UM SISTEMA DE APOIO À AVALIAÇÃO DE CUSTOS DE SOFTWARE

Carlos Alberto Soares Ribeiro

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

Aprovada por:

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

Prof. Jano Moreira de Souza, PhD

Prof. Doris Ferraz de Aragón, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 1989

RIBEIRO, CARLOS ALBERTO SOARES
Um Sistema de Apoio à Avaliação de Custos de Software. (Rio de Janeiro) 1989.
XII, 214p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1989)
Tese - Universidade Federal do Rio de Janeiro, COPPE.
1. Estimativa de custos de software
I. COPPE/UFRJ II. Título (série)

À Teresa

AGRADECIMENTOS

À Profa. Ana Regina pela orientação e amizade ao longo desses anos; ao Prof. Jano pelas sugestões e amizade.

Aos amigos e professores do programa de Engenharia de Sistemas, em particular à professora Sueli Mendes, pelas conversas esclarecedoras e aos colegas Vermelho, Raul, Ednéa, Cacilda, Marcus "Arataca", Luis Sergio, José Alberto, e Guilherme, pelo companheirismo.

Aos colegas da UFF, em especial aos Profs. John, Regina Pereira, Miriam e Rosangela, pelo apoio recebido.

À meu pai e irmãos, pelo estímulo e apoio sempre presentes.

À Teresa, especialmente, pelo seu amor, estímulo e participação.

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

UM SISTEMA DE APOIO À AVALIAÇÃO DE CUSTOS DE SOFTWARE

Carlos Alberto Soares Ribeiro

Dezembro de 1989

Orientadora: Ana Regina Cavalcanti da Rocha

Programa: Engenharia de Sistemas e Computação

Estimar os custos de um produto de software é uma das mais importantes e difíceis tarefas da Engenharia de Software.

Em função da ampla utilização de técnicas baseadas no julgamento de especialistas para estimativa de custos e em função dos possíveis problemas decorrentes da sua utilização, surgiu a idéia de se desenvolver um sistema de computação capaz de apoiar os profissionais que utilizam estas técnicas na elaboração de estimativas de custo.

Como o conhecimento em estimativa de custos não se encontra suficientemente sedimentado optou-se pelo desenvolvimento de um sistema especialista de suporte a decisão.

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

A SYSTEM TO HELP IN THE EVALUATION OF SOFTWARE COSTS

Carlos Alberto Soares Ribeiro

December 1989

Thesis Supervisor: Ana Regina Cavalcanti da Rocha

Department: Engenharia de Sistemas e Computação

Estimating the costs of a software product is one of the most important and difficult tasks of the software engineering area.

Considering both the large use of techniques based on expert judgment to estimate software costs and the problems that may arise when these estimations are only based on people's experience, we had the idea of developing a computer system capable of helping professionals who use these techniques to make software costs estimations.

Since the knowledge in software cost estimation is not mature enough we have decided to develop an expert decision support system.

ÍNDICE

LISTA DE FIGURAS

CAPÍTULO I

INTRODUÇÃO

Estimar os custos de um produto de software é uma das mais importantes e difíceis tarefas da Engenharia de Software.

É através da estimativa de custos de software que se procura entender e controlar os custos, tornando-se possível:[1]

- economizar dinheiro, uma vez que os custos com software estão cada vez maiores;
- aumentar a produtividade do pessoal encarregado de desenvolver e manter software;
- aumentar a qualidade do software produzido, e não apenas a quantidade.

Estimar os custos de software se torna uma tarefa ainda mais complexa quando a estimativa de custos precisa ser elaborada logo no início do desenvolvimento de um sistema, isto é, ao final da proposta de desenvolvimento, quando inúmeros fatores capazes de influenciar os custos são ainda desconhecidos.[2]

Com o objetivo de tentar solucionar este problema, diversos métodos de estimativa de custos vêm sendo desenvolvidos e aperfeiçoados. Atualmente, o método mais utilizado baseia-se no julgamento de especialistas. Para obter suas estimativas, especialistas em estimativa de

custos utilizam sua experiência, que é a peça fundamental para o bom funcionamento do método.

A maior vantagem deste método, que é o fato das estimativas serem realizadas com base na experiência dos profissionais, pode se tornar um risco. Um especialista, ao acreditar que um novo sistema é similar a outro já desenvolvido, pode não notar algumas diferenças que, em conjunto, tornam o novo sistema significativamente diferente. Uma outra situação, ainda mais grave, ocorre quando os profissionais responsáveis pela tarefa de estimar os custos de um sistema não possuem experiência com projetos similares.

Em função da ampla utilização desta técnica de estimativa de custos e dos possíveis problemas decorrentes da sua utilização, surgiu a idéia de se desenvolver um sistema de computação capaz de apoiar profissionais envolvidos na tarefa de estimar os custos de software, assim como aqueles que não têm experiência na área e desejam aprender.

Para desenvolver um sistema com estas características optou-se pela utilização dos recursos da linha de pesquisa dos sistemas especialistas de suporte a decisão (SESD).

SESD são uma junção de duas linhas: sistemas especialistas e sistemas de suporte a decisão. SESD se caracterizam por armazenar o conhecimento de uma determinada área, como os SE, permitindo no entanto que uma

série de decisões sejam tomadas por aquele que está utilizando o sistema, assim como um sistema de suporte a decisão. [3]

I.1 OBJETIVOS DO TRABALHO

Este trabalho tem por objetivo o desenvolvimento do protótipo de um sistema especialista de suporte à decisão com os seguintes componentes:

- interface com o usuário: o usuário e o sistema se comunicam através de perguntas elaboradas pelo sistema, que devem ser respondidas pelo usuário por meio de janelas que fornecem as respostas possíveis. Em função das respostas o sistema fornece diagnósticos a respeito do problema sendo avaliado.

- máquina de inferência: contém métodos de raciocínio utilizados pelo sistema na solução de problemas.

- base de conhecimento: contém o conhecimento (regras, fatos, heurísticas) do sistema. O conhecimento a ser adquirido nessa fase é principalmente o que consta da literatura.

- base de dados históricos: após cada sessão o usuário tem a possibilidade de guardar as informações por ele fornecidas. Estas informações são armazenadas em uma base de dados históricos para possível reutilização futura.

- módulo de explicação: fornece ao usuário o motivo que leva o sistema a efetuar determinada pergunta e

esclarece o significado das possíveis respostas.

- memória dinâmica: é composta pelas respostas fornecidas pelo usuário às perguntas efetuadas pelo sistema.

I.2 CONTEÚDO DO TRABALHO

Este trabalho divide-se em seis capítulos e um apêndice, cujos resumos vêm a seguir.

Capítulo I. Introdução.

Neste capítulo são apresentadas algumas razões que justificam o projeto de um sistema para estimativa de custos e o porquê da opção pelo desenvolvimento de um SESD.

Capítulo II. Estimativa de Custos: Os Métodos Mais Conhecidos, Suas Vantagens e Limitações.

No segundo capítulo apresenta-se um estudo baseado em pesquisa bibliográfica sobre modelos e métodos para estimativa de custos. Ao final do capítulo, algumas conclusões são apresentadas sobre as diferentes abordagens estudadas.

Capítulo III. Fatores de Produtividade na Estimativa de Custos.

O terceiro capítulo apresenta um estudo da bibliografia sobre os fatores de produtividade que constarão do sistema cujo projeto encontra-se no capítulo V. Os fatores de produtividade foram agrupados da seguinte

forma:

- fatores do produto;
- fatores do computador;
- fatores do pessoal;
- fatores do projeto.

Capítulo IV. Sistemas Especialistas.

Neste capítulo é apresentado um estudo a respeito do processo de construção de sistemas especialistas, quando estes sistemas devem ser desenvolvidos, sua aplicação à área gerencial e uma avaliação da viabilidade de se desenvolver um SESD para estimativa de custos.

Capítulo V. O Projeto do Sistema.

Este capítulo descreve o sistema proposto, discrimina o que será realizado em etapas posteriores. Apresenta-se a arquitetura do sistema, com a definição de cada módulo, e são comentados os aspectos relacionados à implementação do produto. Ao final do capítulo é apresentado o exemplo de uma sessão.

Capítulo VI. Conclusão.

Neste capítulo são apresentadas as conclusões finais sobre o sistema desenvolvido, e sugeridas algumas idéias no sentido de aperfeiçoá-lo em novas versões.

Anexo A. Manual do Usuário.

Neste anexo se encontra a descrição dos procedimentos necessários para a operação do sistema.

CAPÍTULO II

ESTIMATIVA DE CUSTOS: OS MODELOS MAIS CONHECIDOS, SUAS VANTAGENS E LIMITAÇÕES

II.1. INTRODUÇÃO

Algumas organizações, reconhecendo a seriedade do problema "estimar custos de software", costumam realizar uma série de estimativas ao longo do processo de desenvolvimento. A primeira estimativa ocorre ao se encerrar a proposta de desenvolvimento do sistema, quando já se possui algum conhecimento do produto a ser construído. Estimativas mais precisas são fornecidas ao longo da especificação de requisitos do novo sistema e finalmente na fase de projeto do sistema. Cada estimativa é um refinamento da estimativa anterior e é baseada nas informações adicionais obtidas durante os trabalhos de desenvolvimento do sistema.[2] É comum algumas estimativas serem realizadas em função das diversas soluções possíveis do problema, de forma que o usuário possa escolher a solução/custo que mais lhe convier.

Estas estimativas podem ser realizadas de duas formas: "top-down" ou "bottom-up". [2,4] E podem ser baseadas em um ou mais modelos de estimativa. (lineares, multiplicativos, etc.).

Esse capítulo tem como objetivo apresentar os dois tipos de estimativa, "top-down" e "bottom-up" e os

modelos e métodos que vêm sendo utilizados para estimar custos.

II.2. TIPOS DE ESTIMATIVA

II.2.1. ESTIMATIVA "TOP-DOWN"

São estimativas "top-down" aquelas que tratam dos custos a nível de sistema, tais como os recursos de hardware, software e de pessoal necessários ao desenvolvimento do sistema. São custos a nível de sistema: aqueles dispendidos com treinamento, controle de qualidade, integração, manuais do usuário, etc. Os custos com pessoal são estimados através do exame dos custos de projetos similares.

Estimativas "top-down" possuem a vantagem de tratar dos custos a nível de sistema, mas podem deixar de considerar fatores técnicos importantes de alguns módulos do sistema, além de não proverem uma base detalhada que justifique os custos estimados.

II.2.2. ESTIMATIVA "BOTTOM-UP"

São estimativas "bottom-up" aquelas que tratam, primeiramente, dos custos dos módulos ou subsistemas. Para se obter o custo total do sistema, somam-se as estimativas parciais.

A forma mais segura de se garantir que os custos a nível de sistema serão estimados em uma estimativa "bottom-up", é através da organização do trabalho de

desenvolvimento em um gráfico de desdobramento do trabalho ("work breakdown structure" (WBS)) que inclua não só a hierarquia dos componentes do software sendo estimado, mas também a hierarquia das atividades a serem executadas. A maior vantagem deste gráfico é tornar claro quais custos foram considerados na estimativa.

II.3. MÉTODOS DE ESTIMATIVA

II.3.1. BASEADOS EM MODELOS ALGORÍTMICOS

Métodos de estimativa baseados em modelos algorítmicos calculam o custo de um software através de equações matemáticas, em função de um certo número de variáveis. Os modelos algorítmicos mais comuns são: [4]

1. Modelos lineares.

Possuem a forma:

$$ESFORÇO = a_0 + a_1 x_1 + \ldots + a_n x_n ,$$

onde $x_1 \ldots x_n$ são as variáveis e $a_0 \ldots a_n$ os coeficientes.

2. Modelos multiplicativos.

Possuem a forma:

$$ESFORÇO = a_0 \; a_1 x_1 \; \ldots \; a_n x_n ,$$

onde $x_1 \ldots x_n$ são as variáveis e $a_0 \ldots a_n$ os coeficientes.

3. Modelos analíticos.

Possuem a forma: ESFORÇO = f(x , ... ,x), onde x ... x são as variáveis e f é uma função matemática não linear e não multiplicativa.

4. Modelos tabulares.

Possuem tabelas que relacionam as variáveis que representam os fatores de produtividade considerados pelo modelo a porções do esforço necessário ao desenvolvimento de um produto de software, ou a multiplicadores utilizados para ajustar o esforço estimado.

5. Modelos compostos.

Estes modelos incorporam as características de dois ou mais modelos citados acima.

A maior diferença entre os diversos métodos baseados em modelos algorítmicos está na utilização ou não, do número de linhas de código fonte (NLCF), como fator primário para a determinação do esforço necessário para se desenvolver um sistema.[5] A objetividade deste fator deu origem a diversos modelos, como por exemplo, SLIM e COCOMO. No entanto, a dificuldade de se estimar o NLCF, especialmente no início do processo de desenvolvimento, provocou o surgimento de outros modelos, como, Pontos por Função e mais recentemente o ESTIMACS. Os quatro modelos algorítmicos mencionados serão comentados nos ítens seguintes.

II.3.1.1 SLIM

Este método foi desenvolvido por PUTNAM e FITZSIMMONS no final dos anos 70. [6] Através da análise do ciclo de vida dos sistemas e da curva de Rayleigh, Putnam observou que a qualquer ponto da curva correspondia um esforço de pessoal naquele tempo. Veja figura 2.1 abaixo.

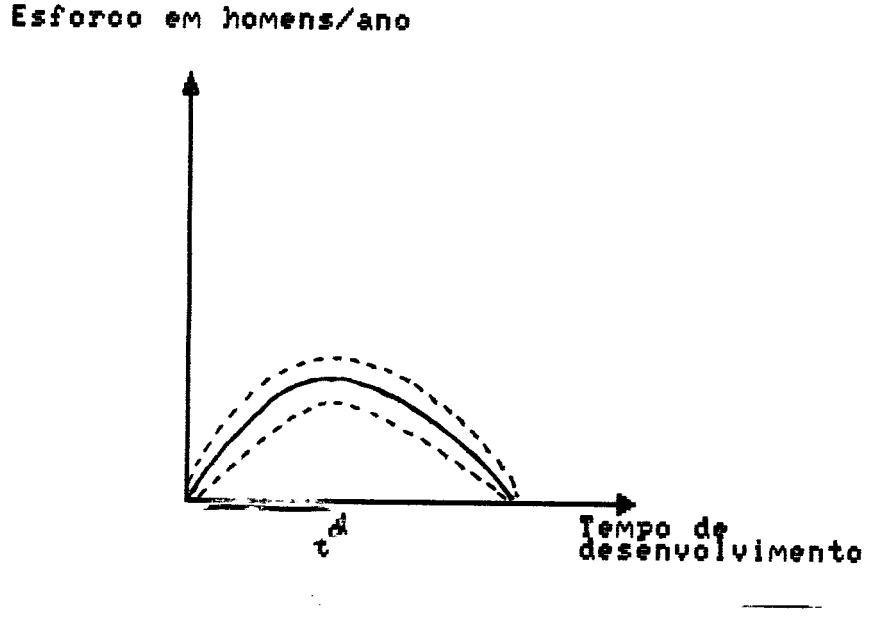

Figura 2.1 – Esforço x Tempo de Desenvolvimento. [6]

A curva de RAYLEIGH é representada pela equação:

$$Y = K / t_d^2 \cdot t \cdot e^{-t^2 / 2t_d^2}$$

onde,

Y – representa o esforço homem/ano em um determinado tempo.

K – é a área abaixo da curva de Rayleigh e denota o

esforço total, em homens/ano, empregado no sistema durante todo o seu ciclo de vida.

t_d - é o tempo necessário ao desenvolvimento de um sistema. É identificado como sendo o tempo em que o esforço é máximo.

O método se baseia na consideração de que existe uma relação entre o NLCF do projeto, o esforço empregado no seu desenvolvimento, o tempo de desenvolvimento e o estado da tecnologia aplicada ao projeto. A equação que descreve esta relação é:

$$T_s = C_k * K^{1/3} * t_d^{4/3}$$

onde,

T é o tamanho do sistema em número de linhas de código fonte entregues;

K é o esforço no ciclo de vida em homem/ano;

t_d é o tempo de desenvolvimento;

C_k é a constante para o estado da tecnologia.

Esse método sugere ainda, a utilização da técnica de PERT para calcular o NLCF entregues e define os momentos em que estas estimativas devem ser realizadas: a primeira, no final da fase de definição do sistema e as duas seguintes na fase de projeto.

A maior crítica que se faz a este método é o fato dele utilizar um único fator (C_k) para representar todos os fatores de produtividade capazes de influir no custo final. Daí se dizer que o método é eficaz apenas para se efetuar estimativas a nível macro, isto é, úteis apenas no início do desenvolvimento do sistema.

II.3.1.2. COCOMO

Este método (COnstructive COst MOdel) foi desenvolvido por BOEHM e publicado em 1981.[4] Baseado na análise de 63 projetos, Boehm desenvolveu um método algorítmico para calcular o esforço e o tempo necessários na construção de um produto de software.

O método COCOMO foi desenvolvido em três versões:

- básico,
- intermediário, e,
- detalhado.

O objetivo do COCOMO básico é fornecer estimativas rápidas, logo no início do processo de desenvolvimento. Nesta fase de um projeto, o conhecimento existente a seu respeito é bastante limitado, e por isso mesmo, o método utiliza como variável de entrada apenas a estimativa do número de instruções a serem programadas pelo pessoal de projeto e entregues como parte do produto final. O esforço necessário ao desenvolvimento de um sistema é calculado através de uma das equações abaixo, de acordo com o nível de complexidade do sistema:

Nivel de Complexidade	Esforço (homems-mes)
Software Aplicativo	HM = 2,4 * (MIFE) ** 1,05
Software Utilitário	HM = 3,0 * (MIFE) ** 1,12
Software do Sistema	HM = 3,6 * (MIFE) ** 1,20

Obs.: MIFE são milhares de instruções fonte entregues.

Através do COCOMO básico é possível, ainda, obter-se uma estimativa de custos das diversas fases do desenvolvimento.

Com objetivo de aperfeiçoar o método, BOEHM acrescentou ao sistema 15 fatores de produtividade e modificou levemente as equações de cálculo do esforço. Esta nova versão do método denomina-se COCOMO Intermediário. O conjunto de fatores de produtividade agregados ao método são agrupados em quatro categorias, conforme vem a seguir:

. Atributos do Produto

Confiabilidade

Tamanho do Banco de Dados

Complexidade

. Atributos do Computador

Restrição Quanto ao Tempo de Execução

Restrição Quanto ao Uso da Memória

Versatilidade da Máquina Virtual

Tempo de Resposta do Computador

. Atributos de Pessoal

Capacidade do Analista

Experiência na Aplicação

Capacidade do Programador

Experiência com a Máquina Virtual

Experiência com a Linguagem de Programação

. Atributos do Projeto

Modernas Práticas de Programação

Utilização de Ferramentas de Software

Estes fatores de produtividade dão uma maior precisão às estimativas do método, tornando-o mais adequado às etapas subsequentes ao início do processo de desenvolvimento, quando o conhecimento que se possui do sistema é bem maior. Para se obter o esforço total de desenvolvimento, o COCOMO intermediário prevê a estimativa do esforço por componente do sistema. Cada componente principal do sistema tem seu tamanho em linhas de código estimado, para em seguida serem aplicados os quinze fatores de produtividade mencionados. Da mesma forma que no COCOMO básico, é possível obter-se uma estimativa de custo das diversas fases do desenvolvimento.

O COCOMO Detalhado provê uma forma minuciosa de se preparar estimativas. O sistema a ser estimado é decomposto em subsistemas e em módulos. Os módulos são descritos através de seu tamanho em número de linhas de código e pelos fatores que variam a este nível: complexidade, capacidade do programador, experiência com a linguagem de programação e com a máquina virtual. Já os

subsistemas são descritos pelos demais fatores, que permanecem iguais para todos os módulos de um subsistema.

Uma segunda característica da versão detalhada do método é o fato de que os quinze fatores de produtividade passam a ser avaliados por fase do ciclo de desenvolvimento, uma vez que o impacto relativo de um fator se comporta de maneira diferente nas diversas fases de um projeto.

Assim, como todo método algorítmico para estimativa de custos, este método precisa ser adaptado ao ambiente onde vai ser utilizado. [7] Em outras palavras, o método precisa ser calibrado.

II.3.1.3. PONTOS POR FUNÇÃO

Este método foi desenvolvido por ALLAN ALBRECHT e publicado pela primeira vez em 1979. [8,9] Pontos por função trata do problema relativo à medição da produtividade no desenvolvimento/manutenção de software e representa uma forma alternativa de se calcular o NLCF de um produto.

O método de ALBRECHT toma por base as funções que um software executa com o objetivo de avaliar o seu tamanho. Fatores como experiência dos profissionais envolvidos, o uso de métodos estruturados para desenvolvimento de sistemas, o uso de ferramentas automatizadas, etc., não são considerados neste momento uma vez que não determinam o tamanho do sistema. Tais fatores

são responsáveis pelo nível de produtividade alcançado pelos desenvolvedores do produto.

A forma de cálculo do número de pontos por função de um software ocorre em duas etapas:

- contagem das funções do usuário, e,
- ajuste ao nível de complexidade do processamento.

A primeira etapa é efetuada contando-se o número de elementos que caracterizam as funções executadas pelo sistema. Em seguida multiplica-se este número pelo peso correspondente, conforme observa-se na figura 2.2:

Elementos do sistema	Peso
Entradas externas	4
Saídas externas	5
Consultas externas	5
Arquivos mestres	10
Interfaces externas	7

Figura 2.2: Peso dos elementos do método Pontos por Função.

Os resultados são somados para fornecer o total de pontos por função (FC).

A segunda etapa do cálculo é efetuada em função do reconhecimento de que o esforço necessário para se desenvolver sistemas varia de acordo com os seguintes fatores de complexidade do processamento:

- uso de teleprocessamento,
- processamento distribuído,
- objetivos de desempenho,
- restricão de configuração,
- volume de transações,
- entrada de dados on-line,
- transações on-line interativas,
- complexidade de processamento,
- projeto visando reutilizabilidade de código,
- facilidade de conversão e instalação,
- facilidade de operação,
- instalação em múltiplos locais, e,
- facilidade de manutenção e uso.

Cada um dos fatores acima deve ser avaliado utilizando-se uma escala de zero (nenhuma influência) a cinco (forte influência). Em seguida, os valores estimados são somados, o resultado da soma é multiplicado por 0,01, e somado a 0,65, conforme vem a seguir.

$$PCA = 0{,}65 + 0{,}01 \sum_{i=1}^{14} C(i),$$

onde,

PCA = Ajuste de Complexidade de Processamento (0,65 <= PCA <= 1,35), e C (i) = fatores de complexidade (0 <= C (i) <= 5).

Este fator é, então, utilizado na equação final:

$$FP = FC \ (PCA)$$

onde, FP = Pontos por Função, e FC = Pontos por Função previamente calculados.

O resultado final (FP) pode variar em mais ou menos 35% dos pontos originalmente calculados (FC).

Uma vez calculado o total de pontos por função do novo sistema, este número poderá ser utilizado na comparação com projetos já desenvolvidos, objetivando a estimativa de custos e o controle da produtividade do pessoal de desenvolvimento.

A maior crítica que se faz ao método é a sua inadequabilidade para avaliar sistemas com alta complexidade algorítmica e baixa complexidade de dados, como por exemplo, a maioria dos sistemas científicos. Segundo JONES [10], falta ao método a capacidade de quantificar as características estruturais do software, tais como, desvios, "loops", chamadas recursivas, etc. Pesquisas, ainda não concluídas, vêm sendo desenvolvidas no sentido de desenvolver um modelo híbrido que reúna os conceitos propostos por ALBRECHT e outras métricas de complexidade funcional que tratem dos problemas não resolvidos em Pontos por Função.

II.3.1.4. ESTIMACS

Assim como Pontos por Função, este modelo de estimativa de custos de software não requer do usuário o NLCF do sistema a ser analisado.

Trata-se de um sistema desenvolvido estritamente

com o objetivo de comercialização, e consequentemente, maiores detalhes do modelo como suas equações, não estão disponíveis. O modelo foi desenvolvido por H. RUBIN [11] e tem como dados de entrada 25 questões descritas em [12,13].

II.3.2. BASEADO EM JULGAMENTO DE ESPECIALISTAS

Alguns métodos baseados em julgamento de especialistas vêm sendo utilizados, assim como variações desses métodos. Tais métodos se caracterizam pela consulta a especialistas, que utilizam sua experiência e entendimento do projeto proposto para calcular seu custo.

A técnica de DELPHI é um exemplo de como avaliar custos usando a abordagem do julgamento de especialista. Seu objetivo é procurar eliminar os problemas que podem surgir quando pessoas se reúnem a fim de se obter uma opinião de consenso.

Para estimativa de custos, segundo FAIRLAY [2], esta técnica funciona assim:

1. Um coordenador entrega aos participantes as especificações do projeto e um formulário (figura 2.3) onde a estimativa será efetuada.

2. Após um estudo das especificações do projeto cada especialista preenche, anonimamente, um formulário com sua estimativa. Caso haja dúvidas os especialistas poderão fazer perguntas ao coordenador mas não deverão discutir suas estimativas.

3. O coordenador prepara e distribui um resumo das estimativas, e inclui neste resumo qualquer justificativa diferente, fornecida para a estimativa.

4. É fornecida nova estimativa, sempre anônima. Os especialistas que fornecerem estimativas fortemente divergentes do grupo poderão ser consultados, anonimamente, para explicarem a razão de sua estimativa.

5. Os processos 3 e 4 são executados até que se chegue a um consenso. Discussões em grupo não são permitidas durante todo o processo.

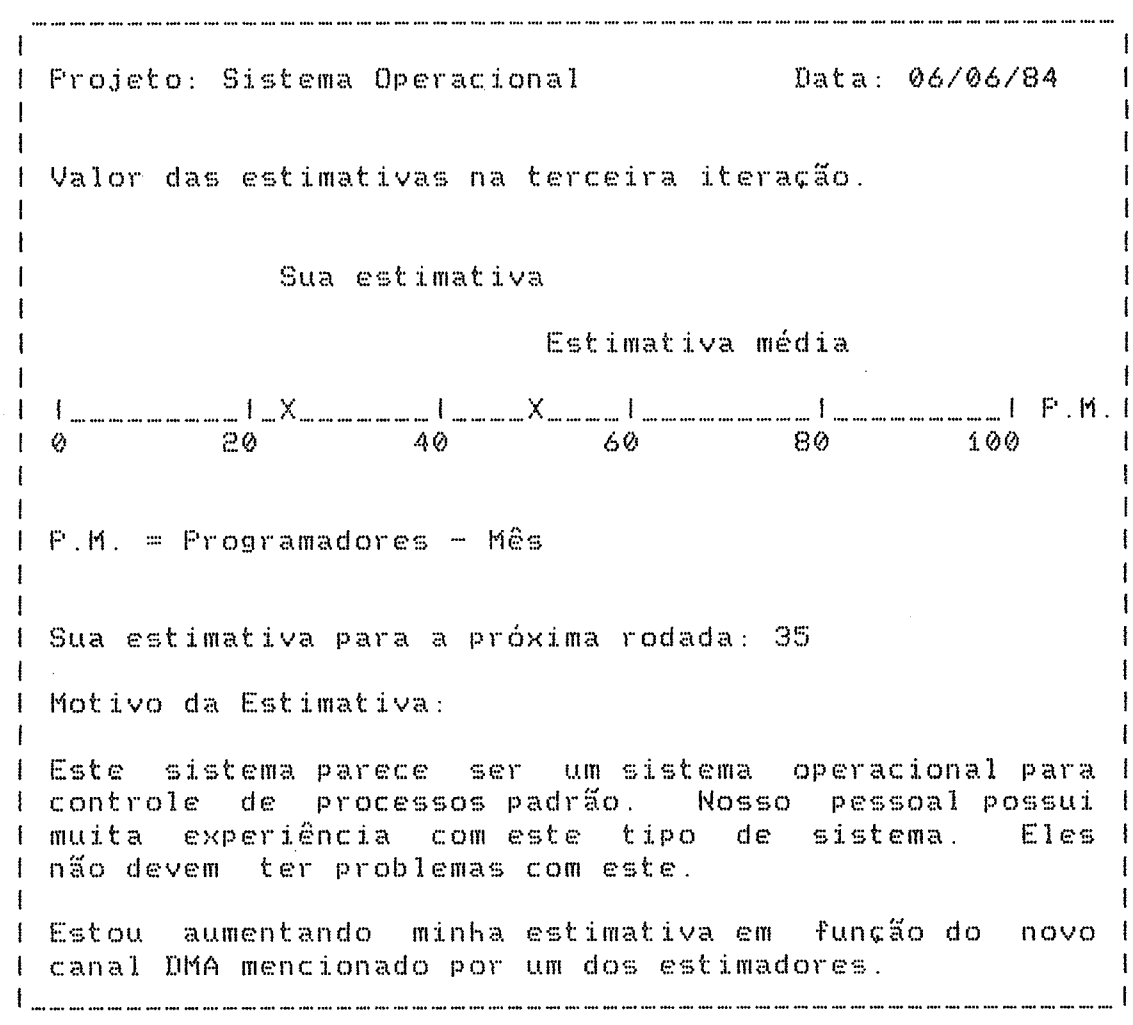

Figura 2.3: Formulário para estimativa usando a Técnica de DELPHI.[4]

II.3.3. OUTROS [4]

Os dois método descritos a seguir, se é que se pode dar-lhes o status de métodos, têm em comum o fato de só produzirem estimativas corretas por acaso. São eles:

- Método Parkinsoniano e
- Método "Price-to-Win".

O método Parkinsoniano baseia-se na lei de PARKINSON que diz que "o trabalho se expande para preencher o volume disponível".

Um exemplo de estimativa usando-se este modelo seria: "Este sistema de controle de vôo precisa caber em uma máquina de 65 K de memória real, logo, o tamanho do sistema será de aproximadamente 65 K. É preciso que o sistema esteje pronto em 18 meses. Como temos 10 pessoas disponíveis para trabalhar no projeto, o seu custo será de aproximadamente 180 homens-mês."

Este método além de gerar estimativas erradas, provoca a utilização de más práticas de desenvolvimento de sistemas.

O método "Price-to-Win" é utilizado quando se deseja ganhar uma concorrência a qualquer preço. Sua lógica é a seguinte: "que preço e que prazo de entrega devo pedir ao cliente para vencer esta concorrência? O preço deve atender à capacidade de caixa do cliente, mesmo que irreal. Deve ser menor do que o menor preço razoável. O mesmo ocorre para o prazo de entrega, se for importante

para o cliente."

II.4. CONCLUSÕES

A experiência tem demonstrado que não é nada fácil estimar o custo de desenvolvimento de software, assim como determinar o tempo necessário à sua construção. Segundo CUELENAERE, GENUCHTEN e HEEMSTRA [7], os fatores responsáveis por estas dificuldades são:

- o grande número de fatores que influenciam os custos e a duração de um projeto de software;

- a inexistência de definições precisas para estes fatores de produtividade;

- a dificuldade de se avaliar o impacto de alguns fatores no custo, assim como de se estabelecer, para cada fator, o significado dos possíveis níveis de atuação do fator, como: alto, médio, baixo;

- o fato de que vários fatores influenciam uns aos outros;

- a falta de dados históricos. O conhecimento e a experiência de desenvolvimento de software com características de produto e de projeto específicas e com a influência de fatores de produtividade somente existe na cabeça de poucas pessoas. Com a coleta de dados históricos torna-se mais fácil difundir este conhecimento e viável a calibração de um eventual software de estimativa de custos.

Como todos os modelos existentes (algorítmicos ou

não) para estimativa de custos, se baseiam em dados históricos sobre o desempenho de organizações no desenvolvimento de certos tipos de sistemas, uma avaliação de custos será tão precisa quanto for nossa habilidade de avaliar o nosso futuro desempenho a partir do desempenho passado. [2]

Comparados a outros métodos de estimativa de custos, os modelos algorítmicos possuem as seguintes vantagens: [4]

- são objetivos, isto é, não se deixam influenciar por fatores tais como o desejo de agradar ou o desejo de ganhar uma concorrência. Também são objetivos no sentido de que são objetivamente calibrados a experiências passadas.

- uma estimativa obtida hoje, também será amanhã, caso as mesmas respostas sejam fornecidas às mesmas questões.

Possuem no entanto as seguintes desvantagens:

- como são calibrados em função de experiências anteriores, fica a dúvida de até que ponto estas experiências serão representativas quando novos tipos de sistemas forem desenvolvidos, ou quando, por exemplo, novas técnicas ou novas arquiteturas de computadores forem utilizadas.

- não são capazes de tratar os fatores de produtividade que, embora previstos no modelo, ocorrem em

condições excepcionais.

- não são capazes de produzir estimativas melhores do que a habilidade do usuário do modelo de estimar o NLCF do produto a ser desenvolvido. (Naturalmente, este ítem só se aplica aos modelos que se baseiam no NLCF do produto.)

- não têm como corrigir ou compensar fracas estimativas do NLCF e dos valores relacionados aos fatores de produtividade considerados pelo modelo. Esta é uma característica comum a todos os modelos existentes.

Como "estimar custos de software" é uma tarefa baseada em dados históricos, é preciso colher informações sobre os atuais projetos em desenvolvimento para que elas possam ser utilizadas no futuro. Segundo FAIRLEY, [2] a consequência mais importante da utilização de modelos algorítmicos talvez seja a necessidade de se coletar e analisar dados com o objetivo de formar essa base de dados históricos.

Os quatro métodos baseados em modelos algorítmicos comentados neste capítulo, foram objeto de uma pesquisa desenvolvida por KEMERER [5]. KEMERER e sua equipe investigaram a diferença entre o esforço homem-mês real, necessário ao desenvolvimento de quinze sistemas comerciais, e o esforço estimado por cada um dos quatro métodos. Nesta validação empírica concluiu-se que:

1. Estes métodos, de modo geral, não funcionam

bem fora dos ambientes em que foram gerados. Variações de erro médias entre 65% e 772%, com muitos casos na faixa dos 500 a 600%, mostraram a necessidade de se obter dados históricos com o objetivo de se calibrar modelos algorítmicos antes de usá-los.

2. Os modelos não baseados no NLCF foram capazes de produzir melhores estimativas. Isto é, na média, Pontos por Função e Estimacs se saíram melhor do que os modelos COCOMO e SLIM. Este fato reforça a tese, já apresentada, de que Pontos por Função e Estimacs (um modelo semelhante ao Pontos por Função) são mais eficientes no cálculo do tamanho dos sistemas com alta complexidade de dados e baixa complexidade algorítmica.

Com relação aos métodos baseados em julgamento de especialistas, BOEHM [4] descreve as seguintes vantagens quando comparados aos demais métodos.

- Vantagens:

- um especialista é capaz de perceber as diferenças entre experiências com projetos anteriores e o desenvolvimento de novos tipos de sistemas, a utilização de novas técnicas ou o uso de novas arquiteturas de computadores.

- especialistas são capazes de perceber a ocorrência de fatores de produtividade em condições não usuais.

- Desvantagens:

- o julgamento de um especialista é rápido, mas difícil de racionalizar. E não será melhor do que seu nível de conhecimento, sua objetividade, seu otimismo ou pessimismo, ou seu desconhecimento de aspectos importantes do projeto.

Para compensar as desvantagens acima mencionadas é comum a formação de grupos de especialistas com o objetivo de obter uma estimativa que elimine, ou pelo menos diminua, estes problemas. Estimativas em grupo, embora bem documentadas e fáceis de analisar, são demoradas e difíceis de serem repetidas com frequência. Estimativas em grupo, quando não realizadas de forma adequada, podem fazer com que membros do grupo não sejam totalmente imparciais, devido por exemplo a questões políticas.

Observa-se que dentre todos os assuntos tratados pela Engenharia de Software, estimar os custos de software parece ser a tarefa mais sujeita a erros. Nenhuma estratégia ou modelo de estimativa de custos tem demonstrado ser o caminho ideal para a solução dos problemas da área.

As abordagens de estimativas de custos top-down e bottom-up, como se sabe, são complementares. Logo, para se obter a melhor estimativa possível, é preciso compensar suas vantagens e desvantagens, usando-se ambas, sempre que possível, independentemente do modelo de custos que se esteja utilizando.

As abordagens de estimativa de custos baseadas no julgamento de especialistas e em modelos algorítmicos são fundamentais e, na prática, mais de uma técnica deveria ser utilizada, e seus resultados comparados e reconciliados.

CAPÍTULO III

FATORES DE PRODUTIVIDADE NA ESTIMATIVA DE CUSTOS

III.1. INTRODUÇÃO

O objetivo deste capítulo é descrever os fatores de produtividade utilizados na construção do sistema proposto no capítulo V. Para a construção desse sistema foram considerados apenas os fatores mais bem documentados na literatura. Consequentemente, não foram considerados diversos fatores tais como restrições de segurança, restrições legais e separação geográfica dos locais de desenvolvimento.

Os fatores descritos neste capítulo estão agrupados em quatro categorias: atributos do produto, atributos do computador, atributos do pessoal e atributos do projeto, conforme vem a seguir:

Atributos do Produto

1. Tamanho do Sistema.
2. Complexidade do Sistema.
3. Confiabilidade Desejada.
4. Classe do Sistema.
5. Grau de Inovação do Sistema.
6. Tipo da Aplicação.
7. Linguagem de Programação.

Atributos do Computador

1. Restrição Quanto ao Tempo de Execução.
2. Restrição Quanto ao Uso da Memória.

Atributos do Pessoal

1. Experiência Profissional.
2. Nível de Satisfação no Trabalho.

Atributos do Projeto

1. Métodos Estruturados no Desenvolvimento de Sistemas.
2. O Ambiente de Desenvolvimento.
3. Métodos de Verificação e Validação.
4. Prototipagem Rápida.
5. Estrutura Organizacional da Equipe.
6. Reutilizabilidade de Código e de Projetos.

III.2. ESTUDO DOS FATORES DE PRODUTIVIDADE.

III.2.1. ATRIBUTOS DO PRODUTO.

1. Tamanho do Sistema.

Estudos em produtividade têm verificado (figura 3.1) que a produtividade no desenvolvimento de software apresenta algum aumento até o ponto em que o projeto se torna grande demais para os recursos aplicados. Este ponto de produtividade máxima depende de fatores, tais como: tipo e classe do sistema, linguagem de programação e habilidade do pessoal de desenvolvimento. [14]

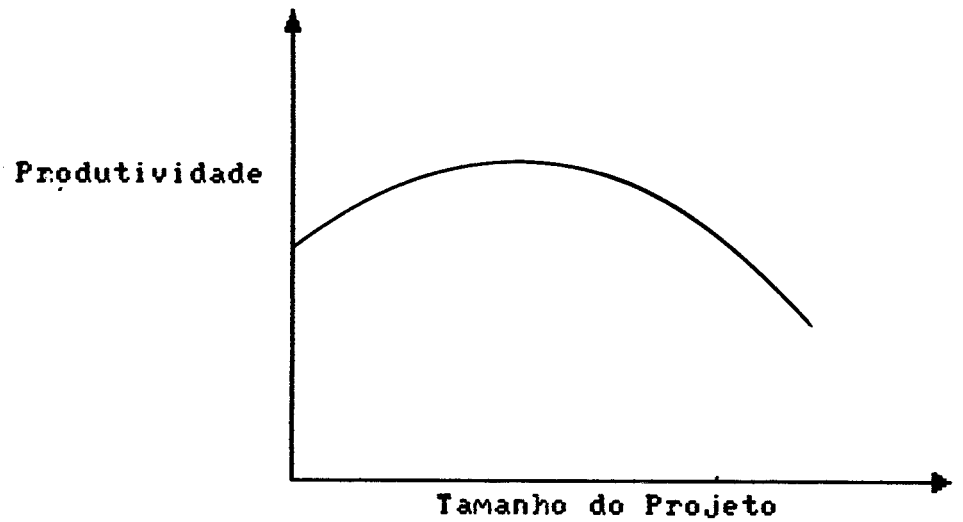

Figura 3.1: Produtividade e tamanho do projeto [14]

A partir deste ponto de produtividade máxima, a medida que os sistemas aumentam de tamanho surgem problemas relacionados com a produtividade. Estes problemas são em função do maior número de erros introduzidos durante as fases de análise e projeto e em função do aumento do volume de documentação necessária. [15]

JONES [10] identificou que, com o aumento dos sistemas, ocorre um aumento simultâneo de forma mais que linear dos seguintes ítens:

- complexidade dos dados tratados pelo sistema;
- complexidade das interfaces entre os módulos do sistema;
- necessidade de comunicação entre os membros que compõem a equipe de desenvolvimento;

- problemas de integração que em sistemas pequenos podem até mesmo inexistir;

- atividades de planejamento e gerenciamento;

- requisitos de desenvolvimento;

- aparecimento de defeitos (problemas relativos à complexidade e integração do sistema);

- volume de dados processados pelo sistema;

- necessidade de documentação.

