da manutenção dos softwares através da redução do tempo e do esforço necessário para tanto. Isto pode ser obtido se o projeto for relacionado com uma característica de resolução de problemas em geral, onde fica mais fácil resolvê-los quando se considera os aspectos do problema separadamente. Com isso, reduz-se a complexidade e aumenta-se a facilidade para implementar mudanças. O objetivo, portanto, do projeto estruturado é projetar programas como estruturas de módulos independentes e unifuncionais.

O programa resultante torna-se mais fácil de ser desenvolvido e mantido. Além disso, ele possuirá as seguintes características:

- maior simplicidade;
- poderá ser escrito, entendido, testado e alterado parte por parte;
- os programadores terão menos possibilidade de cometer erros;
- efeitos colaterais de futuras alterações serão reduzidos drasticamente;
- estimativas de custo mais precisas serão possíveis;
- esforços de otimização poderão ser aplicados nas áreas críticas;
- módulos de função única serão altamente qualificados para serem utilizados em futuros programas.

II.1.1. Qualidades de um Bom Projeto

Nem o Diagrama Hierárquico de Módulos nem a especificação dos módulos por si só mostram a qualidade de um certo projeto. É preciso considerar mais outros pontos e todos em conjunto.

Para um melhor entendimento descreve-se, a seguir, a definição de um módulo: é um conjunto de instruções de programa que pode ser chamado por um nome. Exemplo: programas compilados separadamente, seções ou parágrafos em cobol, segmentos de programação estruturada, etc. Os módulos do projeto estruturado serão mais independentes à

medida em que possuem seu próprio conjunto de variáveis.

Como já foi dito, um dos principais fundamentos do projeto estruturado é o fato de grandes sistemas serem particionados em módulos manuseáveis. Contudo, esta separação deve ser feita de modo que os módulos sejam o mais independentes possível. Este critério é chamado de acoplamento. Além disso, cada módulo deve tratar de um único problema relacionado. Este é o critério de coesão.

Estes são os temas centrais do Projeto Estruturado, mas existem diversas diretrizes que podem ser usadas para avaliar e aprimorar a qualidade do projeto. Estas diretrizes também são apresentadas neste capítulo.

II.1.2. Diagrama Hierárquico de Módulos

O projeto estruturado necessita de especificações como entrada. A análise estruturada produz uma boa entrada para o projeto estruturado. A saída do projeto estruturado é o Diagrama Hierárquico de Módulos (DHM), complementado com as especificações de cada módulo, ROCHA [19] .

O Diagrama Hierárquico de Módulos tem as seguintes vantagens:

- . uma ferramenta gráfica;
- . possível de ser particionado permitindo, assim, uma visão geral do sistema;
- . rigoroso, embora flexível;
- . uma entrada útil para a implementação estruturada;
- . uma boa documentação da fase;
- . um bom auxílio para a manutenção.

A figura abaixo mostra a simbologia básica do Diagrama Hierárquico de Módulos.

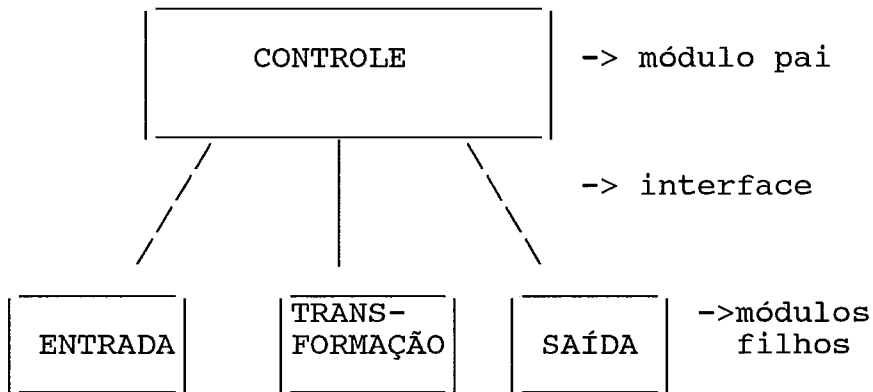


Figura II.1.

Os módulos obedecem a uma hierarquia, onde os módulos subordinados (módulo filho) recebem ordens de seus superiores (módulo pai). Cada módulo executa uma função específica, sendo que módulos de um mesmo nível não se comunicam. A linha que liga os módulos chama-se interface, e por ela transitam os parâmetros que são representados por bolinhas abertas (dados) ou fechadas (flag de controle ou descritivo), associadas a uma seta

que indica o sentido em que estão transitando (para cima ou para baixo).

O projeto estruturado prevê dois tipos de estruturas: de transformação e de transação.

A estrutura de transformação é comum na maioria das aplicações comerciais que em geral, obtém dados, transformam estes dados e os colocam em alguma parte. Sistemas com este tipo de estrutura possuem:

- . uma ramificação de entrada que trata de todas as funções de entrada;
- . uma ramificação para transformação que aceita uma entrada e produz um resultado;
- . uma ramificação de saída que trata de toda saída daquele resultado.

Programas do tipo comercial, que não resultam neste tipo de estrutura, geralmente resultam na estrutura de transação. Vide figura abaixo:

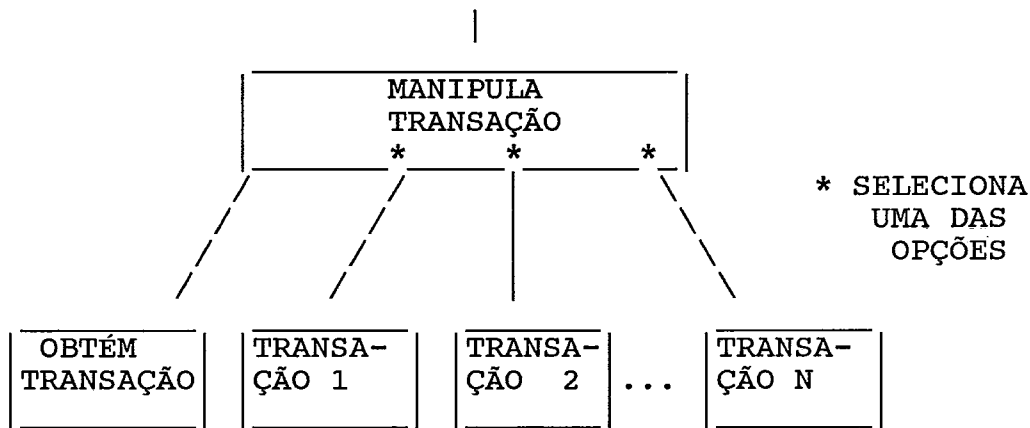


Figura II.2.

Ao se construir um DHM, é conveniente supor que a estrutura do programa se ajusta a uma das duas estruturas básicas. Caso, posteriormente, se verifique que é preferível outro tipo de estrutura, a estrutura básica pode ser modificada e melhorada. Construir estruturas originais para cada programa pode significar muito tempo desperdiçado.

II.1.3. Conceitos Básicos

Duas medidas relativas à obtenção da independência entre módulos: coesão e acoplamento. A coesão mede a força do relacionamento entre os elementos do código dentro de um módulo. O objetivo é maximizar esta força. O acoplamento mede a força das relações entre os módulos. O objetivo é maximizar a independência, minimizando o acoplamento.

A meta é colocar elementos altamente relacionados no mesmo módulo (aumentando a coesão), para que haja um acoplamento baixo entre os módulos (aumentando a independência). A coesão e o acoplamento não são independentes; geralmente a coesão influencia o acoplamento. Uma coesão melhor (maior) leva a um acoplamento melhor (menor) e ambos tornam os sistemas melhores (mais fáceis de se manter).

II.1.3.1. Coesão

Mede a força das relações entre os elementos dentro de um mesmo módulo.

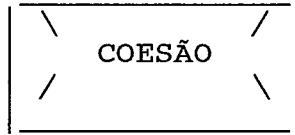


Figura II.3.

Os elementos não devem ser muito relacionados com elementos de outros módulos, pois, isso aumentaria o acoplamento entre os mesmos.

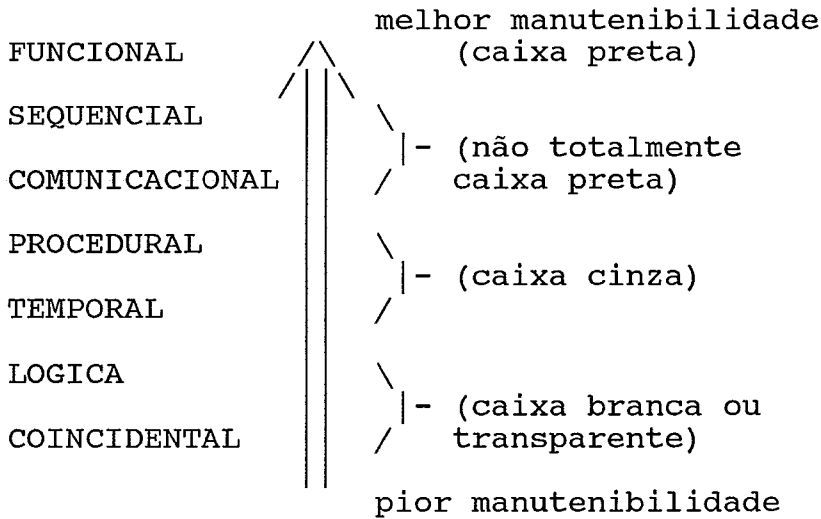


Figura II.4. - Escala da Coesão

Um módulo é coeso quando se tem neste módulo elementos fortemente associados. Entende-se pelo termo elemento, uma instrução, ou grupo de instruções ou até mesmo uma chamada a outro módulo.

Assim, coesão é a medida da força de associação funcional dos elementos dentro de um módulo.

Esta é uma forma de determinar o particionamento de um sistema, focalizando a maneira como as atividades dentro de um módulo singular são relacionadas a um outro módulo.

A pergunta básica para identificar o tipo de coesão é: " Por que certas instruções foram agrupadas ? ".

A seguir faz-se uma descrição sucinta de cada tipo de coesão:

COESÃO FUNCIONAL: Um módulo coeso funcionalmente contém elementos onde todos contribuem para a execução de uma e somente uma tarefa relativa a um determinado problema.

Uma característica de uma unidade funcional é que ela pode facilmente ser substituída por qualquer outro elemento que sirva ao mesmo propósito. Outra característica é que há conexões simples e claras entre um módulo funcional e sua vizinhança.

De um modo geral, se pudermos resumir em uma sentença, composta de um verbo que expresse uma ação e um objeto, as atividades que um módulo executa, então ele

será funcional. Por exemplo: calcular raiz quadrada, obter data, calcular INPS.

O módulo não estará coeso funcionalmente se nesta descrição houver:

- uma proposição composta (E, vírgula, OU);
- mais de um verbo;
- um verbo não específico, como: "manipular", "processar" ou "controlar".

A descrição da função de um módulo deve incluir tudo que ocorra entre o ponto em que é chamado e o ponto em que retorna ao seu chamador.

COESÃO SEQÜENCIAL: Um módulo coeso seqüencialmente é aquele cujos elementos estão envolvidos em atividades tais que os dados de saída de uma atividade servem como dados de entrada da próxima.

Um módulo seqüencial geralmente tem um bom acoplamento e é facilmente mantido. A única desvantagem é quanto à reutilização do módulo, pois ele contém atividades que em geral não são usadas juntas.

Por exemplo: Elemento que obtém e revisa alguns dados; os elementos que criam e armazenam um registro em

um arquivo; os elementos que acumulam resultados e imprimem.

COESÃO COMUNICACIONAL: Um módulo coeso comunicacionalmente é aquele cujos elementos contribuem para atividades que usem os mesmos dados de entrada ou de saída. É mais fraco que o seqüencial porque os elementos apenas referenciam os mesmos dados de entrada e/ou saída. Os elementos de código podem ser executados normalmente em qualquer ordem.

Os módulos com este tipo de coesão são facilmente mantidos, apresentando, entretanto, devido ao compartilhamento de código entre as atividades, problemas na reutilização e dificuldades na manutenção. O que se sugere nestes casos é identificar e separar as atividades ali contidas em módulos funcionais.

O acoplamento entre dois módulos comunicacionalmente coesos é aceitável.

Exemplos de coesão comunicacional: os elementos que produzem múltiplos relatórios do mesmo fluxo de dados; os elementos que tanto imprimem quanto armazenam um item de dados de entrada; os elementos que atualizam e deletam um dado registro em um arquivo.

COESÃO PROCEDURAL: Contém elementos que estão envolvidos em atividades não relacionadas funcionalmente e nem por dados. O controle de execução flui de uma atividade para outra, sendo importante a ordem de execução (na coesão sequencial, o que flui de um módulo para o outro são dados e não controle). Em geral, os dados que entram e saem possuem um pequeno relacionamento, pois módulos com coesão procedural geralmente realizam apenas parte de uma tarefa. Por exemplo : calcular resultados parciais.

Aqui, ultrapassa-se a fronteira de módulos de fácil manutenção, com alto nível de coesão, para módulos de manutenção menos fácil e médio nível de coesão.

COESÃO TEMPORAL: Um módulo coeso temporalmente contém elementos que estão relacionados com o tempo (momento) em que são executados. Por exemplo: módulos com nomes como: "zerar campos", "arrumação", "limpeza"; ou com palavras referentes a tempo como "inicializar", "finalizar" ou "terminar".

As atividades de um módulo temporal estão mais fortemente relacionadas com atividades de outros módulos. Essa situação leva a um acoplamento ruim. Entretanto, este tipo de coesão é melhor que a lógica porque todos os elementos do módulo podem ser executados sempre que este for chamado. Além disso, ele não é chaveado, embora tenda

a compreender compartilhamento de código entre as atividades, dificultando a manutenção e sua reutilização.

Já entre os módulos procedurais e temporais, a diferença esta na ordem de execução das atividades, onde esta é mais importante em módulos procedurais.

COESÃO LÓGICA: Um módulo coeso logicamente contém elementos envolvidos com atividades de uma mesma categoria geral, isto é, os elementos executam uma classe de função geralmente empregando o mesmo verbo (mesma função em dados diferentes). Por exemplo: "editar todos os dados", módulo com um predicado que não seja um objeto específico simples.

A atividade a ser executada é selecionada externamente ao módulo lógico (caixa branca), além disso, muitas vezes importantes porções de código nem mesmo são executadas a cada chamada distinta do módulo, em outras palavras, o código é compartilhado entre as funções existentes no mesmo. Os parâmetros recebidos tem significados diferentes, dependendo da atividade a ser executada.

