UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO
E AUTOMAÇÃO INDUSTRIAL

Este exemplar corresponde à redação final da tese defendida por Manoel Gomes de Mendonça Neto e aprovada pela Comissão Julgadora em 18 de junho de 1990

Mario Jino

# UM AMBIENTE BASEADO EM CONHECIMENTO PARA SUPORTE AO PLANEJAMENTO DO DESENVOLVIMENTO DE SOFTWARE

*POR:* Eng. Manoel Gomes de Mendonça Neto

*ORIENTADOR:* Prof. Dr. Mário Jino

Dissertação apresentada à Faculdade de Engenharia Elétrica da UNICAMP como parte dos requisitos exigidos para obtenção do título de "Mestre em Engenharia Elétrica".

Junho 1990

Dedico este trabalho
à Conça e Gau,
com seu incentivo e carinho,
elas me ajudaram a realizá-lo.

# AGRADECIMENTOS

Ao Prof. Mário Jino pela competente orientação.

Ao Prof. Fernando Gomide pela orientação no uso da engenharia de conhecimento.

Ao Prof. Manuel de Jesus Mendes pelo constante incentivo e amizade.

Aos colegas da FEE, em especial a Guido, João, Rabay, Alexandre, Thomas e José, por tão frutífera convivência.

Aos professores da Faculdade de Engenharia Elétrica, e em especial do DCA, que tanto colaboraram para este trabalho.

Aos funcionários da CPG da FEE, da secretaria do DCA e da biblioteca da FEC que sempre tão prontamente me atenderam.

Aos meus pais e irmãos, sem eles este trabalho não teria sido possível.

À CAPES, pelo apoio financeiro.

A todos que, direta ou indiretamente, colaboraram para execução deste trabalho.

# RESUMO

A evolução das tecnologias de *hardware* dos sistemas computacionais nas últimas duas décadas propiciou o rápido barateamento do seu custo permitindo a produção de máquinas cada vez mais poderosas a custos cada vez mais acessíveis. A tecnologia de *software*, entretanto, não evoluiu com a mesma rapidez. A consequência deste processo é que existe uma demanda crescente por *softwares* maiores e de melhor qualidade. Devido a isto, tem-se observando um aumento constante dos custos dos *softwares* que são necessários nos dias atuais. Dentro desta realidade é primordial se conhecer e controlar melhor os processos de desenvolvimento e manutenção do *software*. Um dos pontos mais importantes para se controlar estes processos, e consequentemente seus custos, é planejá-los de forma eficiente.

Esta dissertação apresenta o trabalho de desenvolvimento de um "Sistema de Auxílio ao Planejamento do Desenvolvimento de *Software*" (SAPDES). Este sistema pretende auxiliar o planejamento do desenvolvimento e manutenção de sistemas de *software* através de estimativas de cronograma e custos de realização destas atividades.

Para cumprir seus objetivos o SAPDES integra de forma modular diversos modelos de estimativa de custo de *software* em um ambiente que permite ao usuário a fácil seleção e utilização destes modelos. O sistema é baseado em conhecimento: ele utiliza heurística na escolha dos modelos que mais se adequam ao problema/usuário, e realiza um diálogo inteligente para obtenção de dados necessários aos modelos.

Além de apresentar o SAPDES, esta dissertação discute ainda diversos modelos e ferramentas automatizadas de estimativa de custo de *software*, compará-os, e situa o SAPDES neste universo.

# ABSTRACT

The evolution of computing hardware technology in the last two decades has caused a steep decrease in its cost, making possible the production of more powerfull and cheaper machines. Software technology, however, has not evolved at the same rate. There is a growing demand for larger, more complex and better softwares, and the production of software is becoming more and more expensive. This situation and the problems related to software development make fundamental a clear understanding of the software development and maintenance processes, as well as, the availability of better ways to control them. One of the most important points for the control of these processes (and their costs) is to plan them efficiently.

This thesis presents SAPDES (System for Support of Software Development Planning). SAPDES intends to help in the planning of development and maintenance of software systems through estimates of costs and schedules of these activities.

In order to achieve its purpose, SAPDES integrates several software cost estimation models in a modular fashion. These models are put together into an environment that permits the user to choose and apply them easily. The System is knowledge-based; it uses heuristics to choose the user/problem models that best apply, and performs an intelligent dialog for input of model data.

In addition to presenting SAPDES, this report discusses and compares several estimation models and automated tools, and locates SAPDES within this universe.