Por outro lado, tem-se verificado que o impacto no custo das atividades de projeto detalhado, codificação e testes de unidades aumenta de forma linear com o tamanho dos sistemas. Consequentemente, a medida que os sistemas crescem estas tarefas passam a representar percentagens cada vez menores no cômputo geral do esforço efetuado.

Sabe-se, no entanto, que a maior parte do esforço gasto no desenvolvimento de grandes sistemas é aplicado na construção de novas funções com o objetivo de aperfeiçoar os sistemas existentes. [16]

O fato do esforço associado às atividades relacionadas à especificação de requisitos e à remoção de erros de análise, projeto e codificação aumentarem de forma mais que linear com o aumento dos sistemas, indica a necessidade de se utilizar técnicas que facilitem o gerenciamento do projeto e que minimizem os custos de eliminação de erros. A eliminação dos erros deve ser realizada através do uso de técnicas de prevenção e/ou pela melhoria da eficiência dos métodos de remoção dos erros.

Melhorar a produtividade requer uma estratégia que, dentro dos recursos disponíveis, dê prioridade aos elementos responsáveis pelos maiores custos. Para pequenos programas, onde a codificação é o maior custo, o uso de boas linguagens e ferramentas de apoio à codificação são eficientes. Entretanto, para grandes sistemas, onde os custos de eliminação de erros e de documentação são maiores que os custos de codificação, a estratégia indicada é composta de ferramentas que facilitem a geração de documentos, de bons métodos de prevenção e remoção de erros e de boas linguagens de programação, dado o grande volume de codificação necessário. [10]

2. Complexidade do Sistema.

De acordo com o método de complexidade ciclomática definido por McCABE [17], à medida que o número de desvios dentro de um programa aumenta, aumenta a sua complexidade. Para que os programas sejam testáveis e manuteníveis McCABE sugere a utilização da complexidade ciclomática, estabelecendo-se um limite superior para esta medida, além do qual o programa deve ser revisto. Segundo JONES [10], essa medida de complexidade tem sido um dos mais bem sucedidos indicadores de problemas em programas até agora descobertos.

Outro método muito utilizado no cálculo da complexidade foi definido por M. HALSTEAD. Por este método a complexidade é calculada através da soma dos operandos (variáveis ou constantes) e operadores de um programa. [18]

O fator complexidade é, talvez, o maior responsável pela baixa qualidade e produtividade no desenvolvimento de software. O problema principal, ainda sem solução, é se a medida de complexidade dos programas reflete a complexidade do problema original ou se ela reflete simplesmente a complexidade da solução encontrada.

Pesquisas realizadas por JONES [10] concluíram que, geralmente, a complexidade encontrada no código dos programas é devida à instabilidade dos requisitos dos sistemas ou a fatores acidentais que podem ser evitados. As principais razões da alta complexidade no código dos programas, são:

- a instabilidade dos requisitos dos sistemas;
- a falta de tempo para programar cuidadosamente;
- falta de familiaridade com a área de aplicação;
- complexidade combinatorial dos dados e interações funcionais;
- falta de conhecimento de programação estruturada.

THADHANI [19] sugere que o tempo gasto por um programador para desenvolver um programa seja levado em consideração na determinação do nível de complexidade do programa. Esta medida de complexidade pode ser útil à gerência para que se estabeleça que programas devem ser extensivamente testados.

Na avaliação da complexidade de um programa dois fatores devem ser considerados:

- Complexidade funcional - Considera o fluxo de controle de um programa e as ligações entre os módulos.

- Complexidade dos dados - Considera o número de relações "booleanas" entre os elementos de dados tratados pelo programa.

Estudos de casos realizados por JONES [10], mostram que a alta complexidade dos dados influi nos custos de forma bem mais acentuada do que a alta complexidade funcional. Sistemas com alta complexidade de dados possuem altos custos nas fases de testes e manutenção. Estes custos decorrem do mau funcionamento dos sistemas a partir de sua entrega aos usuários finais.

A pressão sofrida pela equipe de programação no sentido de terminar o trabalho rapidamente talvez seja a explicação mais comum para o excesso de complexidade encontrado no código dos programas. Neste caso, é preciso que as equipes de programação sejam dirigidas no sentido de produzirem trabalhos de qualidade em vez de apenas programarem rapidamente.

3. Confiabilidade Desejada.

Um software é considerado confiável quando é capaz de executar suas funções satisfatoriamente. É preciso, no entanto, definir, para cada sistema, o que é funcionar satisfatoriamente, uma vez que este conceito varia, em função das características específicas do

software e em função das necessidades de seu usuário.

Segundo BOEHM [4], o esforço adicional necessário para se desenvolver sistemas mais confiáveis é constante durante as três primeiras fases de desenvolvimento (análise, projeto e construção). Na fase de integração e testes, no entanto, o esforço adicional é consideravelmente maior. Além dos cuidados com controle de qualidade, é preciso testá-los exaustivamente e talvez verificá-los formalmente. O esforço adicional em testes pode incluir, por exemplo, o processamento de grandes volumes de dados reais, a simulação de falhas no hardware, a utilização do sistema por pessoal inexperiente, etc.

4. Classe do Sistema.

Em [10] é apresentada uma classificação de sistemas por necessidade de documentação, descrita a seguir. São definidas oito classes diferentes de sistemas organizados de forma que, quanto maior o valor da posição ocupada pela classe maior a necessidade de documentação. A partir da classe número quatro o volume de documentação costuma ser extremamente alto.

As classes mencionadas são:

1 - Sistemas internos sem fins comerciais.

Dentre todas as classes, esta é a mais comum. Os custos de documentação são, basicamente, os relacionados à geração dos documentos editados pelos analistas.

Manutenção é outro importante aspecto dessa classe de sistemas. Quando não realizada internamente e em um único local pode se tornar extremamente cara. O suporte local na instalação de produtos e os serviços de campo são os dois mais caros serviços da fase de manutenção.

2 - Sistemas para uso em comunidade ou objetivando reutilizabilidade.

Sistemas dessa classe são, normalmente, desenvolvidos lentamente nas fases de análise e projeto devido ao grande número de requisitos a serem atendidos simultaneamente.

3 - Sistemas cujos serviços são alugados a terceiros.

Os custos de documentação dessa classe de sistemas são geralmente altos se comparados aos demais sistemas internos. Para causar boa impressão os manuais do usuário são, quase sempre, escritos e editados de forma profissional.

4 - Sistemas desenvolvidos para serem alugados a clientes sob a forma de "leasing".

A documentação desse tipo de sistema é elaborada e composta de forma profissional, sendo, por isso mesmo, muito cara.

Quanto à manutenção, normalmente inclui serviço de campo e comunicação telefônica 24 horas por dia para que os usuários do sistema possam ter seus problemas resolvidos

ou encaminhados a qualquer momento. E para a instalação dos produtos, costuma haver suporte local, o que aumenta consideravelmente os custos de manutenção.

5 - Sistemas desenvolvidos para serem vendidos.

Um novo e relativamente caro aspecto no desenvolvimento dessa classe de sistemas é a proteção dos pacotes contra as cópias não autorizadas.

O nível de documentação dos sistemas à venda é bastante variado.

Embora a documentação de um programa via "HELP" deva custar aproximadamente 50% mais caro do que a simples datilografia do texto desejado, essa nova forma de documentação pode se mostrar bastante vantajosa ao se considerar a inexistência de gastos com impressão, encadernação e distribuição dos manuais do sistema para todos os usuários.

Suporte técnico na entrega do produto e serviços de campo podem existir ou não.

6 - Sistemas desenvolvidos sob contrato privado.

O nível de sofisticação dessa classe de programas é altamente variável.

Um importante fator de produtividade nessa classe de sistemas é o grau de conhecimento do usuário que controla o contrato. Usuários que não sabem o que querem e que desconhecem a dificuldade de se alterar sistemas de

computação após certas etapas podem prejudicar a produtividade de forma significativa.

A natureza do contrato é um fator importante nas classes 6, 7 e 8. É necessário examinar cuidadosamente todos os aspectos legais do contrato que se pretende assinar para, no futuro, não haver surpresas com multas pelo não cumprimento de obrigações contratuais, ações na justiça movidas pelo desenvolvedor do sistema que reivindica a propriedade do "software", etc. As convencionais definições de produtividade terão pouca importância se os custos de multas e ações na justiça forem significativos.

7 - Sistemas desenvolvidos sob contrato governamental.

Assim como os sistemas da classe 6, o nível de sofisticação dos sistemas desta classe também é muito variável. No entanto, o fator documentação é, geralmente, o que causa maior impacto na produtividade dos desenvolvedores. Embora nem todos os governos contratem a entrega de grandes quantidades de documentação, isso geralmente ocorre e em proporções maiores do que nos contratos com empresas privadas.

8 - Sistemas desenvolvidos sob contrato militar.

O nível de sofisticação dos sistemas dessa classe é, também, bastante variável.

Assim como nos sistemas da classe anterior, o

principal atributo dos sistemas desenvolvidos sob contrato militar é a documentação. Um dos frequentes motivos que justificam a preocupação dos órgãos governamentais com a quantidade e qualidade da documentação fornecida pela firma contratada é a futura responsabilidade desses órgãos em manter o sistema após sua entrega.

Concluindo, é preciso aperfeiçoar os métodos de documentação a fim de diminuir estes custos, especialmente na construção de sistemas de grande porte. Provavelmente mais da metade do trabalho realizado na construção de sistemas de grande porte da classe 8 é em documentação e menos de 30% do pessoal envolvido no projeto trabalha em codificação.

5. O Grau de Inovação do Sistema.

Não há dúvida que as aplicações implementadas pela primeira vez possuem um grau de incerteza muito superior àquelas desenvolvidas em terreno conhecido, como as reimplementações ou conversões.

Um fator comum às novas aplicações é o desconhecimento, por parte dos usuários, do que eles realmente desejam, ou melhor, o desconhecimento dos aspectos significativos das tarefas a serem automatizadas.

Examinando-se detalhadamente o desenvolvimento de uma nova aplicação, nota-se que as fases mais problemáticas são: a Especificação de Requisitos e a Especificação de Projeto. Isto decorre do desconhecimento, por parte de

usuários e programadores, de como resolver o problema de forma eficiente. Essa incerteza quanto aos requisitos do sistema costuma ser resolvida, embora de forma desgastante, na fase de codificação, através da reimplementação das funções cujos requisitos são modificados. Isto sugere a utilização de protótipos para a definição do sistema e a redução dos custos.

Na fase de manutenção, os custos costumam ser excessivamente altos em função das incertezas existentes na forma de se testar novas aplicações uma vez que são desconhecidas as situações reais capazes de testá-los completamente.

6. Tipo da Aplicação.

Em [10] é apresentada uma relação de 12 diferentes tipos de aplicação organizados em ordem crescente de dificuldade. Esta relação é descrita a seguir:

1. Sistemas não procedimentais.

São os sistemas desenvolvidos com linguagens do tipo:

- planilha eletrônica;
- gerador de programas ou de aplicações;
- linguagens de consulta a banco de dados, etc.

2. Sistemas "batch".

São sistemas do tipo folhas de pagamento e

contabilidade.

3. Sistemas interativos.

São sistemas controlados pelo usuário através de terminal.

4. Sistemas "batch" que utilizam sistemas de gerência de banco de dados.

Dois exemplos destes sistemas são dBase III e IMS.

5. Sistemas interativos com o uso de sistemas de gerência de banco de dados.

6. Sistemas científicos ou matemáticos.

São sistemas em que há sequências de cálculos relativamente longas e complexas.

7. Sistemas de suporte.

São sistemas como compiladores, ferramentas de teste e sistemas operacionais.

8. Sistemas de controle de processos.

9. Sistemas de comunicações ou de telecomunicações.

10. Sistemas de tempo real.

11. Processamento de gráficos e imagens.

12. Programas que utilizam técnicas de inteligência artificial.

Conforme se verifica na classificação acima, acredita-se que alguns tipos de sistemas são mais difíceis de construir do que outros. Isto ocorre em função de características intrínsecas a cada tipo e em função da tecnologia empregada. Embora não existam dados suficientes para comprovar esta afirmação, na prática isto se verifica através dos altos níveis salariais dos profissionais dos últimos tipos de sistemas.

7. Linguagem de Programação.

O efeito causado na produtividade por linguagens de baixo nível e pelas linguagens de altíssimo nível é bastante conhecido. No entanto, é muito difícil determinar os efeitos relativos causados pelas diversas linguagens de terceira geração, tais como Fortran e Pascal.

Estudos realizados por Boehm [4] concluíram que a produtividade não é alterada pela escolha de uma ou outra linguagem de terceira geração, exceto nos seguintes casos:

- quando os sistemas são grandes e complexos, e,
- no caso de organizações que desenvolvem muitos produtos.

Linguagens de baixo nível, tipo "assembler", provocam um impacto na produtividade muito acentuado. As fases do ciclo de vida tradicional mais prejudicadas com a utilização do "assembler" são a codificação, testes e manutenção, sendo as duas primeiras fases responsáveis por 70% do esforço empregado no desenvolvimento de um sistema.

Comparando com as linguagens de terceira geração, é comum necessitar-se de quatro vezes mais pessoal para dar manutenção a sistemas construídos em "assembler".

A forma mais eficiente de se melhorar a produtividade parece ser através do uso de geradores de aplicações. [15] Um gerador de aplicação consiste de um programa baseado no conhecimento de uma aplicação particular, que opera em um contexto pré-definido através da utilização de uma biblioteca de módulos. A especificação das aplicações é, geralmente, não procedimental ou definida através de "menus". [4] A grande maioria dos geradores de aplicação que vêm sendo atualmente comercializados tratam de aplicações comerciais em ambientes de bancos de dados, e são direcionados para a criação de "menus", geração de relatórios, formatação de telas de saída, pesquisas a dados, e várias outras operações como seleções, "joins", e operações matemáticas sobre os dados. [10]

No entanto, estes geradores, não costumam ser eficientes na utilização do "hardware" o que faz com que estas ferramentas sejam inadequadas ao processamento de grande volume de dados. Suas maiores desvantagens em relação aos programas desenvolvidos manualmente são: [20]

- maior tempo de execução;
- maior gasto de memória;
- dificuldade de interface com sistemas desenvolvidos em outras linguagens.

Consequentemente, as aplicações que tenham pequena vida útil e com pequeno volume de dados, podem ser adequadas.

Ao analisar-se o impacto na produtividade provocado por esses geradores ao longo do ciclo de desenvolvimento de software verificou-se que eles são, normalmente, mais eficientes nas fases de projeto e codificação e provocam, comparativamente, pequeno impacto nas demais fases.

Somente os produtos capazes de acelerar a produção de "software" em todas as fases do ciclo de desenvolvimento, devem ser considerados linguagens de quarta geração (L4G). Espera-se que através do uso dessas linguagens seja possível construir "software" em aproximadamente 10% do tempo e do custo necessários ao desenvolvimento com linguagens de terceira geração.

Alguns exemplos de L4G são:

- linguagens de consulta a banco de dados;
- planilhas eletrônicas;
- algumas ferramentas de propósito especial, como por exemplo, para a geração de gráficos.

As linguagens de quarta geração provocam:

- impacto significativo nas fases de projeto e codificação. A fase de projeto pode ser totalmente eliminada, sendo necessário, apenas, construir a aplicação.

- impacto variado nas fases de documentação interna, reparação de erros e manutenção.

- impacto mínimo nas fases de especificação de requisitos, integração e de documentação externa.

III.2.2. ATRIBUTOS DO COMPUTADOR.

1. Restrição Quanto ao Tempo de Execução.

Este fator representa a relação entre o tempo de máquina necessário para se executar uma aplicação e o tempo disponível.

A produtividade no desenvolvimento de "software" pode ser significativamente melhorada quando se tem uma velocidade de processamento e de memória real suficientes para que os construtores de "software" não precisem se esforçar além do necessário para instalar seus produtos.

Na medida que a restrição quanto ao tempo de execução vai se tornando um fator cada vez mais sério, vão aumentando concomitantemente os custos de desenvolvimento. Este aumento dos custos é mais ou menos por igual em todo o ciclo de vida, exceto nas fases de integração e testes quando a escalada dos custos é acentuada.

Com o objetivo de eliminar os problemas relacionados às restrições de memória e de tempo de execução, BOEHM [4] sugere três importantes critérios para a aquisição de "hardware":

1. Para que os custos sejam minimizados é preciso

possuir um "hardware" com capacidade 30 a 50% superior à capacidade absolutamente necessária.

2. Quanto maior for a relação custo do "software"/custo do "hardware", mais importante será a folga de "hardware".

3. É muito mais arriscado errar na procura de um "hardware" com pouca capacidade de processamento do que com um "hardware" com muita capacidade. Isto é especialmente importante, considerando-se a tendência de se estimar tamanhos de sistemas por baixo e ao fato de que os produtos de "software" costumam aumentar durante as fases de desenvolvimento e de manutenção.

2. Restrição Quanto ao Uso da Memória.

Assim como na restrição quanto ao tempo de execução, este fator representa a relação entre a memória primária necessária à execução de uma aplicação e a memória disponível.

Os custos decorrentes de restrições ao uso de memória primária se comportam de forma similar aos custos por restrições ao tempo de execução. Nas primeiras fases de desenvolvimento há um aumento de custos mais ou menos igual e nas fases de integração e de testes estes custos sobem acentuadamente devido, principalmente, aos altos custos da fase de testes.

III.2.3. ATRIBUTOS DO PESSOAL.

1. Experiência Profissional.

A literatura especializada é unânime em afirmar que a experiência profissional é um dos mais significativos fatores de produtividade no desenvolvimento de "software". Alguns especialistas chegam mesmo a considerá-lo o mais importante fator.

A experiência profissional pode ser de dois tipos:

1. experiência na área do problema a ser automatizado,

2. experiência na utilização das ferramentas e linguagens utilizadas no processo de automação.

Ambos os fatores são importantes na produtividade. No entanto, o impacto geral relativo à experiência na área do problema é um pouco mais acentuado, sendo as fases de análise e de testes as mais prejudicadas quando os desenvolvedores não possuem um bom conhecimento da área da aplicação.

O impacto da experiência na utilização das ferramentas e linguagens é, geralmente, muito intenso em equipes com cinco ou menos programadores sendo, no entanto, amenizado a nível de equipe ou departamento. [2]

BOEHM considera como atributos de pessoal, em seu método de estimativa de custos (COCOMO), os seguintes fatores de produtividade, todos relacionados à experiência

profissional:

- capacidade do analista;
- experiência com a aplicação;
- capacidade do programador;
- experiência com a máquina virtual, e,
- experiência com a linguagem de programação.

a) Capacidade do Analista.

Habilidade de análise, eficiência, esmero no trabalho, facilidade de comunicação e cooperação são os atributos que devem ser considerados na determinação da capacidade dos analistas. Estes atributos devem ser utilizados na avaliação da equipe de desenvolvimento como um todo e não indivíduo a indivíduo.

O impacto deste fator na produtividade se verifica mais acentuadamente nas fases de análise e projeto. Durante os trabalhos de codificação, testes e integração do sistema o impacto é um pouco menor.

b) Experiência com a Aplicação.

A experiência na aplicação é avaliada por Boehm em função do tempo de trabalho com o tipo de aplicação em questão. O impacto deste fator é mais acentuado na fase de análise do sistema. Da fase de projeto em diante o impacto vai sendo reduzido pouco a pouco em função da experiência adquirida durante o desenvolvimento do projeto.

c) Capacidade do Programador.

Os mesmos atributos utilizados na determinação da capacidade do analista devem ser considerados na avaliação da capacidade do programador. Neste caso, o impacto na produtividade ocorre, de forma constante, da fase de projeto em diante.

d) Experiência com a Máquina Virtual.

Com relação à experiência na máquina virtual (complexo de "hardware" e "software") BOEHM também utiliza o tempo como forma de avaliação do nível de experiência. O impacto deste fator é pequeno nas fases de análise e projeto e um pouco maior nas demais fases.

e) Experiência com a Linguagem de Programação.

O fator de produtividade "experiência com a linguagem de programação" não faz parte da máquina virtual em virtude de sua importância. O tempo é mais uma vez utilizado como forma de avaliação do nível de experiência. Nas fases de análise e projeto o impacto na produtividade é relativamente pequeno. Da fase de construção em diante o impacto na produtividade, aumenta substancialmente.

2. Nível de Satisfação no Trabalho.

Uma pesquisa elaborada por DITTRICH, COUGER e ZAWACKI [21] com mais de 1200 funcionários (963 analistas e programadores e 261 operadores) de nove empresas americanas revelou que a satisfação no trabalho está

significativamente relacionada à igualdade de tratamento recebida pelos funcionários. Um outro fator pesquisado foi a questão do que leva os profissionais a desejarem mudar de emprego. A resposta a esta pergunta foi, de longe, a falta de justiça nas regras de recompensa.

A principal conclusão desta pesquisa é que os motivos que provocam mudanças de emprego ou que afetam a satisfação no trabalho estão diretamente sob o controle das gerências das empresas.

Com relação à produtividade x nível salarial, constata-se que nas empresas onde o nível de produtividade é alto, costuma haver planos de recompensa nos quais os membros das equipes técnicas podem alcançar níveis salariais iguais aos alcançados pelos membros de carreiras gerenciais. Por outro lado, as empresas que possuem baixo nível de produtividade normalmente recompensam seus técnicos de forma inadequada, forçando-os a passar para atividades gerenciais ou a deixar a empresa para atingirem maiores salários.

O fato de um profissional ser um bom técnico não significa que ele será um bom gerente o que só vem a prejudicar a empresa que o promove desta forma. A segunda alternativa, que é permitir a saída do pessoal técnico qualificado da empresa por melhores salários traz consequências desastrosas a médio prazo. Após alguns anos, o nível técnico do pessoal contratado será baixo e a empresa, possuindo má reputação, terá dificuldades cada vez maiores de atrair pessoal de qualidade. [10]

III.2.4. ATRIBUTOS DO PROJETO.

1. Métodos Estruturados no Desenvolvimento de Sistemas.

Tem sido observado que o desenvolvimento de sistemas sem a utilização de métodos estruturados nas fases de análise e projeto levam à produção de sistemas relativamente baratos até sua aceitação, mas muito caros no que se refere a manutenção. Isto se explica, uma vez que os problemas que surgem após a entrega do produto ao usuário final têm sua origem, normalmente, nas fases de análise e projeto. Além disso, a ausência de um trabalho estruturado nessas fases faz com que seja muito difícil a elaboração de um plano de testes capaz de detectar os erros que tornam essa filosofia de desenvolvimento tão cara em manutenção.[10]

As companhias conscientes da importância das fases de análise e projeto no desenvolvimento de sistemas têm procurado, na Engenharia de Software, métodos estruturados adequados ao tipo de sistema em construção, com o objetivo de encurtar o tempo total de desenvolvimento. [20] Segundo BOEHM [4], se para um erro ser corrigido durante a fase de análise é preciso gastar, por exemplo, um dólar, para este mesmo erro ser corrigido durante a fase de projeto são necessários cinco dólares. Na fase de construção é preciso dez dólares e se o erro não é detectado até o sistema entrar em operação, este custo será de cem dólares. Logo, é importante detectar os erros o mais cedo possível, especialmente aqueles mais sérios cometidos nas fases de análise e projeto.

Mas o que fazer para melhorar o desempenho nessas duas fases iniciais? Parte da resposta está em estabelecer uma boa comunicação entre desenvolvedores e usuários. Existem muitos métodos estruturados para este fim. No entanto, com o desenvolvimento das linguagens de altíssimo nível e dos geradores de aplicação, a técnica de prototipagem rápida tem-se demonstrado bastante útil em determinados casos. Para os usuários, por exemplo, é bem mais simples entender o que um sistema fará através de um protótipo do que através da leitura do documento que descreve seu funcionamento. [20]

2. O Ambiente de Desenvolvimento.

Sabe-se que a utilização de melhores ambientes de desenvolvimento levam a um aumento de produtividade. Mas, que ferramentas devem ser utilizadas em cada caso? A falta de definição de quais ferramentas formam um bom ou excelente ambiente, assim como que ferramentas são consideradas fundamentais para que se tenha um ambiente satisfatório, são questões ainda não definidas.

Acredita-se que nos anos 90 estarão desenvolvidos ambientes que dêem suporte a todo o ciclo de vida dos sistemas. Esses ambientes serão capazes de integrar mais de cem ferramentas diferentes e incluirão novas gerações de sistemas especialistas para áreas como planejamento e estimativas.

Como ilustração, em [10] foram descritos os componentes de três ambientes de desenvolvimento de

software. No melhor ambiente, as ferramentas utilizadas são as mais avançadas do mercado. No ambiente intermediário as ferramentas utilizadas são aquelas típicas aos ambientes de desenvolvimento de sistemas comerciais. E o pior ambiente, responsável por um impacto bastante negativo na produtividade, é composto de ferramentas inadequadas e espaço físico de trabalho deficiente.

A descrição detalhada dos três ambientes mencionados vem a seguir.

1. MELHOR AMBIENTE

Este ambiente é composto dos seguintes elementos:

- suporte (integrado de textos e gráficos) às tarefas de análise, projeto e documentação em geral;
- boa biblioteca técnica com fácil acesso à informação;
- um terminal para cada programador/analista, em sala individual;
- boas ferramentas de depuração e uma biblioteca de testes automatizada;
- espaço de trabalho adequado;
- número adequado de salas para revisões, e,
- acompanhamento automatizado dos erros detectados.

2. AMBIENTE MÉDIO

O ambiente médio caracteriza-se pela presença dos seguintes componentes:

- utilização de editor para elaboração de textos mas sem nenhum suporte automatizado para a geração de gráficos;

- pequena biblioteca técnica;

- um terminal para cada três programadores/analistas, instalado em sala de terminais;

- algumas ferramentas de depuração e controle manual da biblioteca de testes;

- espaço de trabalho inadequado com uma sala para cada grupo de três ou quatro programadores/analistas;

- número insuficiente de salas para revisões, e,

- sistema manual de acompanhamento dos erros detectados.

Provavelmente, o elemento mais prejudial à produtividade neste ambiente é a falta de suporte adequado à documentação. No desenvolvimento de grandes sistemas a entrega da documentação geralmente se encontra no caminho crítico para a entrega do software.

3. PIOR AMBIENTE

Este ambiente de desenvolvimento é composto dos seguintes elementos:

- utilização de máquinas de datilografia na documentação;

- inexistência de biblioteca técnica ou qualquer fonte de referência;

- um terminal para cada grupo de quatro ou mais programadores;

- número insuficiente de ferramentas de depuração e biblioteca de testes controlada manualmente;

- inexistência de escritórios para o pessoal técnico, que exerce suas atividades em uma sala de trabalho comum;

- inexistência de salas para revisões, e,

- inexistência de um sistema de acompanhamento dos erros detectados.

Neste último ambiente, o elemento mais prejudicial à produtividade também é a falta de suporte adequado à documentação. Esta deficiência, entre outras, faz com que seja necessário um aumento de recursos humanos em mais de 100% para que uma tarefa seja realizada em um tempo razoável.

3. Métodos de verificação e validação.

A tarefa de remoção dos erros inadvertidamente introduzidos em um "software" durante a sua construção é uma das mais caras do processo de desenvolvimento. Documentação, codificação e remoção de erros estão, geralmente, entre os três maiores custos. Logo, é preciso cuidado na escolha dos métodos de remoção e prevenção de erros para que esta tarefa não se torne ainda mais cara. Por outro lado, é preciso lembrar que o sucesso de um "software" está diretamente relacionado à sua qualidade, e qualidade se obtém através de um bom processo de desenvolvimento que inclui bons métodos de prevenção e remoção de erros.

A quantidade de erros potencialmente existente em um produto de "software" decorre de uma série de fatores. Dentre eles pode-se destacar:

- o grau de inovação do sistema;

- a experiência profissional e o conhecimento da aplicação por parte do pessoal de desenvolvimento;

- o conhecimento dos métodos sendo utilizados para análise, projeto e codificação.

Além dos métodos de revisão conhecidos para as tarefas de análise, projeto e codificação, outros tipos de revisões [22] incluem documentação, dados e planos de testes, materiais de treinamento, procedimentos e padrões, etc.

Os métodos mais eficientes que vêm sendo utilizados nas fases de análise e projeto são a inspeção e a prototipação. O exame pessoal do próprio trabalho possui baixa eficiência uma vez que apenas 30% dos erros remanescentes são detectados. Prova de correção é o método mais controvertido para a remoção de erros de projeto. Existem muitas referências a este método na literatura, mas nenhuma demonstra que esta técnica é mais eficaz na remoção de erros do que os métodos convencionais.

A figura 3.2, a seguir, apresenta os métodos mais utilizados na remoção e prevenção de erros, e sua eficiência modal.

Métodos	Eficiência modal
1. Análise pessoal do projeto e da documentação	35%
2. Revisão informal do projeto (em grupo)	40%
3. Inspeção do projeto	55%
4. Prototipação	65%
5. Inspeção do código	60%
6. Análise pessoal do código	40%
7. Teste de unidade	25%
8. Teste funcional	35%
9. Teste de integração	45%
10. Teste de campo	50%

Figura 3.2: Métodos de remoção e prevenção de erros. [10]

A revisão informal caracteriza-se pela não utilização de procedimentos formais com o objetivo de encontrar e documentar os erros detectados.

Quanto à inspeção, trata-se de um método altamente eficiente utilizado na remoção dos erros de qualquer produto criado através de um processo de desenvolvimento cujo resultado possa ser visto e lido. Em informática, este método é normalmente utilizado para se verificar a correção dos produtos gerados nas fases de análise, projeto e codificação. Os participantes de uma sessão de inspeção são o autor do projeto ou do código, o

leitor (quem parafraseia o projeto ou código para os demais participantes), o examinador (analisa o projeto ou código com um ponto de vista crítico) e o moderador (quem conduz a inspeção - não deve estar envolvido no trabalho sendo inspecionado, mas deve ter experiência no assunto). [23] Com o objetivo de ganhar alguma eficiência na inspeção, é comum definir-se um certo número de questões a examinar. Neste caso, esta lista funciona como um "checklist" contendo as possíveis falhas do sistema. [22]

A realização de testes é o método mais utilizado na remoção dos erros de codificação. Este método, no entanto, quando utilizado sozinho possui baixa eficiência. A eficiência acumulada da série composta por testes de unidade, de funções, e de integração pode totalizar um valor de eficiência modal inferior a 75%. Isto é, menos de 75% dos erros de codificação potencialmente existentes são detectados através desta sequência de testes.

A realização de testes com dados reais é um método bastante eficiente mas se esta técnica não for utilizada em conjunto com outras nas fases de análise e projeto, sérios problemas poderão ser descobertos tarde demais. [10]

A forma mais eficiente de se remover erros de codificação é através de inspeções de código e através de prototipação do código, com uma linguagem interpretada ou uma L4G do tipo "linguagem de consulta" caso se trate de um sistema de informações gerenciais.

Para se obter boa eficiência na correção dos erros de um sistema é aconselhável a manutenção de estatísticas do número de erros detectados em cada módulo. Estudos realizados na IBM revelaram que os erros detectados em um sistema não estão espalhados igualmente por todos os seus módulos, e sim agregados em um pequeno número de módulos. A correção dos erros contidos nesses módulos, é, geralmente, muito cara. Através das estatísticas mencionadas seria fácil identificar quais os módulos que precisam ser intensamente inspecionados e testados, principalmente no caso de sistemas externos, que exigem suporte na entrega do produto e manutenção de campo.

De acordo com JONES [10], no balanceamento do cronograma, custos e recursos disponíveis contra o fator qualidade, a abordagem mais bem sucedida para verificação e validação deve ser uma combinação de:

- Inspeção das funções críticas do projeto;
- Prototipagem rápida das principais saídas do sistema para que o usuário possa ver os resultados;
- Revisões e testes.

O tempo gasto em inspeções e prototipagem seria compensado na fase de testes em função do pequeno número de erros a reparar.

Concluindo, qualquer série de métodos que omita revisões e inspeções de projeto terá sérios problemas na fase de testes. Se o sistema tiver sido desenvolvido para ser vendido, terá necessidade de no mínimo uma série

composta por cinco métodos de remoção e prevenção de erros com uma eficiência média.

4. Prototipagem Rápida.

Segundo GILHOOLEY [24] o impacto da prototipagem rápida no desenvolvimento de sistemas se verifica das seguintes maneiras:

- A utilização da técnica de prototipagem rápida no desenvolvimento de "software" facilita intensamente a participação do usuário na fase de análise do problema. Nesta fase, a comunicação entre usuários e desenvolvedores fica bastante prejudicada em função do alto grau de abstração presente na especificação do novo sistema. Com a prototipação aumenta a comunicação e, consequentemente, a aceitação do novo sistema por parte do usuário. Esta aceitação se torna ainda mais fácil com a inclusão, no protótipo, de um comando tipo "HELP" para cada função.

- A filosofia de prototipagem rápida associada a ferramentas como L4G e geradores de aplicações aumentam a produtividade de forma significativa. E a incorporação de bancos de dados relacionais e dicionários de dados a estas ferramentas simplificam substancialmente as manutenções eventualmente realizadas.

- O dicionário de dados e o banco de dados mencionados provêem independência de dados e suportam a extração e manipulação dos dados em contextos diferentes da aplicação em desenvolvimento.

Como regra geral, todos os sistemas com mais de 5000 linhas de código, que necessitam interagir intensamente com seus usuários finais são bons candidatos à prototipação. Se o sistema manipula pequeno volume de dados, é possível que o protótipo gerado possa evoluir diretamente para o produto acabado. Por outro lado, se ele manipula grande volume de dados, seu desempenho em uma linguagem não procedimental pode ser inadequado. Nestes casos, os protótipos tendem a ser jogados fora e o sistema reimplementado em uma linguagem de terceira geração ou "assembler", de forma a atender aos objetivos de desempenho necessários.

Mesmo quando o protótipo é jogado fora a atividade de prototipação é valiosa. Se o tempo e os recursos gastos na prototipação totalizarem em torno de 5 a 10% do total planejado para o desenvolvimento, é comum terminar-se o sistema ligeiramente mais cedo e dentro do orçamento, mesmo após a absorção dos custos e do tempo gasto na construção do protótipo. Este fato decorre da eficiência da prototipação em minimizar as mudanças nos requisitos dos sistemas ocorridas durante o processo de desenvolvimento. [10]

Um estudo realizado recentemente por BOEHM, GRAY e SEEWALDT [40] indicou que os sistemas desenvolvidos com prototipagem rápida consumiram, na média, apenas 55% do esforço necessário ao desenvolvimento de sistemas de forma convencional.

Uma vantagem adicional, introduzida por esta técnica, é a diminuição do esforço necessário na fase de manutenção. Esta diminuição decorre da eficiência da prototipagem rápida na determinação precisa dos requisitos dos sistemas. Desta forma um protótipo pode ser visto como uma poderosa ferramenta na prevenção de erros.

5. Estrutura Organizacional da Equipe.

Há várias estruturas organizacionais propostas na literatura. No entanto, as mais utilizadas em ambientes de desenvolvimento de "software" são a hierárquica e a matricial.

Geralmente, no desenvolvimento de grandes projetos de software (mais de 50.000 linhas de código fonte), 30% dos projetos gerenciados hierarquicamente têm problemas de cronograma e de custos e, aproximadamente, 50% dos projetos gerenciados de forma matricial sofrem os mesmos problemas. Obviamente, outros problemas que nada têm a ver com a estrutura básica organizacional são importantes. No entanto, a organização matricial parece ser, de modo geral, menos eficiente do que a hierárquica. [10]

A organização matricial parece induzir a um certo nível de confusão e ambiguidade nos grandes projetos de "software", o que normalmente provoca um aumento do tempo de desenvolvimento e dos custos. A estrutura matricial de gerência faz com que o número de canais de comunicação entre os gerentes das diversas áreas aumente

geometricamente e não aritmeticamente como ocorre na organização hierárquica.

6. Reutilizabilidade de Código e de Projetos.

Nos últimos tempos a demanda por "software" tem aumentado 40% ao ano e as áreas que necessitam de "software" têm se diversificado intensamente. Para que seja possível atender a esta crescente demanda tem-se trabalhado no sentido de "industrializar" a produção de "software". Acredita-se que a forma mais eficiente de se amenizar os altos custos de desenvolvimento, especialmente dos grandes sistemas, é através da reutilização de código e de projetos padrão. [25]

A publicação de janeiro de 1986 da EDP ANALYZER [15] apresenta a experiência da Hartford Insurance Company que no início de 1980 formalizou a reutilização de módulos de programas. No primeiro trimestre de 1984, 31% do código desenvolvido na companhia foi reutilizado, em vez de desenvolvido. No segundo trimestre este percentual foi de 29%. No terceiro trimestre, foi de 22% e no último trimestre, 30%. Na média, estimou-se um aumento de produtividade, naquele ano, em torno de 5 a 7%.

Os custos de implantação desta filosofia são altos, mas os resultados a nível de esforço e de tempo de desenvolvimento são compensadores. Outro benefício desta filosofia de trabalho é o grande impacto provocado na qualidade do "software" gerado.

Neste capítulo procurou-se comentar os fatores de produtividade mais bem documentados na literatura, e, por isso mesmo, utilizados em diversos produtos de estimativa de custos de software. A grande maioria dos produtos mencionados procuram estimar o custo de sistemas através da avaliação do impacto individual de cada um dos fatores de produtividade, no custo.

O sistema proposto no capítulo V deste trabalho procura identificar quais são os fatores de produtividade, que, uma vez combinados, acarretam aumento nos custos de desenvolvimento. Estes fatores foram dispostos na forma de regras objetivando a construção de um SESD. O próximo capítulo apresenta um estudo na área de sistemas especialistas.

CAPÍTULO IV

SISTEMAS ESPECIALISTAS

IV.1. INTRODUÇÃO

A inteligência artificial (IA) é a área da ciência da computação que tem relação com o projeto de sistemas computacionais inteligentes, isto é, sistemas que exibem as características que associamos à inteligência humana, como o entendimento da linguagem falada e escrita, a capacidade de aprender, de raciocinar, de resolver problemas, etc. [26].

A inteligência artificial teve origem nos anos 50, quando cientistas da área começaram a resolver problemas através do uso de programação simbólica. Os primeiros resultados em dedução automática e resolução de problemas provocaram grande euforia e otimismo. A princípio, as pesquisas foram dirigidas no sentido de encontrar metodologias capazes de resolver problemas de propósito geral (métodos fracos aplicáveis a domínios não específicos). Entretanto, tais métodos nunca foram totalmente eficazes. Quanto mais classes de problemas um único programa podia manipular, mais fraca era a solução de um problema individual.

Nas últimas décadas os pesquisadores da área de IA têm aprendido a apreciar o grande valor do conhecimento de domínios específicos como base para a solução de

importantes problemas. Nesse sentido, se compreendeu que os computadores só seriam capazes de resolver os problemas hoje tratados por especialistas se lhes fossem fornecidos os conhecimentos que tais especialistas detêm, uma vez que, embora os computadores tenham muitas vantagens sobre os homens, como velocidade e consistência, o conhecimento humano compensa largamente tais vantagens. [27,28].

A partir desta idéia começaram a ser desenvolvidos os sistemas especialistas.

Um sistema especialista (SE) pode ser encarado como um programa de computador que faz uso de uma base de conhecimentos obtida a partir de peritos em determinado assunto, de tal forma que a máquina possa oferecer conselhos inteligentes ou tomar decisões inteligentes sobre esse assunto.

O conhecimento num SE é organizado de forma a separar o conhecimento do domínio do problema de outros conhecimentos, como por exemplo, o conhecimento de como interagir com o usuário. O conhecimento do domínio é chamado base de conhecimento, enquanto o conhecimento geral de resolução de problemas é chamado máquina de inferência.

A base de conhecimento contém fatos e regras. A máquina de inferência contém um interpretador que decide como aplicar as regras para inferir novos conhecimentos.

Um SE para resolver problemas de determinada área, utiliza conhecimento específico da área, raciocínio simbólico (em vez de efetuar apenas cálculos) e, através da

utilização de heurísticas deve ser capaz de encontrar, por si só, pelo menos boas respostas para os problemas.

Em determinadas áreas, no entanto, não é possível a construção de sistemas que fiquem responsáveis por todas as decisões. Isto se deve ao fato do conhecimento, dessas áreas, não estar suficientemente sedimentado. Para auxiliar a solução dessa classe de problemas, surgiram os sistemas especialistas de suporte a decisão (SESD) que além de utilizarem as técnicas de raciocínio dos SE, dependem do auxílio de especialistas para funcionar. O objetivo destes sistemas é auxiliar especialistas em determinada área a resolver problemas de maior amplitude. Este casamento se dá quando o sistema provê algum conhecimento e raciocínio, e o homem (especialista) direciona o sistema, bem como fornece o conhecimento não disponível.