A vantagem que poderia se obter de um módulo deste feitio é a economia de memória, através do compartilhamento do código entre as funções. Em contrapartida, enfrenta-se as seguintes desvantagens :

alterabilidade ruim (é difícil identificar o código que está sobreposto e as funções que compartilham este código); sobreposição adicional perde tempo de desenvolvimento (ao contrário do que 'as vezes se imagina) por existir a necessidade de se determinar as porções de código que podem ser sobrepostas.

Por último, o módulo lógico é uma caixa branca, sendo seu entendimento e manutenção difíceis.

COESÃO COINCIDENTAL: Um módulo coincidental, assim como um lógico, contém atividades que não estão relacionadas nem por fluxo de dados, nem por fluxo de controle. Entretanto, no módulo lógico, as atividades pertencem a uma mesma categoria, o que não acontece no coincidental. Em outras palavras, um módulo coincidental contém elementos envolvidos em atividades sem relação entre si, pois "eles tinham que entrar em algum lugar". Por exemplo: juntar alguns comandos sem relação entre si mas que aparecem repetidamente.

As causas para existirem um módulo coincidental são: economia de tempo e memória, divisão arbitrária de código monolítico e alterações mal feita, provocando as seguintes desvantagens : difícil manutenção, pode se tornar chaveado, prejudicando o conceito de caixa preta, alterações necessárias a um chamador não são muitas vezes necessárias a todos chamadores.

Para melhor visualização das diferenças entre os tipos de coesão, segue-se uma tabela com algumas de suas características.

NIVEL DE COESÃO	ACOPLAMENTO	REUTILIZAÇÃO	MANUTENÇÃO	ENTENDIMENTO	CLAREZA DE IMPLEMENT.
FUNC.	bom	bom	bom	bom	bom
SEQUENC.	bom	médio	bom	bom	bom
COMUNIC.	médio	pobre	médio	médio	bom
PROCED.	variável	pobre	variav.	variav.	médio
TEMPORAL	pobre	péssimo	médio	médio	médio
LOGICA	péssimo	péssimo	péssimo	pobre	pobre
COINCID.	péssimo	péssimo	péssimo	péssimo	péssimo

Tabela II.1.

Um módulo será ligado funcionalmente somente se todos os pares possíveis de elementos dentro dele exibirem uma ligação FUNCIONAL.

Classifica-se a ligação de um módulo como a de mais baixa ligação encontrada dentro do mesmo, ou seja, a ligação de um módulo melhora quando qualquer elemento não funcionalmente ligado é removido.

Uma Árvore de Decisão segundo PAGE-JONES [20] para determinar a coesão de um módulo:

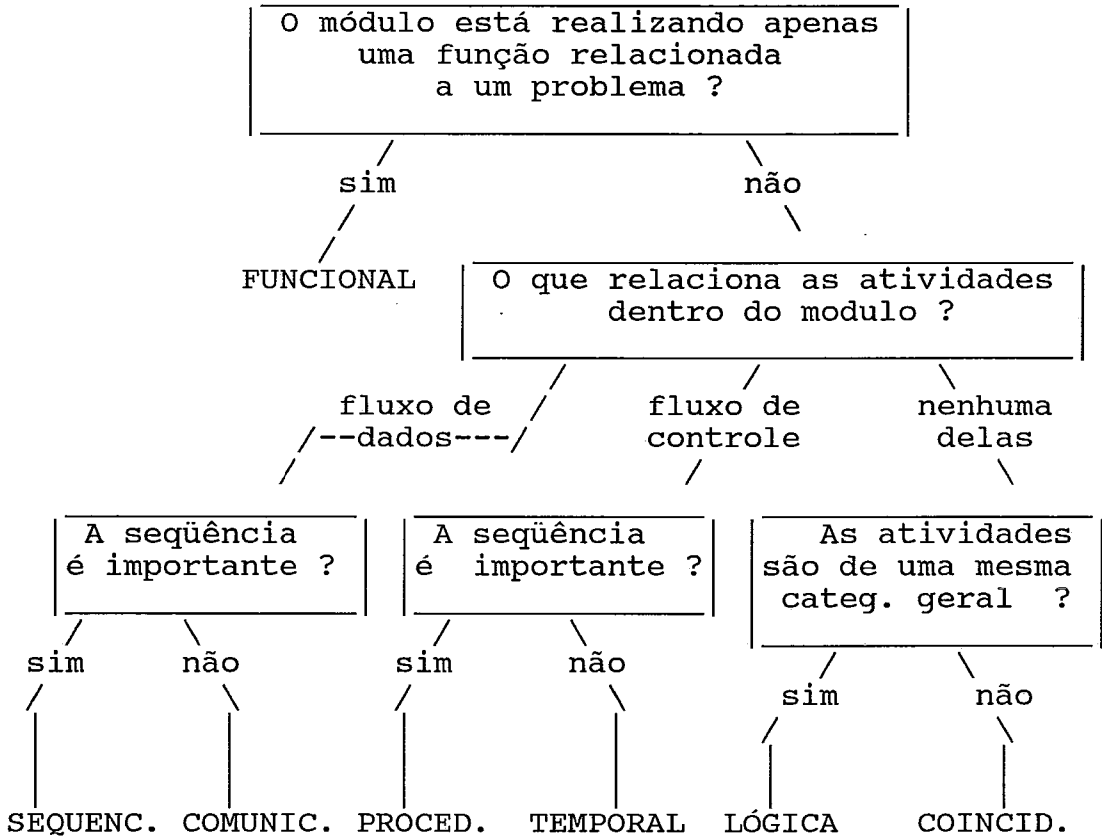


Figura II. 5.

II.1.3.2. Acoplamento

O acoplamento é o grau de interdependência que existe entre dois módulos. O objetivo ao analisar o acoplamento é tentar diminuí-lo ao máximo para que os módulos sejam tão independentes quanto possível.

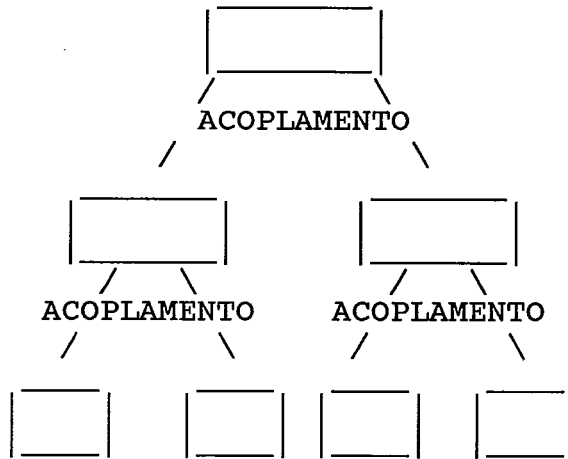


Figura II.6.

Acoplamento baixo entre módulos indica que o sistema foi bem particionado e pode ser alcançado de uma das três maneiras abaixo:

- . eliminando relacionamentos desnecessários
- . reduzindo o número de relacionamentos necessário
- . abrandando o estreitamento dos relacionamentos necessários

Um dos pontos cruciais do baixo acoplamento é que um módulo não precisa se preocupar com os detalhes de funcionamento do outro. Os módulos são vistos pelo seu funcionamento e aparência externa, isto é, como uma "caixa preta".

Resumindo, o baixo acoplamento é necessário porque:

. quanto menos ligações houver entre dois módulos, menos chances haverá de um interferir no outro (o mau funcionamento de um pode parecer como um sintoma no outro);

. deve-se poder alterar um módulo com um risco mínimo de ter que alterar outro módulo também, e essa alteração deve afetar o menor número possível de módulos;

. na manutenção de um módulo, não deve ser necessário preocupar-se com os detalhes internos (código) de outros módulos. O sistema deve ser o mais simples possível de se entender. A clareza (entendimento) é um item muito importante a se considerar, pois quanto mais fácil for o entendimento do código do módulo, menos provável que se tenha de considerar outro módulo para compreender, consertar ou alterar este módulo.

Abaixo os 5 tipos diferentes de acoplamento existentes entre dois módulos (na ordem crescente) :

- | | |
|--|--------|
| 1. Acoplamento de dados | MELHOR |
| 2. Acoplamento por estrutura de dados | |
| 3. Acoplamento por controle | |
| 4. Acoplamento por área comum de dados | v |
| 5. Acoplamento por conteúdo | PIOR |

Dois módulos podem estar acoplados por mais de um tipo de acoplamento, ou pelo mesmo tipo várias vezes.

ACOPLAMENTO DE DADOS: Aparece quando dois módulos que se comunicam através de parâmetros de campo único ou uma tabela homogênea. O acoplamento de dados é uma comunicação de dados necessária entre dois módulos, desde que os módulos realmente precisem se comunicar. Este é o acoplamento menos prejudicial.

O problema que pode surgir com este tipo de acoplamento é o do dado andarilho ("tramp-data"), i.e., um dado que passa por módulos desnecessariamente. Estes por sua vez, ficam obrigados a passá-lo adiante sem causar-lhe danos (como, por exemplo, alterá-lo acidentalmente).

O dado andarilho é um sintoma de má organização, já que o módulo que o cria pode ser diretamente subordinado ao módulo que o usa.

ACOPLAMENTO POR ESTRUTURA DE DADOS: Aparece quando dois módulos se referem à mesma estrutura de dados. Este acoplamento acarreta dois problemas: todos os módulos que se referem a esta estrutura deverão ser modificados de alguma maneira se algum campo da estrutura for modificado, e cria dependência entre módulos que não estão relacionados.

Os problemas que surgem são :

. qualquer mudança na estrutura de dados afeta o módulo, mesmo que ele não use o item alterado;

- . dificulta a reutilização do módulo;

- . expõe o módulo a mais dados do que ele necessita.

Se este módulo causar danos a um campo não relacionado, será difícil detectar de onde veio o erro.

A estrutura de dados é aceitável quando o seu nome exprime o seu conteúdo, facilitando a confecção do DHM.

O empacotamento de dados não relacionados ("bundling") obscurece o problema e não diminui o acoplamento. Sempre que possível, é melhor passar apenas os campos que serão usados.

ACOPLAMENTO POR CONTROLE: Aparece quando um módulo interfere na lógica interna do outro, passando variável de controle. Este acoplamento implica em um módulo conhecer a lógica do outro, fazendo com que o módulo comandado deixe de ser uma "caixa preta".

Se o controle for de baixo para cima, ocorre então uma inversão de autoridade (pois o módulo subordinado está dando ordens a seu pai). Pode ser, neste caso, um sintoma de que uma função foi partida de maneira incorreta entre os dois módulos.

OBS: Segundo PAGE-JONES [20], existem dois tipos de "flag": o de controle, que dá uma ordem (geralmente o seu nome inclui um verbo) e o descritivo, que descreve um dado ou uma situação (geralmente o nome é um adjetivo), sendo

que o primeiro é prejudicial ao sistema e deve ser evitado e o segundo é inofensivo.

ACOPLAMENTO HÍBRIDO: uma variação do acoplamento por controle. Aparece quando um módulo interfere na lógica do outro através de uma variável híbrida. Uma variável híbrida é aquela onde se mistura controle com o dado propriamente dito. A variável híbrida costuma ser formada por várias partes com significados diferentes.

ACOPLAMENTO POR ÁREA COMUM DE DADOS: Aparece quando dois módulos se referem à mesma área global de dados.

Dentre as linguagens que permitem este tipo de acoplamento, incluem-se Fortran, PL/I e Cobol.

Os principais problemas que este tipo de acoplamento oferece são:

- . um problema na área global pode se propagar para todos os módulos que usem essa área;
- . o módulo fica preso ao nome usado na área global, dificultando a reutilização;
- . Diferentes módulos podem dar significados diferentes ao mesmo dado;
- . dificulta o entendimento, já que não temos a visão dos dados que servem ao módulo;
- . quando um dado é modificado, é difícil saber que módulos devem ser modificados.

O uso de área comum degenera o conceito de modularização ao permitir que os dados extrapolem o limite de um módulo.

ACOPLAMENTO POR CONTEÚDO: Aparece quando um módulo se refere de alguma maneira ao conteúdo de outro módulo. Este tipo de acoplamento só é possível em linguagem de baixo nível.

Caracteriza-se por existir, dentro do módulo, referência ao interior de outro módulo, podendo ser um desvio para o interior de outro módulo, modificação de um dado no interior de um outro módulo ou ainda alteração do código de outro módulo.

Problema:

. é preciso conhecer bem o conteúdo do módulo chamado, descaracterizando o conceito de "caixa-preta".

Linguagens que permitem este tipo de acoplamento:
Assembler, Cobol, Fortran, etc.

Quando existir mais de um tipo acoplamento entre dois módulos, o acoplamento resultante será o pior deles.

II.1.3.3. Diretrizes Adicionais

Além do acoplamento e coesão descritos acima, ao

avaliar um projeto deve-se considerar uma série de outras diretrizes. Abaixo são vistas algumas dessas diretrizes:

NÚMERO DE MÓDULOS SUBORDINADOS ("fan-out") : Um número muito alto ou muito baixo de módulos imediatamente subordinados são indicadores de um projeto pobre, embora um número alto seja mais perigoso do que um número baixo. Além disso, quanto mais alto for este número mais difícil será reutilizar o módulo. PAGE-JONES [20] considera que sete (7) subordinados a um módulo é um bom número pois representa um limite da capacidade humana em processar informações simultâneas.

"Fan-out" alto é um sintoma de que o módulo deve ser fatorado em módulos intermediários. Já um "fan-out" baixo é aceitável, porém também merece uma avaliação para uma possível fatoração (união do módulo pai com filho, por exemplo).

NÚMERO DE MÓDULOS SUPERIORES ("fan-in") : Quando mais de um módulo chama um módulo subordinado é porque se está evitando duplicação de código. Quanto maior o "fan-in", maior o grau de reutilização do módulo e conseqüentemente menor será a complexidade da organização por conter módulos individuais. Porém há restrições, o módulo deve ter uma boa coesão e a interface com seus chamadores deve ter os mesmos parâmetros (em número e tipo).

FATORAÇÃO : procura identificar diferentes funções dentro de um mesmo módulo, dividindo-o quando necessário;

As razões para a fatoraçoão são:

- Reduzir o tamanho do módulo;
- Minimizar a duplicação de código dentro do sistema;
- Separar o trabalho da gerência. Geralmente, alterações afetam uma parte ou outra;
- Criar módulos que podem ser usados em outras partes do sistema;
- Simplificar a implementação.

A fatoraçoão deve cessar quando não mais se puder encontrar nenhuma função bem definida dentro de um módulo, pois, extrair aleatoriamente linhas de código, irá provavelmente formar módulos proceduralmente ou temporalmente coesos.