# ÍNDICE

# ÍNDICE (CONTINUAÇÃO)

# LISTA DE FIGURAS

# Lista de Tabelas

# CAPÍTULO 1

# INTRODUÇÃO

A tarefa de desenvolver sistemas de *software*[1] mudou significativamente durante as três últimas décadas. Durante os primeiro dias do desenvolvimento de sistemas computacionais, o grande desafio dos projetistas era construir o *hardware* dos computadores; a construção do *software* deste sistemas era uma tarefa complementar frequentemente relegada a segundo plano. Nos anos que se seguiram à evolução tecnológica na área de semicondutores e microeletrônica propiciou a construção de *hardwares* cada vez mais poderosos a custos cada vez menores. A partir dos meados dos anos 60, o desenvolvimento do *hardware* ocasionou uma demanda cada vez maior de sistemas de *software*; estes sistemas começaram a alcançar tamanhos e níveis de complexidade que os tornaram difíceis de compreender e alterar; como consequência imediata, estes sistemas passaram a não cumprir seus requisitos e, frequentemente, eram não confiáveis, ineficientes e pouco documentados. Este problema ficou conhecido como *a crise do software*.

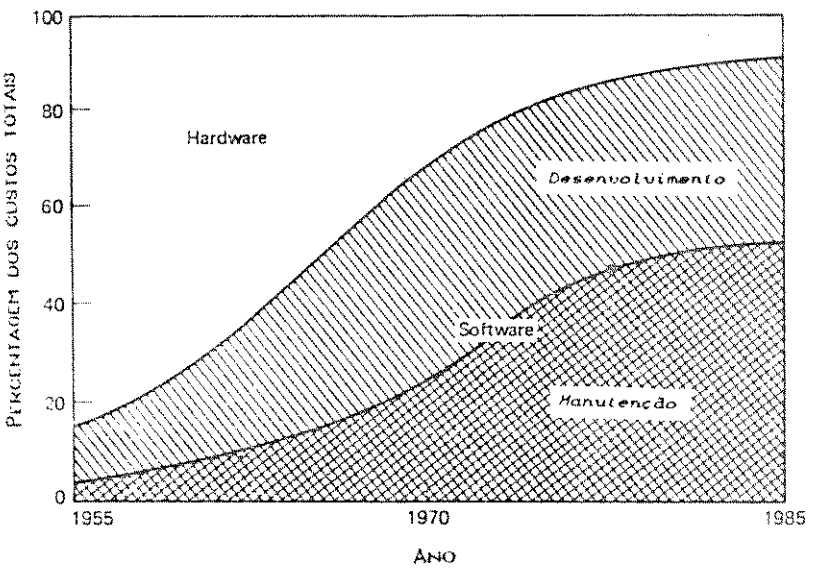


*Figura 1.1* Variação do custo relativo Hardware/Software

[1] a palavra software engloba o conjunto de programas, procedimentos, e documentação relacionada, associados a um sistema computacional.

Com esta crise, o *software* começou a ter um custo relativo cada vez maior no projeto de sistemas computacionais, chegando hoje a ser o ítem de mais alto custo neste sistemas, como ilustra a Figura 1.1 [BOE81]. Como consequência passou-se estudar mais a fundo o problema e, nos anos 70, soluções para a crise do *software* começaram a ser apresentadas, surgiram metodologias e ferramentas[2] para gerência , desenvolvimento e manutenção de sistemas de *software*. Estes sistemas começaram a ser encarados como um produto de engenharia que requer planejamento, análise, projeto, implementação, teste e manutenção. Esta abordagem criou um novo campo de estudos denominado de *Engenharia de Software*.

Dos anos 70 até os dias de hoje a engenharia de *software* tem evoluido rapidamente. Atualmente já se dispõe de ferramentas que procuram integrar diversas metodologias e técnicas de engenharia de *software* com o intuito de gerir, desenvolver e manter sistemas de *software*. Estas ferramentas são chamadas de Ambientes de Desenvolvimento de Software [PEN88].

Se compararmos a engenharia de *software* com outras áreas da engenharia, veremos que, apesar da evolução, ela ainda é um campo muito novo de estudos. A maior dificuldade desta área é que ela possui uma característica própria [LUC87]. Como o *software* é um produto lógico e não físico seus custos se concentram no desenvolvimento e não na produção. Como ele não se 'desgasta', sua confiabilidade é determinada por conceitos abstratos como a sua correção ou tolerância a falhas. As características como variedades de funções, variedades de implementações, evolução, visibilidade e continuidade, criam problemas no controle da complexidade de projetos de *software*.