A linha de SESD surgiu a partir de duas linhas de pesquisa: SE e sistemas de suporte a decisão (SSD).

A figura 4.1 abaixo apresenta com clareza as diferenças e semelhanças existentes entre PD, SSD, SE e SESD.

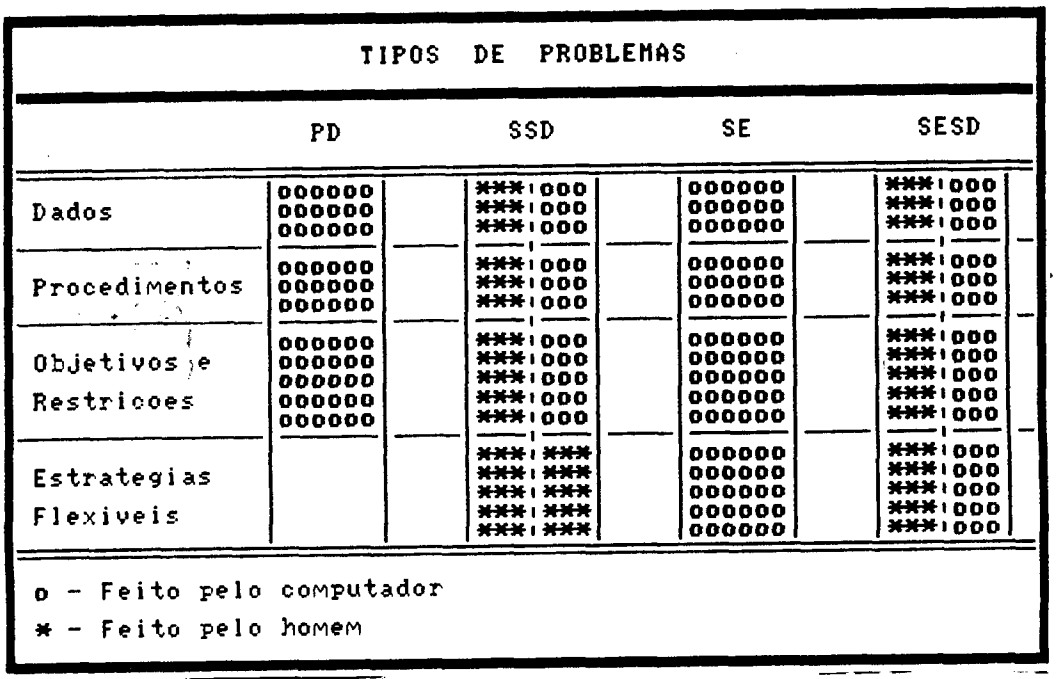

Figura 4.1: PD, SSD, SE e SESD. [3]

O objetivo deste capítulo é a realização de um estudo sobre SE: suas características, o processo de construção, as razões que justificam o seu desenvolvimento, SE que têm sido desenvolvidos para apoiar a atividade gerencial e a viabilidade de construção de um SE para avaliar os custos de software.

IV.2. PROCESSO DE CONSTRUÇÃO

O processo de construção de um SE (de qualquer tipo) é realizado em profunda interação entre o construtor do SE (denominado engenheiro do conhecimento) e um ou mais especialistas na área em que se está construindo o sistema. Durante este processo o engenheiro do conhecimento extrai do especialista seus procedimentos, estratégias e regras, para a solução do problema e incorpora esse conhecimento ao SE. O resultado deverá ser um sistema de computação capaz de resolver problemas de forma semelhante aos especialistas

da área em questão. [3,29]

Não é difícil compreender, portanto, que o fator determinante do sucesso ou não do desenvolvimento de sistemas baseados no conhecimento está na aquisição do conhecimento.

Nesta seção é apresentado um estudo do processo de construção de um SE, que se compõe das seguintes atividades:

1. Enfocar a maneira como os SE vêm sendo desenvolvidos e identificar em que etapas do processo de desenvolvimento a aquisição do conhecimento é mais intensa.

2. Enfocar questões envolvendo a aquisição do conhecimento, como, principalmente, algumas técnicas para a extração e posterior representação do conhecimento na fase de conceituação.

3. Efetuar um estudo do atual estado da arte das ferramentas automatizadas existentes para a aquisição do conhecimento.

4. Apresentar as técnicas de representação do conhecimento mais utilizadas no desenvolvimento de SE.

IV.2.1. AS FASES DE DESENVOLVIMENTO DE UM SE

Não existe, atualmente, uma sequência bem definida de passos que possa ser seguida no desenvolvimento

de um SE. A complexidade inerente ao processo de desenvolvimento desses sistemas impede que essa sequência exista, fazendo com que seus construtores desenvolvam os sistemas de forma evolutiva, isto é, de forma que a organização e a representação do conhecimento do sistema evolua incrementalmente das tarefas mais fáceis para as mais difíceis. Esse processo de desenvolvimento faz com que, em determinado momento, seja preciso reprojetar e reimplementar o sistema que tende a se tornar excessivamente grande e lento.

Apesar de não se conhecer uma sequência exata de passos a seguir, sabe-se que o processo de desenvolvimento é composto das seguintes etapas: identificação, conceituacão, formalização, implementação e testes. Essas etapas ou fases são, na realidade, uma simplificação das atividades complexas e desestruturadas que ocorrem durante o processo de aquisição do conhecimento.

O primeiro objetivo desse método de desenvolvimento deve ser a implementacão de um protótipo do sistema desejado que propiciará ao construtor um melhor entendimento do problema. Uma vez atingido esse objetivo, o engenheiro do conhecimento passa à fase seguinte que consiste em aperfeiçoar o protótipo através de refinamentos sucessivos, de forma que ele evolua tanto em largura quanto em profundidade.

IDENTIFICAÇÃO

O primeiro passo na aquisição do conhecimento

consiste em identificar alguns importantes aspectos do problema, como:

1. Quem são os participantes.

Antes de se iniciar o processo de aquisição do conhecimento é preciso definir quem são os participantes do projeto bem como definir suas funções.

2. Quais as características do problema.

Dentre as atividades realizadas na fase de identificação, o levantamento das características do problema e a determinação de seu escopo é certamente a mais trabalhosa. Nessa fase é preciso definir o problema, levantar suas características e determinar quais são os subproblemas existentes. De acordo com o modelo evolutivo de desenvolvimento de SE, uma maneira rápida de se determinar a complexidade do sistema é através da implementação de rotinas destinadas a resolver subproblemas pequenos mas interessantes.

3. Quais são os recursos disponíveis.

Os recursos são necessários para se adquirir o conhecimento, implementar o sistema e testá-lo. Os mais comuns são: as fontes de conhecimentos (recursos humanos, livros, etc.), tempo disponível, recursos computacionais e dinheiro.

Todos são importantes para o sucesso do desenvolvimento, entretanto, a disponibilidade de tempo,

tanto do engenheiro do conhecimento quanto do especialista, é fundamental, uma vez que serão necessários alguns meses de trabalho árduo para se obter um protótipo do sistema desejado.

4. Que objetivos pretendemos atingir.

Alguns exemplos são: distribuir o conhecimento escasso, ajudar os especialistas a resolver problemas, etc.

CONCEITUAÇÃO

Nessa segunda fase, o engenheiro do conhecimento e o especialista detalham os conceitos e relações definidos na fase anterior. Nessa fase o engenheiro do conhecimento poderá julgar conveniente a diagramação dessas idéias de forma a facilitar a prototipação do sistema. Devido à dificuldade de se produzir uma análise completa do problema logo na segunda fase de desenvolvimento, aconselha-se aos construtores de SE que iniciem logo os trabalhos de implementação visando à prototipação mencionada. O esforço realizado nesse sentido propiciará um melhor entendimento do problema facilitando, assim, os trabalhos dessa fase.

Assim como na fase anterior, a aquisição de conhecimento realizada nesta fase é intensa. Inúmeros contatos entre o engenheiro do conhecimento e o especialista são necessários, consumido tempo considerável desses profissionais.

FORMALIZAÇÃO

O objetivo dessa fase de desenvolvimento é o mapeamento dos conceitos e relações definidos na fase anterior de forma que essas idéias sejam mais formalmente representadas. O engenheiro do conhecimento deverá, juntamente com o especialista, escolher o método de representação do conhecimento apropriado e definir que ferramentas serão utilizadas na fase de implementação. Como mencionado na etapa anterior, a implementação de um protótipo a essa altura do desenvolvimento será de grande ajuda na validação dos trabalhos até então realizados.

O produto final dessa etapa é uma especificação parcial do problema de forma que na etapa seguinte os trabalhos de implementação sejam facilitados.

IMPLEMENTAÇÃO

Na etapa de implementação, o conhecimento formalizado na etapa anterior (estruturas de dados, regras de inferência e estratégias de controle) é codificado nos termos da ferramenta de implementação escolhida.

TESTES

O primeiro objetivo da fase de testes é avaliar o protótipo e a técnica de representação do conhecimento utilizada na implementação do sistema. Cabe ao especialista avaliar o comportamento do sistema e ajudar o

engenheiro do conhecimento na sua revisão.

Uma vez funcionando para alguns casos, o protótipo deverá ser testado exaustivamente de forma a se poder avaliar seu desempenho e utilidade. Essa avaliação pode revelar diversos tipos de problemas, como: esquema de representação do conhecimento inadequado, falta de conceitos e relações, nível errado de detalhamento na representação do conhecimento ou ineficiência com relação aos tempos de execução devido a pesados mecanismos de controle. Tais problemas podem fazer com que os desenvolvedores do sistema sejam obrigados a reexaminar o problema em todas as suas fases de desenvolvimento.

Essas etapas ou fases são, na realidade, uma simplificação das atividades complexas e desestruturadas que ocorrem durante o processo de aquisição do conhecimento.

O primeiro objetivo desse processo de desenvolvimento deve ser a implementação de um protótipo do sistema desejado que propiciará ao construtor um melhor entendimento do problema. Uma vez atingido esse objetivo, o engenheiro do conhecimento passa à fase seguinte que consiste em aperfeiçoar o protótipo através de refinamentos sucessivos, de forma que ele evolua tanto em largura quanto em profundidade.

IV.2.2. A AQUISIÇÃO DO CONHECIMENTO

O termo aquisição do conhecimento compreende a

obtenção do conhecimento especializado necessário à solução de problemas em certo domínio e sua transferência para um programa de computador. São muitas as fontes de conhecimento úteis ao desenvolvimento de SE, como os especialistas no domínio do problema a ser tratado, os livros, os artigos publicados, os estudos de casos e a experiência do engenheiro do conhecimento. [30]

Os princípios gerais que devem nortear os trabalhos de aquisição do conhecimento são: [31]

1. Especialmente nos primeiros estágios de desenvolvimento dos SE, o engenheiro do conhecimento deve procurar ser específico, incentivando o especialista a descrever problemas de particular interesse.

2. É importante que o especialista represente o seu conhecimento da maneira que julgar mais natural. O engenheiro do conhecimento não deve impor métodos de representação estranhos ao especialista.

3. O engenheiro do conhecimento deve evitar interromper o especialista. É preciso ser paciente e considerar a possibilidade de que surjam contradições e inconsistências. No decorrer dos trabalhos as idéias serão esclarecidas sem a necessidade de interrupções que podem cortar a linha de raciocínio do especialista, fazendo com que detalhes importantes sejam esquecidos.

4. A gravação das entrevistas com os especialistas e sua posterior trancrição é aconselhável, já que o construtor dificilmente será capaz de identificar, de

pronto, que informações são relevantes para o sistema. A gravacão metódica de todas as entrevistas e o estabelecimento de referências cruzadas do material com as regras levantadas pode ser um fator fundamental para o sucesso do sistema.

5. O engenheiro do conhecimento deve observar, além dos fatos, teorias e heurísticas, a forma pela qual o especialista usa seu conhecimento, isto é, a ordem em que os problemas são levantados, a importância relativa de diferentes ítens ou a forma de atribuir pesos a evidências.

Uma vez definido e aprovado o desenvolvimento de um SE começa o trabalho do engenheiro do conhecimento (fase de conceituação), que, inicialmente, deverá se preocupar com as seguintes tarefas:[29]

1. **Entender o problema** - Embora o problema já deva ter sido definido na fase de identificação, o engenheiro do conhecimento deverá procurar dirimir todas as suas dúvidas bem como obter a maior quantidade de informações possível antes de contactar o especialista.

2. **Ler a respeito do problema** - Os livros, manuais e documentação existentes a respeito do sistema devem ser estudados. É provável que o conhecimento especializado não seja encontrado em qualquer desses meios, entretanto, eles serão de grande ajuda no aprendizado dos termos utilizados e das rotinas e procedimentos comuns de forma a facilitar a comunicação engenheiro do conhecimento - especialista.

Após executados esses trabalhos preparatórios a tarefa de aquisição do conhecimento pode realmente começar.

O conhecimento necessário ao desenvolvimento de um SE pode se originar de diversas fontes, sendo a principal os especialistas da área em questão. Através de sistemáticas e prolongadas entrevistas o construtor obtém dos especialistas o conhecimento necessário ao desenvolvimento do sistema. Com o objetivo de facilitar a comunicação especialista - construtor algumas técnicas para a extração do conhecimento foram desenvolvidas. O engenheiro do conhecimento deve estar ciente da existência dessas técnicas e deve usá-las sempre que julgar conveniente. Algumas delas são: [31,32]

1. **Estudo de casos interessantes** - O engenheiro do conhecimento deve pedir ao especialista que descreva casos difíceis ou interessantes. Através de seu depoimento o especialista estará transmitindo precisamente os passos seguidos para solucionar o problema. Esse tipo de questionamento, embora possa provocar a omissão de informações essenciais para a solução de problemas mais fáceis, pode trazer à tona, rapidamente, importantes informações que formarão a base de discussões futuras.

2. **Características e decisões** - Embora incapaz de formular regras e heurísticas, o especialista poderá estar apto a fornecer listas de sintomas e de possíveis decisões. Uma vez elaboradas as listas poder-se-á pedir ao especialista que relacione um conjunto de características a uma ou mais decisões. Embora essa técnica seja considerada

de fácil utilização, alguns especialistas julgam-na artificial, uma vez que a ordem em que os sintomas aparecem na vida real não é considerada.

3. **Objetivos diferenciados** - Dado um objetivo específico, pode-se pedir a um especialista que especifique as características necessárias e suficientes na diferenciação deste objetivo dos demais. Se o especialista for capaz de executar esse trabalho para o objetivo final e para cada um dos objetivos intermediários, ele terá construído um modelo estruturado do próprio conhecimento.

4. **Reclassificação** - Esse método é utilizado quando o especialista prefere trabalhar de trás para frente, isto é, do objetivo para as características que o determinam. Cada objetivo é reclassificado em evidências. Se uma evidência é, na realidade, outro objetivo, há nova reclassificação e assim sucessivamente.

5. **Divisão do domínio** - Esse método é o oposto da reclassificação. Aqui, o especialista começa pela descrição dos sintomas e através de agrupamentos sucessivos chega ao objetivo final.

6. **Pensar em voz alta** - Por esse método o especialista ao resolver um problema (real ou não) deverá pensar em voz alta para que seus comentários possam ser gravados e posteriormente analisados, se possível, na presença do especialista. O objetivo é levantar os conhecimentos utilizados pelo especialista, bem como descobrir quais as estruturas de controle utilizadas na

seleção do conhecimento.

7. **Construção por refinamentos** - O especialista entrega alguns problemas (com níveis de dificuldade variados) ao engenheiro do conhecimento que deve resolvê-los à mão utilizando os conceitos, formalismos e regras extraídos, até então, do especialista. A seguir, o engenheiro do conhecimento utiliza o SE para resolver esses mesmos problemas e, por fim, uma análise completa dos resultados é obtida com a participação do especialista.

8. **Validação** - A validação consiste em entregar a outros especialistas os problemas resolvidos pelo protótipo e pelo especialista engajado no desenvolvimento. Assim, será possível comparar as estratégias de diferentes especialistas bem como descobrir as divergências.

A medida que o conhecimento é levantado é preciso analisá-lo e refiná-lo. Essa tarefa, no entanto, não é nada fácil, pois é necessário garantir um resultado consistente, sem redundâncias. Além de um certo número de ferramentas automatizadas (objeto da próxima seção) existem outras muito úteis na fase de conceituacão, como:

1. **Diagramas** [31] - O produto de uma entrevista, normalmente, está gravado ou escrito. Como as linguagens falada e escrita são ambíguas é preciso um grande esforço do engenheiro do conhecimento e do especialista no sentido de levantar os conceitos e relações tratados na entrevista. Os diagramas são muito úteis na representação desse

conhecimento, pois facilitam sua análise e refinamento. Dois exemplos de diagramas são: rede de inferências e árvore de decisão.

2. **Dicionário do conhecimento** [31] - É o dicionário de dados dos SE. Objetiva catalogar informações relevantes ao sistema em desenvolvimento, tais como:

. nome
. tipo (fato, regra, evidência, etc.)
. sinônimos
. atributos ou características
. valores possíveis
. importância
. valor "default"
. pré-requisito
. fatos e regras que geram determinada conclusão
. fonte da informação (referência cruzada)
. comentários (implicações possíveis)

3. **Diagrama de fluxo de dados, Dicionário de dados e Regras** [33] - A notação do DFD é composta por círculos, arcos direcionados e retângulos. Cada círculo representa um pequeno problema independente dos demais. Através dos arcos que conectam os círculos os dados fluem. Os retângulos são chamados fontes ou destinos dos dados.

As informações que constam do DFD são detalhadas no dicionário de dados onde consta para cada decisão (representada no DFD pelo círculo):

- os fatores que influenciam a decisão (fluxos de entrada);

- os possíveis resultados (fluxo de saída);

- as regras nas quais a partir dos fatores chega-se aos resultados.

A figura 4.2 apresenta um exemplo de DFD que visa representar um sistema que determina a adequabilidade de construção de um particular SE.

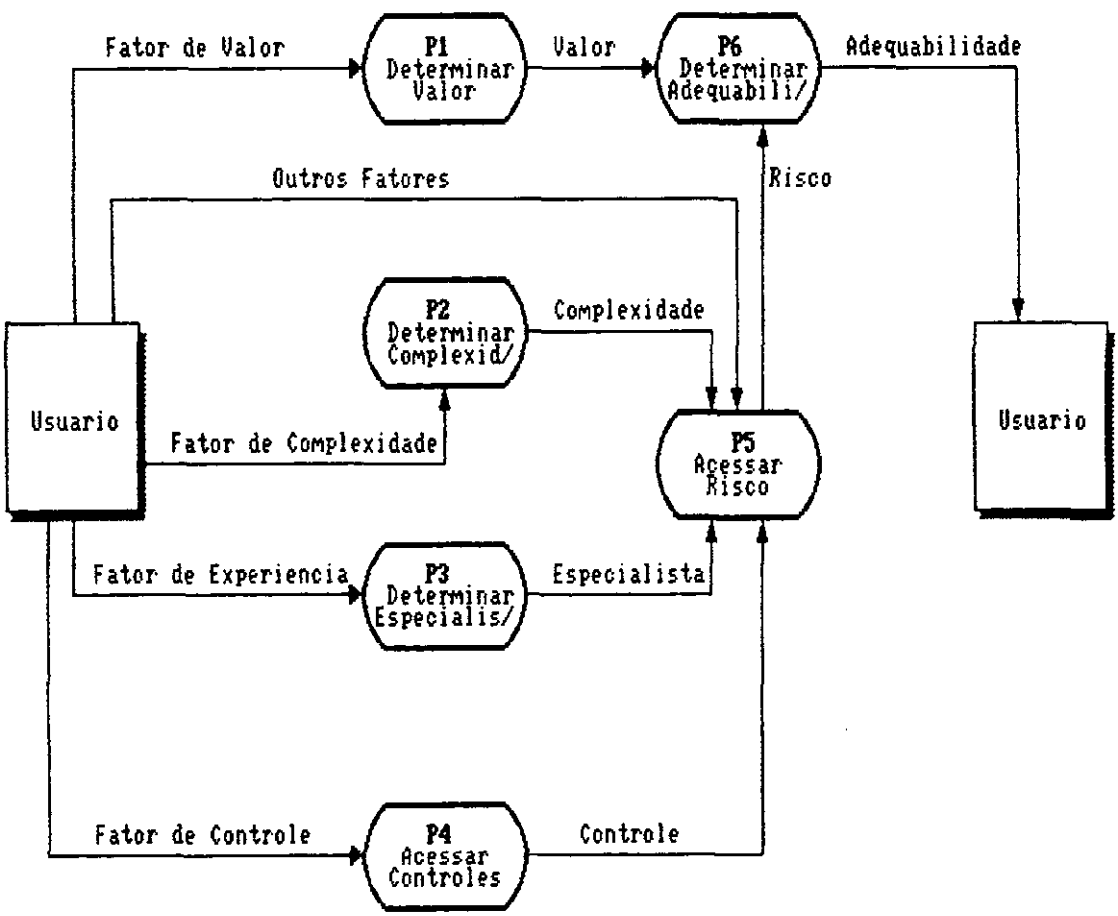

Figura 4.2: DFD para determinar adequabilidade de construção de um SE.[33]

IV.2.3. A AQUISIÇÃO AUTOMATIZADA DO CONHECIMENTO

O típico processo de aquisição do conhecimento envolve um especialista, um construtor e uma base de conhecimentos. A tarefa do construtor é servir de intermediário entre o especialista e a base de conhecimentos, na transferência do conhecimento. Entretanto, sérios problemas de comunicação ocorrem, devido ao pequeno conhecimento que o construtor possui, em geral, sobre o domínio do problema tratado.

Algumas ferramentas automatizadas têm sido propostas no sentido de substituir o construtor (o intermediário no processo de transferência) por programas de edição inteligentes, munidos de sofisticados recursos para a implementação de diálogos e de amplo conhecimento sobre a estrutura das bases de conhecimento. Em Teiresias (projeto Emycin) tem-se, provavelmente, o melhor exemplo desse modelo de aquisição do conhecimento.

Na elaboração das bases de conhecimento é bastante comum a utilização, pelos especialistas, de livros, estudo de casos ou relatórios, como fontes de conhecimento. Embora o assunto seja controvertido, há aqueles que acreditam para o futuro, na plena utilização de máquinas de indução (programas capazes de criar regras a partir de exemplos) para a construção de bases de conhecimento. A entrada de dados dessas máquinas seriam exemplos cuidadosamente escolhidos na literatura utilizada pelos especialistas.

Um terceiro e último método de aquisição do conhecimento, que poderá se tornar viável no futuro, é a extração do conhecimento diretamente dos livros. O vocabulário técnico dos livros é relativamente simples se comparado ao vocabulário popular. Entretanto, esta tarefa requer bem mais sofisticação do que já se conseguiu com os atuais estudos em linguagem natural. [32].

Algumas outras ferramentas automatizadas têm sido desenvolvidas com o objetivo de auxiliar o especialista e o construtor do sistema no processo de aquisição do conhecimento:

1. Editores de texto - Sofisticados editores de textos têm sido desenvolvidos visando facilitar a introdução do conhecimento no sistema e reduzir a probabilidade da ocorrência de erros. Alguns desses editores são capazes de prevenir o usuário não somente quanto aos erros de sintaxe mas também quanto aos de semântica. Os erros de semântica incluem as inconsistências e incompletezas.

Na representação do conhecimento diversos tipos de problemas causam inconsistências: [30]

- quando duas ou mais regras iguais chegam a conclusões diferentes;

- quando há regras duplicadas na base de conhecimentos;

- quando duas regras chegam às mesmas conclusões

embora uma delas seja mais restritiva que a outra, isto é, possua todas as restrições da primeira e mais algumas. Esse tipo de problema gera redundância pois sempre que a regra mais restritiva for executada a menos restritiva também o será.

Quanto à incompleteza, ela se verifica quando em decorrência de determinada situação uma inferência é requisitada sem que exista a regra capaz de resolvê-la.

Nenhum sistema desenvolvido até hoje foi capaz de impedir totalmente a ocorrência de erros durante a construção ou modificação de uma base de conhecimentos. No entanto, o uso de editores capazes de verificar sintática e semanticamente as bases de conhecimentos têm se mostrado extremamente úteis.

2. Explicadores de resultados - Esse tipo de ferramenta faz com que o usuário possa conhecer os motivos que levam o sistema a produzir determinados resultados. Através da análise das explicações fornecidas pelo sistema, o especialista pode descobrir que suposições e inferências são responsáveis pelas falhas ocorridas. [32]

Basicamente, os mecanismos de explicação de resultados presentes nos mais sofisticados SE são:

- Retrospectivo: está presente em quase todos os SE. Produz as explicações necessárias para se saber como o sistema chegou a determinado estado.

- Hipotético: mostra a que conclusão o sistema

chegaria se determinado fato ou regra fosse diferente.

- Justificativo: explica porque uma conclusão esperada não foi encontrada pelo sistema.

3. **Ferramentas de Depuração** - A maior parte das linguagens de programação e das linguagens da engenharia do conhecimento possui técnicas de depuração do tipo "trace" e interrupções programadas. Estes mecanismos de depuração possuem as seguintes características:

- "Trace": informa ao usuário os números ou nomes de todas as regras ativadas assim como os nomes de todas as subrotinas chamadas.

- Interrupções programadas: permitem ao usuário determinar previamente em que pontos do programa a execução deve ser interrompida para que o conteúdo das variáveis possa ser examinado.

Uma outra ferramenta de depuração, existente em poucos SE, prevê a execução automática de um grande número de testes com o objetivo de descobrir erros ou inconsistências nas soluções encontradas. Essa ferramenta é especialmente útil quando o sistema passa por alguma manutenção e se deseja saber se as modificações efetuadas introduziram novos erros.

A extração do conhecimento é um processo muito lento e cansativo. Entretanto, o desenvolvimento de diversas ferramentas automatizadas tem agilizado um grande número de tarefas antes executadas manualmente. Tem-se

observado, no entanto, que a mais séria limitação dessas ferramentas é sua falta de habilidade para a extração do conhecimento do especialista. [34,35]

Apesar do grande progresso na área, as ferramentas existentes para a extração do conhecimento ainda não são capazes de substituir o construtor no processo de desenvolvimento de SE. Consequentemente, é preciso que estes construtores possuam boa experiência com as técnicas de a aquisição do conhecimento, de forma a superar a atual carência de ferramentas automatizadas.

IV.2.4. ALGUMAS TÉCNICAS DE REPRESENTAÇÃO DO CONHECIMENTO

O conhecimento de um SE é representado através de técnicas como lógica, redes semânticas, regras de produção e "frames". Embora não exista um critério formal de escolha da técnica a ser usada para uma determinada aplicação, sabe-se que determinadas técnicas funcionam melhor para alguns tipos de problemas do que para outros. Assim sendo, é muito importante conhecê-las suficientemente bem para que o conhecimento seja representado de forma adequada à natureza do problema.

A seguir são apresentadas as seguintes técnicas de representação do conhecimento: lógica, redes semânticas, regras de produção e "frames".

IV.2.4.1. LÓGICA [26,36,37,38,39] - Muitas das primeiras formas de representação do conhecimento foram baseadas no cálculo de predicados de primeira ordem. Nesta

forma de representação, o conhecimento sobre o domínio do problema é representado na forma de "fórmulas bem formadas" (well-formed formulas - wffs). Através das wffs são representadas as proposições, isto é, os fatos e as regras que compõem a base de conhecimentos do sistema.

Sistemas lógicos utilizam o processo de refutação/resolução para tirar conclusões a partir de uma base de conhecimentos. Esse processo, tipicamente, resolve problemas através da construção de provas a partir de um conjunto inicial de afirmações (a base de conhecimentos) e um objetivo. O processo consiste de: a) negar a wff que representa o objetivo; b) acrescentar a negação do objetivo à base de conhecimentos; c) converter toda a base de conhecimentos para a forma de cláusulas, e, d) através da resolução, procurar chegar a cláusula vazia. Ao chegarmos à cláusula vazia estaremos provando que a wff objetivo é consequência lógica da base de conhecimentos.

Um sistema lógico possue diversas importantes qualidades: a) é modular, isto é, mantida a consistência, assertivas lógicas podem ser inseridas em uma base de conhecimentos independentemente uma das outras; b) possui a força e a expressividade da lógica; c) a lógica é, em geral, uma maneira natural de expressar certas idéias; d) possui regras de inferência corretas, no sentido de que a cláusula vazia é obtida de uma base de conhecimentos, a partir de um número finito de passos de resolução, se e somente se a base é inconsistente, e, e) os programas são fáceis de ler, uma vez que não estão cheios de detalhes a

respeito de como obter resultados. Ao contrário, se parecem com especificações do que deve ser feito.

Entretanto, o poder de expressão das wffs e as características das regras de inferência também resultam em alguns problemas. O maior deles é que a construção de uma prova para algum objetivo pode levar um tempo excessivo. Pesquisadores interessados em linguagens lógicas têm procurado encontrar maneiras de otimizar o tempo de busca a informações relevantes. No caso da linguagem PROLOG, por exemplo, muitos refinamentos ao princípio da resolução têm sido propostos.

IV.2.4.2. REDES SEMÂNTICAS [36,29] - O termo "redes semânticas" (semantic networks) é usado para descrever um método de representação do conhecimento baseado em estrutura de rede.

Sabe-se que uma das formas de se otimizar o tempo de busca da lógica é restringir o formato das wffs. A técnica de redes semânticas, encarada do ponto de vista da lógica, propõe que o conhecimento seja representado através de um conjunto restrito de predicados unários e binários. Os predicados usualmente utilizados para representar hierarquias são "é-um" e "é-parte-de". Algumas outras restrições foram impostas às "wffs" das redes semânticas com a intenção de permitir o desenvolvimento de inferências mais eficientes.

Como as redes semânticas possuem apenas predicados unários e binários, é comum representá-las na

forma de grafos, sendo os arcos ou ligações aqueles predicados mencionados acima. Procedimentos especializados, de acordo com o tipo de arco, são utilizados para melhorar o processo de inferência do método. Esse método suporta raciocínio por meio de deduções lógicas, por "default" e por probabilidade.

A figura 4.3 abaixo demonstra como seria, tanto em lógica como pelo uso de rede semântica, a representação das sentenças: Sócrates é homem; Todo homem é mortal.

Conhecendo-se as propriedades das relações que ligam os nós, isto é, sabendo-se que "é-um" neste caso possui a propriedade transitiva, pode-se concluir, a partir das sentenças acima, que Sócrates é mortal. Em lógica, poderia-se chegar à mesma conclusão utilizando-se o método da refutação / resolução.

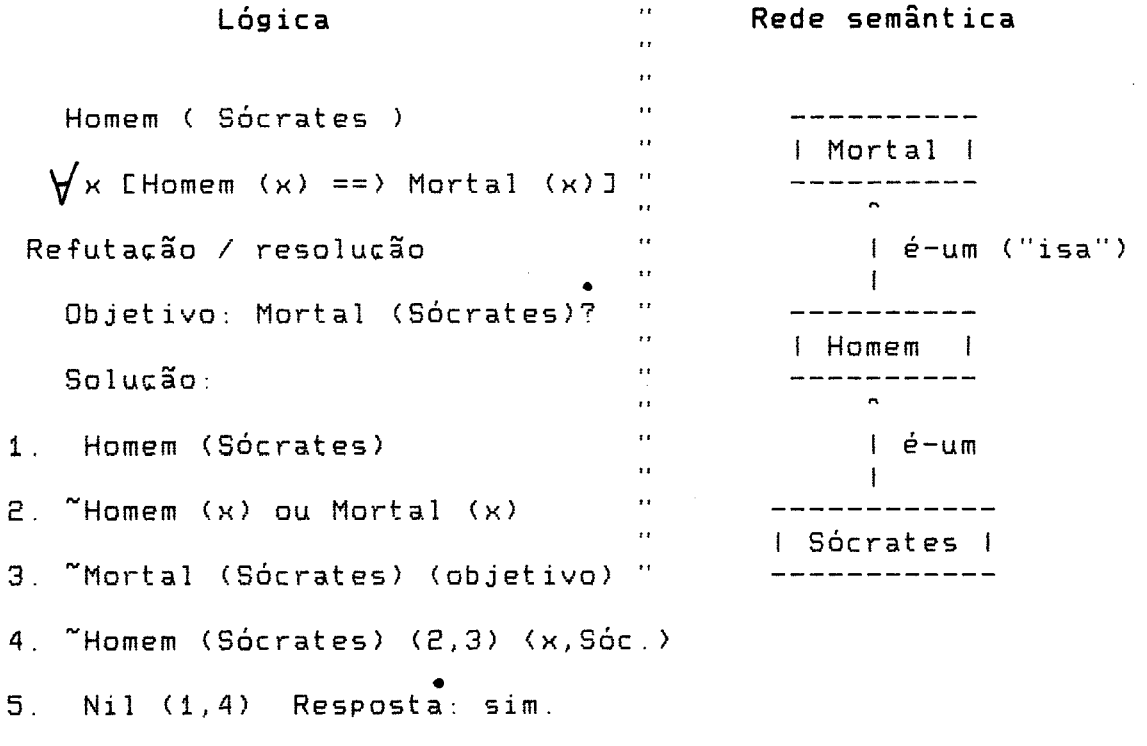

Figura 4.3: Lógica x Rede Semântica

IV.2.4.3. REGRAS DE PRODUÇÃO [26,29,36,38] - Um sistema de regras de produção consiste de um conjunto de regras, uma máquina de inferência e uma base de dados. O conjunto de regras é formado por regras independentes na forma IF <situação> então <ação>. Uma regra é aplicada sobre a base de dados, modificando-a, sempre que o componente <situação> combina com ela. A máquina de inferência é responsável pela escolha da regra aplicável e a computaçao estará encerrada no momento em que a base de dados satisfizer à condição de término. A base de dados, consequentemente, representa o estado corrente do processo de solução do problema, uma vez que ela é modificada pelo componente <ação> sempre que uma regra é ativada.

Para melhor esclarecer pode-se apresentar, como exemplo, um sistema de producao que identifique comidas. O objetivo é, através do uso das regras e da máquina de inferência, identificar que comida é verde e pesa 3 Kg.

Base de dados inicial:

BD = (verde, pesa 3 Kg)

Regras:

R1 - SE está na BD <verde>
ENTÃO acrescente à BD <produto agrícola>.

R2 - SE está na BD <embalado em recipiente pequeno>
ENTÃO acrescente à BD <guloseima>.

R3 - SE está na BD <refrigerado> ou <produto agrícola>
ENTÃO acrescentar à BD <perecível>.

R4 - SE está na BD (pesa 3 Kg) e (barato) e não está na BD (perecível)
ENTÃO acrescentar á BD (de 1a necessidade).

R5 - SE está na BD (perecível) e (pesa 3 kg) e não está na BD (verde)
ENTÃO acrescentar à BD (perú).

R6 - SE está na BD (pesa 3 kg) e (produto agrícola)
ENTÃO acrescentar à BD (melancia).

Máquina de inferência:

1. - Encontre todas as regras aplicáveis.
2. - Se mais de uma regra for aplicável descarte a(s) que adiciona(m) símbolo(s) já existente na BD.
3. - Execute a ação da regra aplicável de menor número. Se nenhuma regra for aplicável, então pare.
4. - Volte a 1.

Sistemas de produção operam em ciclos. A cada ciclo as regras são examinadas da forma especificada pela máquina de inferência, sendo, então, executada a regra escolhida.

Examinando o exemplo acima, observa-se que no primeiro ciclo de execução é aplicada a regra R1 que acrescenta à Base de Dados o símbolo "produto agrícola".

BD = (produto agrícola, verde, pesa 3 Kg)

No segundo ciclo, as regras R1, R3 e R5 são aplicáveis. Entretanto, como R1 gera duplicata e R3 é menor

que R5, R3 é a escolhida.

BD = (perecível, produto agrícola, verde, pesa 3 kg)

No terceiro ciclo as regras aplicáveis são R1, R3, e R6. Como R1 e R3 geram duplicatas, R6 é executada.

BD = (melancia, perecível, produto agrícola, verde, pesa 3 Kg)

No quarto e último ciclo o processamento é encerrado uma vez que não há mais regras a aplicar. E a resposta será o primeiro símbolo da lista de símbolos da base de dados.

Sistemas de diagnóstico médico, entendimento da fala e de exploração mineral são alguns exemplos de sistemas especialistas que têm sido construídos com essa técnica de representação do conhecimento.

Pelos mesmos motivos já expostos quanto a sistemas lógicos, modularidade e naturalidade são também duas qualidades dos sistemas de produção. Outras vantagens decorrem da possibilidade de se poder representar o raciocínio por suposição, probabilidade e "default".

Um problema crítico dos sistemas de produção é a ineficiência em largos domínios. É importante que a máquina de inferência seja hábil na seleção das regras a ativar. Entretanto, sabe-se que quanto maior o domínio, menor a probabilidade dela acertar.

Sistemas baseados em regras, como os de produção,

podem funcionar de duas importantes maneiras, conhecidas como: "forward chaining" e "backward chaining". O problema recém apresentado é um exemplo do funcionamento do primeiro método. O nome "forward chaining" se deve ao fato da procura por novas informações ocorrer, nas regras, da esquerda para a direita, isto é, na ordem natural de execução do comando SE ‹condição› então ‹ação›.

O segundo método, denominado "backward chaining", é muito utilizado quando desejamos percorrer o caminho de trás para frente. No problema de identificação das comidas, suponhamos que não nos interessasse identificar a comida mas apenas saber se melancia é ou não resposta para o problema. Nesse caso, "backward chaining" seria o método indicado. Percorrendo as regras de trás para frente temos: para que melancia seja a resposta é necessário que os atributos ‹pesa 3 Kg› e ‹produto agrícola› estejam na BD. Como ‹pesa 3 Kg› é atributo da base inicial de dados estamos na dependência apenas do atributo ‹produto agrícola›. Se olharmos a regra número 1 veremos que ‹produto agrícola› depende apenas do atributo ‹verde› que também está na base inicial de dados. Logo, podemos concluir que melancia é resposta para o problema.

Em situações em que o objetivo é inferir fatos particulares, "backward chaining" pode ser bem mais eficiente, uma vez que somente as regras relevantes são executadas.

IV.2.4.4. FRAMES [36,29,26] - MARVIN MINSKY, idealizador dessa técnica de representação do conhecimento,

a define como sendo uma estrutura de dados para a representação de situações esteriotipadas como estar indo para uma festa de aniversário de criança, ou estar em um certo tipo de ambiente como uma sala de estar. Ligada a cada "frame" existem vários tipos de informação. Algumas dessas informações relacionam-se ao fato de como utilizar o "frame", outras sobre o que se espera que aconteça em seguida, e outras ainda, sobre o que fazer se essas expectativas não se confirmarem.

A organização desse método é bastante similar à das redes semânticas. Essa estrutura, denominada "frames", é uma rede de nós e relações, organizada hierarquicamente. Os nós superiores representam conceitos gerais enquanto os inferiores, instâncias específicas desses conceitos.

Esse método provê uma estrutura dentro da qual novos dados são interpretados em termos de conceitos adquiridos através de experiências anteriores.

Para representar o conceito genérico de uma cadeira seria necessário o seguinte "frame":

CADEIRA Frame

Espécie : MÓVEL

Número-de-pernas: um número inteiro (DEFAULT = 4)

Estilo-do-encosto: reto, acolchoado, ...

Número-de-pernas: 0, 1 ou 2

A organizaçao desse conhecimento facilita o processamento dirigido a expectativas. O usuário do sistema

especialista, conhecendo o ambiente em que esta inserido, terá expectativas quanto às atitudes do sistema. O mecanismo de representação que torna possível esse tipo de raciocínio são os "slots", locais dentro de um contexto maior denominado "frame", onde é colocado conhecimento.

O conceito genérico de "cadeira" representado acima possui "slots" para representar número de pernas, estilo de encosto, etc. Consequentemente, o "frame" para uma cadeira específica terá os mesmos "slots", uma vez que eles terão sido herdados do "frame" que descreve a cadeira genérica. Entretanto, o conteúdo desses "slots" será precisamente definido.

CADEIRA-DO-JOÃO Frame
Espécie: CADEIRA
Número-de-pernas: 4
Estilo-do-encosto: Acolchoado
Número-de-braços: 0

Se o conhecimento de um determinado domínio pode ser dividido em conceitos e atributos, um sistema com essa estrutura oferece, pelo menos, três vantagens: a) o sistema de inferência desse modelo é bastante eficiente se comparado aos demais. Assim como nas redes semânticas, ligações entre "frames" conduzem o processo de raciocínio. Uma vez que o "frame" relevante tenha sido encontrado, procedimentos específicos dessa estrutura passam a controlar o processo de raciocínio, b) o conhecimento de como raciocinar não está restrito a nenhum modelo. Consequentemente, raciocínio por suposição, por "default" e

por probabilidade são possíveis, c) esse modelo é capaz de representar o conhecimento em nível de detalhe.