PARTICIONAMENTO DA DECISÃO : procura evitar que o módulo de origem (gerador) dos dados fique muito distante do módulo de destino (consumidor);

O particionamento da decisão gera dados ou flags andarilhos, isto é, parâmetros que navegam pelo DHM, passando por módulos sem serem utilizados, o que é, também, um sintoma de má organização do projeto. Porém, há casos em que o particionamento da decisão é

inevitável, por exemplo, no reconhecimento de final de arquivo.

A solução, em geral, é mesmo rearranjar os módulos envolvidos, posicionando o módulo gerador próximo ao módulo consumidor no Diagrama Hierárquico de Módulos.

FORMATO DO SISTEMA : procura evitar que os sistemas fiquem direcionados pela entrada ("input-driven") ou pela saída ("output-driven"), tendo como meta garantir o balanceamento do sistema, onde os módulos do topo do diagrama tratam apenas com dados lógicos, não sendo dependentes das características físicas.

As 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). Note que um sistema balanceado é uma característica lógica, e não estética.

As razões para evitar esses sistemas são:

- Acoplamento ruim pois geralmente ocorre particionamento da decisão;
- Módulos de alto nível sofrerão alterações se o formato da entrada ou saída for alterado;
- Contém módulos de edição não usáveis por outros módulos.

COMPLEXIDADE : afeta a avaliabilidade e manutenibilidade.

ABRANGÊNCIA : os módulos não devem ser muito restritos nem muito genéricos, pois isso dificulta a manutenção e reutilização dos mesmos. O maior problema de um módulo restrito é a dificuldade de sua reutilização. Para se obter maior capacidade de 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 funções. Esta pode levar a mais código, pior acoplamento e pior coesão.

As características de um módulo restritivo são: realiza um trabalho desnecessariamente específico, trata com valores, tipos e estruturas de dados restritivos, faz suposições de onde o dado é usado. Já as características de módulos gerais são: realiza tarefas demasiadamente gerais, trata com valores, tipos e estruturas de dados muito gerais, recebe como parâmetro dados imutáveis.

MEMÓRIA DE ESTADO : normalmente quando um módulo retorna o controle para o seu superior, ele "morre". Quando invocado novamente, o módulo é executado como se fosse pela primeira vez, não tendo memória de sua existência anterior. Entretanto, existe um tipo de módulo que está ciente do seu passado através de sua memória de estado. Um módulo possuidor de memória de

estado torna-se imprevisível , ou seja, para entradas idênticas se comporta diferentemente e/ou produz resultados diferentes a cada vez que é executado. Módulos com memória de estado devem ser evitados pois são de difícil entendimento e manutenção.

Apesar disso, algumas vezes ela se torna inevitável.

TAMANHO DO MÓDULO : o tamanho de um módulo não é uma característica governante mas deve ser considerada. Várias pesquisas têm sido feitas no sentido de estabelecer um limite superior para o tamanho de um módulo. YOURDON [18] considerou 50 comandos um número adequado. Módulos maiores do que uma ou duas páginas devem ser avaliados quanto à possibilidade de serem separados, sempre que isto não comprometa a funcionalidade, e módulos com menos de uma a cinco linhas devem ser examinados para ver se podem ser fundidos com seus superiores.

SIMPLICIDADE : avalia se um módulo atinge seu objetivo da forma mais simples possível satisfazendo, ainda, aos demais requisitos de qualidade. A simplicidade pode ser afetada por excesso de robustez e flexibilidade ou, ainda, por efetuar mais funções do que o necessário.

RELATO DE ERROS : Erros devem ser relatados pelo módulo que os detecta e os identifica. Este procedimento torna o módulo mais legível. Além do mais, reportar o erro

longe do módulo em que foi detectado implica num aumento do acoplamento.

Uma questão que surge é onde colocar as mensagens de erro. Tem-se duas opções: Manter as mensagens de erros juntas em um mesmo módulo ou distribuir as mensagens pelo sistema, ficando esta no módulo que detecta o erro.

As vantagens de manter as mensagens juntas são:

- facilitar a padronização;
- evitar a duplicação;
- facilitar a atualização.

Por outro lado, as desvantagens são:

- é preciso criar um número de mensagem artificial, dificultando a rearrumação das mensagens;
- como consequência da criação do número de mensagem artificial, o entendimento fica mais difícil.

DOCUMENTAÇÃO : este critério avalia a qualidade e a disponibilidade relativa ao módulo. Na fase de projeto esta documentação consiste na especificação do módulo.

MÓDULOS DE INICIALIZAÇÃO E FINALIZAÇÃO: Em geral, os módulos de inicialização são difíceis de manter por causa de sua fraca coesão e alto acoplamento.

Problemas:

- inicialização fica longe da função fazendo dados passarem no diagrama;
- dificulta a reutilização do módulo de inicialização;
- dificulta a reutilização do módulo que precisa da inicialização.

A solução é colocar as inicializações dentro do módulo ao qual estão relacionados funcionalmente. O melhor é inicializar o mais tarde possível e finalizar o mais cedo possível.

II.1.4. Estação de Trabalho CASE

Com a necessidade de empregar as metodologias estruturadas e, tornar seu uso mais amigável, prático e dinâmico, surgiram as estações de trabalho tipo CASE (Computer Aided Software Engineering). Sua finalidade é auxiliar o trabalho de desenvolvimento de sistemas, proporcionando qualidade de produto final - sistemas de aplicação - e produtividade ao processo de desenvolvimento de sistemas. O CASE popularizou-se como palavra que descreve a nova geração de ferramentas poderosas [17, 21, 22].

Com a aceitação crescente do mercado, os produtos

CASE vêm se tornando cada vez mais potentes e completos, exigindo por sua vez, equipamentos cada vez mais poderosos e complexos.

Muitas mudanças foram feitas até se chegar ao CASE de hoje. No início, eles eram meros editores gráficos de DFD (Diagrama de Fluxo de Dados), mais tarde introduziram-se dicionários de dados e a habilidade de validar DFDs. Posteriormente, sentiu-se a necessidade de incluir a documentação de toda a especificação baseada nos preceitos da análise estruturada, projeto estruturado e programação estruturada de sistemas. Atualmente, sua função e capacidade foram extendidas, não se limitando somente aos DFDs, eles editam e armazenam modelos de dados, modelos de arquitetura de código, especificações da lógica de processos, layouts de telas e relatórios, estruturas lógicas de dados e até esquemas físicos de bases de dados. Já há produtos capazes de auxiliar o desenvolvimento de sistemas, desde o plano diretor até a geração de código-fonte.

As estações de trabalho CASE têm como característica proporcionar um nível muito mais alto de integração dos componentes do que os produtos CASE básicos. As atividades de planejamento, análise, projeto e construção são suportadas por ferramentas de estações de trabalho separadas e distintas. Estas estações de trabalho devem ser completamente integradas, e deve existir uma interface

continua entre as fases, de forma que uma estação utilize informação a partir de outra.

As ferramentas CASE usam técnicas automatizadas para reduzir o custo e o tempo de desenvolvimento e manutenção de programas. A qualidade dos programas fica substancialmente melhorada com a utilização das técnicas de análise automatizadas. As ferramentas CASE têm a potencialidade de acelerar o desenvolvimento, reduzir drasticamente os gastos de manutenção, simplificar as técnicas de desenvolvimento, deixar que os desenvolvedores concentrem-se nos aspectos criativos do desenvolvimento, facilitar o envolvimento dos usuários finais, aperfeiçoar a qualidade dos softwares e melhorar a sua portabilidade.

Segundo LORENÇO [21], hoje já podemos qualificar quatro tipo de estações de trabalho para profissionais de desenvolvimento de sistemas :

a) Estação de Trabalho para o Planejamento - Uma estação de trabalho que auxilia o profissional incumbido do planejamento de sistemas da empresa. Consiste de Editores de Matrizes e Dicionário de Dados que apóiam técnicas de planejamento estratégico de informática. Em alguns casos, essas estações de trabalho se ligam a dicionários de dados empresariais que as realimentam e subsidiam o replanejamento de sistemas. Entre as mais conhecidas estão a IEW/Planning Workstation, da Arthur Young knowledgeware, que se liga ao DataManager II em

equipamento mainframe, e ao Excelerator, o best-seller dos CASE.

b) Estação de Trabalho para Análise - Apóia o trabalho do analista de sistemas auxiliando-o no trabalho de especificar, rigorosa e detalhadamente, sistemas de aplicação segundo as técnicas estruturadas de análise de sistemas. Este é o tipo de estação de trabalho com maior concorrência no mercado e que, por isso mesmo, tem tido o maior desenvolvimento e inovação. O próprio termo CASE está, em grande parte, ligado a esse tipo de software.

Os principais produtos estrangeiros com representação comercial no Brasil são o IEW (Information Engineering Workbench), representado pela Arthur Young; o Design/1, componente do Foundation, da Arthur Andersen, e o Excelerator, da Index Technology, representado no Brasil pela Systems Advisers.

Os produtos nacionais mais difundidos são o Smart, da FG&A, representado pela Compucenter e pela Intertec; o PC-DFD plus, componente da família PC-CASE, fabricado e vendido pela Base Tecnologia; e o Mosaico II, IESA-TS.

c) Estação de Trabalho para Projetos - Voltado para o projetista de sistemas, possui editores gráficos capazes de auxiliar no projeto da arquitetura de código dos sistemas, de acordo com métodos que variam entre Yourdon-Constantine (YOURDON [18]), WARNIER [23], JACKSON [24] e

MARTIN [20], com o seu Action Diagram . Possuem, em alguns casos, editores de telas e relatórios , além de construtores e simuladores de navegação e diálogo, permitindo um certo grau de prototipação.

Os softwares mais representativos dessa linha são o Design Workstation, componente do IEW; alguns componentes do Design/1 e o Excelerator e, no Brasil, o Mosaico DEM, da IESA/TS.

d) A última classe de CASE é a formada pelos **geradores de aplicações** para qualquer ambiente, a partir de estações de trabalho baseadas em microcomputadores. Nesta classe destacam-se o Gamma do pacote IEW, o MicroFocus Workbench e o Tellon, que se ligam ao Excelerator, e o Install/1 da Arthur Andersen, integrado ao Foundation. Alguns geradores de aplicação, que ainda não se ligam a outras classes de CASE, merecem ser citados: o AES, antigo Global, da Villares, e o PC-Hibol, representado pela Sun Software, que geram aplicações para o ambiente CICS/VS.

Algumas das estações de trabalho CASE citadas acima e outras tantas podem ser vistas em GANE [13], onde ele faz um apanhado geral das ferramentas comercializadas no mercado internacional.

O mercado de CASE nos Estados Unidos vem crescendo muito e por acreditarem neste mercado, eles têm investido

cada vez mais no desenvolvimento e aprimoramento de CASE. No Brasil, algumas empresas e universidades têm investido consideravelmente no desenvolvimento de CASE, prevendo sua utilização pelo mercado num futuro próximo.

Os CASE brasileiros vêm crescendo em funcionalidade, acompanhando o amadurecimento da comunidade brasileira de informática no uso de metodologias baseadas em técnicas estruturadas.

Apesar de haver uma preocupação crescente em adquirir e utilizar estações de trabalho CASE, isto não significa que os resultados serão obtidos automaticamente. É necessário preocupar-se também, com a formação dos profissionais da área. São poucos os especialistas em técnicas estruturadas existentes no mercado e as ferramentas disponíveis não prevêm seu uso por pessoas iniciantes ou sem experiência no uso prático destas metodologias.

Não há dúvidas de que houve uma grande evolução na engenharia de software com a utilização destas ferramentas, principalmente no que diz respeito à manipulação quantitativa das metodologias, agilizando as atualizações, permitindo que modificações de projeto sejam feitas rapidamente e as consequências sejam verificadas para validade, automatizando a documentação, criando padrões para os diagramas, técnicas e documentação, etc, o

que indiretamente aprimora o produto final e desenvolvimento dos sistemas.

O problema é : será que estas técnicas estão sendo usadas corretamente ? Será que existem especialistas suficientes na área para exercer tal tarefa ?

O fato é que ainda não há no mercado ferramentas que auxiliem o desenvolvedor no que diz respeito a qualidade em si dos sistemas, isto é, uma ferramenta que critique, pergunte, avalie enfim o sistema, enquanto está sendo utilizada.

Este é exatamente o ponto abordado a seguir.

II.1.5. Explicação do Sistema Especialista WALK como assistente do projetista

Segundo GANE [20] e FREEMAN [25], o futuro das ferramentas CASE está relacionado com a utilização de técnicas de Inteligência Artificial na Engenharia de Software, em particular no uso de Sistemas Especialistas aprimorando os produtos CASE.

Os Sistemas Especialistas podem amenizar o problema acima descrito, complementando as ferramentas CASE . A dificuldade em automatizar a experiência e conhecimento dos especialistas está em expressar o entendimento humano

em termos viáveis os quais a máquina possa processá-los. Sendo um problema altamente subjetivo e que ainda está em estudos, torna-se árdua a tarefa de gerar heurísticas e codificar regras que avaliem corretamente um certo sistema. Além disso, o tamanho da base de conhecimento envolvida, e os recursos computacionais necessários para processá-la de uma maneira proveitosa também devem ser levados em consideração. Contudo, superados os obstáculos iniciais, à medida em que tais sistemas especialistas forem sendo implementados e difundidos, sua abrangência, confiabilidade e aceitação serão cada vez maiores.

O Sistema Especialista aqui proposto objetiva, exatamente, complementar os CASE, descritos anteriormente, no que diz respeito ao projeto de sistemas.

Para tanto, ele inicia sua investigação a partir de um primeiro Diagrama Hierárquico de Módulos já descrito. Ele não se preocupa com a passagem da Análise para o Projeto, isto é do DFD para o DHM. A partir da descrição do primeiro DHM e do Dicionário de Dados, o WALK avalia interativamente com o usuário, se projeto está dentro dos padrões propostos pelas metodologias usadas para seu desenvolvimento, se está consistente e se pode ser melhorado.

O WALK procura avaliar não apenas a consistência do DHM no que diz respeito à existência de dados excessivos, etc, mais principalmente se a metodologia escolhida está

sendo bem aplicada, através do uso de módulos funcionais e independentes, de parâmetros integros e bem relacionados e de sistemas balanceados.

O próximo capítulo se destina a descrever como este sistema foi concebido e implementado.