O problema central da área de engenharia de *software* hoje é que a demanda por *softwares* maiores e melhores ainda cresce mais rápido que a capacidade de produzí-los [BOE88] e [MAR83]. Em artigo recente , Ware Myers [MYE89] afirma que é provável que se necessite de sistemas com 10 milhões de linhas de código fonte na década de 90. Esta demanda acelerada e o tamanho dos produtos tem gerado o acúmulo de softwares não desenvolvidos e aumentado os custos necessários para desenvolvê-los.

---

[2] define-se 'ferramenta de software' como um programa ou uma coleção de programas que pode dar suporte para aplicação de uma metodologia.

Compreender e controlar os custos do *software* tornou-se um dos pontos chaves na engenharia de *software* (Boehm [BOE88] estima que em 1990 se gastará 250 bilhões de dólares na produção de *software*). Dentro do objetivo de se controlar custo de *software* é fundamental a capacidade de prever e planejar custo, tempo e cronograma de desenvolvimento de um projeto de *software*.

Esta dissertação apresenta uma ferramenta que auxilia na previsão e planejamento de um projeto de *software*. A ferramenta é denominada de Sistema de Auxílio ao Planejamento de Desenvolvimento de Software - SAPDES; ela integra de forma inteligente alguns modelos de estimativa de custo, tempo e cronograma de projetos de *software*. O SAPDES orienta o usuário na escolha dos modelos de estimativas que melhor se aplicam ao seu projeto, e ajuda o usuário a aplicar estes modelos e interpretar os seus resultados.

A dissertação está organizada em 6 capítulos. O primeiro capítulo contém esta introdução. O segundo capítulo apresenta o campo de estimativas e métricas em engenharia de *software*. O terceiro capítulo apresenta os principais modelos e ferramentas de estimativa de custo e tempo de desenvolvimento de software. O quarto capítulo apresenta o projeto funcional do SAPDES e como se pretende usar engenharia de conhecimento nas estimativas. O quinto capítulo descreve o SAPDES e sua implementação. O Capítulo 6 contém as conclusões do trabalho realizado e discute as pendências e desenvolvimentos futuros no SAPDES.

Neste trabalho são adotadas as seguintes convenções:

- Palavras da língua inglesa consideradas como jargão na área de computação serão escritas em itálico, por exemplo: *software*, *hardware*, *interface*, *frame*, etc.

- Estas palavras ou abreviações que tenham origem na língua inglesa serão explicadas em notas de pé de página.

- Termos que, de forma inversa, foram traduzidos para o português mas são frequentemente utilizados em inglês, terão, também, a palavra inglesa mencionada e explicada em notas de pé de página.

- As referências contêm as primeiras três letras do nome do autor e o ano do trabalho; as referências estão listadas em ordem alfabética no fim da dissertação.

# CAPÍTULO 2

# MEDIDAS E ESTIMATIVAS

Na programação de computadores o interesse de se estimar custo, esforço, produtividade e outros recursos necessários para o desenvolvimento e manutenção de sistemas de *software* começou nos anos 60 ( [FAR65], [NEL66] e [ARO69] ). Este interesse se confunde com o próprio surgimento da crise do software, pois ele refletia a preocupação com o crescimento dos custos relativos do *software* nos custos totais dos sistemas computacionais.

Nos anos 70 o desenvolvimento da tecnologia de *hardware* se tornou mais acentuado fazendo com que os preços das máquinas diminuissem e sua capacidade de processamento aumentasse. A demanda por *softwares* maiores e melhores se tornou tão grande que perguntas como *"Quanto vai custar ?"* passaram a ser acompanhadas frequentemente por questões do tipo *"É viável fazer ?"* ou *" É possivel cumprir os requisitos ?"*.

Durante a última década a "Engenharia de Software" progrediu muito e novas metodologias e ferramentas para especificar, desenvolver, testar e manter um sistema computacional estão surgindo a todo momento. Entretanto, hoje, um gerente ao se deparar com uma nova proposta de projeto de *software* enfrenta exatamente as mesmas perguntas de muitos anos atrás [PUT80] :

- É possivel fazê-lo ?
- Quanto ele vai custar ?
- Quanto ele vai durar ?
- Quantas pessoas serão utilizadas nele ?
- Qual será o esforço necessário para sua construção ?
- Qual será o fluxo de caixa necessário ?
- Quais os riscos associados ao projeto ?