Um dos problemas com esse esquema de representação é que nem todo conhecimento pode ser representado facilmente na forma de conceitos e atributos. Além disso, essa técnica talvez seja a de implementação mais trabalhosa.

IV.3 RAZÕES QUE JUSTIFICAM O DESENVOLVIMENTO DE SISTEMAS ESPECIALISTAS

São muitos os motivos que levam à construção de SE [3,27,28,29].

1. A importância da disseminação do conhecimento especialista escasso: a transferência desse conhecimento de uma pessoa para outra é cara e demorada. O conhecimento de um especialista não é o tipo de conhecimento denominado público (obtido em conferências, livros, etc) mas sim privado, obtido através da experimentação em diferentes situações no decorrer de vários anos. Esse tipo de conhecimento raramente é escrito, e, por isso mesmo, deve ser salvo antes que não mais exista.

2. A capacidade do SE aplicar o conhecimento de forma sistemática e consistente: um especialista pode ser levado a tomar decisões diferentes em situações idênticas, devido a fatores emocionais. O mesmo não ocorre com um SE.

3. O fato do conhecimento armazenado em um SE

ser permanente: um especialista ao deixar de utilizar seus conhecimentos por determinado tempo corre o risco de ter seu desempenho seriamente afetado.

4. A capacidade dos Sistemas especialistas de manipular informações contraditórias e incompletas: o uso de fatores de certeza e medidas de probabilidade têm sido incorporado a vários SE.

5. O fato dos sistemas especialistas terem mostrado uma relação custo / benefício aceitável: fica assim demonstrada sua viabilidade econômica pois o baixo custo de operação desses sistemas tem compensado o alto custo de desenvolvimento.

6. Sistemas especialistas são capazes de explicar suas decisões: ao contrário dos sistemas de software convencionais, os SE são capazes de explicar seus resultados e comportamentos. Tal habilidade é fundamental para o desenvolvimento e manutenção do software assim como para a manutenção da credibilidade do sistema.

Entretanto SE possuem limitações que, na realidade, não refletem limitações fundamentais da inteligência artificial mas apenas o atual estado da arte:

1. A criatividade: nem mesmo o programa mais inteligente é capaz de se aproximar do homem nesta área. Um especialista é capaz de reorganizar informações e usá-las de forma a sintetizar um novo conhecimento, enquanto um SE se comporta sempre da forma como foi programado.

2. A capacidade de aprender: embora já existam programas com essa capacidade, tais programas só têm funcionado para domínios muito restritos.

3. O raciocínio dos sistemas especialistas através da manipulação de símbolos que representam idéias e conceitos: para que um SE seja capaz de fazer uso de informações visuais, táteis e auditivas, sensores devem captar tais informações e transformá-las em símbolos. Nessa transformação, muitas informações relevantes são perdidas.

4. O fato dos sistemas especialistas não possuirem o que se poderia chamar de "senso comum do conhecimento", isto é, um largo conhecimento do mundo que todos os seres humanos possuem e usam: sem esse tipo de conhecimento, um SE, caso inquirido, pode tentar responder às perguntas mais descabidas, cujas respostas nem mesmo existem.

IV.4. SISTEMAS ESPECIALISTAS APLICADOS À ÁREA GERENCIAL

Através da obtenção do conhecimento especialista utilizado por gerentes para identificar e resolver problemas, um SE nessa área pode ser útil das seguintes formas [34]: a) ajudando gerentes a tomar decisões; b) como meio de treinamento, e, c) na análise e aperfeiçoamento dos processos de decisões gerenciais.

Suporte a decisões é o uso mais óbvio. O usuário descreve o problema e o sistema faz recomendações e as justifica. Além disso, o sistema deve ser capaz de

discutir com os seus usuários as principais características do problema, o relacionamento com outros problemas e as razões que levam uma estratégia a ser melhor do que outra.

Como meio de ensino um SE pode se muito útil na educação e treinamento de gerentes inexperientes, bem como na reciclagem dos demais.

O processo de construção de um SE aplicado à gerência além de contribuir com a geração do sistema em si, leva a um melhor entendimento dos processos de decisões gerenciais.

Quatro possíveis áreas de aplicação para SE em gerência de projetos de sistemas são [34]: a) alocação de recursos; b) diagnóstico de problemas; c) planificação e distribuição de tarefas, e, d) gerenciamento de informações.

Sistemas voltados para a alocação de recursos possuem regras heurísticas, obtidas de gerentes experientes que objetivam uma melhor utilização de recursos limitados.

Sistemas desenvolvidos para diagnosticar problemas recebem, como entrada, informações que descrevem sintomas de possíveis problemas e usam as regras retiradas dos especialistas para diagnosticar o problema.

A partir de requisitos especificados pelo usuário, os sistemas de programação e atribuição de tarefas estipulam tarefas e determinam os responsáveis pelo seu cumprimento. Caso esses requisitos estejam inconsistentes,

o sistema deverá sugerir ao usuário possíveis modificações nos requisitos de forma a eliminar as inconsistências.

Um sistema para gerenciamento de informações seria composto de uma base de conhecimentos, (o resultado do agrupamento de informações contidas em arquivos, procedimentos de análise e modelos de decisão) e de uma máquina de inferência que utilizaria a técnica de IA adequada para se determinar que tipo de pergunta poderia ou não ser respondida com os conhecimentos disponíveis. Este tipo de sistema ainda não foi implementado, embora se encontre na literatura muitos estudos a respeito.

IV.5. VIABILIDADE DE CONSTRUÇÃO DE UM SE PARA AVALIAR CUSTOS DE SOFTWARE

Para que seja considerada a viabilidade de construção de um SE para estimativa de custos é preciso que o seu desenvolvimento seja possível, justificado e apropriado. [29]

Para que o desenvolvimento seja possível é necessário que o domínio do problema possua todas as seguintes características:

1. A área do problema considerado deve possuir especialistas pois para que o engenheiro do conhecimento seja capaz de produzir programas verdadeiramente habilidosos é necessária uma poderosa fonte de conhecimentos.

2. Os especialistas devem, de forma geral,

concordar na solução dos problemas. Caso contrário será impossível avaliar o desempenho do sistema.

3. Os especialistas devem ser capazes de explicar os métodos que utilizam na resolução de problemas, uma vez que tais métodos são parte do conhecimento que deve ser inserido no sistema.

4. A solução dos problemas deve requerer habilidades cognitivas e não físicas, embora não seja impossível implementar um SE que possua um pouco de cada. Nesse caso, a parte física seria tratada por outros métodos.

5. Assuntos extremamente difíceis não são adequados à construção de SE pois o processo de extração do conhecimento fica prejudicado.

6. Assuntos cujos conhecimentos não estão suficientemente estruturados, por serem muito novos ou por não estarem ainda bem entendidos, também não devem ser tratados pela engenharia do conhecimento.

7. Para que seja possível a implementação de um SE, a tarefa não pode requerer significativa quantidade de senso comum.

O simples fato do desenvolvimento de um SE ser possível não significa que ele deva ser implementado. É preciso que os motivos justifiquem tal esforço. Alguns desses motivos são apresentados a seguir:

1. Quando a relação custo / benefício é altamente vantajosa.

2. Quando há falta de especialistas no mercado de trabalho.

3. Quando, por fatores diversos (serviço militar, rotatividade de mão de obra, aposentadorias, etc), o conhecimento especialista de uma empresa esteja sendo perdido.

4. Quando o conhecimento especialista é requerido em diversos lugares, simultaneamente, ou quando requerido em locais hostis ao ser humano, como em estações espaciais e ambientes radioativos.

Por fim, os fatores relativos à determinação de quando um SE é apropriado. Tais fatores estão relacionados à natureza, complexidade e escopo do problema a ser tratado.

1. Quanto à natureza, para que um problema possa ser resolvido por um SE é necessário que ele possa ser naturalmente resolvido via manipulação de símbolos.

2. A complexidade deve ser tal que um ser humano leve anos de estudo ou prática para se tornar um especialista na área. Entretanto, como já foi dito, o problema não deve ser extremamente difícil.

3. Quanto ao escopo, não deve ser amplo demais, para que o problema seja manipulável, nem restrito demais, para garantir o interesse prático no assunto.

De forma geral, a área de Engenharia de Software não possui um conhecimento suficientemente sedimentado. Isto é motivado pela pouca idade da Engenharia de Software assim como pelo contínuo e acelerado desenvolvimento da tecnologia computacional. Este fato compromete o desenvolvimento de um SE para a estimativa de custos mas não torna impossível o desenvolvimento de um sistema especialista de suporte a decisão (SESD) de apoio à estimativa de custos de desenvolvimento de software, pois estes sistemas não têm por objetivo encontrar soluções por si só, e sim auxiliar especialistas na solução de problemas de difícil solução.

Por outro lado, embora o conhecimento nessa área não esteja completamente sedimentado, estudos no sentido do estabelecimento de métricas para a determinação do verdadeiro impacto dos principais fatores responsáveis pela produtividade no desenvolvimento dos produtos de software têm avançado significativamente.

Assim como os SE, os SESD também utilizam raciocínio simbólico, no entanto, necessitam do auxílio humano para atingir os objetivos desejados. Enquanto o SESD provê algum conhecimento do assunto e formas de raciocínio, o homem fornece as diretrizes gerais do problema, não incorporada pelo sistema.

Neste caso específico, pretende-se incorporar ao sistema o conhecimento relativo aos fatores responsáveis pela produtividade no desenvolvimento dos produtos de software e estabelecer relações entre estes fatores. Esta

última característica não existe na maioria dos sistemas de estimativa de custos.

Quanto a ser justificado o desenvolvimento, pode-se afirmar que pelo menos dois dos fatores mencionados por WATERMAN [29] são atendidos. Embora existam pouquíssimos especialistas nessa área (fator 2), a tarefa de implementar o sistema não parece ser trabalhosa a ponto de comprometer sua relação custo / benefício. (fator 1)

Finalmente, quanto à determinação de ser ou não apropriado seu desenvolvimento, é necessário analisar o problema com relação à sua natureza, complexidade e escopo.

Quanto à natureza, trata-se de um problema a ser resolvido, em sua maior parte, via manipulação de símbolos.

Quanto à complexidade, estimar custos não parece ser um problema extremamente difícil. Mas também não é muito simples, haja visto o pequeno número de especialistas no assunto.

E quanto ao escopo, acredita-se ser de tamanho razoável, uma vez que já existem métricas precisas para os principais fatores responsáveis pela produtividade e qualidade no desenvolvimento dos produtos de software. Certamente, com o passar do tempo, novas métricas serão definidas pelos especialistas da área quando, então, o escopo do sistema poderá ser aumentado tornando o sistema cada vez mais abrangente.

No próximo capítulo, serão apresentadas a

arquitetura do sistema proposto e as regras que compõem sua base de conhecimentos.

CAPÍTULO V

O PROJETO DO SISTEMA

V.1. INTRODUÇÃO

A idéia de se desenvolver um sistema que desse apoio a profissionais de informática na tarefa de estimativa de custos de desenvolvimento de software surgiu em decorrência da ampla utilização, por estes profissionais, de métodos baseados no julgamento de especialistas, na realização de estimativas de custo.

Embora vários métodos de estimativa de custos venham sendo desenvolvidos e aperfeiçoados, tais métodos baseiam-se, principalmente, em processos algorítmicos.

O objetivo deste trabalho foi apoiar os profissionais habituados a elaborar estimativas de custo através de métodos baseados no julgamento de especialistas. Para atingir estes objetivos optou-se pela construção de um sistema especialista de suporte a decisão, capaz de armazenar o conhecimento dos melhores especialistas da área de forma a poder auxiliar seus usuários na tomada de decisão.

Este capítulo descreve o sistema proposto neste trabalho. São apresentados a arquitetura do sistema e os diversos componentes que o integram.

V.2. O SISTEMA

O objetivo principal deste sistema é apoiar a realização de estimativas de custos de software elaboradas por profissionais experientes, que utilizem em suas estimativas o método baseado no julgamento de especialistas.

Durante a realização de uma sessão, a participação do especialista é fundamental. O sistema elabora uma série de perguntas a respeito do software cujo custo se deseja estimar e a respeito do ambiente de desenvolvimento. As respostas a estas perguntas farão com que o sistema, após consulta à base de conhecimentos, chegue a algumas conclusões que são apresentadas ao usuário. Cabe ao usuário, então, examinar as conclusões elaboradas pelo sistema, e, com base em sua própria experiência, estimar o custo final do produto de software em questão.

A base de conhecimentos do sistema foi elaborada a partir de uma pesquisa bibliográfica que tratou da influência de diversos fatores de produtividade no custo de desenvolvimento e manutenção de software. Consequentemente, o conhecimento inserido no sistema deve ser considerado apenas como um ponto de partida, isto é, novas pesquisas precisam ser realizadas, estudos de casos elaborados e, principalmente, especialistas precisam ser consultados para que o conhecimento seja mais abrangente.

Considerando as perguntas que são formuladas pelo

sistema, aconselha-se que o mesmo comece a ser utilizado no decorrer da proposta de desenvolvimento, uma vez que será preciso conhecer bem o sistema cujo custo se deseja estimar.

A versão (protótipo) do sistema apresentada neste trabalho possui 52 regras. Algumas possuem o objetivo específico de apoiar estimativas de custos. Outras, procuram orientar os usuários no sentido do que deve ser feito para se reduzir o custo final de um projeto.

V.3. ARQUITETURA

A figura 5.1 apresenta o relacionamento existente entre os diversos módulos do sistema. A descrição de cada módulo vem a seguir.

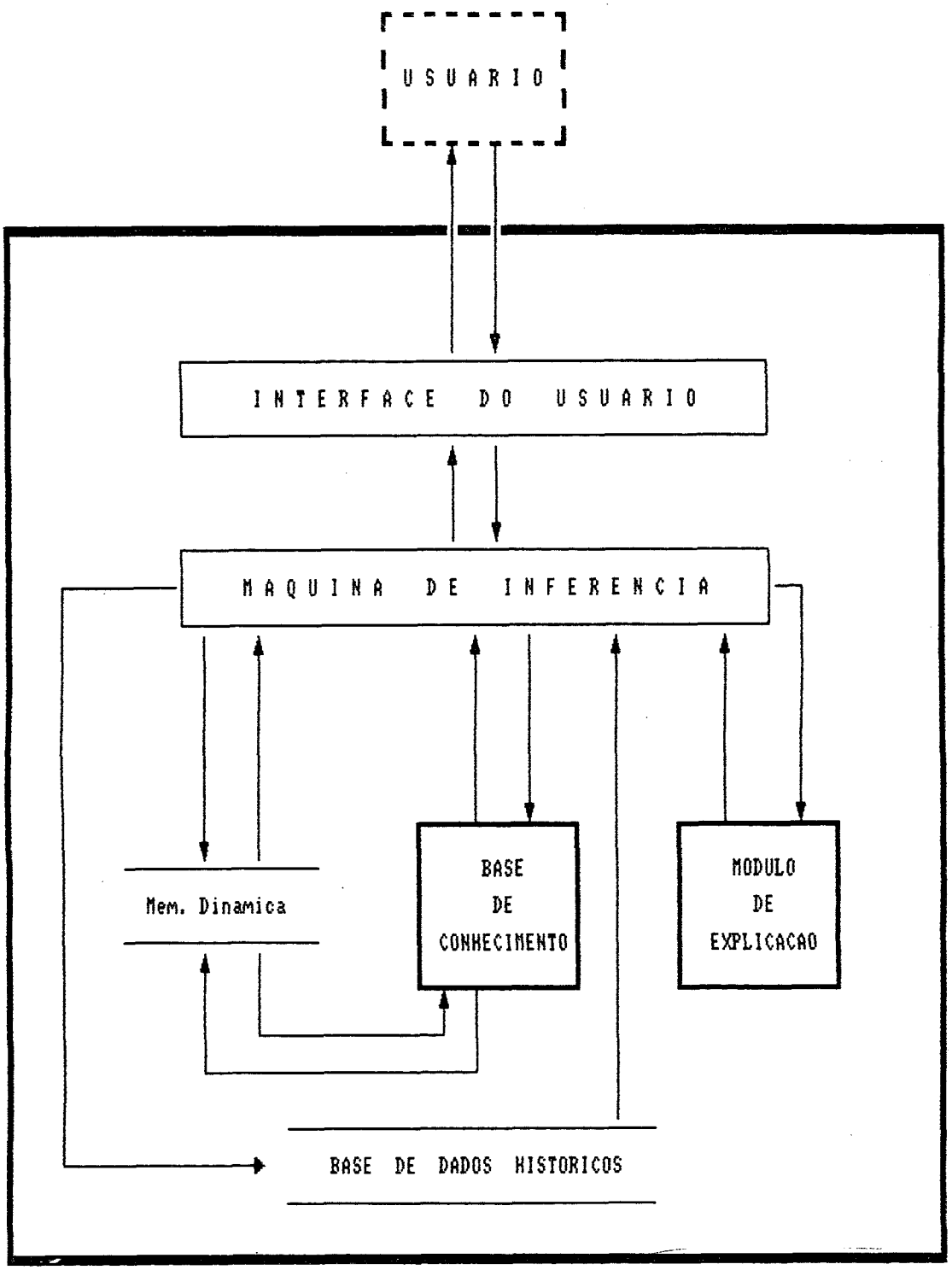

Figura 5.1: Arquitetura do Sistema.

V.3.1. MÁQUINA DE INFERÊNCIA

Contém os métodos de raciocínio utilizados na solução de problemas. Como este sistema foi desenvolvido em PROLOG (Turbo) sua máquina de inferência está baseada na lógica - cálculo de predicados de primeira ordem.

V.3.2. BASE DE DADOS HISTÓRICOS

Após cada sessão o usuário tem a possibilidade de guardar as informações por ele fornecidas. Estas informações são armazenadas em uma base de dados históricos para possível reutilização futura.

V.3.3. MEMÓRIA DINÂMICA

É composta pelas respostas fornecidas pelo usuário às perguntas efetuadas pelo sistema ao longo de uma sessão. Estas respostas são os "fatos" que passam a compor a base de conhecimento do sistema em análise. Através do mecanismo denominado "what if" é possível modificá-los para que se possa verificar o comportamento do sistema sob novas condições.

V.3.4. MÓDULO DE EXPLICAÇÃO

O sistema é capaz de fornecer dois tipos de explicação ao usuário:

a) Após cada pergunta efetuada pelo sistema é possível saber o motivo que levou o sistema a efetuar aquela pergunta. A explicação fornecida menciona a

principal razão para a pergunta, bem como as regras constantes da base de conhecimentos relacionadas à pergunta;

b) Como o sistema sempre apresenta ao seu usuário uma janela com as possíveis respostas às perguntas formuladas, é possível que haja alguma dúvida quanto ao significado destas respostas. Todas as respostas duvidosas podem ser esclarecidas através de um comando do tipo HELP.

V.3.5. INTERFACE

A interface com o usuário é realizada através de telas. A tela da figura 5.2 apresenta na janela denominada "Menu Principal" as funções executadas por este sistema:

1. Tutorial do Sistema: Apresenta uma descrição detalhada do funcionamento do sistema.

2. Comandos DOS: Permite que o usuário deixe o sistema para ficar sob o DOS pelo tempo que desejar.

3. Carrega Banco de Dados: Permite que o usuário carregue na memória do computador uma sessão anteriormente salva.

4. Imprime Banco de Dados: Permite ao usuário imprimir todos os diagnósticos fornecidos pelo sistema ao longo de uma sessão.

5. Assistente Especialista: Ativa o sistema.

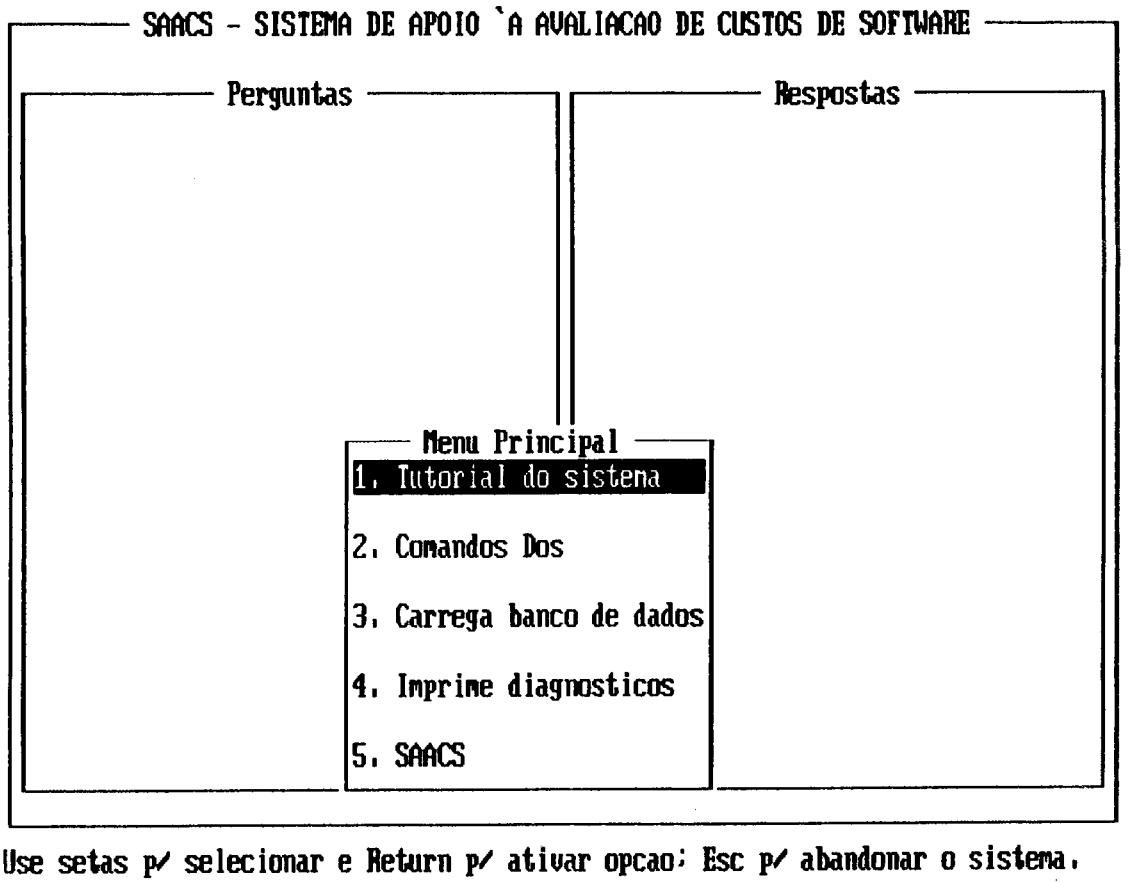

Figura 5.2: Tela de apresentação

As possíveis respostas às perguntas efetuadas pelo sistema aparecem em janelas, conforme consta da tela da figura 5.3. Caso o usuário não entenda a pergunta ou caso deseje algum esclarecimento sobre o significado de alguma possível resposta, poderá inquirir o sistema através das teclas "p" ou "?" respectivamente. Todas as respostas fornecidas pelo usuário ficam visíveis na janela de respostas.

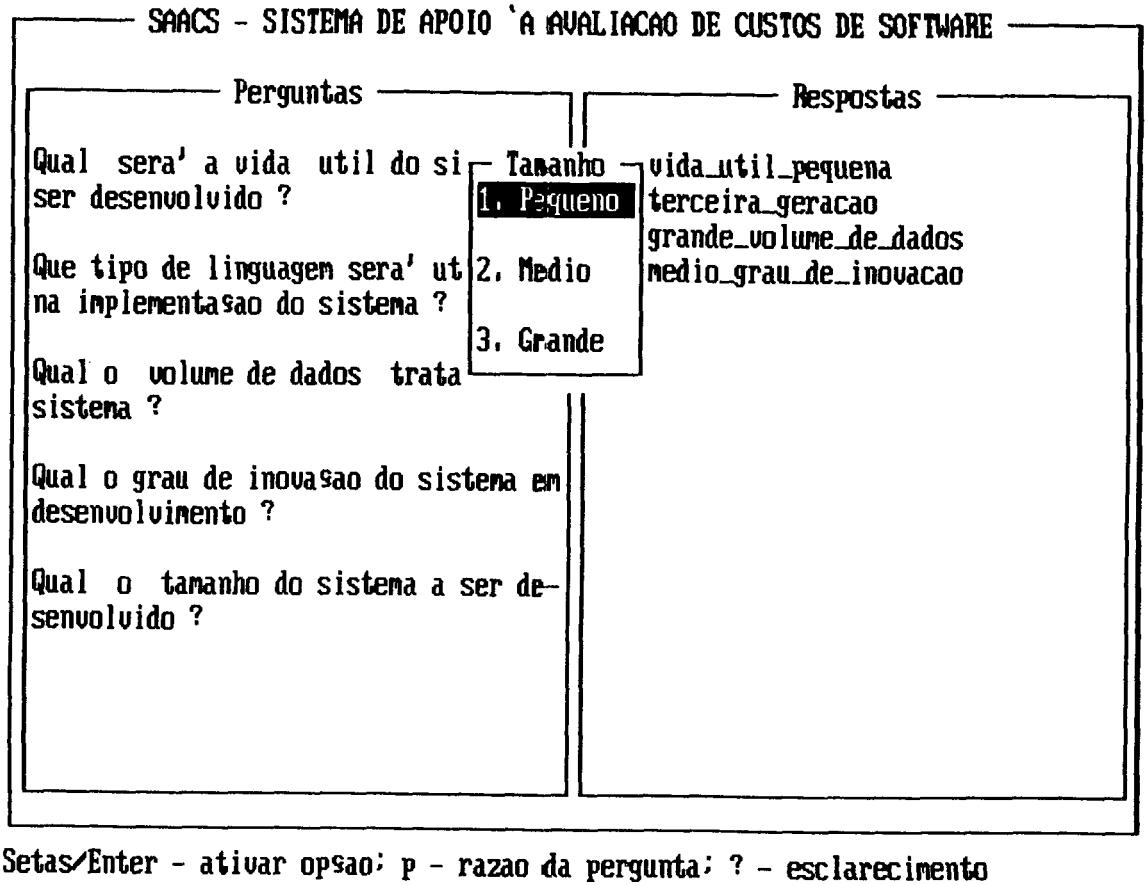

Figura 5.3: Tela de perguntas e respostas.

Os diagnósticos aparecem na janela da esquerda sempre que alguma regra é satisfeita, como consta da figura 5.4.

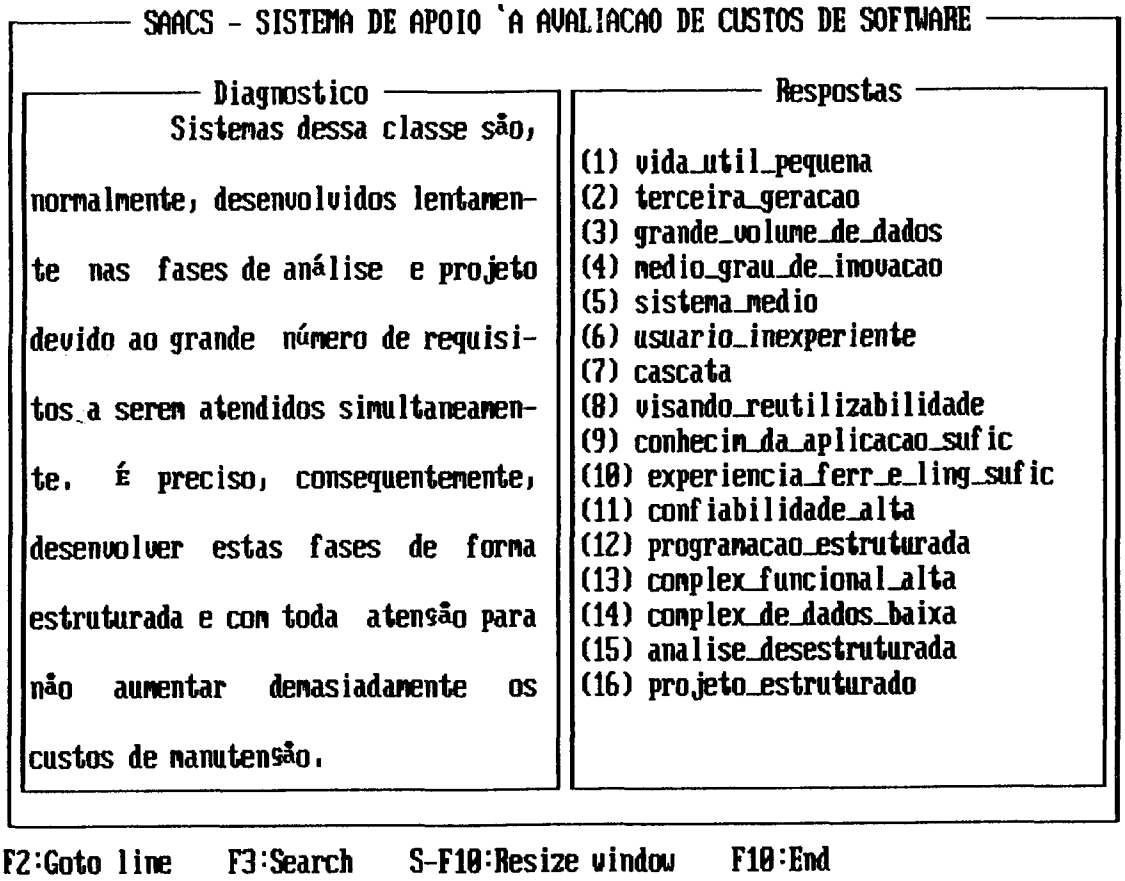

Figura 5.4: Tela de diagnósticos.

Outra característica do sistema é a capacidade de modificar alguma resposta fornecida pelo usuário anteriormente. Este mecanismo é conhecido na literatura como "What if". Caso o usuário deseje alterar algumas das respostas dadas ele poderá fazê-lo, mesmo que depois daquela resposta outras perguntas tenham sido feitas. Nesse caso todas as respostas que dependiam daquela alterada são apagadas e o sistema continua seu processamento.

V.3.6. BASE DE CONHECIMENTO

Contém o conhecimento (regras, fatos, heurísticas) do sistema. O conhecimento que consta do sistema é principalmente o encontrado na literatura.

Para a construção da base de conhecimentos do sistema foi elaborado um estudo bibliográfico a respeito dos fatores de produtividade mais bem definidos na literatura. Consequentemente, diversos fatores tais como restrições de segurança, restrições legais, separação geográfica dos locais de desenvolvimento, entre outros, não foram considerados. Os fatores de produtividade considerados na construção do sitema encontram-se relacionados no capítulo III deste trabalho.

A partir do estudo bibliográfico efetuado, foi elaborado o projeto do sistema, de acordo com o modelo para definição de regras proposto por KELLER [33], descrito no capítulo IV.

A figura 5.5 apresenta o DFD do sistema proposto e a seguir, para cada decisão do diagrama são apresentados seus valores, fatores e regras.

DFD do SISTEMA

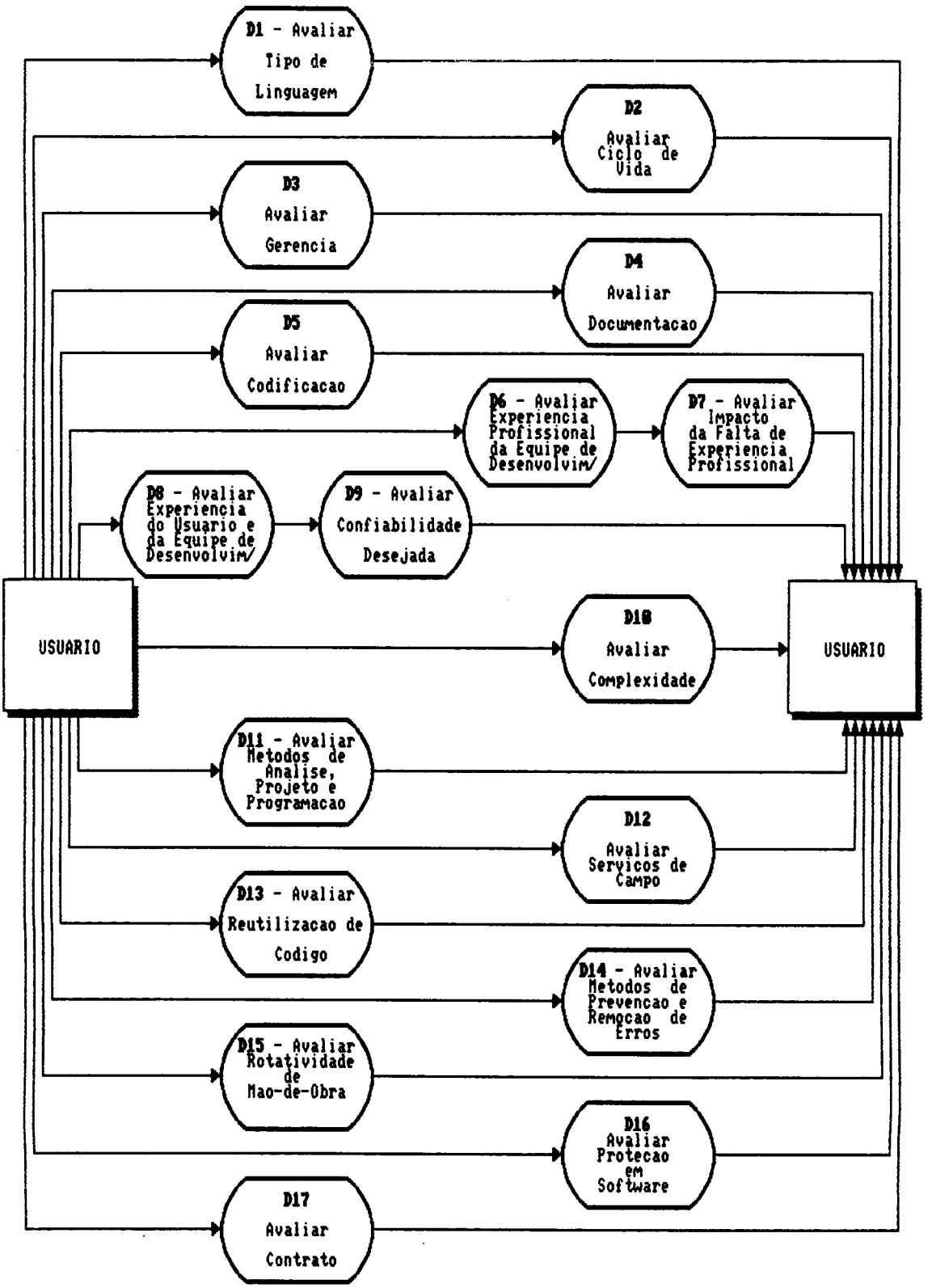

Figura 5.5: DFD do Sistema Proposto.

1) DECISÃO - D1	Valores
Avaliar Tipo de Linguagem	Avaliações descritas em: Regra1.1, Regra1.2, Regra1.3, Regra1.4, Regra1.5.

Fatores	Valores
Vida útil	pequena e longa
Tipo de linguagem	assembler, L3G e L4G
Restrição quanto ao uso do computador	existente e inexistente
Volume de dados	pequeno e grande

Regra 1.1

SE	Vida útil	É	pequena
E	Tipo de Linguagem	É	assembler

ENTÃO

Já que se trata de um sistema com pequena vida útil, se possível, evite programá-lo em assembler pois os custos com codificação e testes serão muito altos. Use, de preferência, uma linguagem de quarta geração. Caso não seja possível, procure utilizar uma linguagem de terceira geração.

Regra 1.2

SE	Vida útil	É	pequena
E	Tipo de Linguagem	É	L3G

ENTÃO

Em função da pequena vida útil do sistema, estude

a possibilidade de se utilizar uma linguagem de quarta geração no desenvolvimento do sistema. Com uma linguagem de quarta geração o custo do sistema poderá ser reduzido em aproximadamente 85%.

Regra 1.3

SE	Vida útil	É	longa
E	Tipo de Linguagem	É	L4G
E	Restrição qto ao uso do computador	É	inexistente

ENTÃO

A substituição de uma linguagem de terceira geração por uma de quarta geração pode acarretar uma economia de, aproximadamente 85%.

No entanto, caso a frequência de execução do sistema seja alta, esta economia poderá ser largamente superada pelos custos de operação do sistema, em função da baixa eficiência de processamento das linguagens de quarta geração.

Embora não haja nenhuma restrição quanto ao uso da máquina, sabe-se que linguagens de quarta geração costumam consumir muitos recursos computacionais. É conveniente, então, avaliar a eficiência da linguagem antes de decidir sobre a sua utilização.

Regra 1.4

SE	Vida útil	É	longa
E	Tipo de Linguagem	É	L4G
E	Restrição qto ao uso do computador	É	existente

ENTÃO

A substituição de uma linguagem de terceira geração por uma de quarta geração pode acarretar uma economia de, aproximadamente 85%.

No entanto, parece totalmente desaconselhável a utilização de uma linguagem de quarta geração neste caso, pois a economia obtida na fase de desenvolvimento será facilmente superada pelos custos de operação do sistema.

No entanto, pelo fato do custo de hardware ser mais baixo que o custo de software valeria a pena aumentar os recursos computacionais de sua instalação.

Regra 1.5

SE	Volume de dados	É	grande
E	Tipo de Linguagem	É	L4G

ENTÃO

Linguagens de quarta geração são, geralmente, ineficientes no processamento de grandes massas de dados. Avalie o custo operacional do sistema antes de se decidir sobre a linguagem a ser utilizada.

2) DECISÃO - D2	Valores
Avaliar Ciclo de Vida	Avaliações descritas em: Regra2.1, Regra2.2, Regra2.3, Regra2.4, Regra2.5, Regra2.6.

Fatores	Valores
Tipo de Linguagem	assembler, L3G e L4G
Grau de inovação	pequeno, médio e grande
Ciclo de vida	cascata, prototipagem evolutiva e prototipagem descartável
Tamanho do sistema	pequeno, médio e grande
Experiência do usuário com a aplicação	insuficiente e suficiente
Interação com o usuário final	não significativa e significativa
Volume de dados	pequeno e grande
Vida útil	pequena e longa

Regra 2.1

SE	Tipo de Linguagem	É	assembler
E	Grau de inovação	É	médio OU alto
E	Ciclo de vida	É	cascata

ENTÃO

Sistemas em assembler são, aproximadamente, 140% mais caros do que os desenvolvidos com linguagens de terceira geração, sendo as fases mais caras a de codificação e a de testes.

Já que o grau de inovação deste sistema não é

baixo e será utilizado o assembler, considere a possibilidade de se desenvolver um protótipo, antes do produto final.

Experiências recentes realizadas por B. Boehm demonstram que o custo de sistemas podem ser reduzidos em até 45% se for utilizada a prototipagem.

Regra 2.2

SE	Experiência do usuário com a aplicação	É	insuficiente
E	Ciclo de vida	É	cascata

ENTÃO

Em função baixo nível de experiência do usuário na área da aplicação, considera-se conveniente prototipar o sistema.

Através da prototipagem o custo de desenvolvimento poderá ser reduzido em até 45%.

Regra 2.3

SE	Experiência do usuário com a aplicação	É	suficiente
E	Ciclo de vida	É	cascata
E	Grau de inovação	É	médio OU alto
E	Tamanho do sistema	É	médio OU grande

ENTÃO

Em virtude do grau de inovação do sistema e do seu tamanho, considera-se conveniente prototipar o sistema.

Através da prototipagem o custo de desenvolvimento poderá ser reduzido em até 45%.

Regra 2.4

SE	Interação com usuário final	É	significativa
E	Tamanho do sistema	É	médio OU grande
E	Ciclo de vida	É	cascata
E	Grau de inovação	É	médio OU alto

ENTÃO

Em função do grau de inovação da aplicação, da significativa interação com o usuário final e do tamanho do sistema, o ciclo de vida com prototipação parece ser o mais indicado.

Segundo JONES, se o tempo e recursos gastos na prototipação totalizarem em torno de 5% a 10% do total planejado para o desenvolvimento, é comum terminar-se o sistema ligeiramente mais cedo e dentro do orçamento.

Pesquisas recentes realizadas por B. BOEHM, verificaram que a prototipagem rápida é capaz de reduzir os custos em até 45%.