CAPÍTULO III

APRESENTAÇÃO DA SOLUÇÃO

III.1. Etapas de Desenvolvimento

Durante o período de desenvolvimento foram realizadas as seguintes etapas:

1. Determinação da área a ser investigada e, mais especificamente, do domínio a ser abordado;

2. Levantamento da literatura especializada e dos possíveis especialistas a serem consultados posteriormente;

3. Estudo das heurísticas sobre Projeto Estruturado encontradas na literatura e treinamento no ambiente de desenvolvimento de sistemas especialista escolhido (ESE/IBM);

4. Adoção de uma estratégia de solução;

5. Representação do conhecimento retirado da literatura.

6. Implementação, isto é, definição e codificação dos objetos (parâmetros, regras e blocos de controle) do

ESE e rotinas externas que lhe são agregadas (a fim de suprir algumas deficiências do ambiente).

Como não se tinha experiência na construção de sistemas especialistas, achou-se que seria conveniente escolher um assunto relacionado com a computação, pois o mesmo não seria de todo desconhecido e o especialista estaria ao alcance. Além disso, o tema deveria ser relativamente novo, sem ter sido muito explorado, baseado em heurísticas e na experiência dos especialistas. Por esses motivos chegou-se a Engenharia de Software e mais especificamente ao Projeto Estruturado.

Quanto à escolha do domínio a ser estudado, inicialmente pensou-se na transformação do Diagrama de Fluxo de Dados para o Diagrama Hierárquico de Módulos e no seu aperfeiçoamento até que estivesse convertido num bom Projeto. Far-se-ia um verdadeiro "WALKTHROUGH" em cima do primeiro DHM, daí o nome do sistema, WALK (abreviação de walkthrough). Entretanto, verificou-se que este domínio era muito complexo e extenso, pois envolvia mais de um problema. Com isso, resolveu-se particioná-lo e trabalhar somente em cima da 2a. parte, ou seja, do DHM já convertido, ficando a cargo do usuário fazer a passagem da análise para o projeto.

Definido o domínio, passou-se para o levantamento da literatura especializada. Por parecer mais prático e esquemático, optou-se por basear os estudos principalmente

na metodologia descrita por PAGE-JONES [20]. Assim, prosseguiu-se reunindo informações e construindo um protótipo com conhecimento básico para futuras interações com o especialista, desta forma seria possível evitar perguntas elementares ao mesmo, diminuindo assim, prováveis desgastes no relacionamento, MOTTA [26].

O estudo das heurísticas foi uma das etapas mais demoradas do desenvolvimento, já que não se tem um método esquemático para extrair o conhecimento dos livros e dos especialistas. Quanto à ambientação no ESE, trabalhando com uma versão que ainda não tinha sido liberada para comercialização, sentiu-se uma certa dificuldade inicial. Entretanto, superaram-se todos os obstáculos necessários para a construção do sistema.

A estratégia de solução adotada foi muito discutida e modificada diversas vezes antes da última versão implementada. A representação no ESE foi mais rápida do que a aquisição. A implementação foi gradual, já que não se codificaram todas as regras de uma só vez. O número de heurísticas foi aumentando com o maior entendimento do problema, da construção de SE e da ferramenta utilizada. Por sinal, a prototipação foi muito útil, pois deu forma ao conhecimento e direcionou bastante o processo de conceituação e formalização do conhecimento.

III.2. Codificação das Heurísticas

Existem heurísticas que visam a correção, consistência e melhoria de um DHM. Estas heurísticas tratam principalmente das características dos módulos isoladamente, do relacionamento entre módulos, da natureza dos dados e da estrutura de um DHM como um todo. Cada uma dessas heurísticas pode estar relacionada com uma ou mais regras

Embora tenha-se levado em consideração as heurísticas feitas por STEVENS [27], FAIRLEY [28] e outros [18,19,12], foram as de PAGE-JONES [20] que realmente direcionaram este trabalho.

O processo de aquisição do conhecimento passou por diversas etapas:

1) Traduziram-se para o português as heurísticas encontradas na literatura. É interessante notar que como foram consultados vários textos , algumas heurísticas apareceram mais de uma vez, diferindo apenas na forma;

2) Escreveram-se as heurísticas em forma de regras, pensando na detecção dos sintomas de problemas e na sua possível solução;

3) Procurou-se melhorar as regras, tornando-as mais claras;

4) Determinou-se em qual categoria as regras deveriam ser classificadas;

5) Procurou-se determinar qual a ordem mais natural para checar os problemas.

A princípio foram geradas 30 regras relacionadas com os seguintes itens : Acoplamento excessivo de dados, Acoplamento de área comum de dados, Acoplamento de estrutura, Acoplamento de controle, Acoplamento híbrido, Dado andarilho, Fatoração, Particionamento da Decisão, Reportagem de Erro, Fan-in/Fan-out, Balanceamento do Sistema, Restritividade / Generalidade, Inicialização / Terminação e Memória de Estado. Além disso, mais 16 regras relacionadas com a coesão do módulo.

A seguir, alguns exemplos das heurísticas e regras:

1. Módulo servindo apenas como interface (MÓDULO FUNIL).

As vezes, projeta-se um módulo que não cumpre nenhum objetivo específico a não ser transmitir os dados fornecidos por seu chamador para os módulos que ele chama e vice-versa. Tais módulos tem, normalmente, uma coesão lógica ou comunicacional.

Para se determinar se os módulos que o módulo funil chama podem ser chamados diretamente do seu chamador procede-se da seguinte maneira :

.verifica-se se seu chamador já tem os dados necessários para chamar os módulos inferiores e,

.certifica-se de que os dados de retorno do módulo

inferior não são necessários dentro do módulo funil.

Codificando-se esta heurística em forma de regra, tem-se algo deste tipo:

SE (módulo A chama B que chama C)
e (o conjunto de parâmetros passados de A para B
contém os parâmetros passados de B para C)
e (os parâmetros retornados de C para B não são
usados em B)

ENTÃO

(Chamar C diretamente de A)

Quando o módulo serve de interface de seu pai para todos os seus filhos, ele deverá ser eliminado.

2. Eliminando um parâmetro.

Se um parâmetro não estiver sendo usado no módulo para o qual está sendo passado e em nenhum de seus filhos, ele pode ser eliminado da interface (parâmetro excessivo).

A regra inicial gerada para este problema foi:

SE (existe parâmetro de entrada para o módulo que não é usado na lógica deste módulo nem dos seus subordinados)

ENTÃO

(este parâmetro está gerando um Acoplamento

Excessivo de Dados)

e (avisar ao usuário que este parâmetro deve ser eliminado)

e (verificar a coesão do módulo)

3. Eliminando as chaves do programa ("flags" de controle).

PAGE-JONES [20] faz uma distinção entre flag descritiva e flag de controle. A primeira geralmente é binária e indica um estado (ex: sexo masculino/feminino) e a segunda passa uma ordem, ação. O perigo está no flag de controle porque ele implica em que o módulo transmissor do mesmo afete na lógica do módulo receptor. O problema é que, em geral, o projetista não percebe se um flag é de controle ou descritivo.

As chaves de programa transmitidas para baixo indicam geralmente que o módulo chamado contém funções múltiplas. As funções podem ser identificadas muitas vezes considerando as ações tomadas pelo módulo chamado em resposta a cada valor da chave de programa. Já as chaves de programa que são transmitidas para cima indicam normalmente uma função mal colocada (inversão de autoridade).

Este problema gerou regras do tipo:

SE (existe flag de controle passado de pai para

filho)

ENTÃO

(verifique a coesão do módulo filho)

e (verifique se existe Particionamento da Decisão)

SE (existe flag de controle passado de filho para o pai)

ENTÃO

(verifique se existe Particionamento de Decisão)

Na figura a seguir, temos o problema de INVERSÃO de AUTORIDADE onde o módulo filho dá uma ordem para o módulo pai através de um flag de controle.

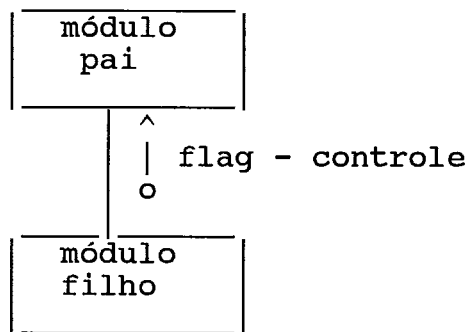


Figura III.1.

4. Dado andarilho ("tramp-data").

Uma das maneiras de eliminar os parâmetros que navegam no diagrama é colocar o módulo de origem do parâmetro o mais próximo possível do módulo destino. Por exemplo: detecção e reportagem de erros devem aparecer juntos.

A seguir temos o problema de um dado andarilho, i.e., um dado que fica vagando no sistema até ser utilizado.

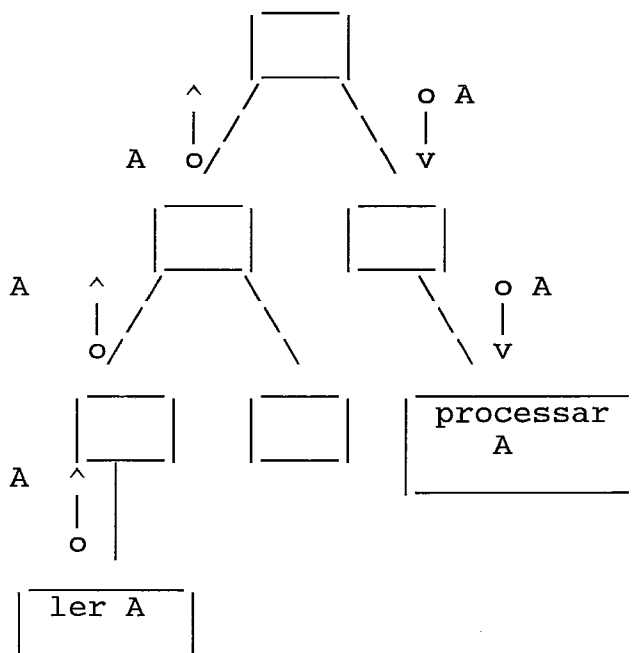
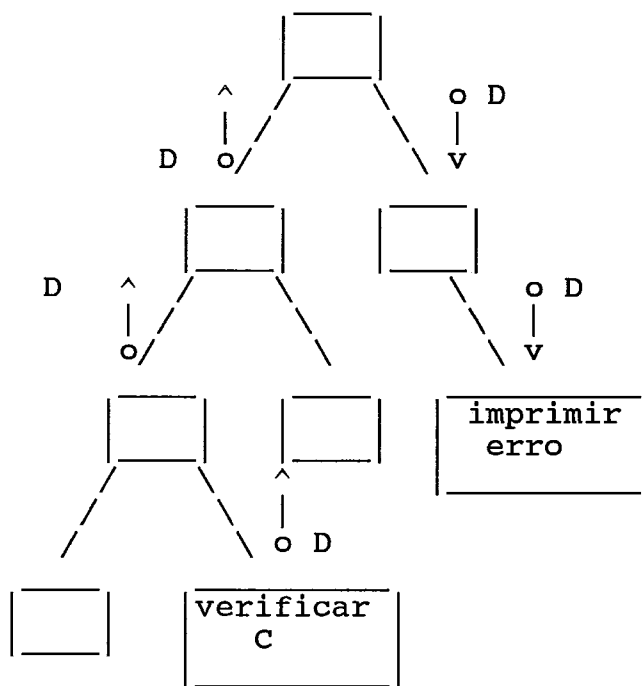


Figura III.2.

Entre outros, um caso particular do particionamento de decisão ("decision-split"), o relato de erro feito longe do local onde foi encontrado, i.e., o módulo que relata o erro não é o mesmo que o encontrou.



onde D é um flag de controle que diz o tipo de erro encontrado em C.

Uma outra observação que pode ser feita sobre o exemplo acima é que ao se deparar com um parâmetro andarilho, uma análise mais detalhada dever ser feita pois ele pode indicar um sintoma de problemas mais sérios.

5. Função duplicada.

Quando uma mesma função está sendo usada em dois módulos distintos, o ideal é criar um módulo que a realize e que seja chamado pelos dois pais. Isto acarreta numa piora do acoplamento, mas é mais fácil resolver um problema de um parâmetro que navega pelo diagrama, que está explícito, do que tentar eliminar uma implícita

duplicação de função mais tarde.

Seguem-se, uma seqüência de heurísticas utilizadas no WALK.

1. Um dado que não é utilizado na lógica do módulo ao qual foi passado como parâmetro sugere a existência de um dado andarilho que navega pela estrutura do projeto.

Esta frase sugere algo do tipo :

Se um dado passa por dois ou mais módulos sem ser utilizado na lógica dos mesmos

Então

este dado é andarilho

2. Particionamento da decisão gera dados ou flags andarilhos, portanto, também é um sintoma de má organização do projeto.

Se o dado é andarilho

Então

é provável que exista o problema de particionamento de decisão

3. Nos casos em que ocorre o particionamento da decisão, a solução, em geral, é mesmo rearranjar os módulos

envolvidos.

Se existe o problema de particionamento da decisão

Então

trazer o módulo consumidor para junto do gerador

Examinando vários casos de projetos com problemas, consegue-se extrair a razão de tais problemas, isto é, seus sintomas. Mas existem diversos casos em que só o especialista é capaz de detectar o principal problema.

III.3. Estratégias de Solução

Pelo o que foi visto no capítulo anterior, o conhecimento é apresentado através das heurísticas, sendo que essas heurísticas podem ser agrupadas em quatro categorias:

- a) Dados do sistema;
- b) Módulos que compõem o sistema;
- c) Interface entre os módulos;
- d) Estrutura geral do projeto.

A estratégia de solução adotada foi dividir a análise em duas etapas. Na primeira são coletados sintomas de possíveis problemas. Na segunda etapa faz-se um diagnóstico completo e sugere-se um tratamento que pode

significar mudanças estruturais no projeto.

Inicialmente, cada categoria é focalizada isoladamente, correspondendo a uma fase de levantamento de fatos, objetivando detectar problemas isolados. A partir deste levantamento, entram as heurísticas de solução. Para conjuntos específicos de problemas isolados são dadas soluções parciais. Não há preocupação, neste ponto, de avaliar a repercussão de cada uma daquelas soluções. Basta que ela resolva aquele problema imediato.

Por fim, faz-se a análise sinérgica dos problemas e suas soluções parciais. São avaliados possíveis efeitos colaterais, pesadas as vantagens e desvantagens de um enfoque em detrimento de outro, para serem dadas, então, as melhores soluções para cada conjunto de problemas relacionados.

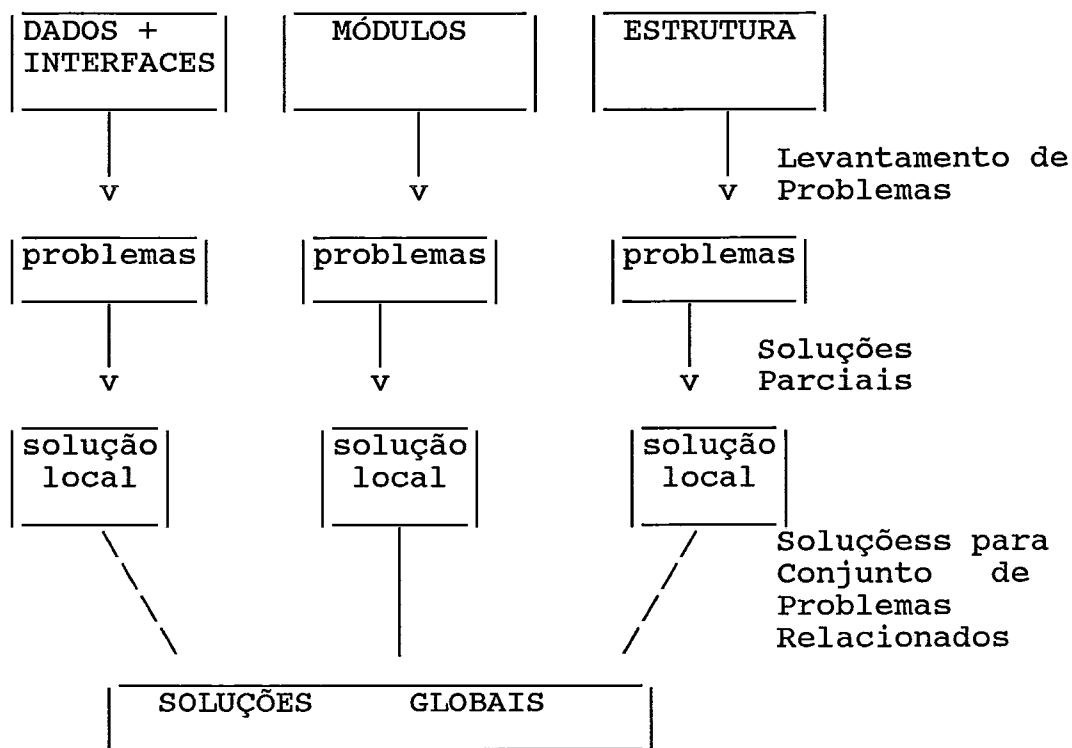


Figura III.4.

ESQUEMA DA ESTRATÉGIA DE SOLUÇÃO DO WALK

Para dar uma idéia melhor da estratégia sugerida acima, segue-se o esquema adotado para a solução do problema.

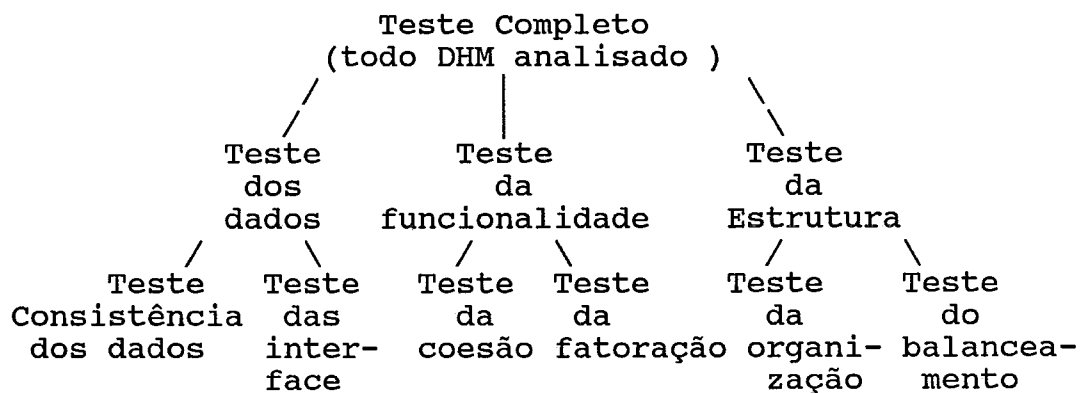


Figura III.5.

O Diagrama Hierárquico de Módulos só estará completamente analisado se forem analisados individualmente todos os dados do diagrama (consistência em si e todas as interfaces pelo qual navegam), a funcionalidade de cada módulo (através de sua coesão e da fatoração). Por último, analisa-se a estrutura como um todo, levando-se em consideração os problemas já então detectados (verifica-se se a organização dos módulos está boa e se o DHM está balanceado).

Descendo um pouco mais neste esquema tem-se:

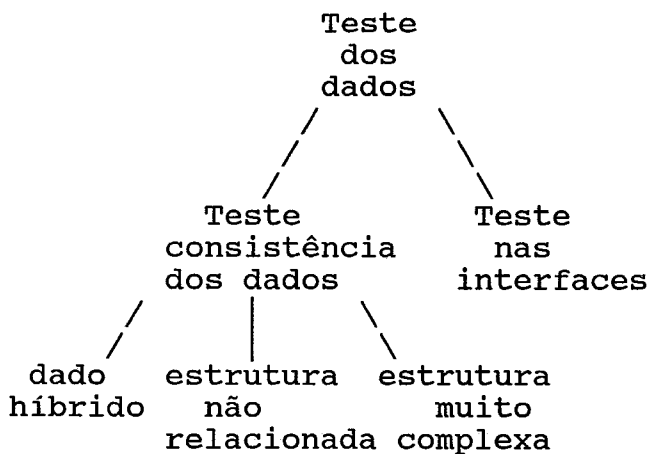


Figura III.6.

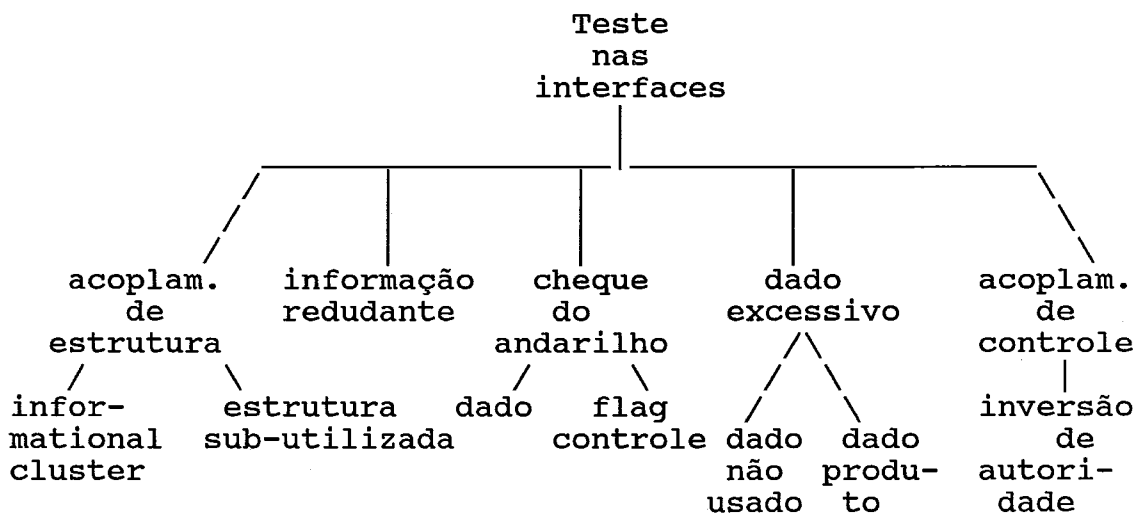


Figura III.7.

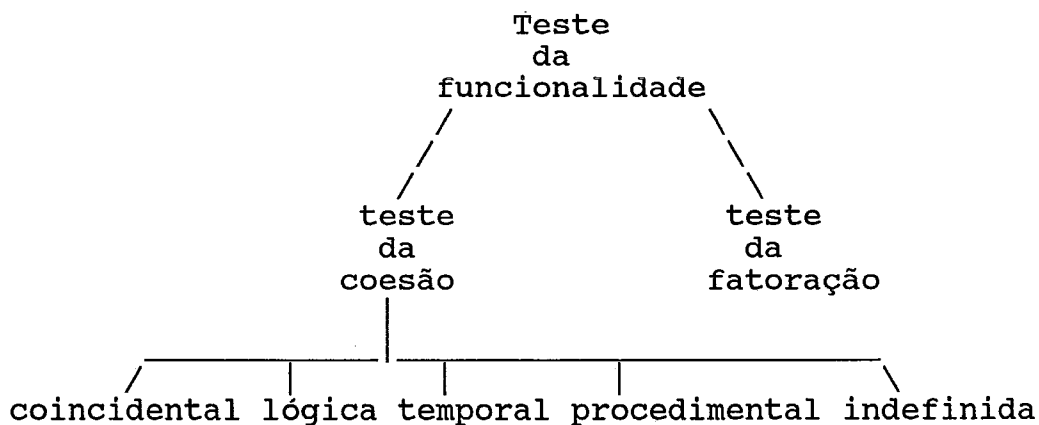


Figura III.8.

As coesões do tipo comunicacional, seqüencial ou funcional são consideradas boas para um projeto.

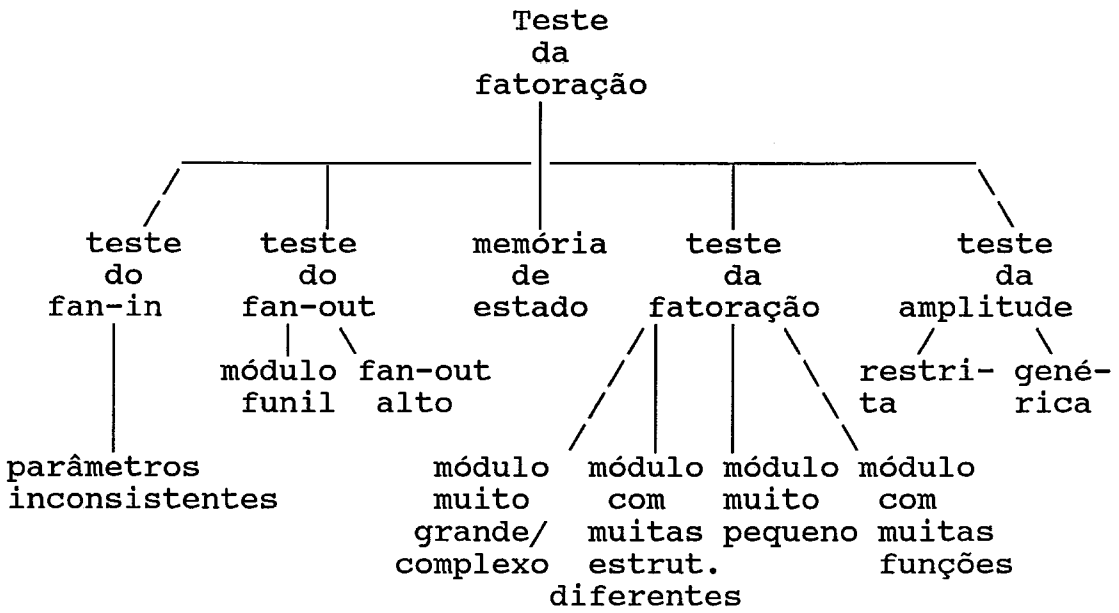


Figura III.9.

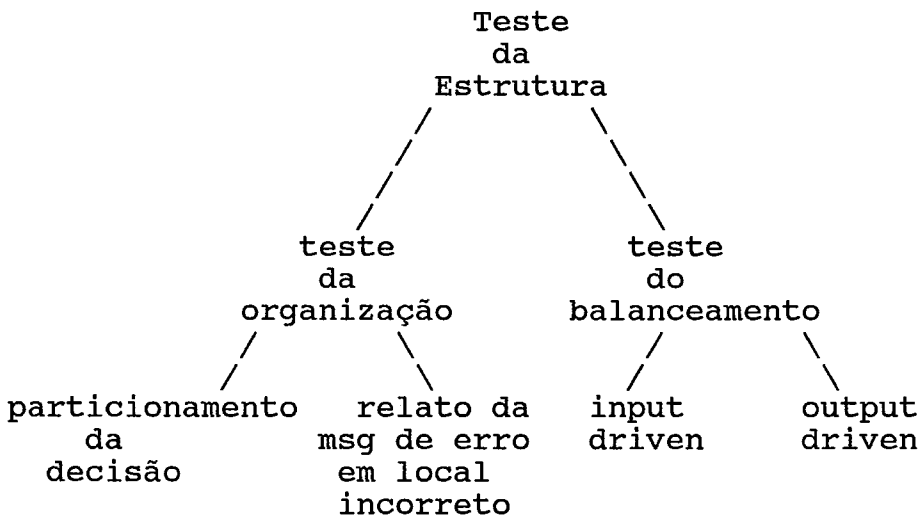


Figura III.10.

Com essa estratégia de solução pode-se trabalhar com a idéia de níveis de problemas, isto é, objetivos intermediários foram determinados antes de se alcançar o principal objetivo e ao mesmo tempo, pode-se utilizar as

conclusões já determinadas para chegar a outras conclusões.

Nivel3 Nivel2 Nivel1 Nivel 0
 (interm.) (interm.) (objetivo)

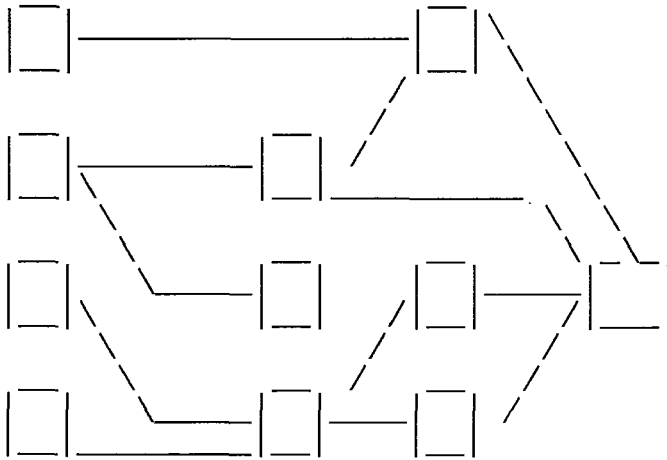


Figura III.11.

III.4. Implementação

III.4.1. Expert System Environment (ESE/IBM)

O ambiente de desenvolvimento utilizado para o desenvolvimento do WALK foi o Expert System Environment da IBM (ESE/VM) que é um sistema de propósito geral para a construção e execução de aplicações de Sistemas Especialistas e roda numa arquitetura IBM/370 sob o

sistema operacional VM/CMS. O ESE/VM é composto de dois programas: um para o desenvolvimento de aplicações (ESDE - Expert System Development Environment) e o outro para execução de tais aplicações (ESCE - Expert System Consultation Environment) [29,30].

Não sendo necessário grande conhecimento em computação, os usuários em potencial do ESE são:

- . o cliente ou usuário final da aplicação,
- . o especialista (médico, engenheiro, advogado, etc)
- . o engenheiro do conhecimento.