Regra 2.5

SE	Ciclo de vida	É	prototipagem descartável
E	Volume de dados	É	pequeno
E	Vida útil	É	pequena

ENTÃO

Os fatores "pequena vida útil" e "pequeno volume de dados" parecem indicar que não deverá haver grande preocupação com eficiência e que o produto gerado provavelmente não sofrerá muita manutenção. Logo, em vez de se desenvolver um protótipo descartável, talvez seja conveniente fazer com que o protótipo evolua para o produto acabado.

Regra 2.6

SE	Ciclo de vida	É	prototipagem evolutiva
E	Vida útil	É	longa
E	Volume de dados	É	grande

ENTÃO

Sistemas desenvolvidos com prototipagem evolutiva costumam possuir baixa eficiência. Como o volume de dados é alto, convém desenvolver um protótipo descartável.

3) DECISÃO - D3	Valores
Avaliar Gerência	Avaliações descritas em: Regra3.1 e Regra3.2.

Fatores	Valores
Tamanho do sistema	pequeno, médio e grande
Gerência	hierárquica, matricial e outras.

Regra 3.1

SE Tamanho do sistema É grande

E Gerência É (não) hierárquica

ENTÃO

A necessidade de comunicação entre os membros que compõem a equipe de desenvolvimento é um dos fatores que aumentam de forma mais que linear com o aumento dos sistemas. É importante, portanto, que os sistemas de grande porte sejam gerenciados de forma a minimizar esta comunicação.

Para sistemas com mais de 50.000 linhas de código a organização hierárquica tem demonstrado ser a de maior sucesso.

Em alguns casos este método de gerência tem apresentado uma redução nos custos da ordem de 12% em relação aos sistemas gerenciados com métodos não hierárquicos.

Regra 3.2

SE Tamanho do sistema É grande

E Gerência É matricial

ENTÃO

A organização matricial de gerência faz com que o número de canais de comunicação entre os membros que compõem a equipe de desenvolvimento aumente geometricamente.

No desenvolvimento de grandes sistemas esta estrutura de gerência induz a um certo nível de confusão, o que, normalmente, leva a um aumento do tempo e dos custos de desenvolvimento.

A organização hierárquica é, geralmente, o método de maior sucesso no desenvolvimento de grandes sistemas.

4) DECISÃO - D4	Valores
Avaliar Documentação	Avaliações descritas em: Regra4.1, Regra4.2, Regra4.3, Regra4.4, Regra4.5, Regra4.6.

Fatores	Valores
Classe do sistema	uso de uma única empresa, uso em comunidade, visando reutilização, prestação de serviço, sistemas para leasing, sistemas para venda, contrato privado, contrato governamental e contrato militar
Ferramenta para documentação ..	inadequada e adequada
Tamanho do sistema	pequeno, médio e grande
Volume de documentação	pequeno e grande
Documentação via HELP	existente e inexistente

Regra 4.1

SE Classe do sistema É prestação de serviço

E Ferramenta para documentação É inadequada

ENTÃO

É conveniente investir em ferramentas para documentação.

Os custos de documentação de sistemas para prestação de serviço são geralmente altos se comparados aos demais tipos de sistemas internos da empresa. Os manuais do usuário são, por exemplo, quase sempre escritos e editados de forma profissional.

Regra 4.2

SE Classe do sistema É sistemas para leasing

E Tamanho do sistema É pequeno OU médio

ENTÃO

Os custos de documentação de sistemas para leasing são, geralmente, muito altos.

Convém adquirir, se necessário, boas ferramentas para esse fim.

Regra 4.3

SE Classe do sistema É sistemas para leasing

E Tamanho do sistema É grande

E Ferramenta para documentação É inadequada

ENTÃO

Os custos de documentação de sistemas de grande porte, para comercialização na forma de leasing podem representar aproximadamente 43% do custo total de desenvolvimento.

A utilização de boas ferramentas de documentação será fundamental para se aumentar a produtividade no desenvolvimento de sistemas com estas características.

Regra 4.4

SE Classe do sistema É sistemas para venda

E Documentação via HELP É inexistente

ENTÃO

A documentação via help deve custar aproximadamente 50% mais caro do que a simples datilografia do texto desejado. No entanto, ela pode se tornar bastante vantajosa no caso de sistemas para venda ao se considerar a inexistência de gastos com impressão, encadernação e distribuição dos manuais para todos os usuários. Esta decisão dependerá do número de usuários do software.

Regra 4.5

SE Classe do sistema É contrato privado
OU contrato governamental
OU contrato militar

E Tamanho do sistema É grande

E Volume de documentação É grande

E Ferramenta para documentação É inadequada

ENTÃO

Os custos de documentação de sistemas de grande porte dessa classe podem representar entre 40% e 50% do custo total de desenvolvimento.

A utilização de boas ferramentas de documentação será fundamental para se aumentar a produtividade no desenvolvimento de sistemas com estas características.

O volume de documentação solicitado pelo contratante costuma ser ainda maior quando o contrato não prevê a realização da manutenção pelo contratado.

Regra 4.6

SE	Classe do sistema	É contrato privado OU contrato governamental OU contrato militar
E	Tamanho do sistema	É pequeno OU médio

ENTÃO

Embora não se possa precisar o impacto da documentação nos custos de desenvolvimento de sistemas de pequeno ou médio porte pertencentes a esta classe, sabe-se que ele é geralmente alto. Recomenda-se, portanto, a utilização de boas ferramentas de documentação com o objetivo de se diminuir estes custos.

5) DECISÃO - D5	Valores
Avaliar Codificação	Avaliações descritas em: Regra5.1.

Fatores	Valores
Tamanho do sistema	pequeno, médio e grande
Classe do sistema	uso de uma única empresa, uso em comunidade, visando reutilização, prestação de serviço, sistemas para leasing, sistemas para venda, contrato privado, contrato governamental e contrato militar

Regra 5.1

SE	Tamanho do sistema	É pequeno OU médio
E	Classe do sistema	É uso de uma única empresa OU uso em comunidade OU visando reutilizabilidade OU prestação de serviço

ENTÃO

Em função do tamanho e da classe do sistema, recomenda-se especial atenção à fase de codificação. Se preciso, invista em linguagens e ferramentas de apoio à codificação.

6) DECISÃO - D6	Valores
Avaliar Experiência Profissional da Equipe de Desenvolvimento.	Experiência profissional alta da equipe de desenvolvimento, Experiência profissional média da equipe de desenvolvimento, Experiência profissional baixa da equipe de desenvolvimento, Experiência profissional muito baixa da equipe de desenvolvimento

Fatores	Valores
Experiência da equipe de desenvolvimento com a aplicação	insuficiente e suficiente
Experiência da equipe de desenvolvimento com linguagens e ferramentas	insuficiente e suficiente

Regra 6.1

SE Experiência da equipe de desenvolvimento com a aplicação É suficiente

E Experiência com linguagens e ferramentas É suficiente

ENTÃO

Experiência profissional alta da equipe de desenvolvimento

Regra 6.2

SE Experiência da equipe de desenvolvimento com a aplicação É suficiente

E Experiência com linguagens e ferramentas É insuficiente

ENTÃO

Experiência profissional média da equipe de desenvolvimento

Regra 6.3

SE Experiência da equipe de desenvolvimento com a aplicação É insuficiente

E Experiência com linguagens e ferramentas É suficiente

ENTÃO

Experiência profissional baixa da equipe de desenvolvimento

Regra 6.4

SE Experiência da equipe de desenvolvimento com a aplicação É insuficiente

E Experiência com linguagens e ferramentas É insuficiente

ENTÃO

Experiência profissional muito baixa da equipe de desenvolvimento

7) DECISÃO - D7	Valores
Avaliar Impacto da Falta de Experiencia Profissional.	Avaliações descritas em: Regra7.1, Regra7.2, Regra7.3, Regra7.4.

Fatores	Valores
Experiência profissional da equipe de desenvolvimento (decisão - D6)	alta, média, baixa e muito baixa.
Tamanho da equipe de desenvolvimento	pequena, média e grande

Regra 7.1

SE Experiência profissional da equipe de desenvolvimento É média

ENTÃO

A experiência profissional da equipe de desenvolvimento pode ser muitíssimo melhorada com algum treinamento.

Uma equipe com experiência na área da aplicação e com linguagens e ferramentas poderá diminuir o custo total de desenvolvimento em aproximadamente 25%.

Regra 7.2

SE Experiência profissional da equipe de desenvolvimento É baixa

ENTÃO

A falta de experiência na área da aplicação é mais prejudicial a uma equipe de desenvolvimento do que a

falta de experiência com linguagens e ferramentas.

Uma equipe com experiência na área da aplicação e com linguagens e ferramentas, poderá diminuir o custo total de desenvolvimento em aproximadamente 50%.

Regra 7.3

SE Experiência profissional da equipe de desenvolvimento É muito baixa

E Tamanho da equipe de desenvolvimento É pequena

ENTÃO

Uma equipe de desenvolvimento de sistemas com experiência tanto na área da aplicação quanto na utilização de linguagens e ferramentas poderá diminuir o custo total de um sistema em aproximadamente 150%.

Em equipes pequenas o custo pode ser ainda maior.

Regra 7.4

SE Experiência profissional da equipe de desenvolvimento É muito baixa

E Tamanho da equipe de desenvolvimento É média OU grande

ENTÃO

Talvez seja conveniente contratar pessoal com experiência na área da aplicação a ser desenvolvida. Quanto à falta de experiência com as linguagens e ferramentas a serem utilizadas, treine-os.

O custo de um sistema desenvolvido com pessoal inexperiente é aproximadamente 150% superior ao custo dos sistemas desenvolvidos com pessoal experiente.

8) DECISÃO - D8	Valores
Avaliar Experiência do usuário e da equipe de desenvolvimento com a aplicação	Experiência insuficiente do usuário e da equipe de desenvolvimento com a aplicação Experiência suficiente do usuário e da equipe de desenvolvimento com a aplicação

Fatores	Valores
Experiência da equipe de desenvolvimento com a aplicação	insuficiente e suficiente
Experiência do usuário com a aplicação	insuficiente e suficiente

Regra 8.1

SE Experiência da equipe de desenvolvimento com a aplicação É insuficiente

E Experiência do usuário com a aplicação É insuficiente

ENTÃO

Experiência insuficiente do usuário e da equipe de desenvolvimento com a aplicação.

Regra 8.2

SE Experiência da equipe de desenvolvimento com a aplicação É suficiente

E Experiência do usuário com a aplicação É insuficiente

ENTÃO

Experiência insuficiente do usuário e da equipe de desenvolvimento com a aplicação.

Regra 8.3

SE Experiência da equipe de desenvolvimento com a aplicação É insuficiente

E Experiência do usuário com a aplicação É suficiente

ENTÃO

Experiência insuficiente do usuário e da equipe de desenvolvimento com a aplicação.

Regra 8.4

SE Experiência da equipe de desenvolvimento com a aplicação É suficiente

E Experiência do usuário com a aplicação É suficiente

ENTÃO

Experiência suficiente do usuário e da equipe de desenvolvimento com a aplicação.

9) DECISÃO - D9	Valores
Avaliar Confiabilidade Desejada	Avaliações descritas em: Regra9.1.

Fatores	Valores
Confiabilidade desejada	moderada, alta e muito alta
Programação	estruturada e desestruturada
Experiência do usuário e da equipe de desenvolvimento com a aplicação (decisão - D8) .	insuficiente e suficiente
Requisitos do sistema	instáveis e estáveis
Desenvolvimento dos programas	cuidadoso e não cuidadoso

Regra 9.1

SE	Confiabilidade desejada	É alta OU muito alta
E (	Programação	É desestruturada
OU	Experiência do usuário e da equipe de desenvolvimento com a aplicação	É insuficiente
OU	Requisitos do sistema	SÃO instáveis
OU	Desenvolvimento dos programas	É (não) cuidadoso)

ENTÃO

Para que um sistema seja de alta ou altíssima confiabilidade é preciso:

- que a programação seja estruturada;

- que haja familiaridade com a área da aplicação por parte dos usuários e desenvolvedores;

- que os requisitos sejam estáveis;

- que os programadores tenham tempo para programar cuidadosamente.

Estes são os mais importantes requisitos para a construção de sistemas confiáveis.

O custo de um sistema de alta ou altíssima confiabilidade é aproximadamente 15% a 40% superior ao de um sistema de confiabilidade moderada. Sistemas de confiabilidade moderada são aqueles que em caso de falha provocam perdas moderadas e recuperáveis.

10) DECISÃO - D10	Valores
Avaliar Complexidade	Avaliações descritas em: Regra10.1, Regra10.2 e Regra10.3.

Fatores		Valores
Complexidade funcional	.....	alta e baixa
Complexidade de dados	.....	alta e baixa

Regra 10.1

SE Complexidade funcional É alta

E Complexidade de dados É baixa

ENTÃO

A alta complexidade funcional faz com que as fases de codificação e de testes sejam 19% mais caras e o esforço total aproximadamente 15% superior ao que seria necessário no desenvolvimento de sistemas com baixa complexidade funcional e com baixa complexidade de dados.

Regra 10.2

SE Complexidade funcional É baixa

E Complexidade de dados É alta

ENTÃO

A influência da complexidade de dados é superior à da complexidade funcional.

Sistemas com alta complexidade de dados costumam ter altos custos na fase de testes e, posteriormente, na

fase de manutenção.

O esforço total de desenvolvimento é aproximadamente 100% superior ao esforço necessário no desenvolvimento de sistemas com baixa complexidade funcional e de dados.

Regra 10.3

SE Complexidade funcional É alta

E Complexidade de dados É alta

ENTÃO

No desenvolvimento de sistemas com alta complexidade de dados e com alta complexidade funcional, é preciso utilizar métodos eficientes para a prevenção e remoção de erros. A prototipagem rápida é indicada, sempre que possível.

O esforço total de desenvolvimento é aproximadamente 200% superior ao esforço necessário para se desenvolver sistemas com baixa complexidade funcional e de dados.

Para se reduzir a complexidade do sistema sugere-se a contratação de pessoal com experiência na área da aplicação, a elaboração cuidadosa do projeto do sistema e o desenvolvimento de programas estruturados. Os programadores devem ser orientados no sentido de programar cuidadosamente, objetivando a redução dos custos em testes e manutenção.

11) DECISÃO - D11	Valores
Avaliar Métodos de Análise, Projeto e Programação.	Avaliações descritas em: Regra11.1, Regra11.2, Regra11.3, Regra11.4, Regra11.5, Regra11.6, Regra11.7, Regra11.8, Regra11.9.

Fatores	Valores
Análise	estruturada e desestruturada
Projeto	estruturado e desestruturado
Programação	estruturada e desestruturada
Classe do sistema	uso de uma única empresa, uso em comunidade, visando reutilização, prestação de serviço, sistemas para leasing, sistemas para venda, contrato privado, contrato governamental e contrato militar

Regra 11.1

SE	Análise	É	estruturada
E	Projeto	É	desestruturado
E	Programação	É	estruturada

ENTÃO

Em relação aos sistemas com análise, projeto e programação estruturada, a construção de um sistema de forma desestruturada na fase de projeto deve provocar um acréscimo de aproximadamente 20% no custo de desenvolvimento e de 85% no custo de manutenção, considerados os cinco primeiros anos de vida útil do

sistema.

Regra 11.2

SE	Análise	É	desestruturada
E	Projeto	É	estruturado
E	Programação	É	estruturada

ENTÃO

Em relação aos sistemas com análise, projeto e programação estruturada, a construção de um sistema de forma desestruturada na fase de análise deve provocar um acréscimo de aproximadamente 20% no custo de desenvolvimento e de 85% no custo de manutenção, considerados os cinco primeiros anos de vida útil do sistema.

Regra 11.3

SE	Análise	É	estruturada
E	Projeto	É	estruturado
E	Programação	É	desestruturada

ENTÃO

Em relação aos sistemas com análise, projeto e programação estruturada, a construção de um sistema de forma desestruturada na fase de programação deve provocar um acréscimo de aproximadamente 27% no custo de desenvolvimento e de 70% no custo de manutenção,

considerados os cinco primeiros anos de vida útil do sistema.

Regra 11.4

SE	Análise	É	desestruturada
E	Projeto	É	desestruturado
E	Programação	É	estruturada

ENTÃO

Sistemas desenvolvidos sem a utilização de métodos estruturados nas fases de análise e projeto são relativamente baratos na fase de desenvolvimento: apenas 20% mais caros do que os sistemas desenvolvidos com análise, projeto e programação estruturada.

A fase de manutenção, no entanto, é extremamente cara: aproximadamente 300% mais cara do que os sistemas com análise, projeto e programação estruturada. Isto ocorre porque a maior parte dos erros encontrados nos sistemas após sua aceitação têm origem nas fases de análise e projeto, que neste caso não foram bem definidas.

A ausência de um trabalho estruturado nas fases de análise e projeto torna muito difícil a elaboração dos testes, fazendo com que os problemas só apareçam após a aceitação do sistema.

Regra 11.5

SE Análise É desestruturada

E Projeto É estruturado

E Programação É desestruturada

ENTÃO

Em relação aos sistemas com análise, projeto e programação estruturada, a construção de um sistema de forma desestruturada nas fases de análise e programação deve provocar um acréscimo de aproximadamente 30% no custo de desenvolvimento e de 80% no custo de manutenção, considerados os cinco primeiros anos de vida útil do sistema.

Regra 11.6

SE Análise É estruturada

E Projeto É desestruturado

E Programação É desestruturada

ENTÃO

Em relação aos sistemas com análise, projeto e programação estruturada, a construção de um sistema de forma desestruturada nas fases de projeto e programação deve provocar um acréscimo de aproximadamente 30% no custo de desenvolvimento e de 80% no custo de manutenção, considerados os cinco primeiros anos de vida útil do sistema.

Regra 11.7

SE	Análise	É	desestruturada
E	Projeto	É	desestruturado
E	Programação	É	desestruturada

ENTÃO

Não utilizar nenhum método estruturado no desenvolvimento de sistemas faz com que tanto o custo de desenvolvimento quanto o de manutenção sejam extremamente altos.

A não utilização de técnicas estruturadas nas fases de análise e projeto, torna o custo da fase de manutenção aproximadamente 330% superior ao mesmo custo nos sistemas desenvolvidos com análise, projeto e programação estruturadas.

Isto ocorre porque a ausência de um trabalho estruturado nas duas primeiras fases do desenvolvimento de sistemas faz com que seja muito difícil definir que testes devem ser realizados.

A não utilização de técnicas estruturadas na fase de programação provoca um aumento do esforço em codificação e testes da ordem de 40%.

Regra 11.8

SE	Classe do sistema	É	(uso em comunidade OU visando reutilizabilidade)
E	(Análise	É	desestruturada
OU	Projeto	É	desestruturado)

ENTÃO

Sistemas dessa classe são, normalmente, desenvolvidos lentamente nas fases de análise e projeto devido ao grande número de requisitos a serem atendidos simultaneamente. É preciso, consequentemente, desenvolver estas fases de forma estruturada e com toda atenção para não aumentar demasiadamente os custos de manutenção.

Regra 11.9

SE	Classe do sistema	É	(uso em comunidade OU visando reutilizabilidade)
E	Programação	É	desestruturada

ENTÃO

Essa classe de sistemas não deve ser programada de forma desestruturada, uma vez que os programas precisam possuir alta qualidade.

12) DECISÃO - D12	Valores
Avaliar Serviços de Campo	Avaliações descritas em: Regra12.1.

Fatores	Valores
Suporte local na instalação de produtos	existente e inexistente
Serviço de campo	existente e inexistente

Regra 12.1

SE	Suporte local na instalação de produtos	É	existente
OU	Serviço de campo	É	existente

ENTÃO

Serviço de campo e suporte local na instalação de produtos são os dois mais caros serviços da fase de manutenção.

13) DECISÃO - D13	Valores
Avaliar Reutilizabilidade	Avaliações descritas em: Regra13.1.

Fatores	Valores
Reutilizabilidade de código	existente e inexixtente
Tamanho do sistema	pequeno, médio e grande

Regra 13.1

SE Reutilizabilidade de código É existente

E Tamanho do sistema É médio OU grande

ENTÃO

Através da reutilizabilidade de código e/ou de projetos padrão é possível obter reduções da ordem de 78% do tempo de desenvolvimento e de 50% do custo.

A estabilização da equipe de programadores e a eliminação do "backlog" de sistemas são alguns benefícios desta técnica de desenvolvimento de software.

14) DECISÃO - D14	Valores
Avaliar Métodos de Prevenção e Remoção de Erros	Avaliações descritas em: Regra14.1, Regra14.2, Regra14.3.

Fatores	Valores
Execução somente de testes	positivo e negativo
Revisão de projeto	existente e inexistente
Inspeção de projeto	existente e inexistente
Confiabilidade desejada	moderada, alta e muito alta

Regra 14.1

SE Execução somente de testes É positivo

ENTÃO

A simples realização de testes visando a remoção dos erros potencialmente existentes em um sistema, além de ineficiente é tão cara quanto a realização de revisões, inspeções e testes.

Regra 14.2

SE Revisão de projeto É inexistente

E Inspeção de projeto É inexistente

ENTÃO

Um sistema que não tenha passado por revisões e inspeções de projeto terá uma fase de testes muito cara.

Devido à baixa eficiência dos testes como método

de remoção de erros, muitos problemas só aparecerão durante a operação do sistema, o que aumentará acentuadamente os custos de manutenção.

Regra 14.3

SE Confiabilidade desejada É alta OU muito alta

ENTÃO

Em função do nível de confiabilidade desejada para o sistema, a melhor combinação de métodos de prevenção e remoção de erros parece ser:

- inspeção das funções críticas do projeto;
- prototipação utilizando uma L4G ou uma linguagem interpretada para que o usuário possa ver as principais saídas do sistema;
- teste de unidade, funcional e de integração.

15) DECISÃO - D15	Valores
Avaliar Rotatividade de Mão-de-Obra	Avaliações descritas em: Regra15.1.

Fatores	Valores
Plano de recompensa	existente e inexistente

Regra 15.1

SE Plano de recompensa É inexistente

ENTÃO

A médio prazo haverá uma alta rotatividade de mão-de-obra o que provocará a formação de uma equipe de baixa qualidade.

Como a experiência do pessoal de desenvolvimento é um importante fator de custos, a falta de um plano de recompensa provocará um aumento do custo dos futuros sistemas.

16) DECISÃO - D16	Valores
Avaliar Proteção em Software	Avaliações descritas em: Regra16.1.

Fatores	Valores
Classe do sistema	uso de uma única empresa, uso em comunidade, visando reutilização, prestação de serviço, sistemas para leasing, sistemas para venda, contrato privado, contrato governamental e contrato militar
Proteção contra cópias	existente e inexistente

Regra 16.1

SE Classe do sistema É sistemas para venda

E Proteção contra cópias É existente

ENTÃO

A proteção dos pacotes contra cópias não autorizadas é um aspecto relativamente caro no desenvolvimento de sistemas dessa classe e deve ser considerado.

17) DECISÃO - D17	Valores
Avaliar Contrato	Avaliações descritas em: Regra17.1.

Fatores	Valores
Classe do sistema	uso de uma única empresa, uso em comunidade, visando reutilização, prestação de serviço, sistemas para leasing, sistemas para venda, contrato privado, contrato governamental e contrato militar
Experiência do usuário com a aplicação	insuficiente e suficiente

Regra 17.1

SE	Classe do sistema	É	contrato privado
		OU	contrato governamental
		OU	contrato militar
E	Experiência do usuário com a aplicação	É	insuficiente

ENTÃO

Tenha cuidado na formulação do contrato. Usuários que não sabem o que querem podem prejudicar a produtividade de forma significativa.

V.4. EXEMPLO DE UMA SESSÃO DO SISTEMA

O objetivo principal deste exemplo é mostrar que diagnósticos podem ser fornecidos ao usuário do sistema, no decorrer de uma sessão.

A figura 5.6 apresenta a tela de abertura do sistema. Nesta tela encontram-se definidas na janela "Menu Principal" as principais funções do sistema. Para ativá-las o usuário deve mover o cursor para a linha da função desejada e teclar <enter>. A definição detalhada de cada função pode ser encontrada na sessão V.3.5., que trata da interface com o usuário. Com relação ao modo de operação do sistema deve ser consultado o Manual do Usuário.

As telas apresentadas nas figuras 5.7, 5.8 e 5.9 mostram de que forma as perguntas elaboradas pelo sistema devem ser respondidas, isto é, pela escolha de uma das opções apresentadas ao usuário através de janelas. As respostas às perguntas são colocadas pelo sistema na janela de respostas para que o usuário possa consultá-las sempre que julgar necessário.

Em seguida são apresentadas telas com todos os diagnósticos fornecidos pelo sistema ao longo da sessão exemplo. Estes diagnósticos são mostrados nas figuras 5.10 a 5.24.

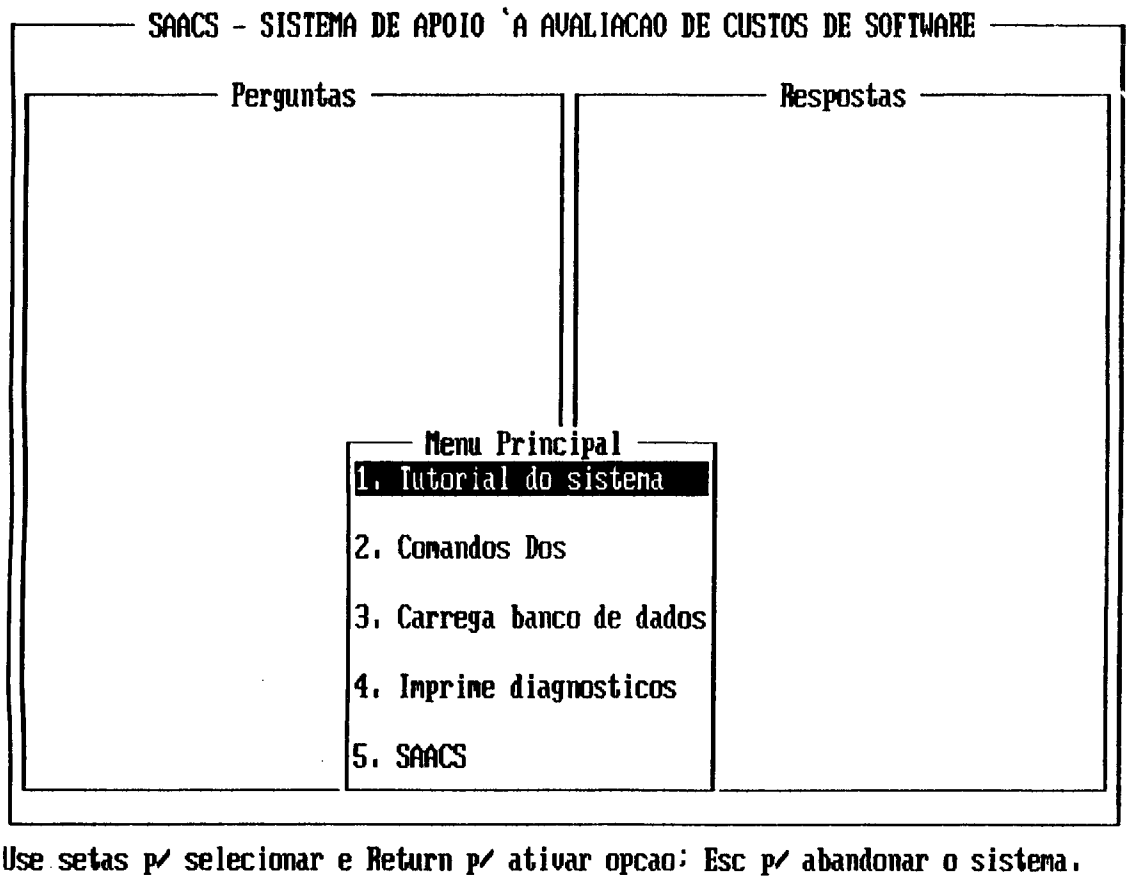

Figura 5.6: Tela de Apresentação.

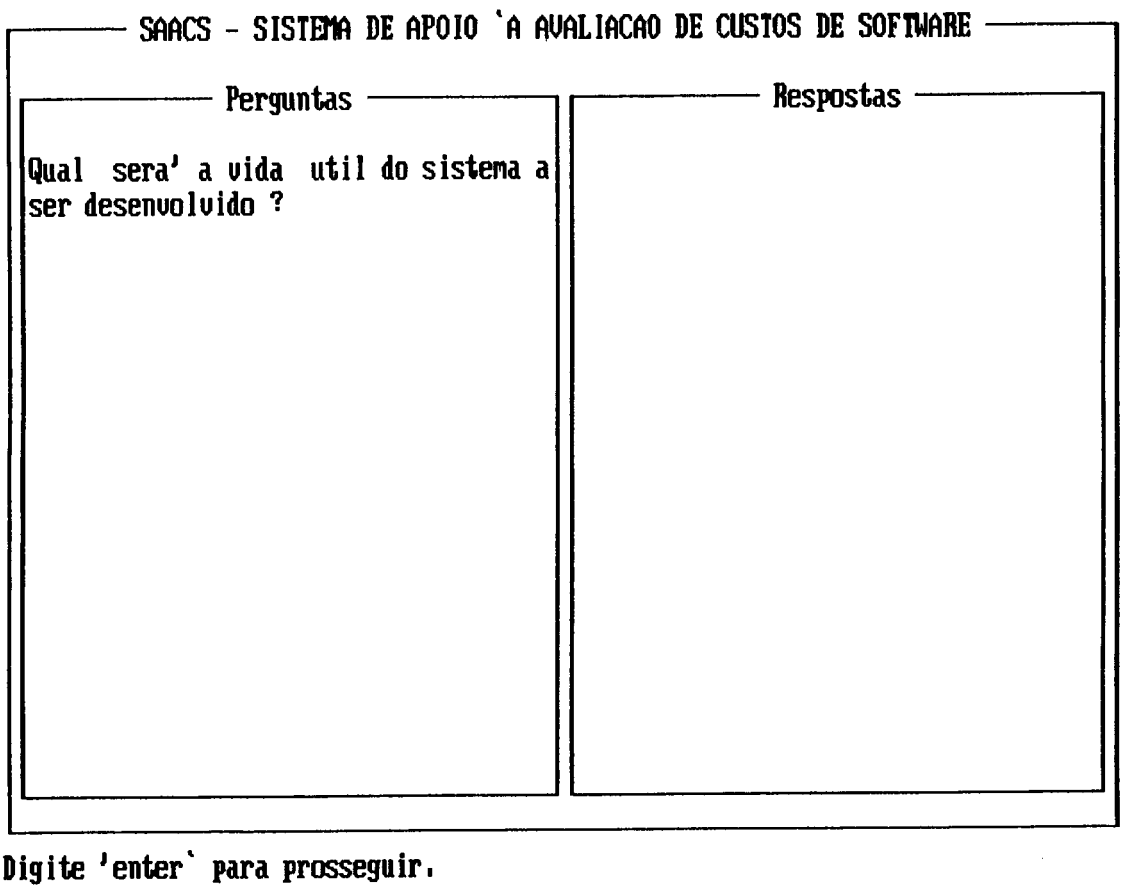

Figura 5.7: Pergunta formulada.

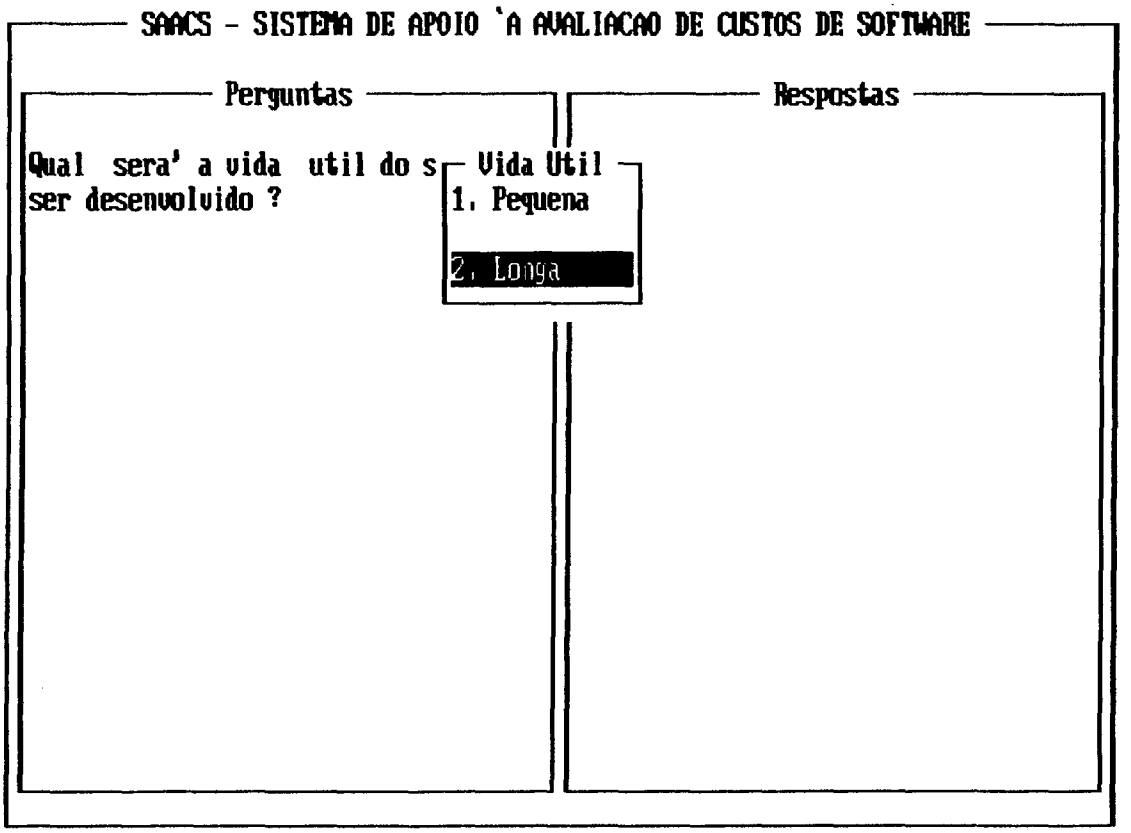

Figura 5.8: Possíveis Respostas.

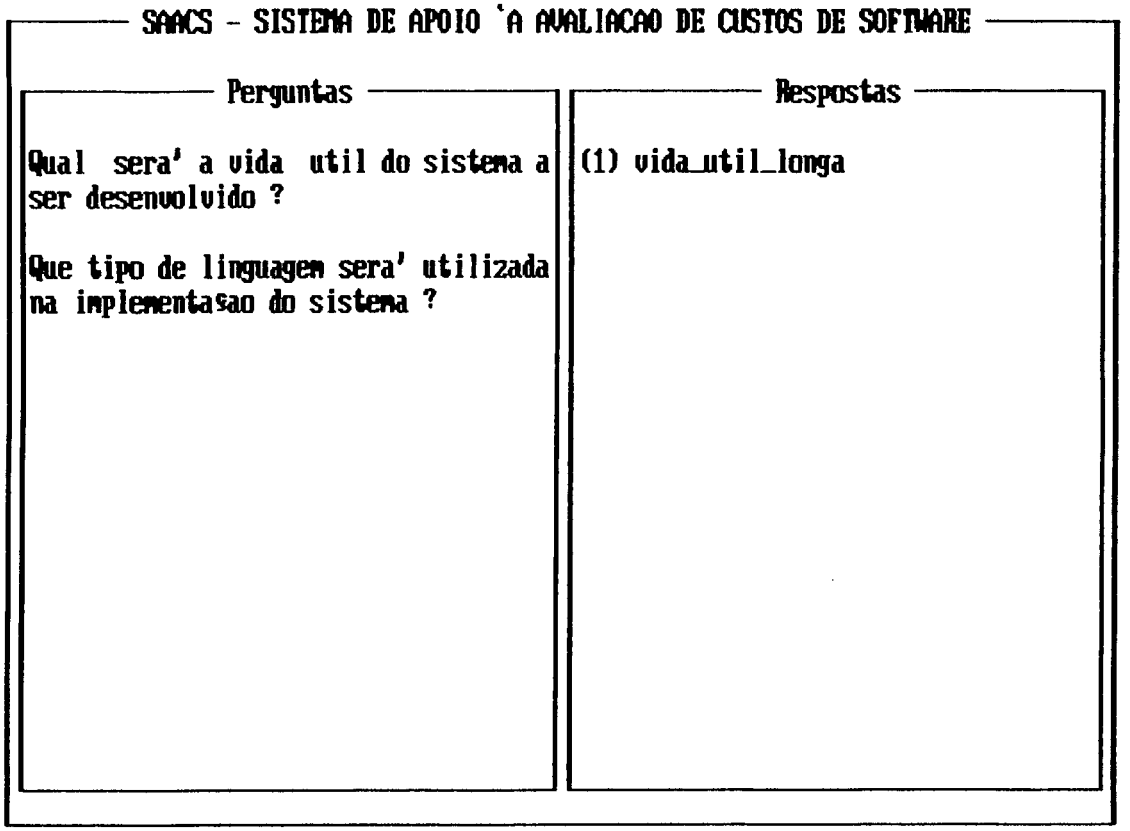

Figura 5.9: Perguntas e respostas.

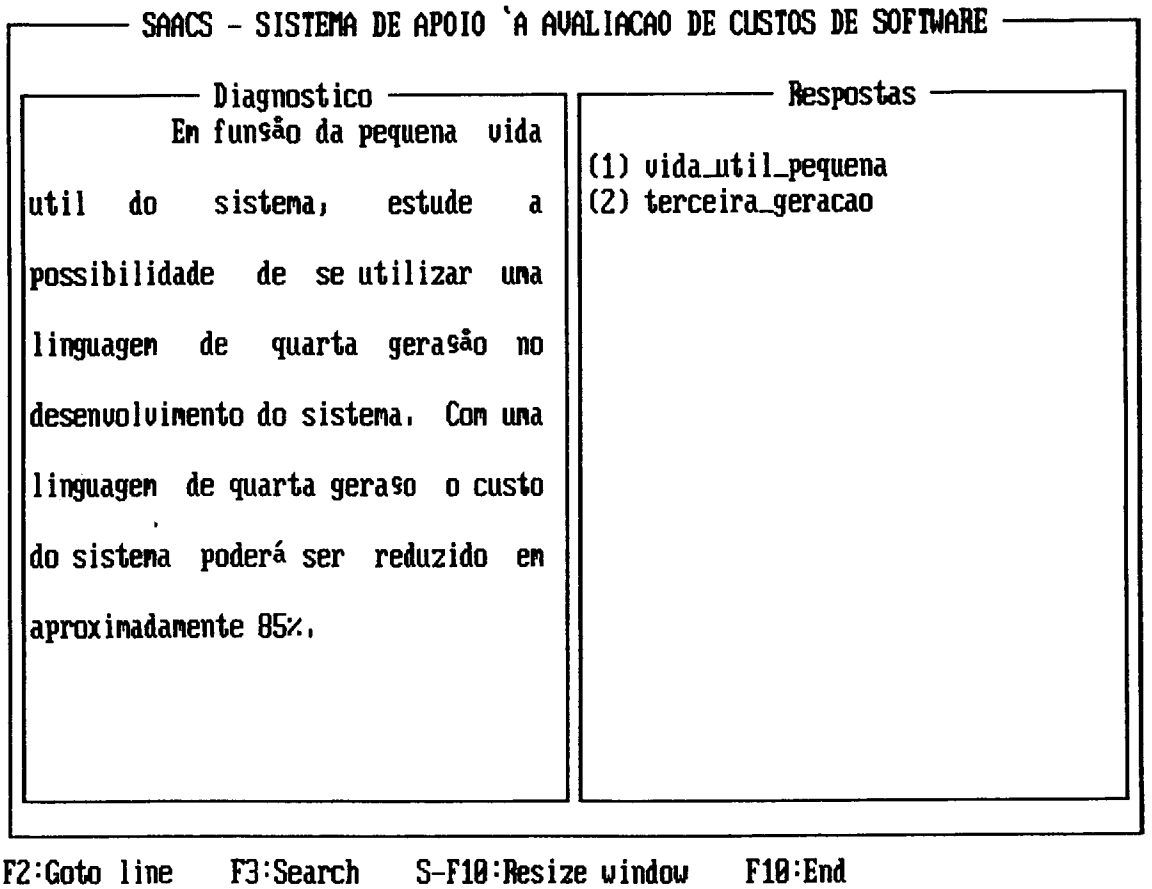

Figura 5.10: Diagnóstico 1.

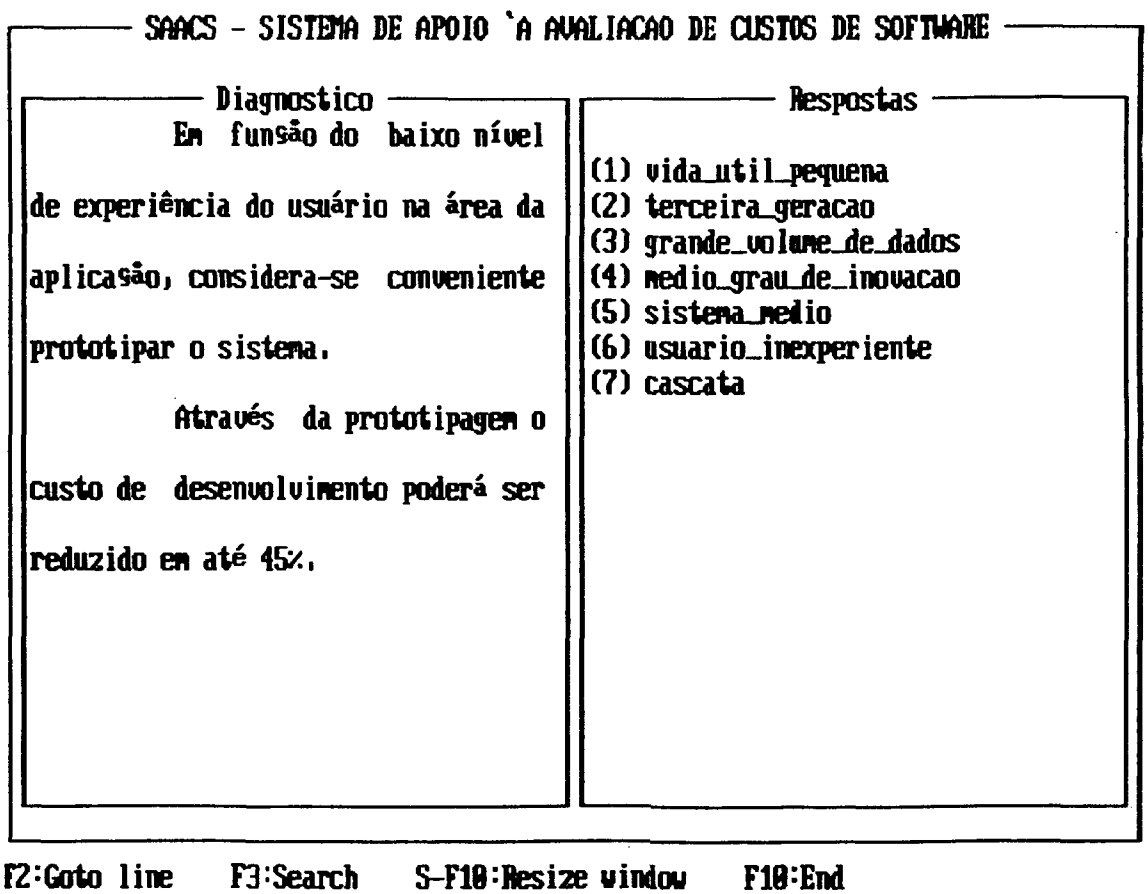

Figura 5.11: Diagnóstico 2.

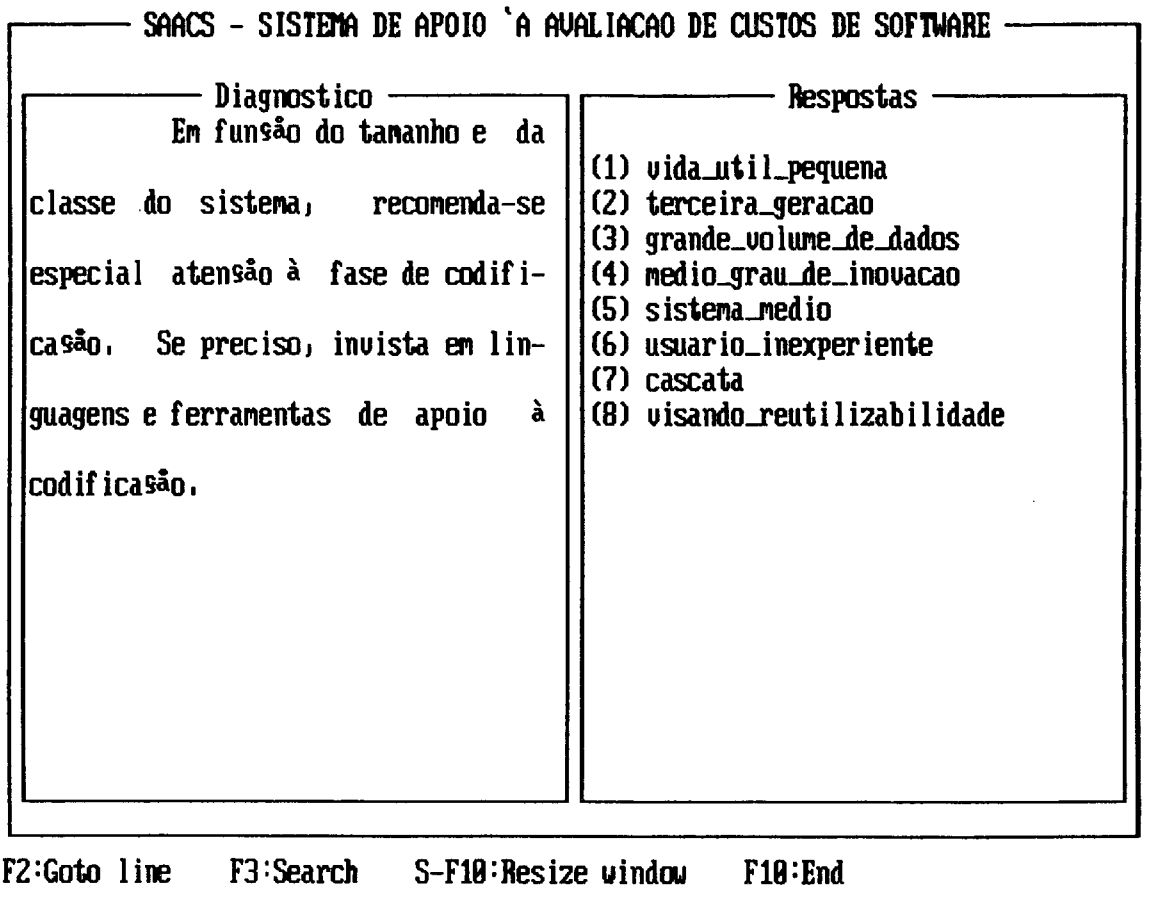

Figura 5.12: Diagnóstico 3.

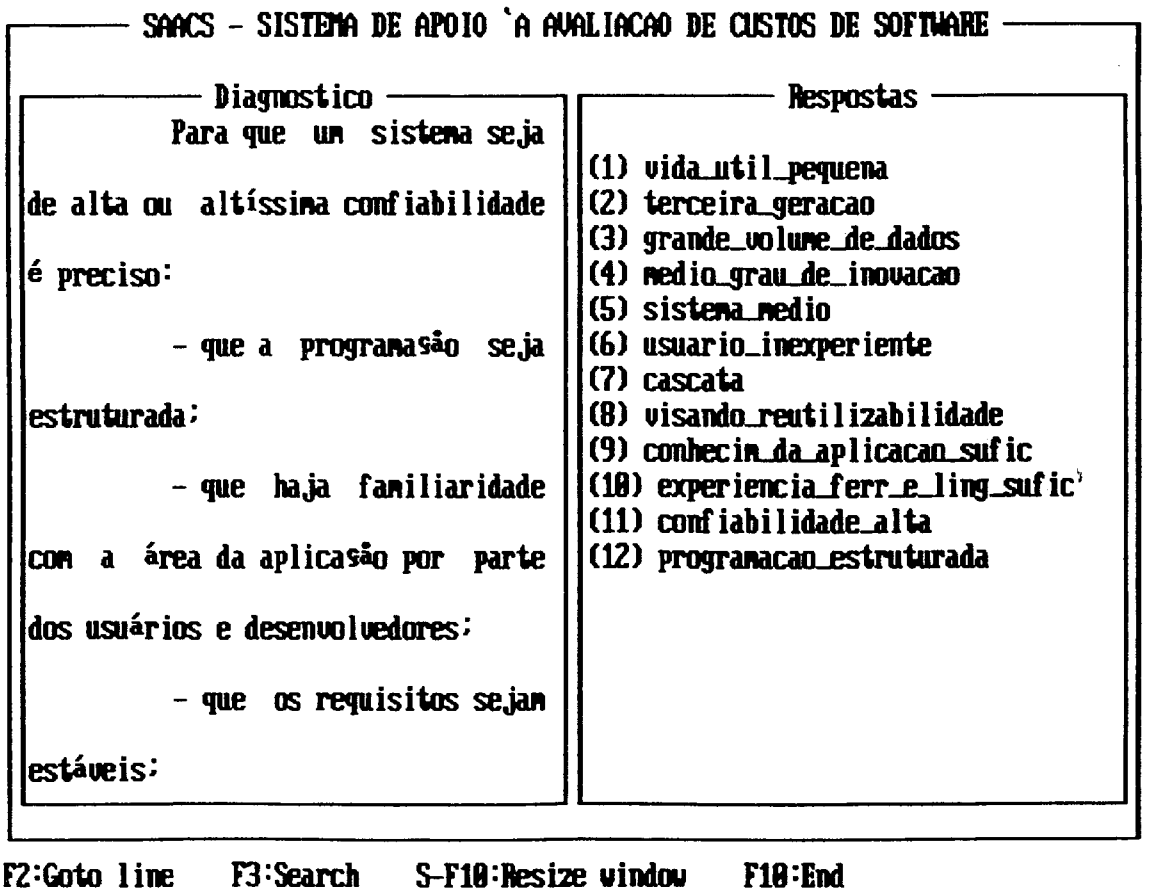

Figura 5.13: Diagnóstico 4.1.

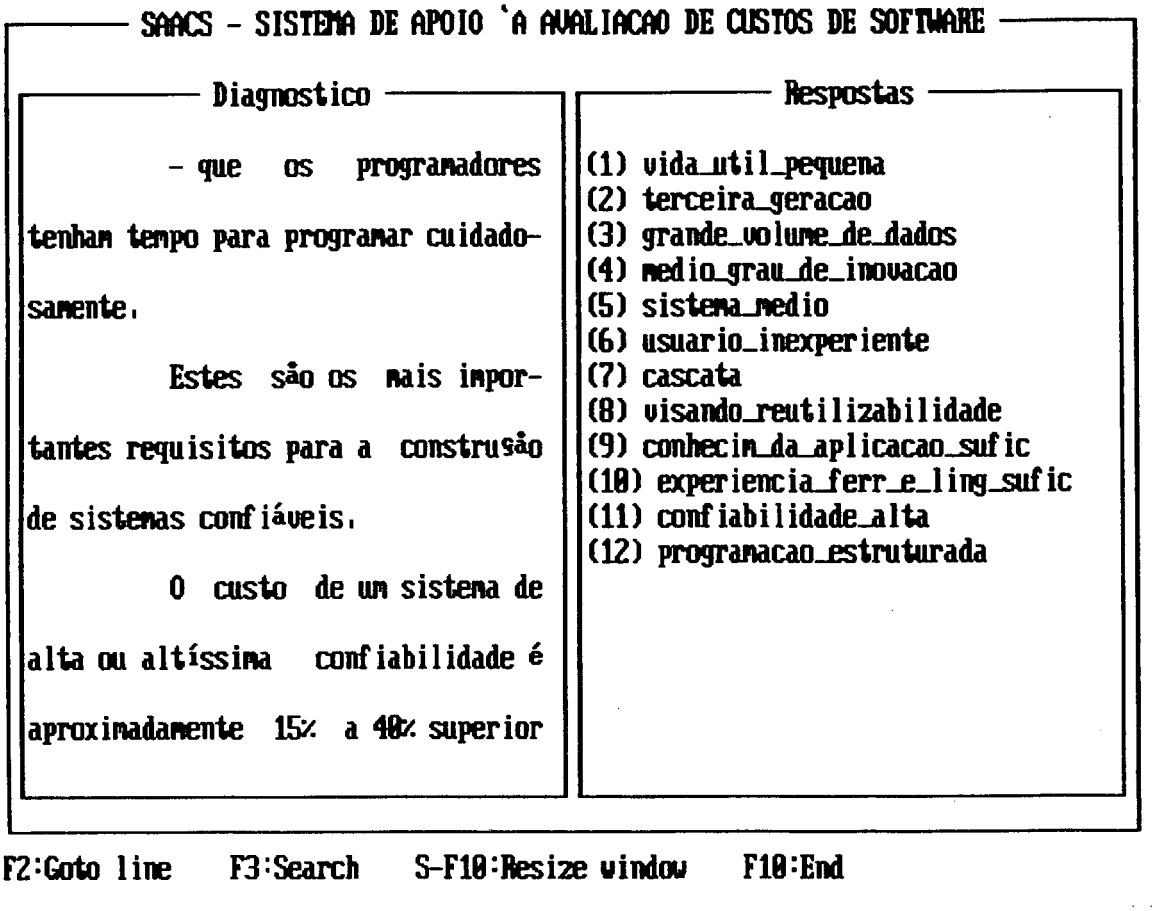

Figura 5.14: Diagnóstico 4.2.

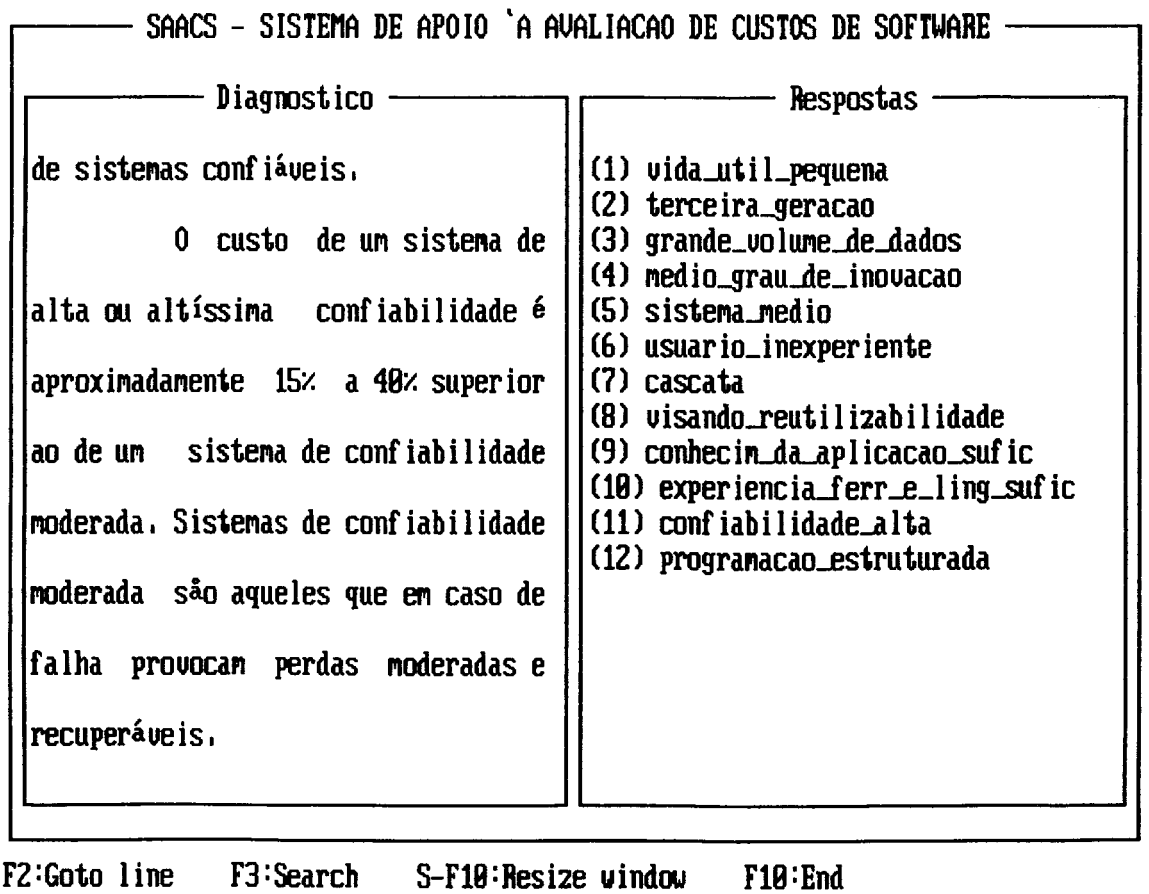

Figura 5.15: Diagnóstico 4.3.

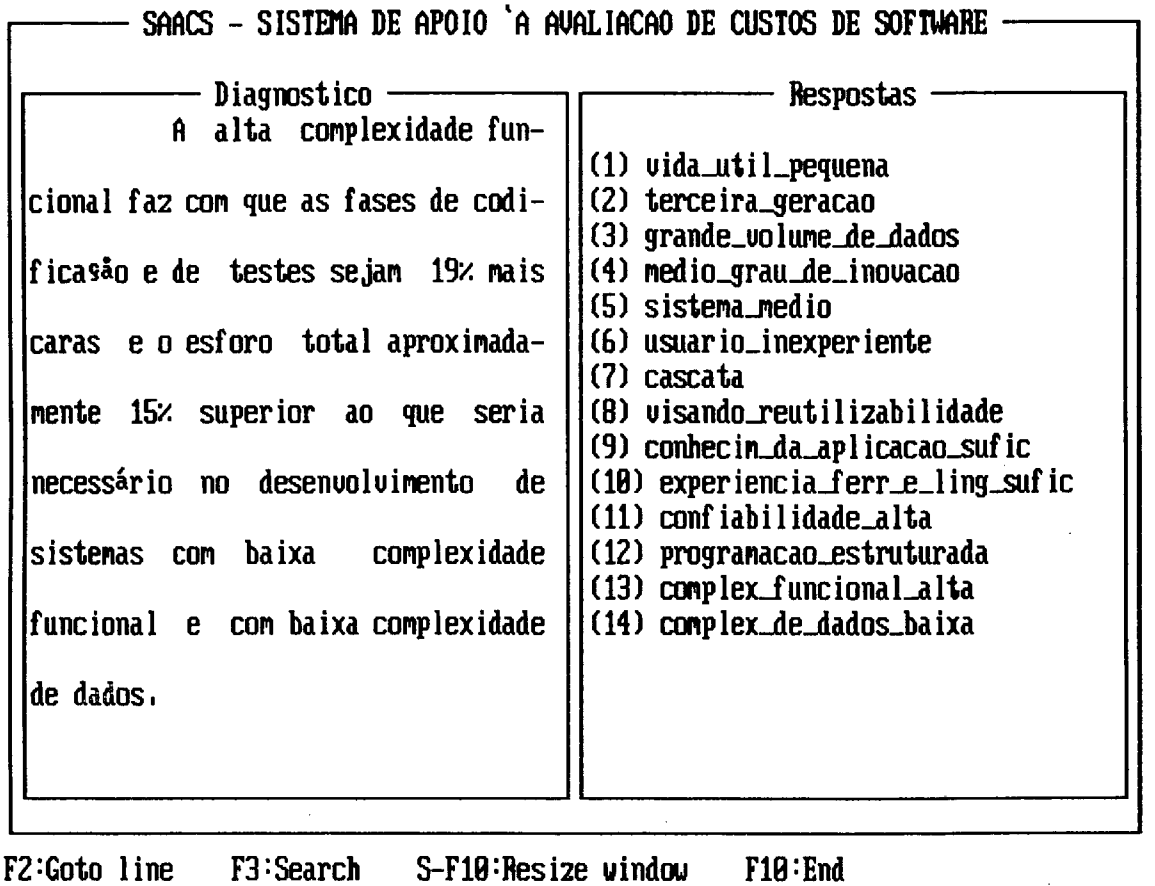

Figura 5.16: Diagnóstico 5.

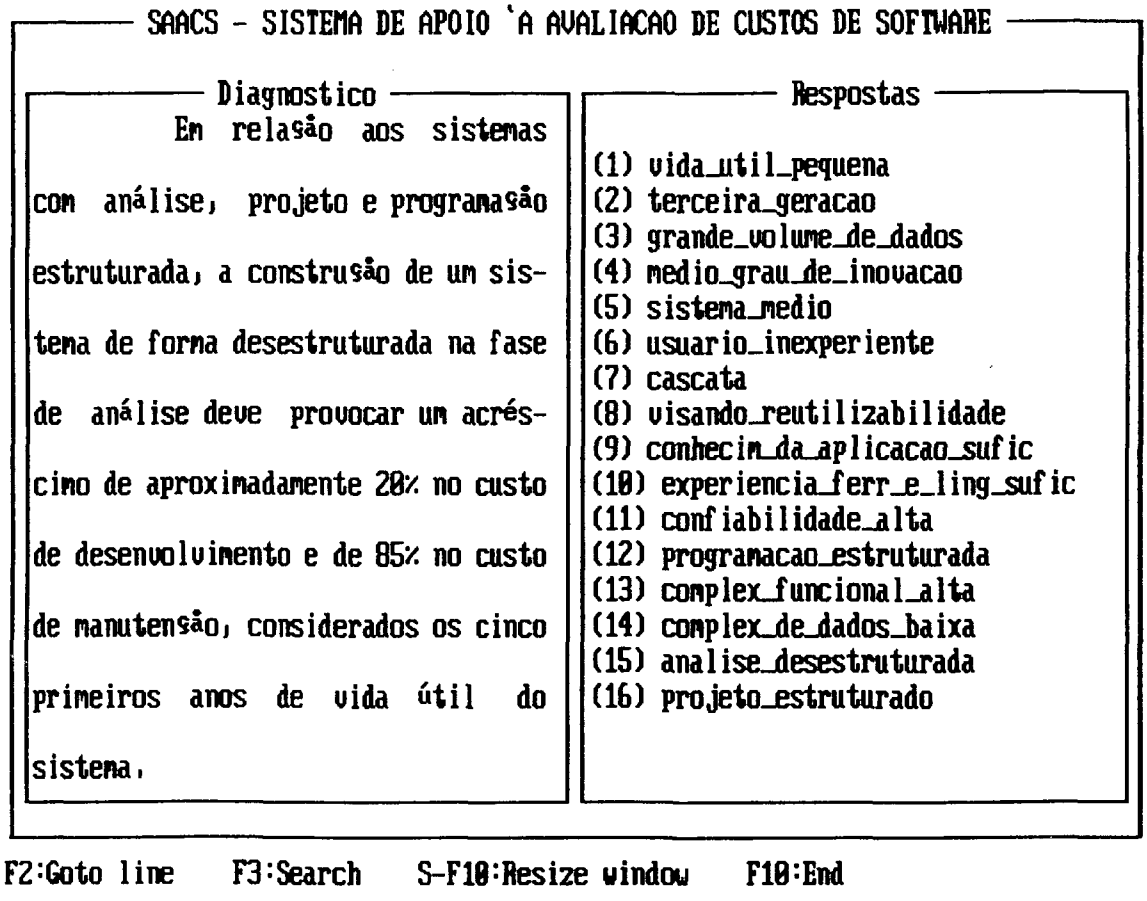

Figura 5.17: Diagnóstico 6.

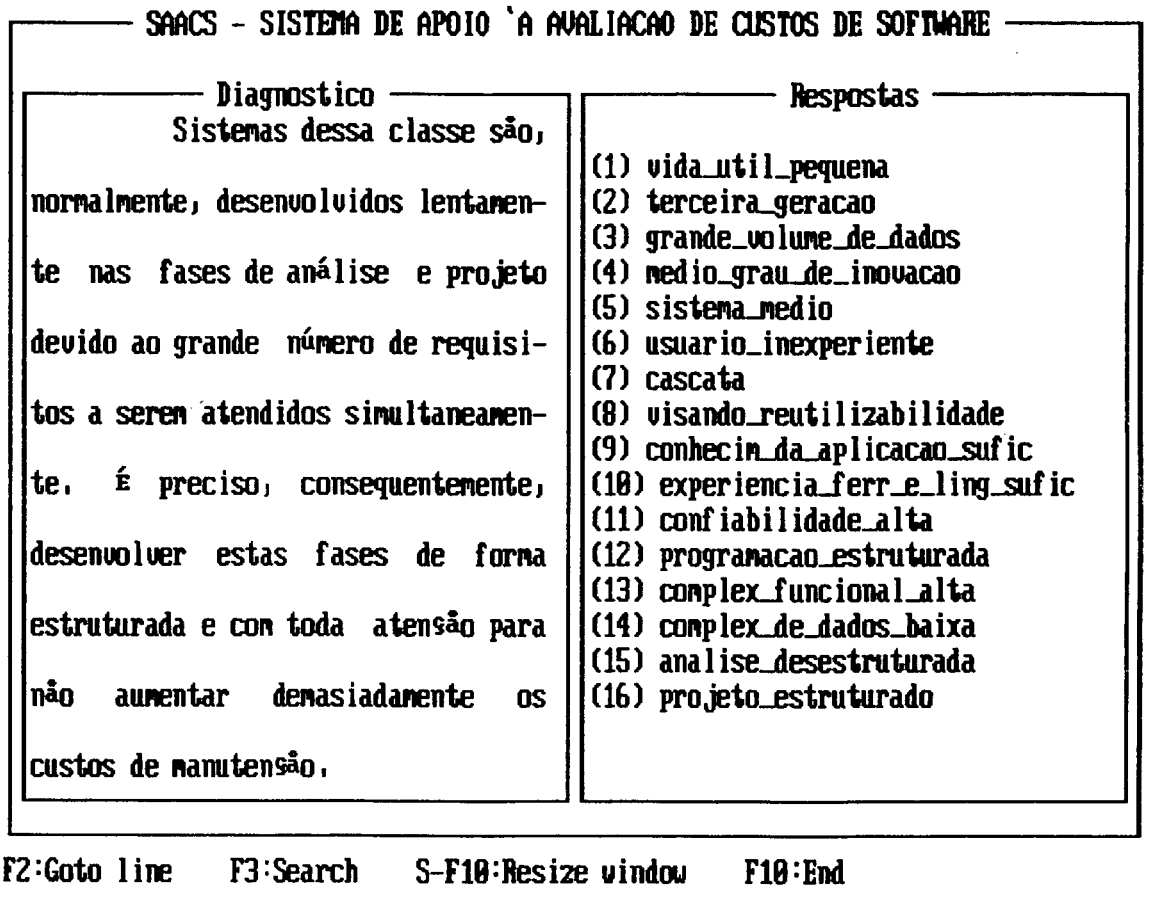

Figura 5.18: Diagnóstico 7.

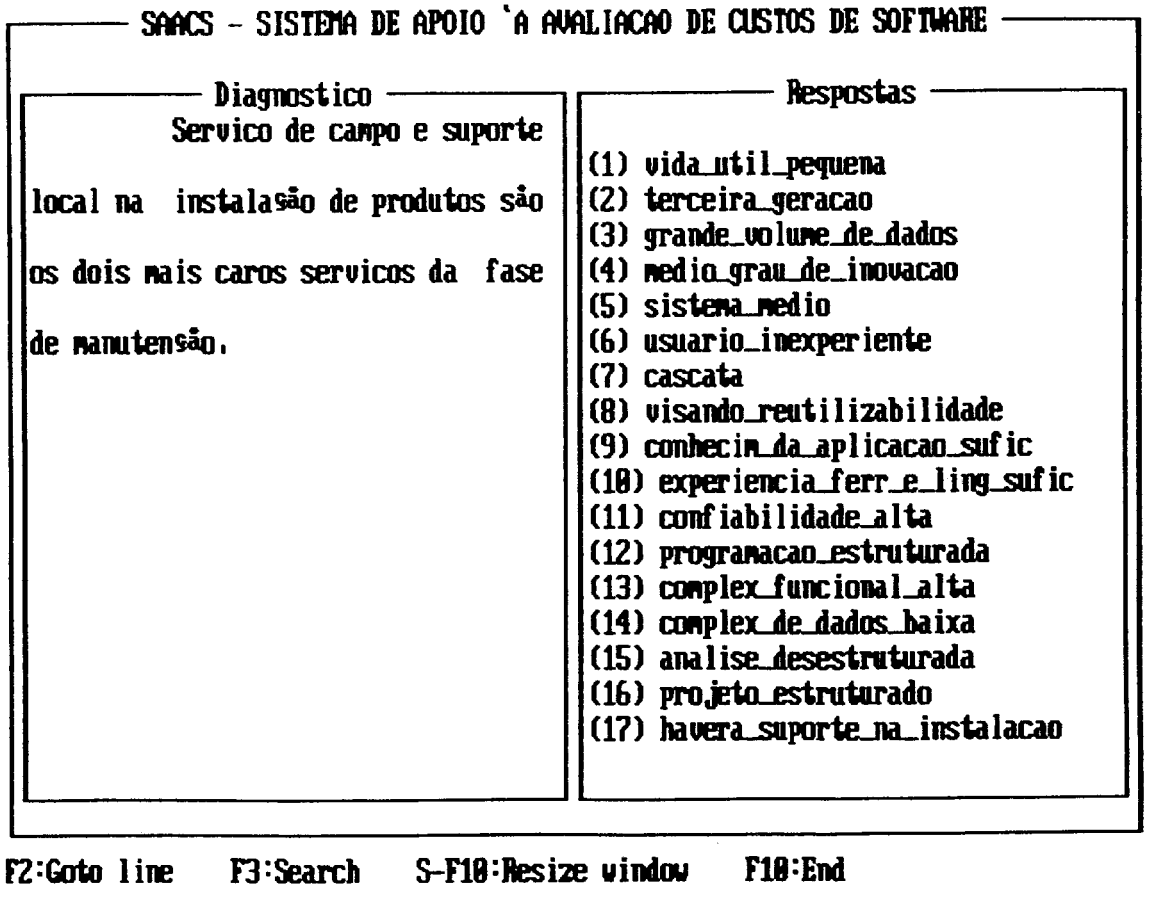

Figura 5.19: Diagnóstico 8.

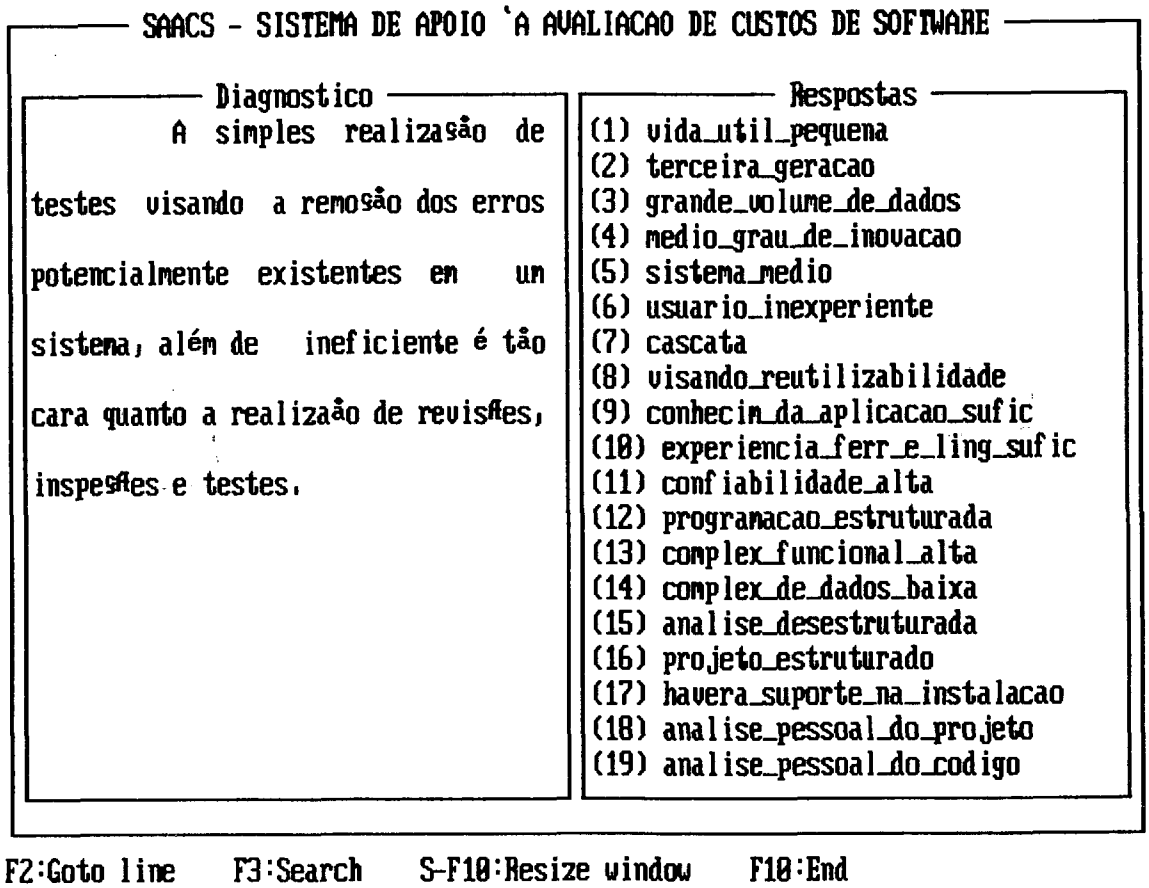

Figura 5.20: Diagnóstico 9.

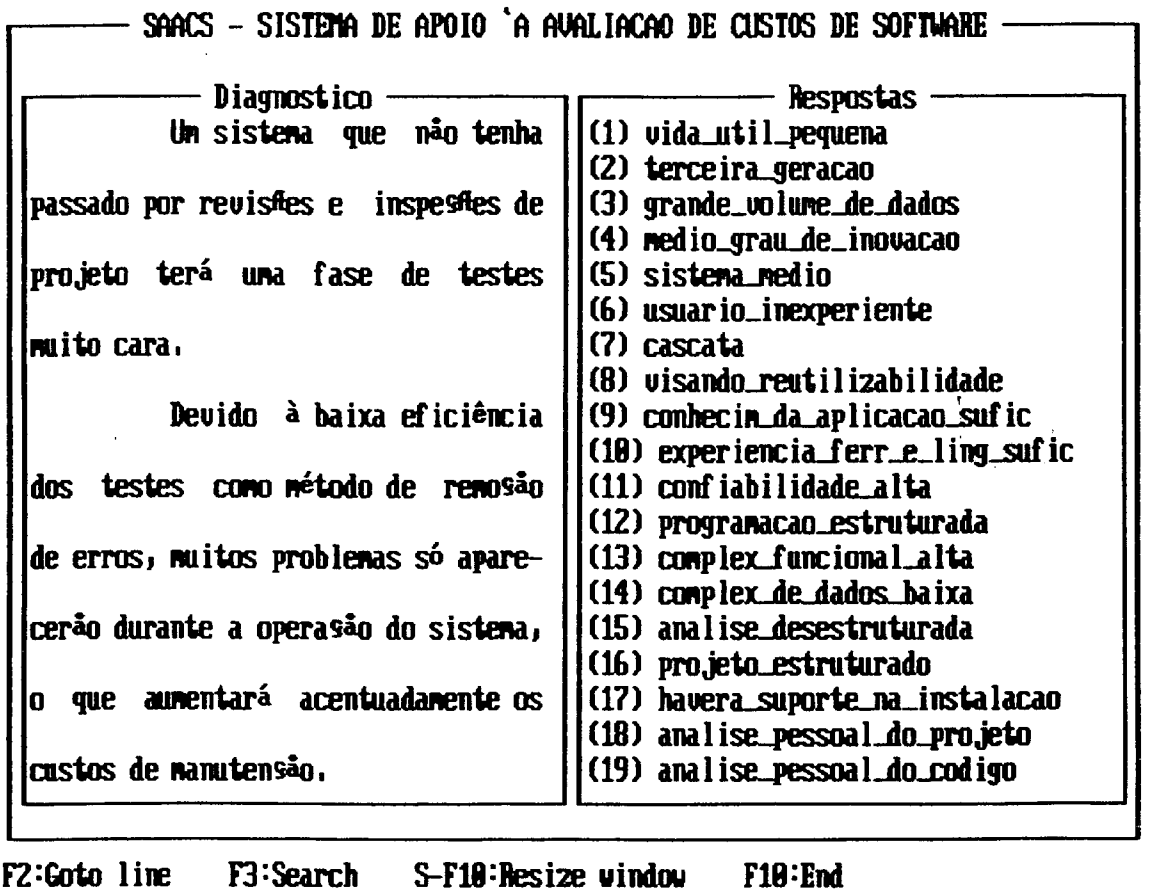

Figura 5.21: Diagnóstico 10.

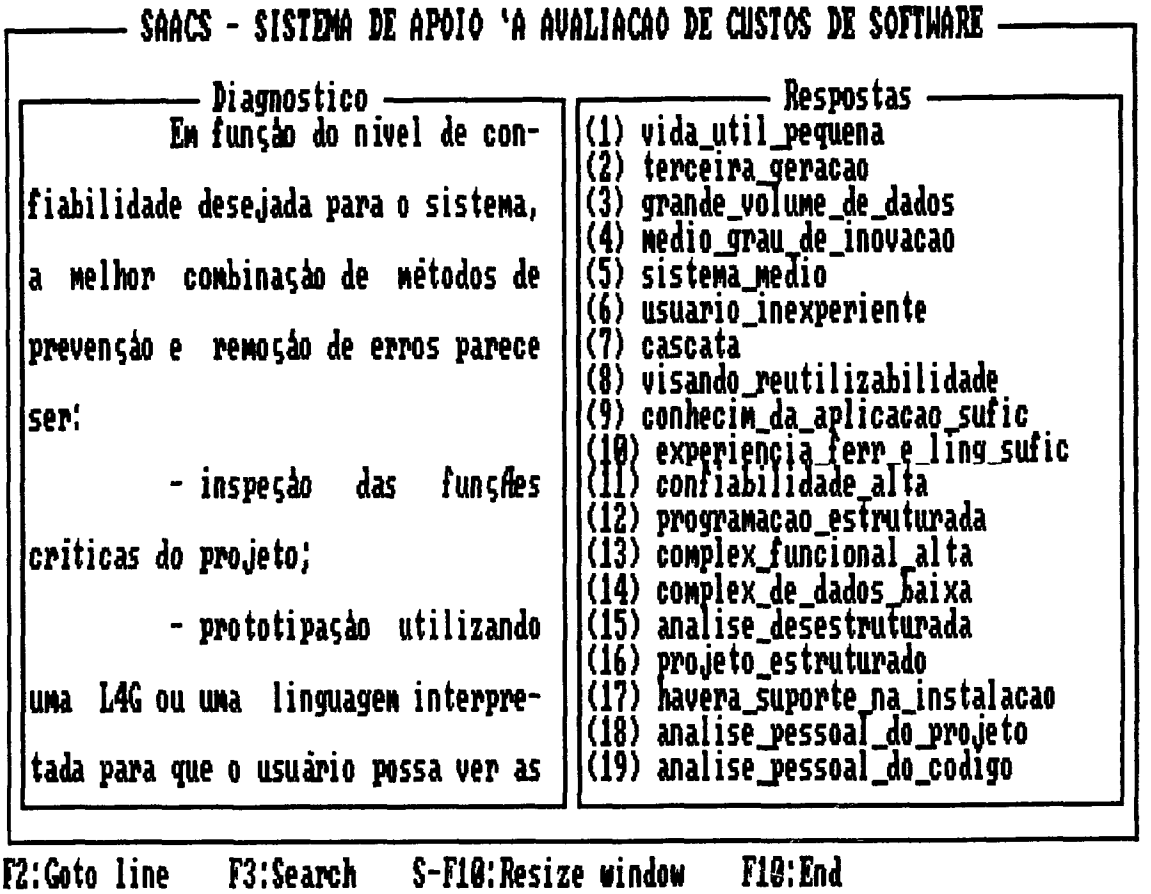

Figura 5.22: Diagnóstico 11.1.