Incorporando um conjunto de rotinas que podem ser usadas para desenvolver uma variedade de aplicações, o ESE pode ser encarado como um sistema "vazio" no qual se inserem regras próprias, para definir a base de conhecimento e, se escolhe um método de raciocínio, i.e., uma estratégia de inferência. O resultado é um Sistema Especialista específico, pronto para o usuário final. A base de conhecimento é inserida através de um editor dedicado que verifica a consistência das informações editadas, acusando erros sintáticos ou semânticos durante a própria edição. A escolha da estratégia de inferência irá basicamente determinar a ordem de aplicação das regras da base de conhecimento. Duas estratégias são oferecidas: uma em que a obtenção do valor de um objetivo é conseguida através do encadeamento reverso das regras

até que as premissas sejam todas conhecidas e a outra em que se consideram somente as regras em cujas premissas só sejam referenciados valores conhecidos, fazendo-se um encadeamento direto e descobrindo-se assim novos valores.

Outra característica do ESE é a possibilidade de organizar ou agrupar a base de conhecimento em estruturas hierárquicas, chamadas de Blocos de Controle (FCB). Cada FCB pode ter sua própria máquina de inferência e seu próprio controle de passos. Pode-se também fazer chamadas de rotinas externas escritas em outras linguagens e acessar outros arquivos ou bancos de dados.

Apesar de ser um sistema de propósito geral, o ESE se adequa melhor a um certo domínio de aplicações que tenha sua base de conhecimentos representada na forma dos seguintes objetos:

.Parâmetros : fatos da aplicação e suas restrições

.Regras : combinação de fatos que implicam em outros fatos (relações entre parâmetros)

.Bloco de Controle : agrupamento de regras e parâmetros que impõe uma organização da base de conhecimento e especificações de controle;

O ESE oferece também as seguintes facilidades:

.Grupos : conjunto de regras, parâmetros ou controle;

.Telas : telas definidas pelo construtor do Sistema Especialista para interação com o usuário.

Um parâmetro é um objeto que tem um nome, alguns atributos e que irá possuir um ou mais valores. Os principais atributos de um parâmetro são:

.seu tipo, que pode ser numérico, lógico, cadeia alfanumérica, binária ou cadeia hexadecimal; além disso, um parâmetro pode guardar um único valor ou muitos valores e também pode ser vetorado, sem limite explícito;

.uma seqüência que determinará a forma que o valor do parâmetro será determinado; a obtenção do valor pode ser através de regras, de pergunta ao usuário ou de execução de rotina externa provida pelo construtor do sistema;

.restrições impostas aos valores que o parâmetro pode receber.

As regras especificam relações entre parâmetros e são da forma SE <premissa> ENTÃO <ação>, fazendo com que ações sejam realizadas quando o resultado da avaliação das premissas for verdadeiro. A cada parâmetro da base de conhecimento está associado um fator de confiabilidade que é utilizado nas regras e através destas pode ser propagado. As ações de uma regra são tipicamente atribuições, mensagens nas telas e alterações no controle de execução das regras.

Os blocos de controle oferecem um mecanismo eficiente para modularização do sistema. Cada bloco contém um conjunto de regras e parâmetros e representa uma unidade de trabalho a ser considerada durante a execução do sistema. Os blocos de controle são organizados hierarquicamente e cada um deles pode possuir múltiplas instâncias. Entre os principais atributos de um bloco de controle estão as listas de regras e de parâmetros que lhe pertencem, o número máximo de instâncias e seu texto, que contém as ações e as estratégias que serão realizadas quando da execução do bloco. Cada bloco de controle só pode acessar a lista de parâmetros que lhe pertence e a de seus ascendentes.

Os grupos são conjuntos de objetos similares, como parâmetros, regras e blocos de controle. A função de um grupo é prover um nome a um conjunto de objetos de modo a facilitar a referência a esse conjunto.

As telas servem para facilitar a interação do sistema com o usuário. Elas podem ser definidas durante a construção do sistema, o que irá proporcionar uma interação personalizada. A execução de uma tela escrita sem referenciar-se a uma em particular produzirá uma saída em uma das telas padrões do sistema, de layout não muito interessante e com textos em inglês.

Rotinas externas, preferencialmente escritas em Pascal, podem ser agregadas ao ESE ampliando

consideravelmente o poder de processamento de um Sistema Especialista. Desta forma é possível o acesso a arquivos, a Bancos de Dados, a pacotes gráficos e ao CMS em geral.

Um Ambiente Personalizado de Desenvolvimento/Consulta pode ser criado escrevendo-se rotinas externas, que ampliem o ambiente original, e gerando-se um módulo executável que tenha o ESDE/ESCE e as rotinas da aplicação. Para que seja possível a troca de informações entre o ESDE/ESCE e as rotinas externas existe uma biblioteca de comunicação que permite às rotinas:

- conhecer o nome e o tipo do parâmetro ao qual será atribuído o valor retornado pela rotina;

- conhecer o número de argumentos passados para a rotina;

- conhecer o nome e tipo de cada argumento da rotina;

- conhecer o número de valores que um determinado argumento possua;

- acessar o valor do argumento (ou cada um de seus valores, se o argumento possuir mais de um) assim como o fator de confiabilidade de cada valor;

- enviar mensagens para o arquivo de "trace" da consulta;

- retornar um valor, ou uma série de valores, ao parâmetro objeto da execução da rotina, junto com seu(s) coeficiente(s) de confiabilidade.

III.4.2. Representação no ESE

Recebendo como definição do projeto seu Diagrama Hierárquico de Módulos e seu Dicionário de Dados, o sistema irá verificar heurísticamente se são atendidas as características de estruturação de um bom projeto, apresentando diagnósticos e sugerindo modificações.

A estratégia é refletida na organização dos Blocos de Controle da implementação no ESE. Descrever a implementação do Sistema envolve a estruturação dos Blocos de Controle, a definição dos Parâmetros e a construção de regras, bem como a alocação destes dois últimos nos Blocos, SILVA [30]. Das heurísticas codificadas em forma de regras foram identificados os objetos que seriam os parâmetros. Cada parâmetro recebeu um nome e teve seu tipo definido. Feito isso, codificaram-se as regras no formato ESE.

Os Blocos de Controle são organizados hierarquicamente e essa hierarquia pode ser mostrada com o auxílio de uma árvore que possua o Bloco Global como raiz.

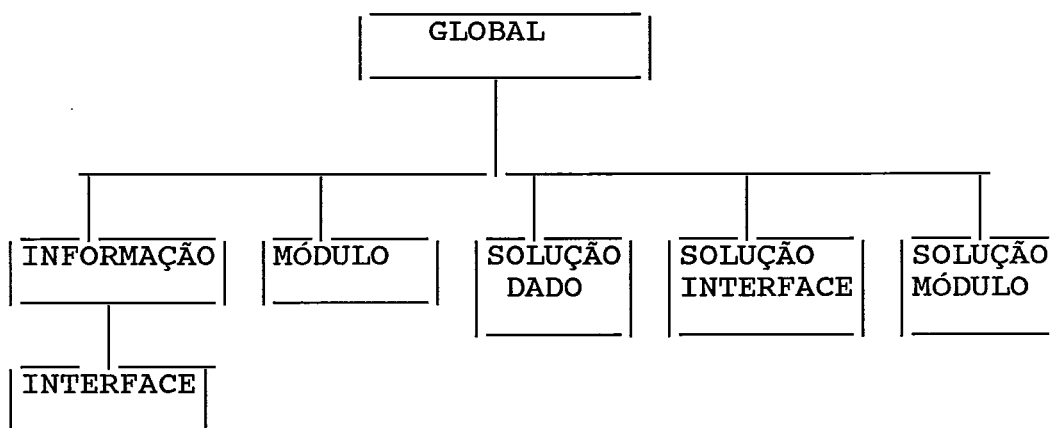


Figura III.12.

Bloco **GLOBAL** - Pertencem a este bloco os parâmetros que guardam as características do projeto a ser criticado e a série de sintomas detectados. Os parâmetros referentes ao projeto são preenchidos através da execução de rotinas externas do ESE, escritas em Pascal. Os sintomas são coletados durante a execução dos blocos **INFORMAÇÃO**, **INTERFACE** e **MÓDULO**. Após a execução desses blocos são considerados os aspectos sobre a organização e o balanceamento da estrutura e são ativados os blocos **SOLUÇÃO DADO**, **INTERFACE** e **MÓDULO**. Assim sendo, o texto de controle deste bloco é :

- a) Mostrar tela de apresentação;
- b) Preencher os parâmetros com informações sobre o projeto;
- c) Executar o bloco **INFORMAÇÃO**;
- d) Executar o bloco **MÓDULO**;
- e) Aplicar as regras referentes à estrutura;
- f) Executar o bloco **SOLUÇÃO DADO**;

- g) Executar o bloco SOLUÇÃO INTERFACE;
- h) Executar o bloco SOLUÇÃO MÓDULO;
- i) Executar soluções referentes à estrutura;

Bloco **INFORMAÇÃO** - Na definição deste bloco diz-se que ele possui múltiplas instâncias, uma para cada dado do projeto que está sendo considerado. Pertencem a este bloco as regras referentes às heurísticas que tratam dos dados e, cada vez que esse bloco for instanciado, são preenchidos seus parâmetros que descrevem as características do dado em questão. Seu texto de controle:

- a) Preencher os parâmetros referentes ao dado;
- b) Aplicar as regras que fazem a coleta de sintomas;
- c) Executar o bloco INTERFACE (em que este dado flui);
- d) Executar nova instância do bloco INFORMAÇÃO, se ainda houver algum dado a ser criticado.

Bloco **INTERFACE** - Este bloco também possui múltiplas instâncias, uma para cada interface entre dois módulos, por onde flui o dado que estiver sendo considerado pelo bloco INFORMAÇÃO. Pertencem a este bloco as regras referentes às heurísticas que tratam do relacionamento entre módulos, e cada instância do bloco possui parâmetros com informações da interface em estudo. Seu texto de controle :

- a) Preencher os parâmetros referentes interface;
- b) Aplicar as regras que coletam sintomas;
- c) Executar nova instância do bloco INTERFACE, se ainda houver alguma interface onde o dado em estudo flui.

Bloco **MÓDULO** - Este bloco, como os anteriores, possui instâncias, uma para cada módulo do projeto. Pertencem a este bloco as regras que implementam as heurísticas aplicáveis aos módulos, em particular as que analisam sua coesão. Os parâmetros do bloco, que descrevem um módulo, são preenchidos quando da ativação de uma instância. O texto de controle do bloco **MÓDULO** :

- a) Preencher os parâmetros referentes ao módulo;
- b) Aplicar as regras do bloco;
- c) Executar nova instância do bloco **MÓDULO**, se ainda existir algum módulo a ser considerado.

Bloco **SOLUÇÃO** (DADO, INTERFACE, MÓDULO)- Nestes blocos são analisados os sintomas encontrados pelo sistema, muitos deles relacionados, e feito um diagnóstico dos problemas que causaram esses sintomas. Uma vez localizado um problema, é sugerida uma mudança na forma de estruturação do projeto.

Ao se armazenar em um local de acesso comum (no caso, o bloco Global), todos os sintomas/problemas detectados em diferentes etapas da consulta, pretende-se deixá-los disponíveis para a solução do conjunto de problemas

relacionados. Isto torna possível que futuramente um certo problema envolvendo diversos sintomas possa ser localizado e tratado como um todo e não apenas parte dele.

Exemplo:

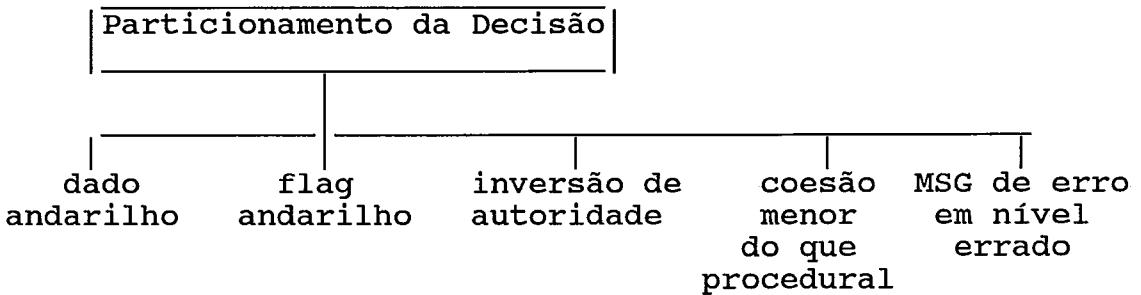


Figura III.13.

Se sintoma1 é flag andarilho

e modulo1 é fonte do flag andarilho

e modulo2 é destino do flag andarilho

e a coesão do modulo1 é < procedural

Então

existe particionamento de decisão entre módulo1

e módulo2

Se existe particionamento de decisão entre módulo1

e módulo2

e módulo1 e módulo2 não tem o mesmo pai

e módulo1 não é pai de módulo 2

e módulo2 não é pai de módulo 1

Então

mover o módulo1 para junto do módulo2

Com a estrutura de Blocos de Controle concebida,

deu-se às regras, nomes que sugerissem a que bloco cada uma delas iria pertencer. A elaboração das regras nos blocos foi feita de maneira indireta. Se a regra não fosse alocada explicitamente a um bloco, o ESE coloca-la-ia no bloco de posição hierárquica mais inferior de tal forma que todos os parâmetros referenciados na regra pertencessem ao bloco ou a algum bloco hierarquicamente situado entre este e o bloco raiz da estrutura de controle. Terminada essa alocação automática, restava conferir se ela estava feita da maneira prevista, e caso contrário far-se-iam as adaptações necessárias.

Para exemplificar este processo, tomemos como exemplo a codificação a seguir.

Heurística: Um dado de controle que não é utilizado na lógica do módulo ao qual é passado como parâmetro sugere a existência de um "flag" que navega pela estrutura do projeto.

PARÂMETRO	PERTENCE AO BLOCO
fd-tipo	INFORMAÇÃO
fi-sentido	INTERFACE
fi-usado-filho	INTERFACE
fi-usado-pai	INTERFACE
fd-tramp-flag	INFORMAÇÃO

Tabela III.1.

Regra para o ESE:

```
If fd-tipo = "controle" and
    (( fi-sentido = "down" and fi-usado-filho = false)
    or (fi-sentido= "up" and fi-usado-pai= false))
then
    There is 0.11 evidence that fd-tramp-flag is true.
```

O nome desta regra é INTER07 que será alocada no bloco INTERFACE e indica que na 2a. ocorrência o flag será considerado andarilho.

III.4.3. Interação com o Usuário

A interação com o usuário se dá de três formas distintas. Começa quando a modularização do projeto é codificada na forma de um texto, que é compilado e posteriormente lido pelo Sistema. Durante a etapa da coleta de sintomas, alguns parâmetros do ESE são determinados através de perguntas ao usuário. E finalmente, os diagnósticos e os tratamentos sugeridos são mostrados ao usuário empregando-se telas personalizadas.

III.4.3.1. Representação do DHM e do DD

Um problema que se apresenta é como coletar e como armazenar o DHM e o DD que possuem um volume

significativo e razoavelmente complexo de informação nas estruturas de dados do ambiente que se pretende utilizar.

Como foi visto anteriormente, o ESE só permite que se definam parâmetros simples e vetorados, mesmo assim possuindo uma única dimensão. Na solução encontrada foi utilizar inúmeros parâmetros vetorados (ORDERED) e representar as informações contendo um número variável de componentes através de poços e de apontadores de início e fim desses componentes no poço, SILVA [30].

Devido à grande quantidade de informações necessárias para a descrição de um DHM, optou-se por descrever o projeto através de uma linguagem criada para este propósito, SILVA [30]. O texto é compilado e são gerados arquivos com formatos que simplificam o processo de carga dos parâmetros do ESE. A carga é feita através de rotinas, que são agregadas ao ambiente.

O usuário dever entrar com o diagrama de seu projeto mais as informações contidas nele.

Interface : identificação -> nome dos módulos pai e filho

parâmetros "up" -> tipo

parâmetros "down" -> tipo

Módulo : nome
função
módulo(s) pai(s)
módulo(s) filho(s)
parâmetros de ENTRADA usados na lógica
parâmetros de RETORNO usados na lógica

Parâmetro : nome
tipo
complexidade (estrutura, simples)

Onde:

.parâmetro "up" são aqueles que vão do módulo filho para o módulo pai e "down" o inverso. (sentido: "up" e "down");

.os tipos podem ser: dado, flag-controle, flag-descriptivo e dado híbrido (controle e dados), sendo que o dado pode ser simples (com apenas 1 campo) ou estrutura (com mais de um campo);

.a complexidade dos parâmetros só é necessária no caso de ser uma estrutura. Nesse caso, entra-se com o número de dados que compõem essa estrutura.

III.4.4. Melhorando a Performance e Heurísticas para Soluções

Uma das dificuldades encontradas foi a de codificar

as regras de forma que não fosse necessário fazer muitas perguntas ao usuário, utilizando apenas informações obtidas através de consultas ao DHM e ao DD. Devido a esta dificuldade, adiou-se esta preocupação para quando fossem feitas entrevistas com o especialista. A princípio, as perguntas que o sistema faz ao usuário estão num nível muito alto, i.e., pergunta-se direto se tal sintoma existe. Mas, pretende-se, através de rotinas externas, consultar o Banco de Dados com informações do projeto e tirar conclusões evitando ao máximo perguntas diretas ao usuário.

III.4.5. Exemplo

Um exemplo completo de como foram codificadas as regras é dado a seguir através da determinação do tipo de coesão:

PAGE-JONES [20] sugere que se use a seguinte árvore de decisão para identificar qual tipo de coesão caracteriza o módulo analisado:

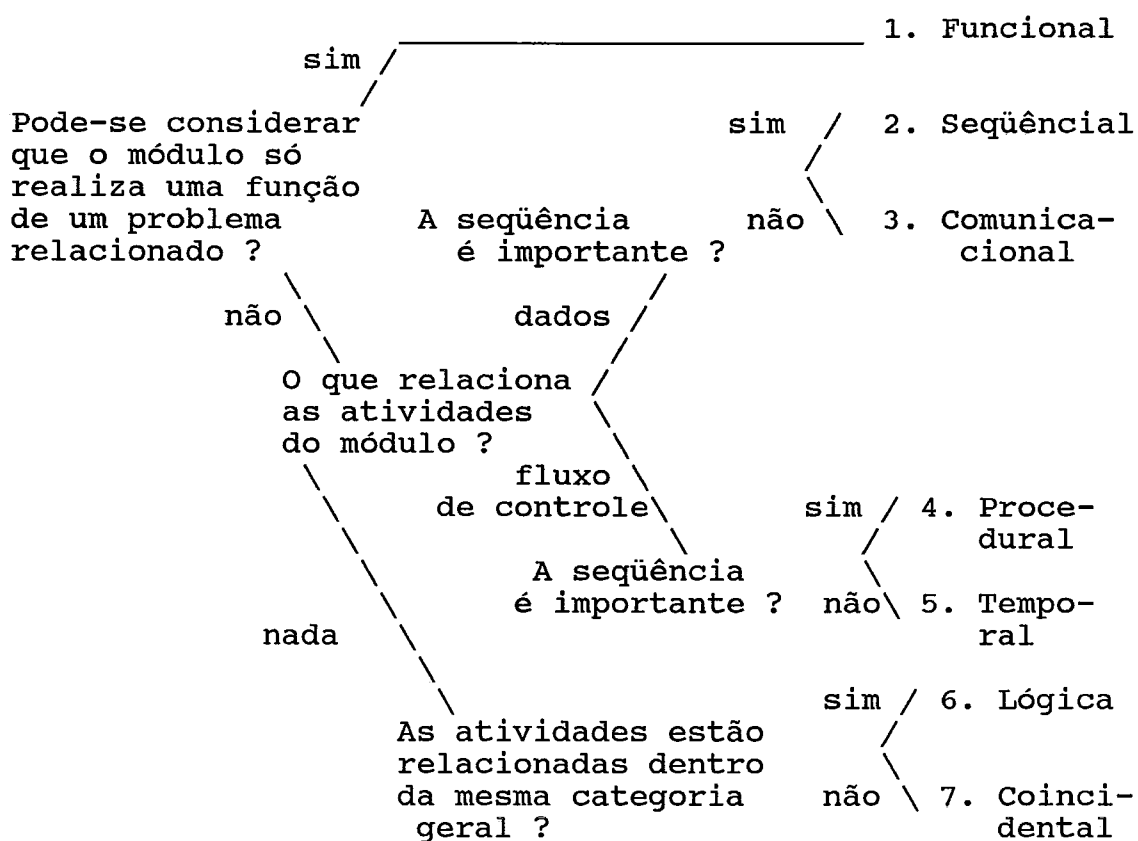


Figura III.14. - outra versão gráfica da Figura II.5. para melhor visualização e acompanhamento.

A partir desta árvore de decisão foram geradas as seguintes regras :

1. SE

(só uma parte do módulo é executado a cada chamada)

e (a seleção do trecho a ser executado é feita externa ao módulo)

e (o comando é passado ao módulo por flag)

ENTÃO

a coesão é lógica ou coincidental

2. SE

(existe parâmetro de INPUT que em chamadas diferentes assume significados diferentes)

ENTÃO

a coesão é lógica ou coincidental

3. SE

(a coesão é lógica ou coincidental)

e (funções que o módulo executa podem ser classificados como um mesmo tipo genérico de função)

ENTÃO

a coesão é lógica

4. SE

(a coesão é lógica ou coincidental)

e (funções que o módulo executa não podem ser classificados como um mesmo tipo genérico de função)

ENTÃO

a coesão é coincidental

5. SE

(a única relação entre as funções do módulo é terem que ser executadas juntas no tempo)

ENTÃO

a coesão é procedural ou temporal

6. SE

(parâmetro de saída não é transformação dos
parâmetros de entrada)

ENTÃO

a coesão é procedural ou temporal
ou lógica ou coincidental

7. SE

(coesão é procedural ou temporal)

e (a ordem de execução das funções do módulo é
única)

ENTÃO

a coesão é procedural

8. SE

(coesão é procedural ou temporal)

e (ordem de execução das funções do módulo
não é única)

ENTÃO

a coesão é temporal

9. SE

(funções do módulo compartilham dados de entrada
ou de saída)

ENTÃO

a coesão é funcional ou seqüencial
ou comunicacional

10. SE

(coesão é funcional ou seqüencial ou

comunicacional)

e (existe outro possível contexto onde o uso do módulo teria mesmos parâmetros de entrada e subconjunto dos parâmetros de saída)

ENTÃO

a coesão é comunicacional

11. SE

(coesão é funcional ou seqüencial ou comunicacional)

e (funções do módulo usam como entrada saídas de outras funções)

e (existe uma única seqüência de execução dessas funções)

ENTÃO

coesão é funcional ou seqüencial

12. SE

(a coesão é funcional ou seqüencial)

e (é possível sumariar tudo que o módulo faz como uma função única)

ENTÃO

a coesão é funcional

13. SE

(a coesão é funcional ou seqüencial)

e (não é possível resumir tudo que o módulo faz
como uma função única)

ENTÃO

a coesão é seqüencial

No ESE essas regras codificadas ficaram alocadas no
bloco Módulo no formato mostrado a seguir:

COESÃO01 IF

fm_execução_parcial is true and

fm_seleção_externa is true and

fm_flag is true

THEN

fm_lógica_coincidental is true

COESÃO02 IF

fm_significados_diferentes is true

THEN

fm_lógica_coincidental is true

COESÃO03 IF

fm_lógica_coincidental is true and

fm_mesmo_tipo_genérico is false

THEN

fm_coesão is "coincidental"

```
COESÃO04  IF
           fm_lógica_coincidental is true and
           fm_mesmo_tipo_genérico is true
           THEN
           fm_coesão is "lógica"
```

```
COESÃO05  IF
           fm_única-relação_tempo is true
           THEN
           fm_procedural_temporal is true
```

```
COESÃO06  IF
           fm_procedural_temporal is true and
           fm_ordem_execução is false
           THEN
           fm_coesão is "temporal"
```

```
COESÃO07  IF
           fm_procedural_temporal is true and
           fm_ordem_execução is true
           THEN
           fm_coesão is "procedural"
```

```
COESÃO08  IF
           fm_compartilham_dados is true
           THEN
           fm_func_seq_com is true
```

```
COESÃO09  IF
           fm_func_seq_com is true and
           fm_subconjunto_saida is true
        THEN
           fm_coesão is "comunicacional"
```

```
COESÃO10  IF
           fm_func_seq_com is true and
           fm_dados_em_seqüencia is true
        THEN
           fm_funcional_seqüencial is true
```

```
COESÃO11  IF
           fm_funcional_seqüencial is true and
           fm_única_função is false
        THEN
           fm_coesão is "seqüencial"
```

```
COESÃO12  IF
           fm_funcional_seqüencial is true and
           fm_única_função is true
        THEN
           fm_coesão is "funcional"
```

```
COESÃO13  IF
           fm_coesão is not known and
           (fm_mod_inicialização is true or
           fm_mod_terminação is true)
```

```
THEN  
    fm_coesão is "temporal"
```

```
COESÃO14 IF  
    fm_coesão is not known  
THEN  
    fm_coesão is "indefinida"
```

```
COESÃO15 IF  
    fm_coesão is "funcional" or  
    fm_coesão is "seqüencial" or  
    fm_coesão is "comunicacional"  
THEN  
    gm_coesão is fm_coesão and  
    coesão_check is true
```

Assim que se chega a um valor para "fm_coesão", dá-se o teste da coesão deste módulo por encerrado. Independente do tipo de coesão detectado, armazena-se o resultado do mesmo para uma consulta posterior, se necessário. Se for o caso de uma coesão considerada baixa, armazena-se também essa informação como um problema referente ao módulo que deverá ser tratado.

```
COESÃO15 IF  
    fm_coesão is not "funcional" and  
    fm_coesão is not "seqüencial" and  
    fm_coesão is not "comunicacional"
```

THEN

```
gm_coesão is fm_coesão and  
gm_prob_erro is fm_coesão and  
gm_prob_mod is fm_nome_mod and  
coesão_check is true
```

Onde os parâmetros recebem verdadeiro ou falso através de perguntas feitas diretamente ao usuário ou são ativados através de outras regras.

Com este exemplo pode-se perceber como funciona a idéia de trabalhar com vários níveis de conclusões, sendo que a coesão é também um sub-objetivo que faz parte da análise dos módulos e assim por diante. Para simplificar, colocou-se somente o nome do parâmetro com (u) se foi obtido através de uma pergunta e (r) se foi concluída através de uma regra.

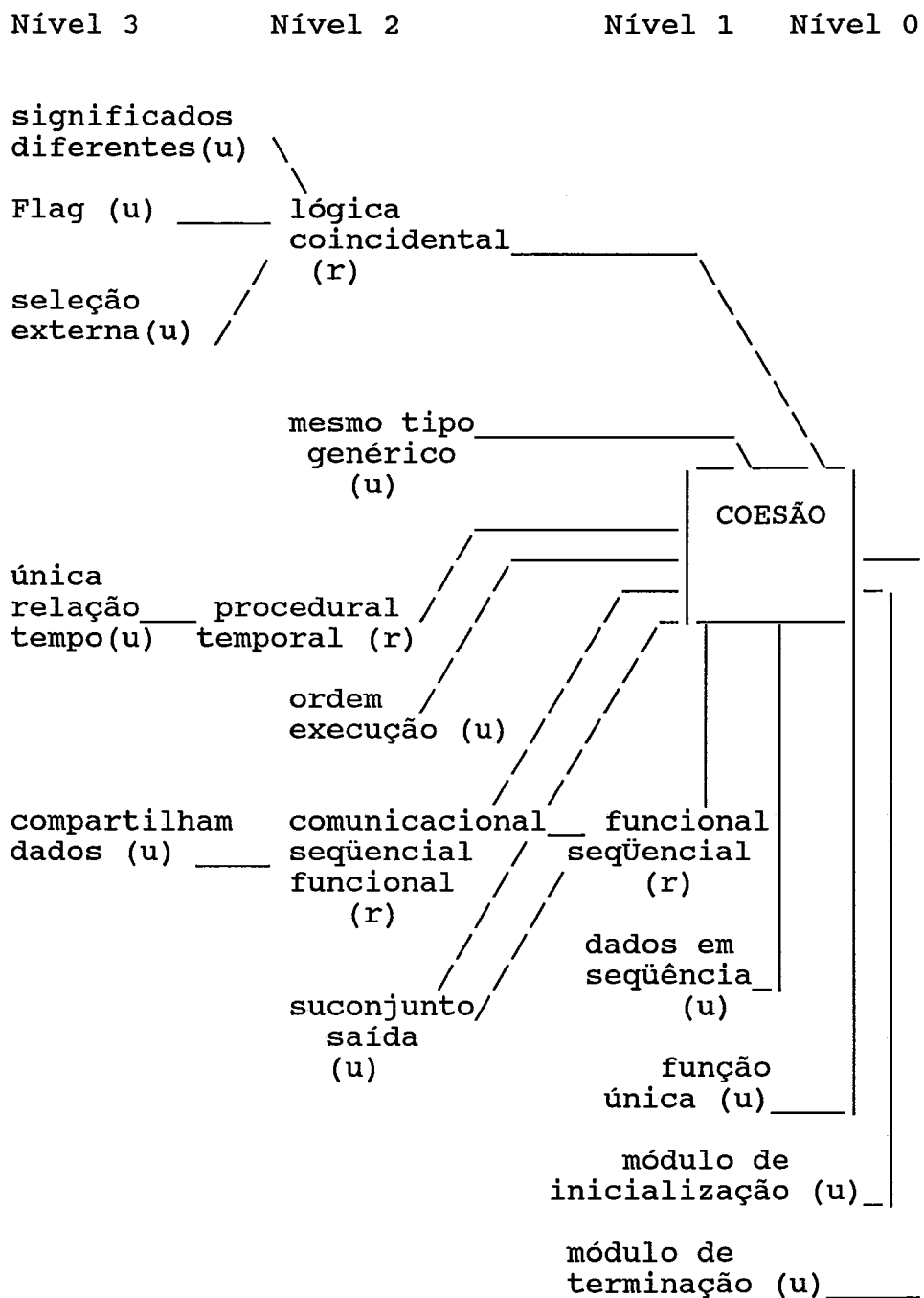


Figura III.15.