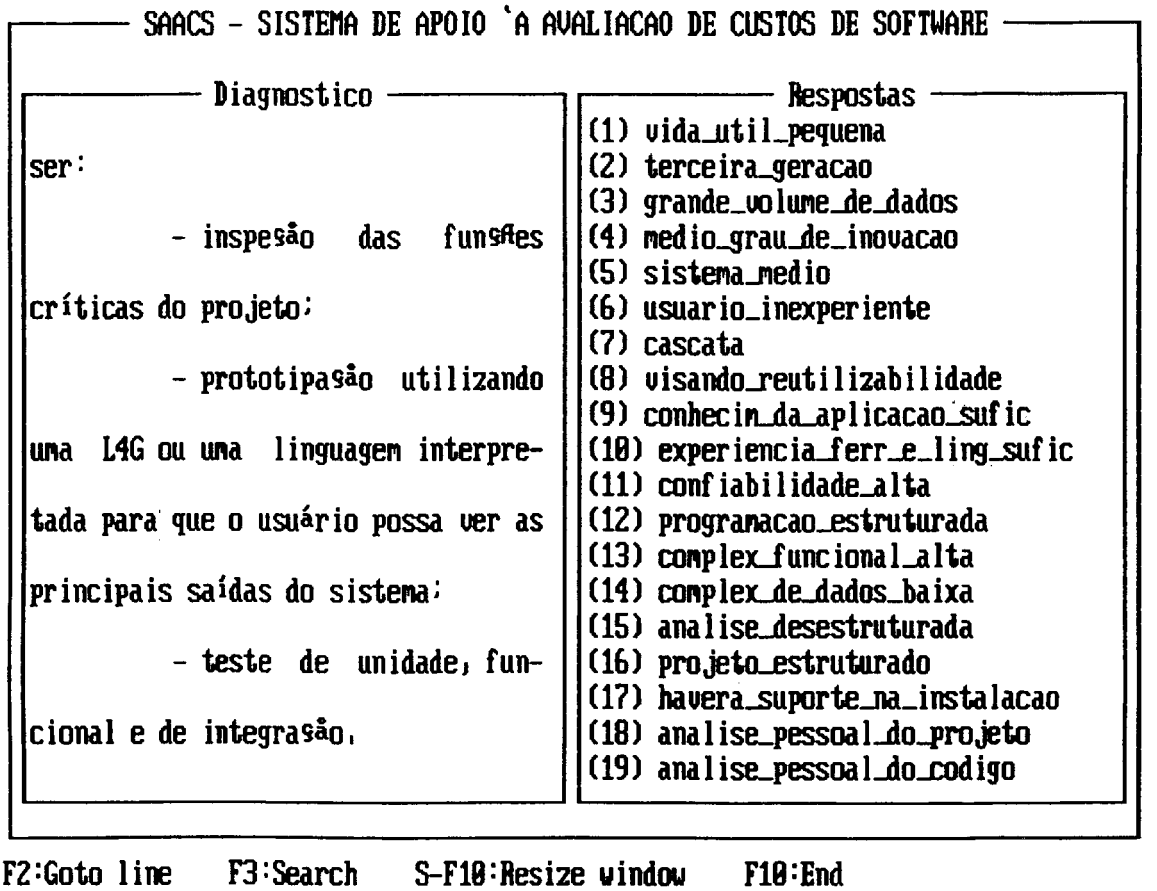

Figura 5.23: Diagnóstico 11.2.

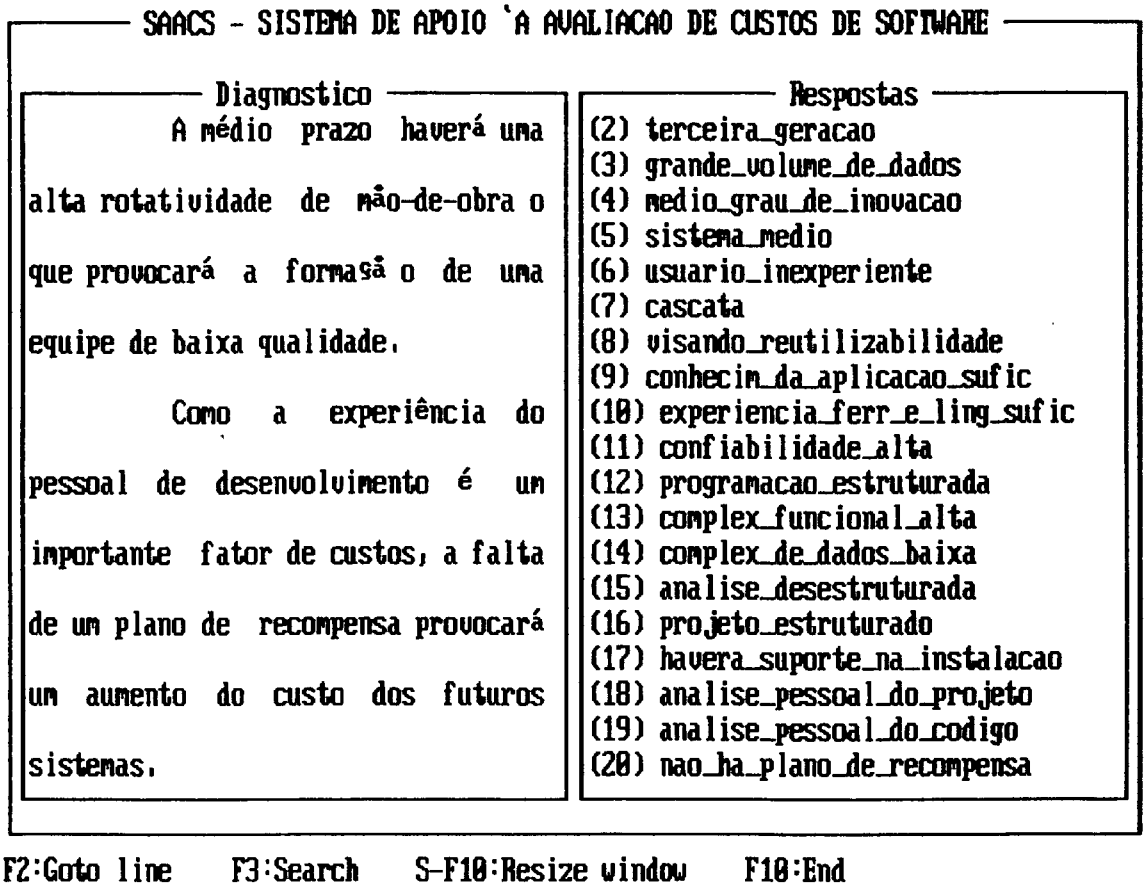

Figura 5.24: Diagnóstico 12.

CAPÍTULO VI

CONCLUSÕES

Este trabalho teve como objetivo o desenvolvimento do protótipo de um sistema de computação capaz de auxiliar profissionais de PD a produzirem melhores estimativas de custos de desenvolvimento de software.

Como o conhecimento em estimativa de custos não se encontra sedimentado o suficiente para se permitir o desenvolvimento de um sistema especialista, julgou-se conveniente desenvolver um sistema que permitisse a participação do usuário (o profissional responsável pela avaliação do custo) no processo de tomada de decisão. Esta participação foi assegurada construindo-se um sistema especialista de suporte a decisão cujo objetivo principal é o de ajudar o usuário a tomar decisões.

Em virtude da complexidade do assunto tratado pelo sistema julgou-se conveniente desenvolver um sistema protótipo a partir de um estudo bibliográfico realizado sobre os fatores de produtividade mais bem documentados na literatura, deixando-se de lado os fatores considerados intangíveis, isto é, aqueles fatores pouco conhecidos. Em uma segunda etapa, a consulta a especialistas será bastante simplificada com o uso do protótipo.

A primeira versão do sistema possui 52 regras, algumas com o objetivo específico de apoiar estimativas de

custos e outras com a intenção de orientar o usuário na redução do custo final do sistema que está sendo avaliado, sem, no entanto, determinar o montante da economia.

VI.1. OS USUÁRIOS

Este sistema se dirige em primeiro lugar a profissionais envolvidos da tarefa de estimar os custos do processo de desenvolvimento de software. No entanto, sua utilização por profissionais menos experientes pode ser bastante interessante, principalmente no tocante à disseminação do conhecimento armazenado em sua base de conhecimentos. Neste caso o sistema possuiria objetivo puramente educacional.

VI.2. A IMPLEMENTAÇÃO

Na implementação deste sistema houve a preocupação de fornecer ao usuário uma interface amigável e esclarecedora. Além do mecanismo "What if", que permite que as respostas que já tenham sido fornecidas pelo usuário possam ser alteradas forçando o sistema a tomar novo caminho, o sistema fornece ao usuário dois outros mecanismos com os seguintes objetivos:

- esclarecer o sentido das perguntas, bem como informar com que regras as perguntas se relacionam, e,

- definir claramente o significado das possíveis respostas.

VI.3. UTILIZAÇÃO DO SISTEMA COM OUTRA BASE DE CONHECIMENTO

Embora este sistema não tenha sido desenvolvido com as características de um shell, uma vez que o engenheiro do conhecimento precisa conhecer prolog para poder retirar as regras do sistema atual e armazenar na base de conhecimentos as regras do novo sistema, foi realizada em julho de 1989 uma experiência neste sentido cujo objetivo foi desenvolvimento do protótipo do especificador de ambientes da estação TABA. [4i]

VI.4. MELHORAMENTOS PROPOSTOS

Para uma próxima versão do sistema são propostos os seguintes melhoramentos:

- o refinamento do conhecimento do sistema, a partir de consultas a especialistas;

- a integração de métodos algorítmicos de estimativa de custos a este sistema de forma que o usuário se sinta estimulado a utilizar ambos, atendendo assim a recomendação proposta no item VI.4;

- o acesso a uma base de dados contendo informações a respeito de projetos já executados, tais como: custo real dos sistemas, tempo de desenvolvimento, número de linhas de código, e número de pontos por função, e,

- a construção de uma interface amigável com o engenheiro do conhecimento capaz dar suporte à aquisição do conhecimento.

IV.5. CONCLUSÃO FINAL

Embora este sistema apóie as estimativas de custos baseadas no julgamento de especialistas, é recomendado, como mencionado no item II.4, que métodos como este sejam utilizados juntamente com um modelo de custos algorítmico para que as diferenças eventualmente apuradas possam ser comparadas e avaliadas. Por outro lado, esta segunda estimativa faz com que se obtenha um maior conhecimento dos fatores de produtividade capazes de afetar os custos de um produto de software possibilitando, inclusive, a atualização do conhecimento do sistema apresentado neste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Boehm B.W., Papaccio P.N., Understanding and Controlling Software Costs, IEEE Transactions on Software Engineering, vol. 14, no. 10, oct. 1988.

[2] Fairley R.E., Software Engineering Concepts; McGraw-hill, 1985.

[3] Luconi F.L., Malone T.W., Morton M.S.S., Expert Systems: The Next Challenge for Managers, Sloan Management Review, 1986.

[4] Boehm B.W., Software Engineering Economics, Prentice-Hall, 1981.

[5] Kemerer, C.F., An Empirical Validation of Software Cost Estimation Models, Communications of the ACM, vol.30,n.5 (maio 1987), pp.416-429.

[6] Putnam, L. e Fitzsimmons, A., Estimating Software Costs, Datamation 25, 10-12 (Set.-Nov. 1979).

[7] Cuelenaere, A.M.E., Genuchten, M.J.I.M. e Heemstra, F.J., Calibrating a Software Cost Estimation Model: Why and How, Information and Software Technology, vol 29, n. 10, 1987.

[8] Albrecht, A.J., Measuring application development, productivity, In Proceedings of the IBM Applications Development Symposium, GUIDE/SHARE (Monterey, Calif., out. 14-17). IBM, 1979, pp.83-92.

[9] Albrecht, A.J., Gaffney, J., Jr. Software function, source lines of code, and development effort prediction: A Software science validation, IEEE Trans. Softw. Eng., SE-9, 6(Nov. 1983), pp.639-648.

[10] Jones, C., Programming Productivity, McGraw-Hill, New York, 1986.

[11] Rubin, H.A., Macroestimation of Software Development Parameters: The Estimacs System, In SOFTFAIR Conference on Software Development Tools, Techniques and Alternatives, (Arlington, Va., Julho 25-28), IEEE Press, New York, 1983, pp.109-118.

[12] Rubin, H.A., The Art and Science of Software Estimation: Fifth Generation Estimators, In Proceedings of the 7th Annual ISPA Conference, (Orlando, Fla., Maio 7-9), International Society of Parametric Analysts, McLean, Va., 1985, pp.56-72.

[13] Rubin, H.A., Using ESTIMACS E. Managment and Computer Services, Valley Forge, Pa., março de 1984.

[14] D.R. Jeffery, The relationship Between Team Size, Experience and Attitudes and Software Development Productivity, IEEE Computer, 1987.

[15] EDP Analyzer, Canning Publications Inc., vol. 24, n.1, 1986.

[16] Flaherty, M.J., Programming Process Productivity Measurement System for System / 370, IBM Systems Journal, vol. 24, N. 2, 1985.

[17] McCabe, T.J., A Complexity Measure, IEEE Trans. Soft. Eng., SE-2, n. 4, 1976.

[18] Carver, D.L., Simmons, D.B., The Impact of Programming Methodology on Program Complexity, The Journal of Systems and Software, N. n, 1985.

[19] Thadhani A.J., Factors Affecting Programmer Productivity During Application Development, IBM

Systems Journal, vol. 23, n. 1, 1984.

[20] EDP Analyzer, Canning Publications Inc., vol. 24, n. 6, 1986.

[21] Dittrich J.E., Couger J.D., Zawacki R.A., Perceptions of Equity, Job Satisfaction and Intention to Quit Among Data Processing Personnel, North-Holland, Information and Management, N. 8, 1985.

[22] Weinberg, G.M. e Freedman, D.P., Reviews, Walkthrough and Inspections, IEEE Trans. Soft. Eng., SE-10, n. 1, 1984.

[23] Fagan M.E., Advances in Software Inspections, IEEE Trans. Soft. Eng., SE-12, n. 7, 1986.

[24] Gilhooley I.A., A Methodology for Productivity Systems Development, Journal of Information Systems Management, Winter, 1986.

[25] Matsumura K., Furuya K., Yamashiro A., Obi T., Trend Toward Reusable Module Component: Design and Coding Technique 50SM, IEEE Computer, 1987.

[26] The Handbook of Artificial Intelligence, Vol. 1, Ed. Avron Barr e Edward A. Feigenbaum, 1981.

[27] Hayes-Roth F., Knowledge Based Expert Systems, Computer, oct. 1984.

[28] Leadbetter P.A., A Current Review of the Expert Systems Phenomenon, Interfaces in Computing 3, 67-81, 1985.

[29] Waterman D.A., A Guide to Expert Systems, Addison-Wesley, 1986.

[30] Buchanan B.G., Shortliffe E.H., Rule-Based Expert Systems - The Mycin Experiments of the Stanford

Heuristic Programming Project, Addison-Wesley, 1984.

[31] Hart A., Knowledge Acquisition for Expert Systems, McGraw-hill, 1986.

[32] Hayes-Roth F., Waterman D.A., Lenat D.B., Building Expert Systems, Addison-Wesley, 1983.

[33] Keller, R., Expert System Technology = Development and Application, Prentice Hall, 1987, New Jersey.

[34] Blanning R.W., Expert Systems for Management: Research and Applications, Journal of Information Science, 9, North-Holland, 1985.

[35] Simon H.A., Whether Software Engineering Needs to be Artificial Intelligent, IEEE Transactions on Software Engineering, vol. SE-12, no. 7, julho de 1986.

[36] Baldwin D., Kasper G.M., Toward Representing Management-domain Knowledge, Decision Suport Systems 2, 159-172, North-Holland, 1986.

[37] Clocksin W.F., Mellish C.S., Programming in Prolog, Springer-Verlag, 1984.

[38] Nilsson N.J., Principles of Artificial Intelligence, Springer-Verlag, 1982.

[39] Hogger C.J., Introduction to Logic Programming, Academic Press Inc., 1984.

[40] Boehm, B.W., Gray, T.E. e Seewaldt, T.; "Prototyping Versus Specifying: A Multiproject Experiment"; IEEE Transactions on Software Engineering, vol. SE-10, n. 3, maio de 1984.

[41] Aguiar, T.C.;"Protótipo do Especificador de Ambientes da Estação TABA";ES-208/89;COPPE/UFRJ

ANEXO A

MANUAL DO USUARIO

1. Introdução.

Este manual tem como objetivo definir os requisitos de hardware necessários à execução do SAACS (Um Sistema de Apoio ao Avaliador de custos de Software), descrever seu modo de operação e relacionar os arquivos que o compõem, mencionando seus usos.

2. Objetivo.

O objetivo do sistema é apoiar aqueles profissionais que utilizam unicamente sua experiência na avaliação de custos de desenvolvimento de software.

3. Requisitos de Hardware.

O SAACS foi desenvolvido para microcomputadores da linha IBM-PC. Sua única restrição de hardware é a necessidade da máquina hospedeira possuir, pelo menos, 512K de memória RAM e um acionador de discos flexíveis.

4. Usuários.

Este sistema se dirige em primeiro lugar a profissionais envolvidos na tarefa de estimar os custos de desenvolvimento de software. No entanto, sua utilização por profissionais menos experientes pode ser bastante

interessante, principalmente no tocante à disseminação do conhecimento armazenado em sua base de conhecimentos. Neste caso o sistema possuiria objetivo puramente educacional.

5. Arquivos do Sistema.

Este sistema é composto por quatro tipos de arquivos. São eles:

- os arquivos SAACS.PRO e C2.PRO contém código executável do programa;

- os arquivos REGRAxx.DOC contém os textos que são exibidos na janela de diagnósticos a medida que as regras são satisfeitas. O conteúdo destes arquivos encontra-se definido no capítulo V deste trabalho;

- os arquivos MTVxx.DOC contém as justificativas para as diversas perguntas efetudas pelo sistema ao usuário, e,

- os arquivos TERxxxx.DOC contém as definições dos termos fornecidos pelo sistema como respostas às perguntas formuladas.

A seguir encontram-se as perguntas formuladas pelo sistema acompanhadas do nome dos arquivos onde foram definidos os motivos que levam o sistema a efetuar tais perguntas:

1) A que classe de aplicação o sistema pertence ?

Arquivo: MTV01.DOC

2) Que ciclo de vida pretende utilizar ?

Arquivo: MTV02.DOC

3) Qual o nivel de interação do sistema com o usuario final ?

Arquivo: MTV03.DOC

4) Que tipo de linguagem será utilizada na implementação do sistema ?

Arquivo: MTV04.DOC

5) Qual o nivel de experiência da equipe de desenvolvimento no uso das ferramentas e linguagens a serem utilizadas ?

Arquivo: MTV05.DOC

6) Qual o nível de experiência da equipe de desenvolvimento com a aplicação ?

Arquivo: MTV06.DOC

7) Será utilizado algum método estruturado para a especificação dos requisitos do sistema ?

Arquivo: MTV07.DOC

8) Será utilizado algum método estruturado para a elaboração do projeto dos programas ?

Arquivo: MTV08.DOC

9) Será utilizada programação estruturada na elaboração dos programas ?

Arquivo: MTV09.DOC

10) Qual o tamanho do sistema a ser desenvolvido ?

Arquivo: MTV10.DOC

11) Qual o grau de complexidade funcional do sistema ?

Arquivo: MTV11.DOC

12) Qual o grau de complexidade de dados do sistema ?

Arquivo: MTV12.DOC

13) Qual será a vida útil do sistema a ser desenvolvido ?

Arquivo: MTV13.DOC

14) Qual o grau de confiabilidade requerido para o sistema ?

Arquivo: MTV14.DOC

15) Qual o grau de inovação do sistema em desenvolvimento ?

Arquivo: MTV15.DOC

16) Há alguma restrição ao uso do computador ?

Arquivo: MTV16.DOC

17) Qual o volume de dados tratado pelo sistema ?

Arquivo: MTV17.DOC

18) De que forma o projeto será gerênciado ?

Arquivo: MTV18.DOC

19) Qual o nível das ferramentas de documentação existentes ?

Arquivo: MTV19.DOC

20) O sistema será documentado via HELP ?

Arquivo: MTV20.DOC

21) Qual o volume de documentação a ser gerado ?

Arquivo: MTV21.DOC

22) Qual o tamanho da equipe de desenvolvimento ?

Arquivo: MTV22.DOC

23) Você acredita que os requisitos do sistema serão estáveis ?

Arquivo: MTV23.DOC

24) Os programas serão desenvolvidos cuidadosamente ?

Arquivo: MTV24.DOC

25) Qual o nível de experiência do usuário final com a aplicação ?

Arquivo: MTV25.DOC

26) Haverá suporte local na instalação dos produtos ?

Arquivo: MTV26.DOC

27) Haverá serviço de campo ?

Arquivo: MTV27.DOC

28) Haverá reutilização de código ou de projetos padrão ?

Arquivo: MTV28.DOC

29) Dentre os métodos de remoção / prevenção de erros abaixo relacionados quais você pretende utilizar durante as fases de especificação de requisitos e projeto ?

1. Analise pessoal do projeto e do cumentos.
2. Revisão (em grupo) informal do projeto.

3. Inspeção do projeto.

Arquivo: MTV29.DOC

30) Dentre os métodos de remoção / prevenção de erros abaixo relacionados, quais você pretende utilizar durante a fase de codificação ?

1. Analise pessoal do codigo.
2. Inspeção do código.

Arquivo: MTV30.DOC

31) Dentre os métodos de remoção de erros abaixo relacionados, quais você pretende utilizar durante a fase de testes ?

1. Teste individual e funcional dos módulos.
2. Teste de integração do sistema.
3. Teste de campo com dados reais.

Arquivo: MTV31.DOC

32) Existe na organização algum plano de recompensa ?

Arquivo: MTV32.DOC

33) O sistema a ser desenvolvido possuirá proteção contra cópias não autorizadas ?

Arquivo: MTV33.DOC

Para cada pergunta efetuada, o sistema apresenta ao usuário as possíveis respostas. Os arquivos listados a seguir contém as definições dos termos utilizados pelo sistema na apresentação destas respostas.

ARQUIVO	TERMO DEFINIDO
TER0100.DOC	- Sistema para uso de uma única empresa.
TER0101.DOC	- Sistema para uso em comunidade.

ARQUIVO		TERMO DEFINIDO
TER0102.DOC	-	Sistema visando reutilizabilidade futura.
TER0103.DOC	-	Sistema cujo serviço será alugado a terceiros.
TER0104.DOC	-	Sistema alugado a clientes (Leasing).
TER0105.DOC	-	Sistema para ser vendido no mercado.
TER0106.DOC	-	Sistema desenvolvido sob contrato privado.
TER0107.DOC	-	Sistema desenvolvido sob contrato governamental.
TER0108.DOC	-	Sistema desenvolvido sob contrato militar.
TER0200.DOC	-	Ciclo de vida - Cascata.
TER0202.DOC	-	Ciclo de vida - Prototipagem descartável.
TER0204.DOC	-	Ciclo de vida - Prototipagem evolutiva.
TER0300.DOC	-	Grau de interação do sistema com o usuário - Não significativo.
TER0302.DOC	-	Grau de interação do sistema com o usuário - Significativo.
TER0400.DOC	-	Linguagem de quarta geração
TER0500.DOC	-	Experiência profissional com ferramentas e linguagens - Insuficiente.
TER0502.DOC	-	Experiência profissional com ferramentas e linguagens - Suficiente.
TER0600.DOC	-	Nível de experiência da equipe de desenvolvimento com a plicação - Insuficiente.
TER0602.DOC	-	Nível de experiência da equipe de desenvolvimento com a plicação - Suficiente.
TER0700.DOC	-	Utilização de método estruturado na fase de análise - Sim.
TER0702.DOC	-	Utilização de método estruturado na fase de análise - Não.
TER0800.DOC	-	Utilização de método estruturado na fase de projeto - Sim.
TER0802.DOC	-	Utilização de método estruturado na fase de projeto - Não.

ARQUIVO	TERMO DEFINIDO
TER0900.DOC	- Utilização de método estruturado na fase de programação - Sim.
TER0902.DOC	- Utilização de método estruturado na fase de programação - Não.
TER1000.DOC	- Tamanho do sistema - Pequeno.
TER1002.DOC	- Tamanho do Sistema - Médio.
TER1004.DOC	- Tamanho do Sistema - Grande.
TER1100.DOC	- Complexidade funcional - Baixa.
TER1102.DOC	- Complexidade funcional - Alta.
TER1200.DOC	- Complexidade de dados - Baixa.
TER1202.DOC	- Complexidade de dados - Alta.
TER1300.DOC	- Vida útil - Pequena.
TER1302.DOC	- Vida útil - Longa.
TER1400.DOC	- Confiabilidade - Moderada.
TER1402.DOC	- Confiabilidade - Alta.
TER1404.DOC	- Confiabilidade - Muito alta.
TER1500.DOC	- Grau de inovação - Pequeno.
TER1502.DOC	- Grau de inovação - Médio.
TER1504.DOC	- Grau de inovação - Alto.
TER1600.DOC	- Restrição no uso do computador - Sim.
TER1602.DOC	- Restrição no uso do computador - Não.
TER1700.DOC	- Volume de dados - Pequeno.
TER1702.DOC	- Volume de dados - Grande.
TER1800.DOC	- Gerência - Hierárquica.
TER1802.DOC	- Gerência - Matricial.
TER1900.DOC	- Ferramentas para documentação - Inadequandas.
TER1902.DOC	- Ferramentas para documentação - Adequadas.
TER2000.DOC	- documentação via HELP - Sim.
TER2002.DOC	- Documentação via HELP - Não.

ARQUIVO	TERMO DEFINIDO
TER2100.DOC	- Volume de documentação - Pequeno.
TER2102.DOC	- Volume de documentação - Grande.
TER2200.DOC	- Tamanho da equipe - Pequena.
TER2202.DOC	- Tamanho da equipe - Média.
TER2204.DOC	- Tamanho da equipe - Grande.
TER2300.DOC	- Estabilidade dos requisitos - Sim.
TER2302.DOC	- Estabilidade dos requisitos - Não.
TER2400.DOC	- Tempo para programar cuidadosamente - Sim
TER2402.DOC	- Tempo para programar cuidadosamente - Não.
TER2500.DOC	- Experiência do usuário - Insuficiente.
TER2502.DOC	- Experiência do usuário - Suficiente.
TER2600.DOC	- Suporte local - Sim
TER2602.DOC	- Suporte local - Não
TER2700.DOC	- Serviço de campo - Sim
TER2702.DOC	- Serviço de campo - Não
TER2800.DOC	- Reutilização de código - Não
TER2802.DOC	- Reutilização de código - Sim
TER3200.DOC	- Plano de recompensa - Sim
TER3202.DOC	- Plano de Recompensa - Não
TER3300.DOC	- Proteção contra cópias - Sim
TER3302.DOC	- Proteção contra cópias - Não

6. Operação do Sistema.

Para ativar o SAACS coloque o disquete que o contém no acionador de discos flexíveis A: e digite:

SAACS <ENTER>

Deste momento em diante o sistema se comportará conforme a descrição a seguir:

Sistema: apresentará uma tela contendo as seguintes funções: (Figura A.1)

- tutorial;
- comandos dos;
- carrega base de dados;
- imprime diagnósticos, e,
- SAACS

Usuário: deverá mover o cursor para a linha da função desejada, que ficará em vídeo reverso, e teclar <enter>. Caso o usuário deseje cancelar a execução do programa, deverá teclar <esc>.

A seguir, este manual descreve o comportamento do sistema para cada uma das cinco funções acima mencionadas.

Função Tutorial do Sistema.

Sistema: apresentará ao usuário uma janela (figura A.2) contendo informações a respeito do SAACS.

Usuário: poderá paginar o texto apresentado. Para retornar à tela da figura A.1 deverá teclar <F10>.

Função Comandos DOS.

Sistema: passará o controle para o sistema operacional. (Figura A.3).

Usuário: poderá emitir comandos DOS. Para retornar ao sistema (tela da figura A.1) deverá digitar "exit".

Função Carrega Banco de Dados.

Sistema: solicitará do usuário o nome do arquivo DOS onde se encontra a sessão que se deseja recuperar (figura A.4). Uma vez recuperado o arquivo (que contém as respostas fornecidas pelo usuário em sessão anterior) o usuário poderá rever os diagnósticos daquela sessão, se desejar.

Usuário: deverá fornecer o nome do arquivo que se deseja recuperar.

Sistema: caso o arquivo solicitado pelo usuário não exista, será emitida a mensagem "Arquivo mencionado não encontrado - Digite <ENTER>". No caso do arquivo ser encontrado, o sistema carregará na memória do computador as respostas constantes do arquivo recuperado e emitirá a mensagem "Banco de dados carregado - Aperte<ENTER>". (Figura A.5)

Usuário: em ambos os casos o usuário deverá teclar <ENTER> e a tela da figura A.1 reaparecerá. Caso o arquivo solicitado tenha sido carregado na memória, o usuário poderá, a seguir, ativar a função "Imprime Diagnósticos" ou a função SAACS.

Função Imprime Diagnósticos.

Sistema: caso as respostas às perguntas formuladas pelo sistema estejam na memória do computador, o sistema imprimirá os diagnósticos correspondentes. Caso contrário, o sistema emitirá a mensagem "Carregue base de dados - Digite <ENTER>".

Usuário: deverá informar o nome de um arquivo DOS, se solicitado.

Função SAACS

Sistema: caso o usuário tenha carregado na memória do computador as respostas fornecidas em uma sessão anterior, tais respostas serão listadas na janela de respostas (figura A.6) e será formulada a pergunta "Deseja omitir diagnósticos? (S/N)".

Usuário: se o usuário desejar examinar os diagnósticos fornecidos para a sessão, deverá responder "SIM", caso contrário, deverá responder "NÃO".

Sistema: caso o usuário não tenha carregado previamente uma sessão na memória do computador, o sistema apresentará ao usuário, na janela reservada a perguntas a primeira questão (figura A.7) a respeito do produto de software que será tratado.

Usuário: após ler a pergunta, deverá teclar <ENTER>.

Sistema: apresentará ao usuário, em uma janela, as

possíveis respostas à pergunta formulada. (figura A.8)

Usuário: deverá posicionar o cursor na linha onde se encontra a resposta mais adequada e teclar <ENTER>. Caso o usuário não tenha entendido a pergunta e deseje alguma explicação (figuras A.8 e A.9) deverá apertar a letra "p". Por outro lado, se a dúvida do usuário estiver no significado de alguma resposta, deverá posicionar o cursor na linha correspondente a esta resposta e teclar "?". (Figuras A.8 e A10)

Sistema: se o usuário não tiver qualquer dúvida e responder à pergunta, o sistema porá a resposta fornecida na janela reservada às respostas (figura A.11) e e nova pergunta será formulada. (figura A.12)

Se o usuário apertar a tecla "p" o sistema fará com que apareça uma tela (figura A.9) contendo uma justificativa para a pergunta formulada e uma lista de todas as regras relacionadas à pergunta.

E se o usuário apertar a tecla "?" uma janela aparecerá com os esclarecimentos necessários. (figura A.10)

Os últimos cinco passos se repetem para todas as perguntas formuladas pelo sistema.

No decorrer da sessão algumas regras serão satisfeitas o que provocará a emissão de diagnósticos, conforme vem a seguir:

Sistema: os diagnósticos são mostrados ao usuário do sistema em uma janela que ficará superposta à janela reservada às perguntas. (Figura A.13 e A.14)

Usuário: poderá paginar a janela que contém o diagnóstico. Para prosseguir com a execução do sistema deverá apertar <F10> e nova pergunta será formulada.

Caso o usuário deseje modificar a resposta a uma pergunta formulada anteriormente, poderá fazê-lo. Este mecanismo é conhecido na literatura como "what if" e tem por objetivo permitir que o usuário navegue através das regras do sistema, tomando conhecimento, rapidamente, do impacto no custo causado pelas diversas respostas. Para isto, deverá teclar a letra "w" (que não aparece no vídeo) e em seguida o número da resposta que deseja modificar. O número mencionado deverá ser extraído da janela de respostas. (Figuras A.14 e A.15)

Após efetuar sua última pergunta o sistema se comportará como segue:

Sistema: perguntará ao usuário se a sessão deve ser salva. (Figura A.16)

Usuário: deverá responder "s" ou "n".

Sistema: caso o usuário tenha respondido "s", o sistema solicitará o nome de um arquivo DOS onde as respostas fornecidas deverão ser salvas. (Figuras A.17 e A.18)

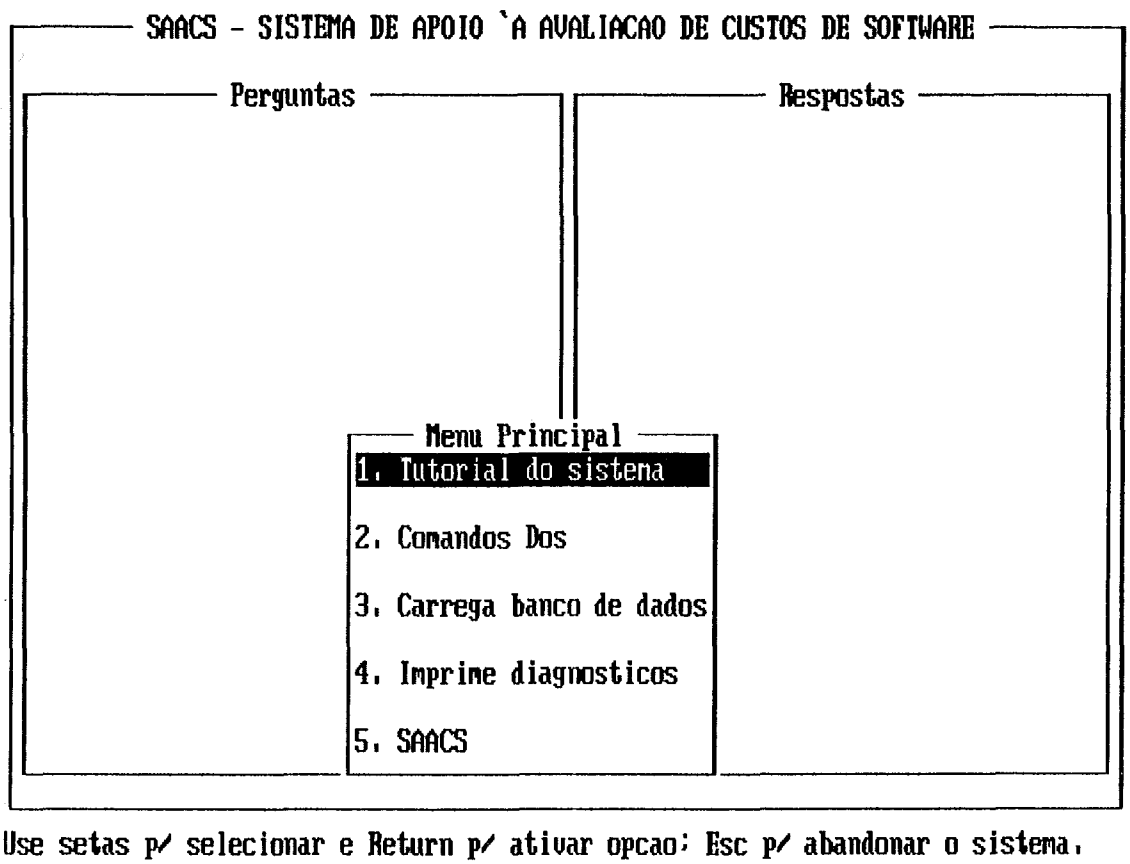

Figura A.1: Tela de apresentação.

SAACS - SISTEMA DE APOIO `A AVALIACAO DE CUSTOS DE SOFTWARE

T U T O R I A L
****

Este sistema tem como objetivo apoiar a estimativa de custos de desenvolvimento de software. O usuario devera fornecer ao sistema, entre outras, as seguintes informacoes:

- dados a respeito do produto a ser desenvolvido;
- as caracteristicas do pessoal envolvido no trabalho;
- os metodos a serem utilizados no processo de desenvolvimento.

E o sistema, baseado nestas informacoes, procurara informar ao usuario sobre o impacto no custo, dos diversos fatores de produtividade identificados.

A atual versao contem 50 regras a respeito dos mais importantes fatores de produtividade comentados na literatura.

O usuario devera' responder as perguntas formuladas selecionando uma das opcoes que aparecem no "menu". O usuario pode pedir um esclarecimento so-

F2:Goto line F3:Search S-F10:Resize window F10:End

Figura A.2: Tutorial.

```
Type EXIT to return from DOS

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

MONYDOS - Sistema Operacional Compativel com MSDOS Ver 2.1

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

A>
```

Figura A.3: Sistema Operacional.

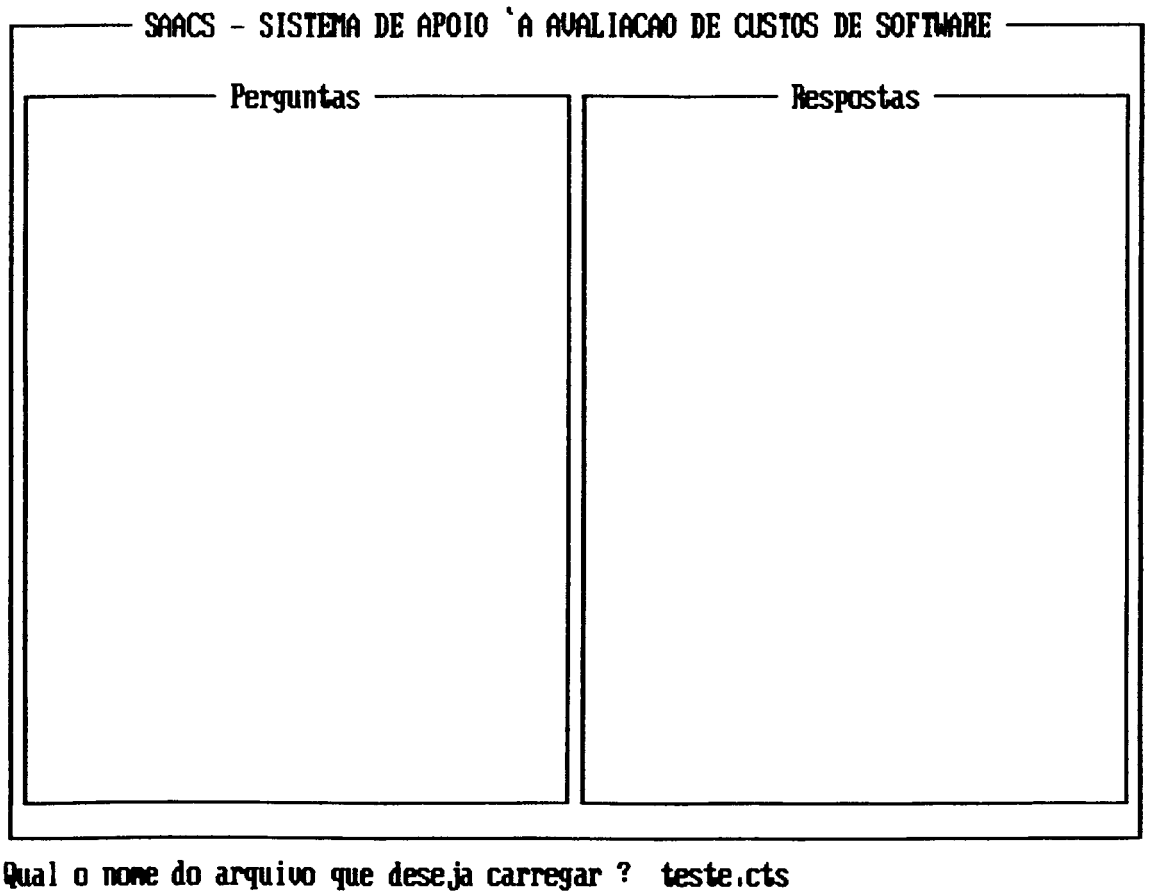

Figura A.4: Carrega banco de dados.

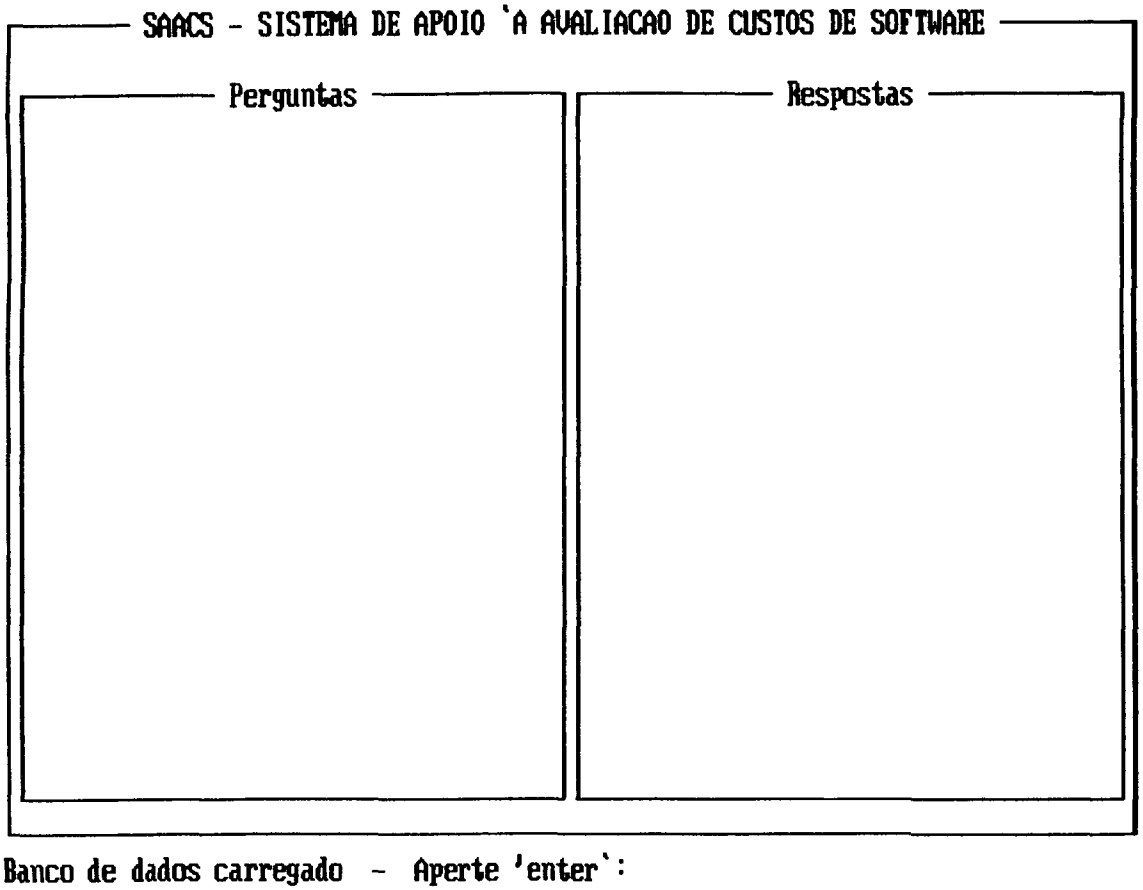

Figura A.5: Banco de dados carregado.