CAPÍTULO IV

AVALIAÇÃO, CONCLUSÕES E SUGESTÕES PARA EXTENSÃO

IV.1. Avaliação e Conclusões

A aplicação de Sistemas Especialistas em Engenharia de Software vem se mostrando cada vez mais viável e necessária. Essa integração complementa-se numa ferramenta que auxilia o usuário não só quantitativamente, agilizando o processo de construção do sistema, como qualitativamente, auxiliando na análise do mesmo.

Durante o desenvolvimento do WALK, passou-se por algumas fases críticas as quais serão comentadas a seguir. A primeira diz respeito a fase de aquisição do conhecimento. Como não há na literatura uma abordagem sistemática para a captura de conhecimento de um especialista ou mesmo diretamente dos livros especializados, escrever regras contendo as heurísticas se tornou uma tarefa extremamente demorada e muitas vezes difícil, mesmo no caso do Projeto Estruturado, onde pode-se encontrar diversas heurísticas para a análise da modularização, o problema não foi amenizado, pelo contrário. As heurísticas encontradas são muito vagas, pois estão baseadas em situações difíceis de serem determinadas. Imagine, por exemplo, que deseja-se estimar

a coesão de um módulo. As heurísticas referem-se à existência de mais de uma função no módulo para determinar que ele não é funcional. O problema de aplicar essas heurísticas se reduz ao problema de determinar a existência de mais de uma função no módulo, o que em si é um outro problema grande a se abordar.

O segundo ponto crítico refere-se ao uso de um ambiente de desenvolvimento de propósito geral. De um modo geral, o ambiente (ESE) utilizado pode ser visto como uma ferramenta muito útil ao desenvolvimento de SE. Suas principais virtudes estão no fato de ser um ambiente completo (possuindo facilidades de edição, verificação de consistência, execução de consultas e "trace"), de possuir uma estrutura de controle poderosa e de manipular informações aproximadas através de fatores de confiabilidade. Como principais deficiências estão a dificuldade de se representar informações complexas (como, por exemplo, grafos) e sua documentação, por demais descritiva e com pequena quantidade de exemplos. A dificuldade de lidar com informações complexas foi superada através do uso de rotinas externas para manipulação com Diagrama Hierárquico de Módulos.

O processo de construção foi bastante beneficiado pela utilização de um protótipo, MOTTA [26]. Durante essa experiência observou-se que é importante construir um protótipo de SE inicialmente, sem a participação direta do especialista (pequena interação apenas para indicar a

literatura, dar uma idéia do problema e orientar na determinação do domínio a ser atacado). Acredita-se que o aproveitamento do especialista no desenvolvimento de um SE é muito maior se o sistema já tiver "algum conhecimento". Ao invés de se interagir com o mesmo diretamente desde o início, constrõe-se um núcleo básico com regras e fatos obtidos através da literatura especializada.

As vantagens observadas na construção de um protótipo de SE com a interação com especialista humano, numa fase posterior do desenvolvimento do sistema são:

.sendo o estudo inicial baseado em livros, o tempo do especialista será poupado, pois ele não precisará participar da fase em que as informações são ainda elementares (isto quando esta fase puder ser obtida através da literatura especializada);

.desperta-lhe o interesse ao ver o protótipo funcionando com informações de sua área;

.entende-se com mais facilidade seu "jargão", facilitando-se assim a interação;

.direciona-se a aquisição do conhecimento uma vez que já se tem noção do que é preciso perguntar.

IV.2. Sugestões para Extensão

Uma das sugestões para continuação do projeto WALK no que se refere a implementação é o de transportá-lo para a linguagem PROLOG. Acredita-se que tal linguagem é mais adequada para problemas relacionados com grafos, como é o caso da modularização de projetos, lidando muito bem com hierarquias e relações. Aliás, chegou-se a transportar o WALK para um IBM PC compatível usando-se a linguagem PROLOG. Este transporte foi rápido, se comparado ao tempo total de desenvolvimento, já que a base de conhecimento já estava construída.

O WALK ainda é um sistema pouco inteligente, e sua interface com usuário é um pouco cansativa devido a excessivas interações durante sua execução. Contudo, pretende-se nas próximas etapas de desenvolvimento detalhar suas heurísticas junto ao especialista e aumentá-las em número, depurar e avaliar sua base de conhecimento através de casos de testes, ampliando desta maneira a inteligência do sistema.

Uma outra sugestão para tornar o uso do sistema mais amigável é dividir a consulta em etapas. Por exemplo: na primeira análise poderiam ser considerados apenas os erros primários, cometidos normalmente por iniciantes, "limpando-se" assim o projeto; na segunda, levar-se-ia em consideração os problemas comumente encontrados em projetos como a coesão, o acoplamento, etc; na terceira e

última análise, ter-se-ia a preocupação em otimizar o projeto. Desta maneira, diminuiria-se a quantidade de problemas analisados por consulta, o que a tornaria também mais rápida.

A última sugestão para continuação do WALK é sua integração com softwares tipo CASE (Engenharia de Software Assistida por Computador) formando um ambiente de apoio ao desenvolvimento de sistemas baseados em metodologias estruturadas, MOTTA [43].

Um exemplo a curto prazo seria a integração do WALK com um editor gráfico de DHM. Desta forma o desenvolvimento do projeto de sistemas tornar-se-ia automatizado desde sua elaboração até sua avaliação, além do mais, o usuário teria um ambiente amigável para trabalhar.

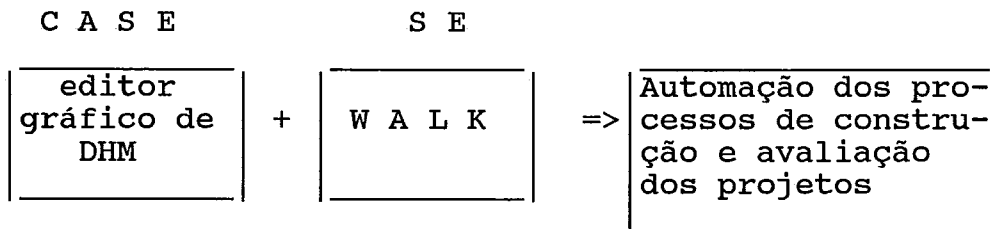


Figura IV.1.

Por fim, durante o desenvolvimento desse trabalho, ficou constatada a carência de especialistas em projeto de sistemas no mercado. Este fato reforça a necessidade da construção de ferramentas deste tipo, lembrando que um dos objetivos dos Sistemas Especialistas é exatamente difundir o conhecimento especializado onde ele é escasso.

CAPÍTULO V

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] DREYFUS, HUBERT L., O que os Computadores não podem fazer, A Casa do Livro Eldorado, Rio de Janeiro S.A., 1975.
- [2] HARMON, PAUL e KING, DAVID, Expert Systems - Artificial Intelligence in Bussiness, Wiley Press Book, 1985.
- [3] WATERMAN, DONALD A., A Guide to Expert Systems, Addison-Wesley Publishing Company, 1986.
- [4] FORSYTH, RICHARD, Expert Systems - Principles and Case Studies, Chapman and Hall Computing, 1984.
- [5] ALTY, JL e COOMBS, MJ, Expert System - Concepts and Examples, Ncc Publications, 1984.
- [6] RICH, ELAINE, Artificial Intelligence, McGraw-Hill International Book Company, 1983.
- [7] BUCHANAN, BRUCE G. e SHORTLIFFE, EDWARD H., Rule-Based Expert Systems : The Mycin Experiments of Stanford, Addison-Wesley Publishing Company, 1984.
- [8] JACKSON, PETER, Introduction to Expert Systems, Addison-Wesley Publishing Company, 1986.
- [9] THOMPSON, B. e THOMPSON, W., Finding Rules in Data, Revista Byte, 1986.
- [10] WINSTON, PATRICK H., Artificial Intelligence, Addison-Wesley Publishing Company, 1984.

- [11] HSU, CHUN YIN, Representação de Conhecimento usando Regras e "Frames", Tese de mestrado COPPE/SISTEMAS da UFRJ, 1988.
- [12] STEWARD, DONALD V., Software Engineering with Systems Analysis and Design, Brooks/Cole Publishing Company, 1987.
- [13] LUCENA, CARLOS, Inteligência Artificial e Engenharia de Software, Publicações Acadêmico-Científicas, PUC-RJ/IBM Brasil, Jorge Zahar Editor Ltda, 1986.
- [14] ADELSON, BETH e SOLOWAY, ELLIOT, The Role of Domain Experience in Software Design, IEEE Transactions on Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol.SE-11, No.11, pp.1351-1360, Novembro de 1985.
- [15] KANT, ELAINE, Understanding an Automating Algorithm Design, IEEE Transactions on Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol.SE-11, No.11, pp.1361-1374, Novembro de 1985.
- [16] STEIN, DAVID M. e KANT, ELAINE, The Roles of Execution and Analysis in Algorithm Design, IEEE Transactions on Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol.SE-11, No.11, pp.1375-1386, Novembro de 1985.
- [17] GANE, CHRIS, "Computer-Aided Software Engineering The Methodologies, The Products, The Future", Instituto Brasileiro de Pesquisa em Informática,

IBPI, Março, 1988.

- [18] YOURDON, EDWARD e CONSTANTINE, LARRY, Structured Design, Prentice-Hall, 1979.
- [19] ROCHA, ANA REGINA CAVALCANTI, Análise e Projeto Estruturado de Sistemas, Editora Campos, 1987.
- [20] PAGE-JONES, MEILIR, The Practical Guide to Structured Systems Design, Yourdon Press, 1980.
- [21] LOURENÇO, HERMÍNIO, Estações de Trabalho CASE, Dados & Idéias, No. 129, Março, 1989.
- [22] MARTIN, JAMES, A Revolução do CASE, REVISTA INFO, No. 77, Junho, 1989.
- [23] WARNIER, J.D., LCP - Lógica de Construção de Programas, Editora Campos, 1986.
- [24] JACKSON, M., "Constructive Methods of Program Design", in Software Design Strategies, Bergland, G, Gordon, R.D. (eds), IEEE Computer Society, 1981.
- [25] FREEMAN, PETER, "Strategic Directions in Software Engineering: Past, Present and Future", III Simpósio Brasileiro de Engenharia de Software, Recife, Outubro de 1989.
- [26] MOTTA, CLAUDIA; SILVEIRA, PEDRO; TELES, ANTÔNIO e SILVA FILHO, YSMAR V., "Desenvolvendo Sistema Especialista a partir de Protótipo", Anais do XX Congresso Nacional de Informática, vol. II, 1987.
- [27] STEVENS, WAYNE P., "Using Structured Design", A Wiley-Interscience Publication, 1981.
- [28] FAIRLEY, RICHARD E., "Software Engineering Concepts",

- McGraw-Hill Book Company, 1985.
- [29] IBM, Expert System Development Environment/VM, Reference Manual, setembro, 1985.
- [30] SILVA FILHO, YSMAR V.; TELES, ANTÔNIO e MOTTA, CLAUDIA, "Proposta de Um Sistema Especialista em Projeto Estruturado", Relatório Técnico, NCE0486, 1986.
- [31] SHIRAI, YOSHIAKI e TSUJII, JUN-ICHI, Artificial Intelligence - Concepts, Techniques and Applications, John Wiley & Sons, 1982.
- [32] YOURDON, EDWARD, Structured Walkthroughs, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.
- [33] RIBEIRO, HORÁCIO C. S., Introdução aos Sistemas Especialistas, Livros Técnicos e Científicos Editora S.A., 1987.
- [34] SELL, PETER S., Expert Systems - A Practical Introduction, MacMillan, 1985.
- [35] DAVIS, RANDALL e LENAT, DOUGLAS B., Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill International Book Company, 1982.
- [36] NOGUEIRA, DURVAL L., "Editor de Gráfico de Estrutura: Uma Ferramenta para Apoio ao Projeto Estruturado", Anais do I Simp"σιο Brasileiro de Engenharia de Software, 1987.
- [37] AGUIAR, TERESA; BLASCHEK, JOSÉ R.; NOGUEIRA, DURVAL e ROCHA, ANA REGINA, "Ferramenta Automatizada para apoiar as fases de Análise e Projeto", Anais do XX Congresso Nacional de Informática,

vol. II, 1987.

- [38] IBM, Pascal/VS - Language Reference Manual, 1985.
- [39] MARTIN, JAMES, Vantagens estratégicas, Ferramentas CASE integradas, INFO, No. 72, Janeiro, 1989.
- [40] BOEHM, BARRY W., Seven Basic Principles of Software Engineering, The journal of Systems and Softwares 3, 3-24 (1983), Elsevier Science Publishing Co, Inc. 1983.
- [41] HIRSCH, P.; KATKE, W.; MEIER, M.; SNYDER, S. e STILLMAN, R., Interfaces for Knowledge-base Builders Control Knowledge and Application-specific Procedures, IBM J. RES. DEVELOP., VOL 30, NO. 1, JANUARY, 1986.
- [42] SEYDEN, ERIC, "Systems Experts: Simuler L'Intelligence", Sciences & Techniques, França, 1978.
- [43] MOTTA, CLAUDIA e SOUZA, MARCOS G., "Uma proposta de integração de software tipo CASE com Sistemas Especialistas", Anais do II Simpósio Brasileiro de Engenharia de Software, p.113 a 122, Canela, RS, 1988.