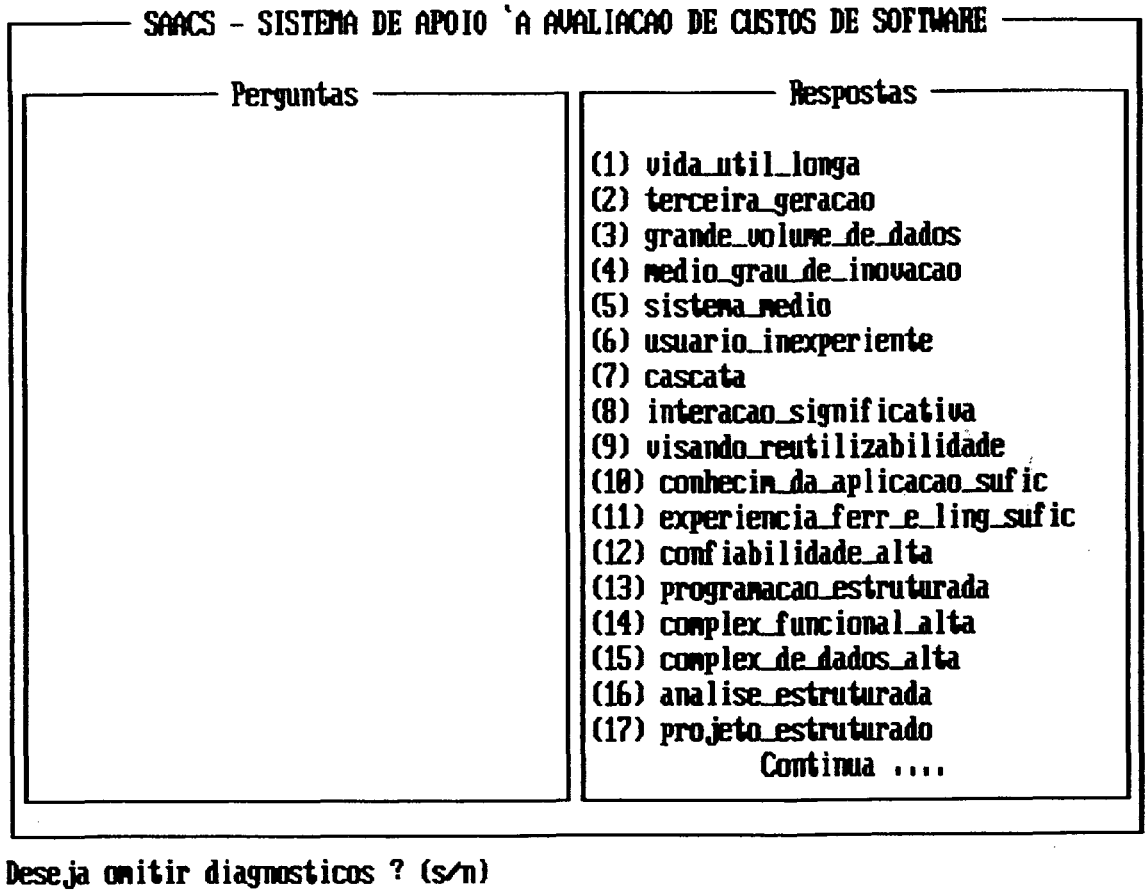

Figura A.6: Lista de respostas.

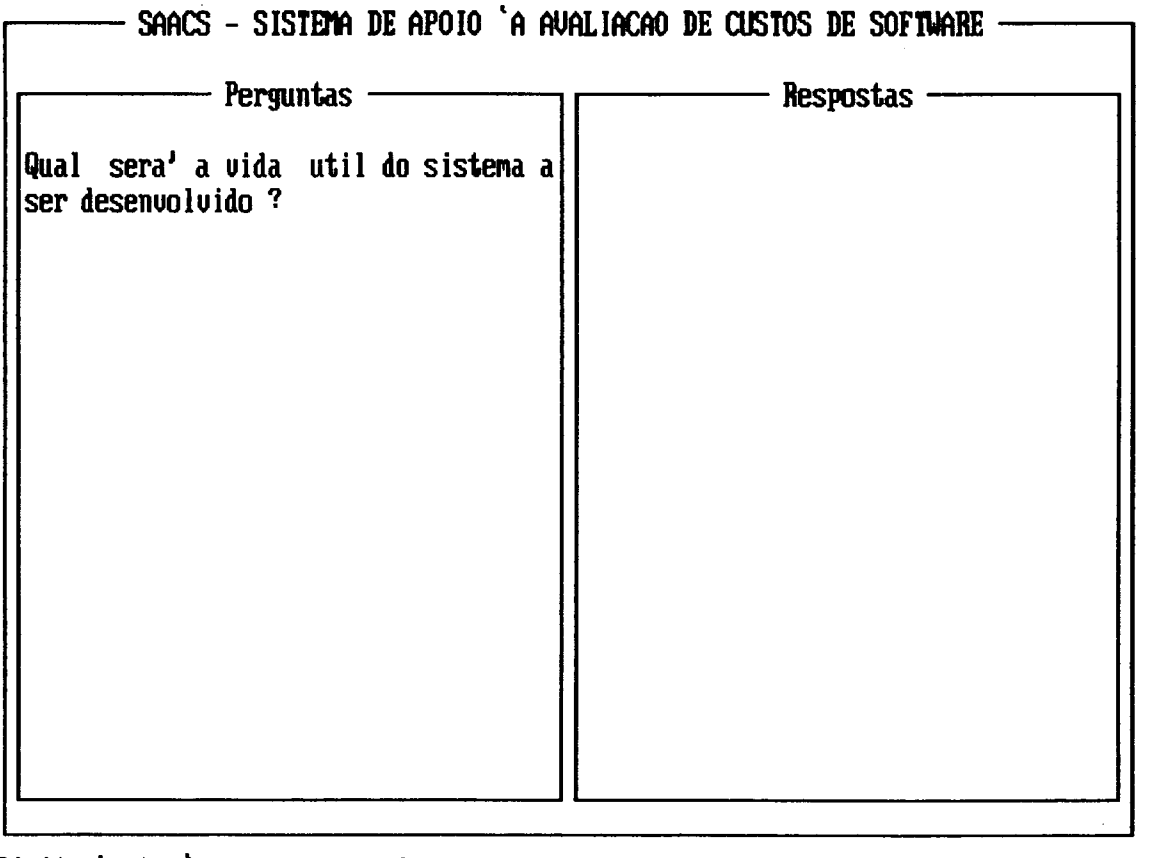

Figura A.7: Perguntas do sistema.

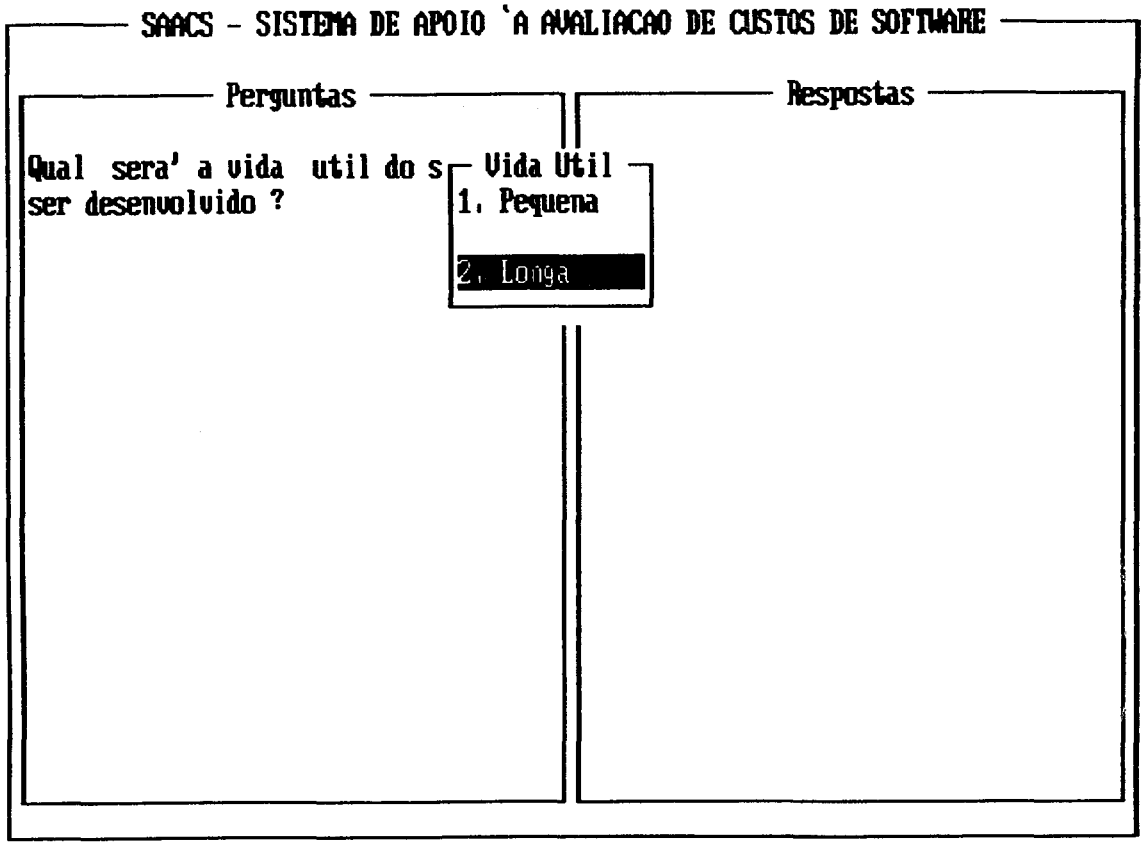

Figura A.8: Possíveis respostas.

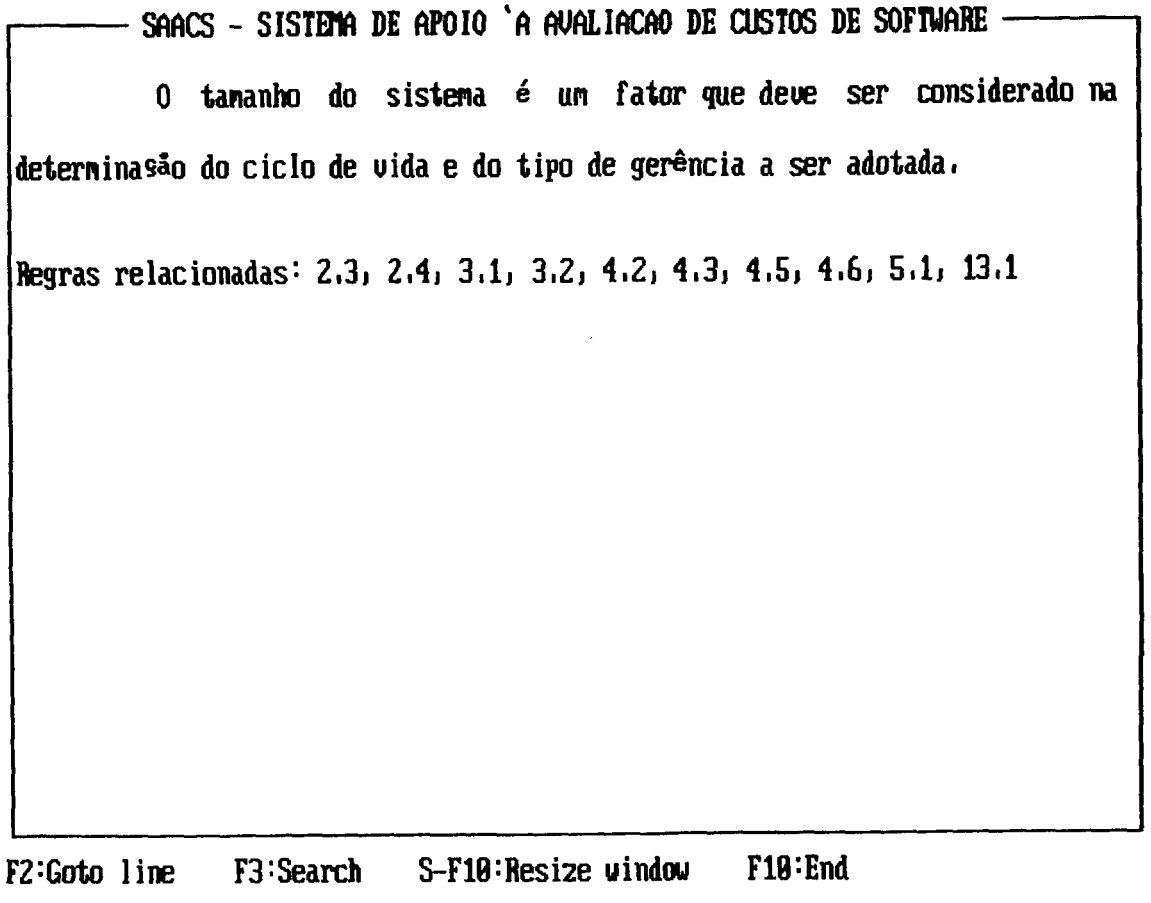

Figura A.9: Justificativa da pergunta formulada.

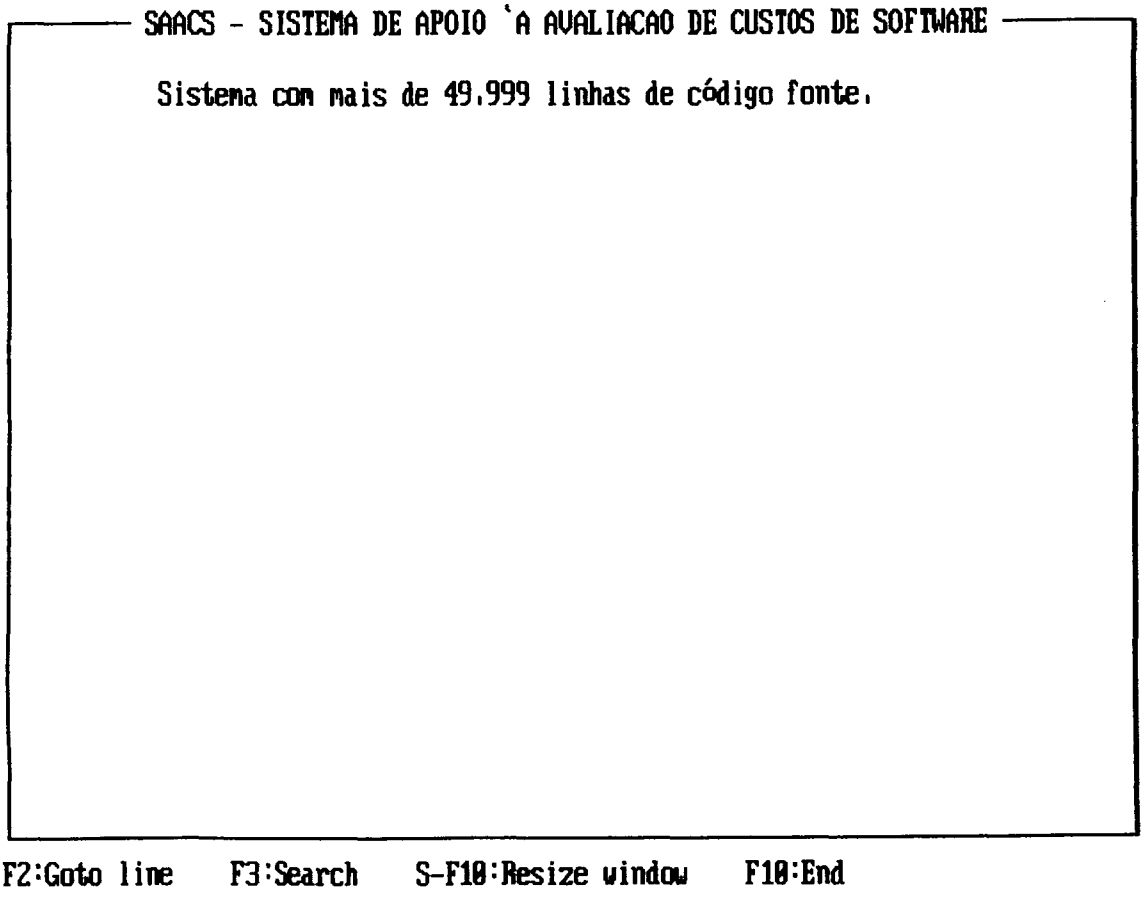

Figura A.10: Esclarecimento do que significa "um sistema de médio porte.

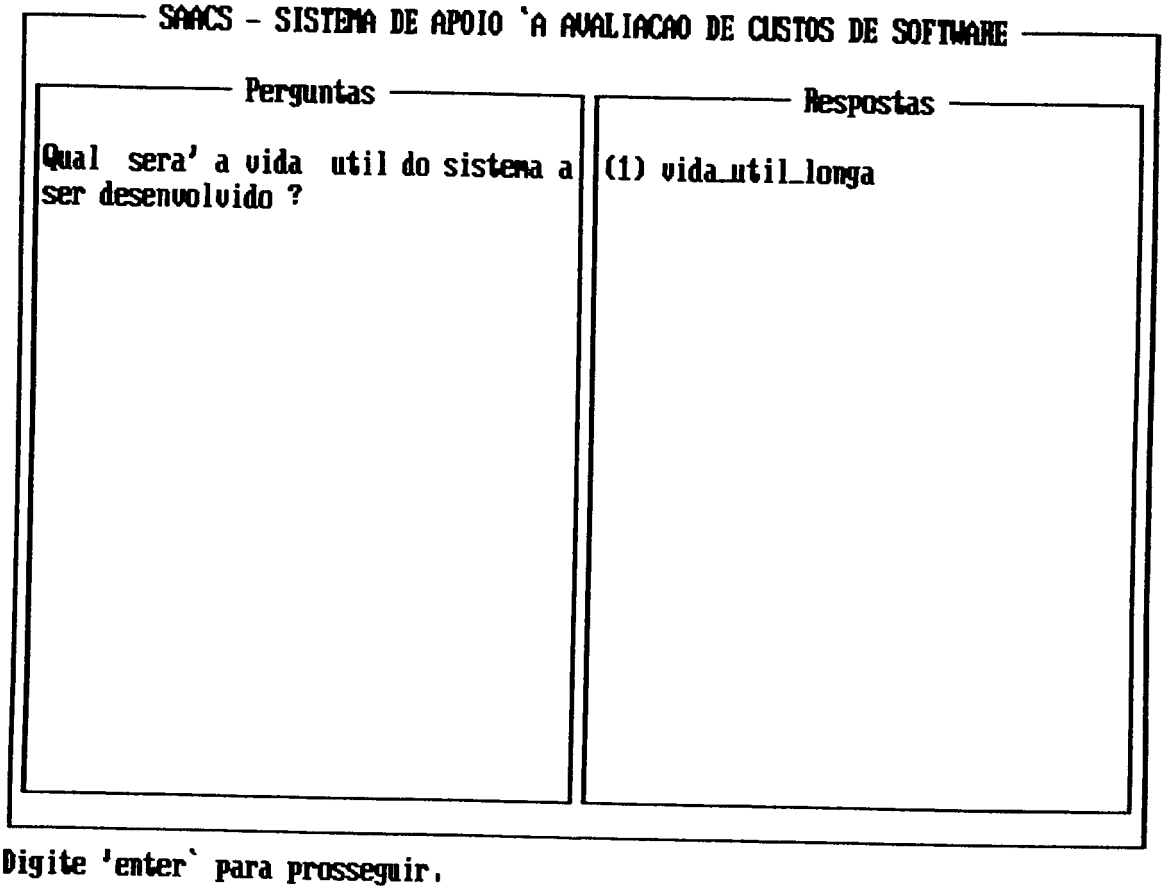

Figura A.11: Tela de respostas.

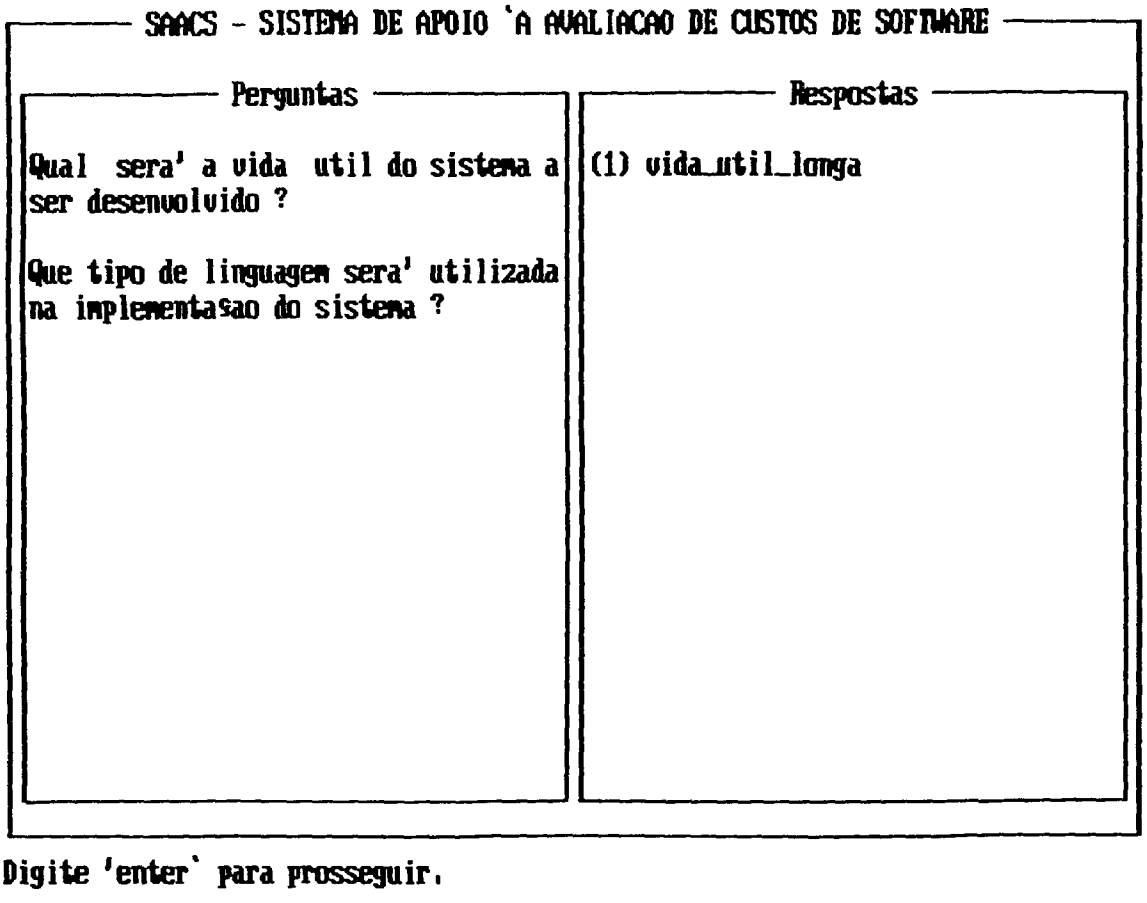

Figura A.12: Tela de perguntas.

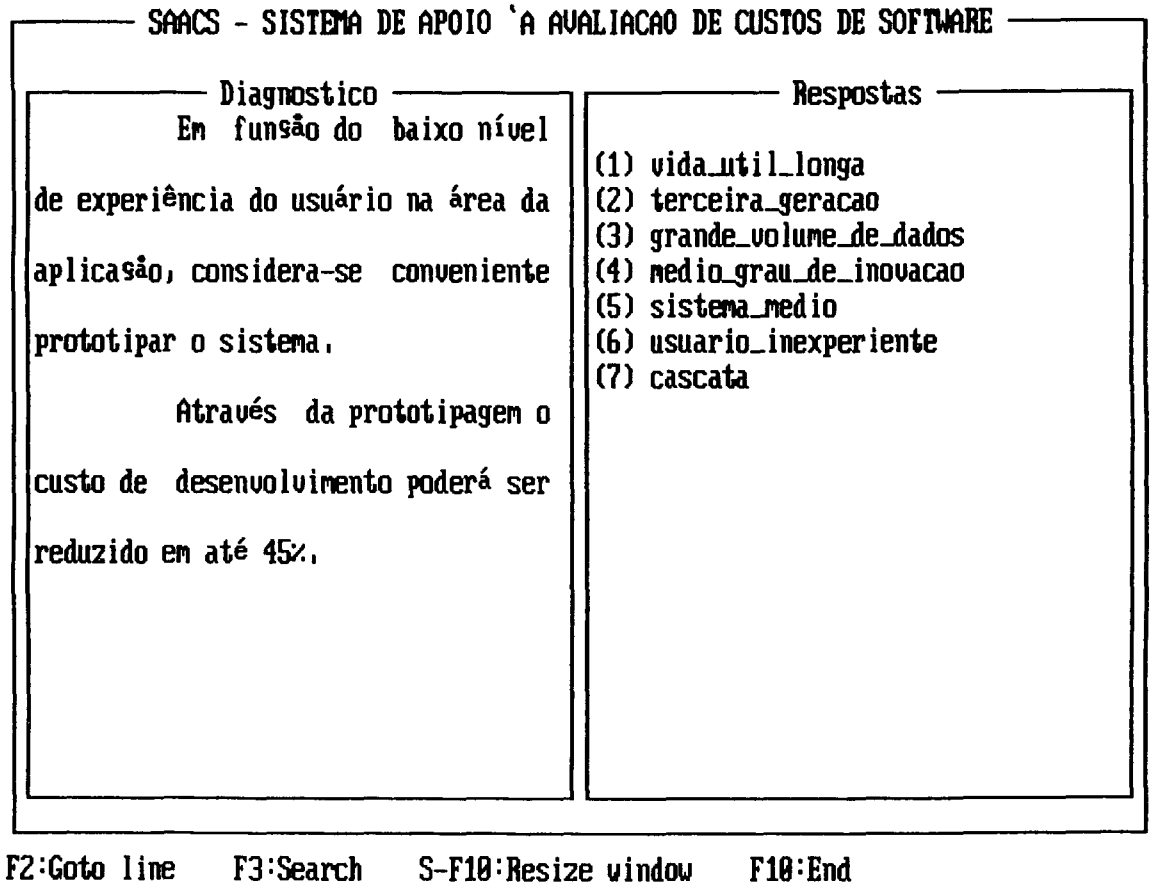

Figura A.13: Tela de diagnósticos.

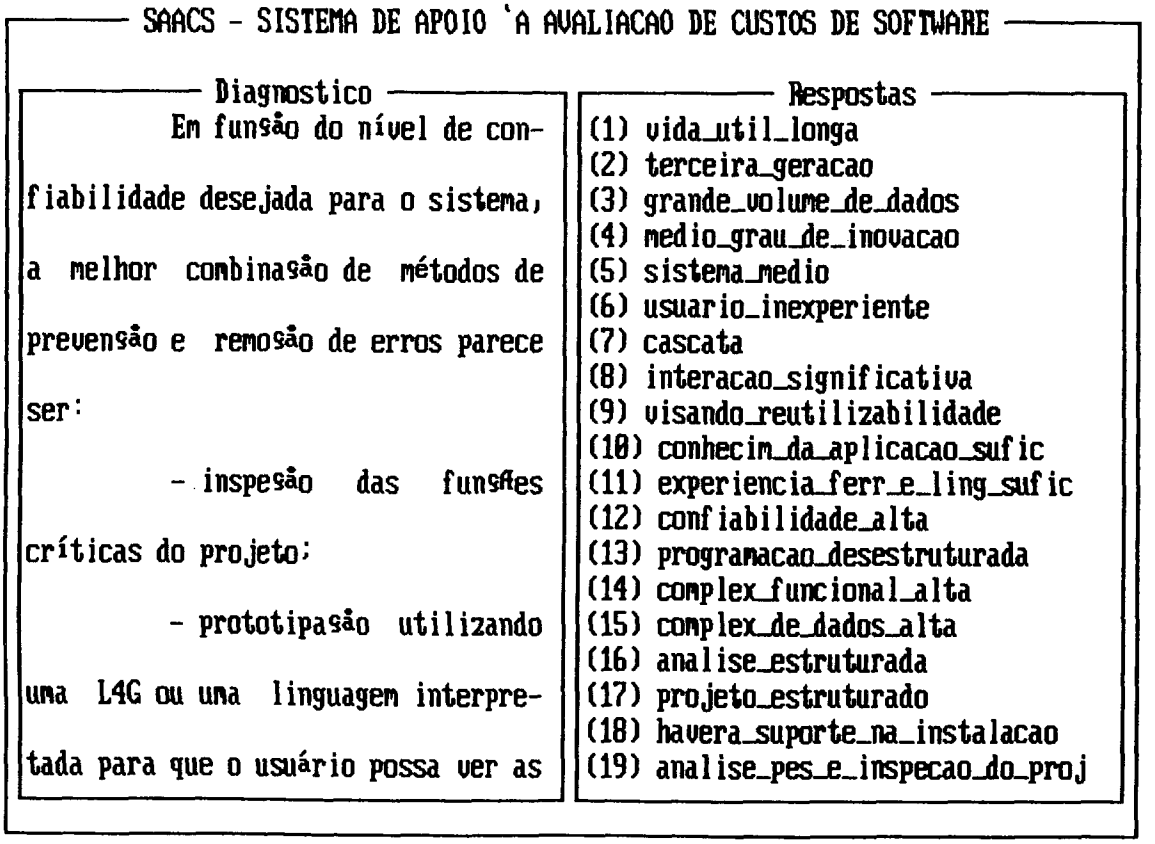

Figura A.14: Diagnóstico e "what if".

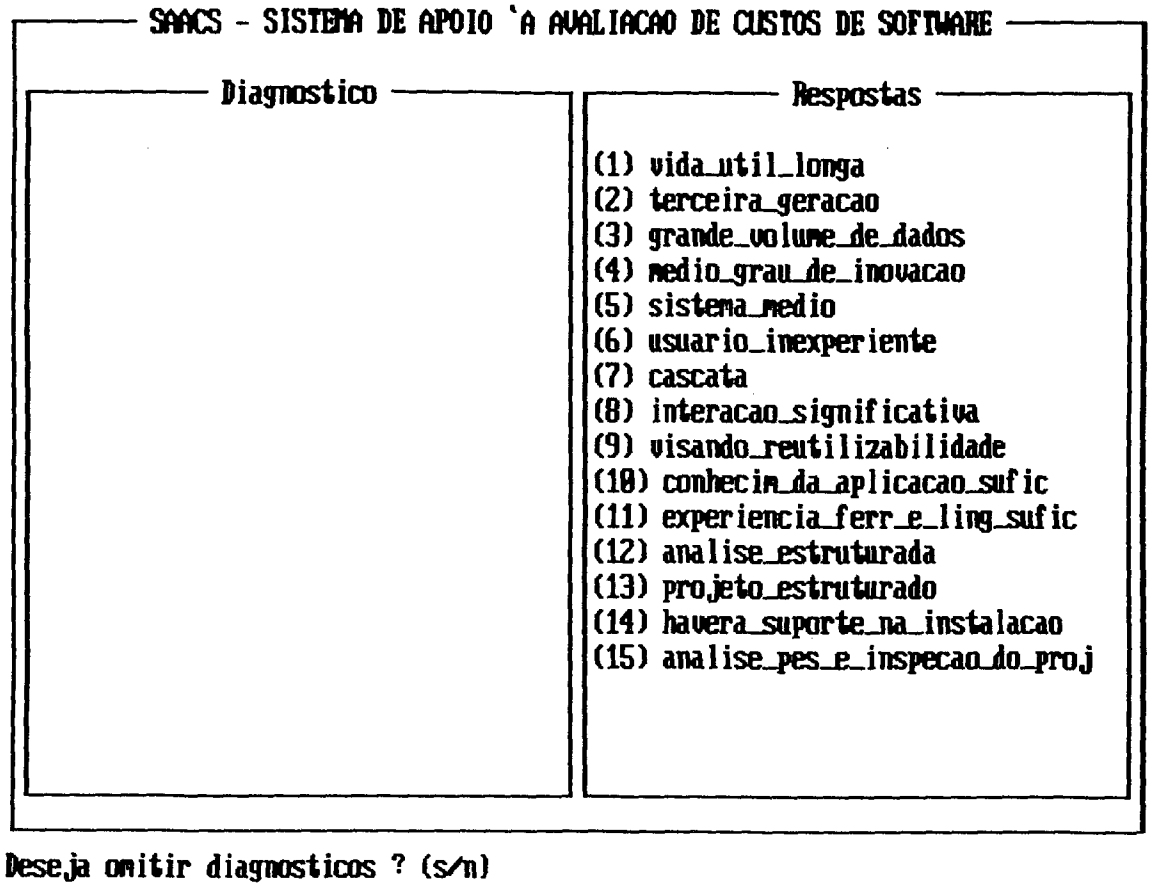

Figura A.15: Janela de respostas e "what if".

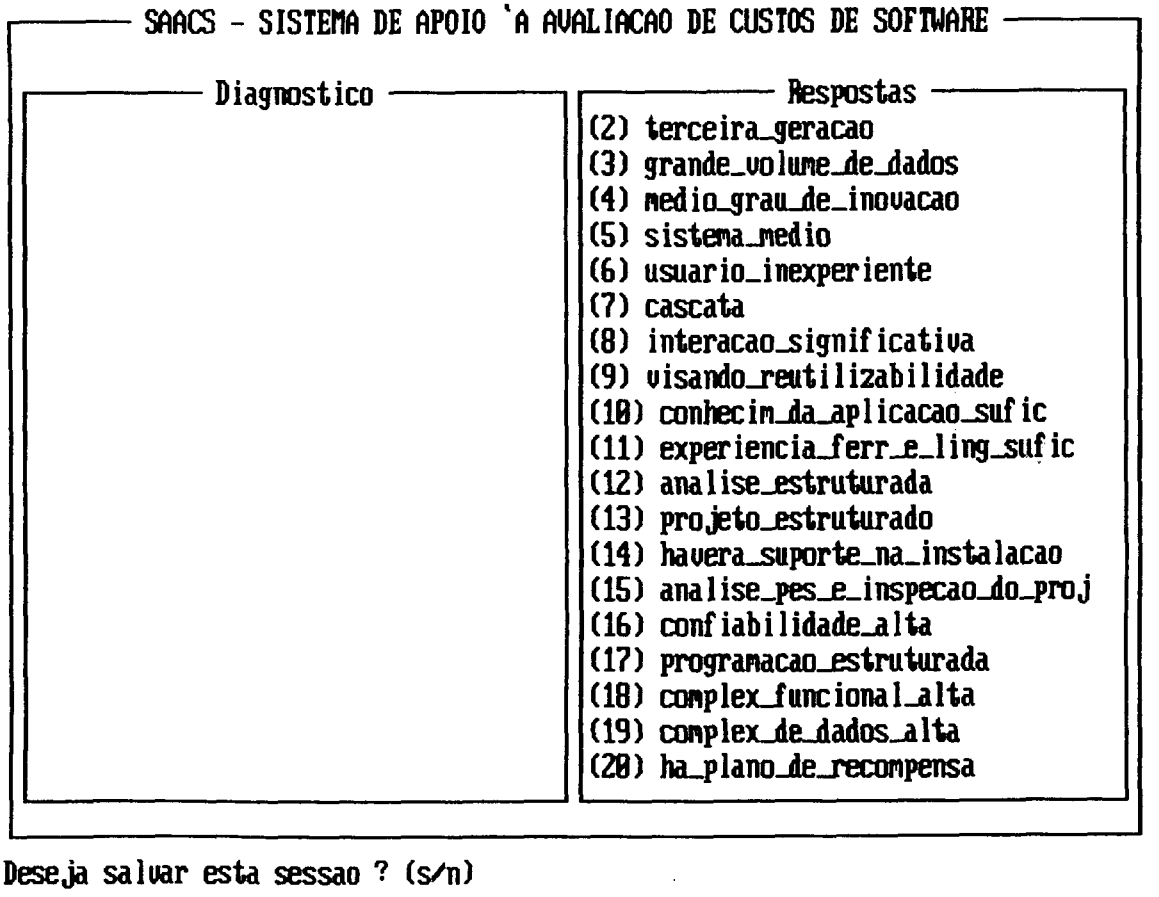

Figura A.16: Salvamento da sessão.

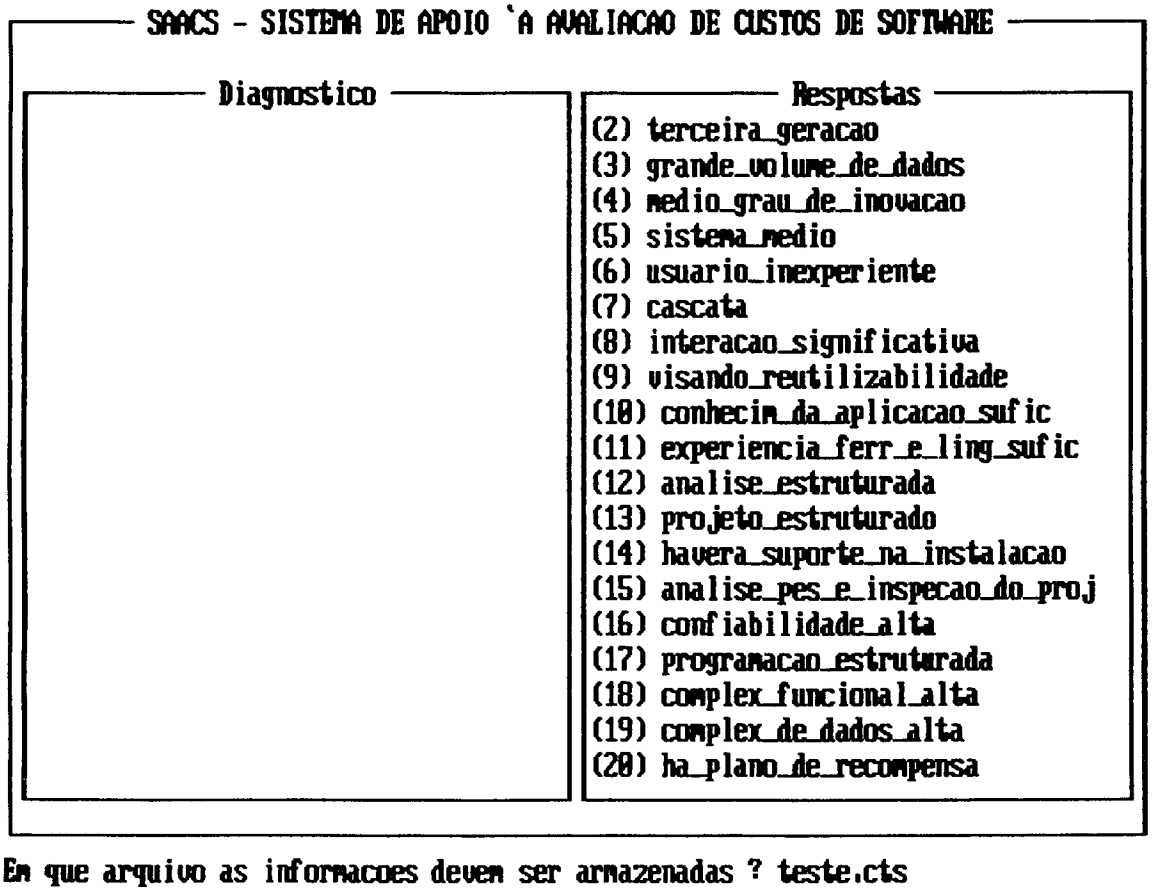

Figura A.17: Especificação do arquivo DOS.