As respostas para este conjunto de perguntas estão interligadas e o seu estudo pode ser enquadrado em um único campo, o das "Estimativas de Recursos para o Desenvolvimento e Manutenção de Sistemas de *Software*". Este capítulo se dedica a introduzir este campo de estudos e situar este trabalho dentro dele.

## 2.1 O QUE SE QUER ESTIMAR

As grandezas que desejamos conhecer são as que respondem às perguntas formuladas pelo gerente no início de um projeto (seção anterior) e aquelas que respondem ao mesmo conjunto de perguntas aplicadas aos esforços de manutenção e alteração no *software*. Elas estão listadas abaixo :

- Custo (em alguma unidade monetária)
- Esforço (em alguma unidade de trabalho)
- Distribuição de custo por tempo
- Tempo de desenvolvimento e/ou manutenção

- Qualidade do *software*
- Confiabilidade do *software*
- Complexidade do *software*
- Facilidade de manutenção do *software*
- Eficiência do *software*

As quatro primeiras grandezas são classificadas como medidas diretas pois podem ser coletadas facilmente durante o processo de desenvolvimento do *software*. As outras grandezas são classificadas como medidas indiretas pois são obtidas a partir de outras medidas determinadas durante o ciclo de vida do software. A seguir listamos algumas grandezas fundamentais na determinação das grandezas medidas indiretamente.

- Taxa de erros
- Tempo entre erros
- Tempo de reparo de erros
- Esforço de reparo de erros
- Esforço em modificações de projeto
- Nivel de documentação

A maioria destas medidas se apóia em algum tipo de métrica básica. A mais conhecida destas métricas é o tamanho do *software* em linhas de código fonte (LOC[1]); por exemplo, a taxa de erros

---

[1] abreviatura do inglês, lines of code

pode ser expressa como número de erros por linhas de código fonte, o nível de documentação pode ser expresso como número de páginas de documentação por linhas de código fonte , etc. Outra métrica básica de comparação são os *FUNCTION-POINTS*[2] [ALB83] que procura medir a funcionalidade ou utilidade de um programa; é uma medida indireta e sua determinação depende da contagem dos pontos de *interfaces* externas e da caracterização do sistema (o processo de contagem dos *FUNCTION-POINTS* será detalhado no capítulo 3). Merecem ainda menção as métricas de Halstead [HAL77] e de McCabe [MCC 76], usualmente utilizadas na medida de complexidade de sistemas; entretanto, elas não serão mais mencionadas aqui pois elas só ficam disponíveis muito tarde no ciclo de desenvolvimento de software tornando assim nula sua utilidade em estimativas. As quatro métricas citadas acima são classificadas como clássicas devido à grande quantidade de trabalhos publicados sobre elas. Outras métricas têm sido alvo de estudo de diversos pesquisadores; uma lista bastante completa e atualizada delas pode ser obtida no trabalho de Côte, Bourque, Oligny e Rivard [COT88].

Apesar da importância de todas as grandezas citadas, os modelos de estimativas, devido aos seus estágios ainda primários, buscam fundamentalmente determinar as grandezas de medidas diretas, pois são as de mais fácil tratamento e as que geram maior interesse nos momentos iniciais de um projeto. Porisso as grandezas básicas que vão ser discutidas, estudadas e estimadas neste trabalho serão:

- Custo
- Esforço
- Distribuição de custo por tempo
- Tempo de desenvolvimento e/ou manutenção

Como o recurso mais crítico é o custo e sua componente maior tem sido invariavelmente o esforço humano (custo de pessoal)[3], de aqui em diante usaremos as expressões "Estimativa de Custo ou Esforço" para denominar o processo de previsão destas grandezas básicas pelos modelos que serão estudados neste trabalho.

---

[2] do inglês, pontos de função

[3] Neste trabalho as palavras Custo e Esforço serão intercambiadas frequentemente, a seção 2.2.2 discute a relação entre as duas grandezas

## 2.2 GRANDEZAS ESTIMADAS: DEFINIÇÕES, UNIDADES E CONVERSÕES

Nesta seção estabelecem-se algumas definições importantes sobre as grandezas: linhas de código fonte, tempo, custo e esforço de desenvolvimento. Discutem-se a relação entre custo e esforço de desenvolvimento ,e alguns aspectos da distribuição de esforço por tempo de desenvolvimento.

### 2.2.1 Definições

Na seção 2.1 citou-se a grandeza linhas de código fonte como uma das mais importantes métricas em engenharia de software; de fato ela é utilizada como a principal grandeza para estimativa em diversos modelos. Daí a necessidade de uma definição precisa do que consideramos como linhas de código fonte. O termo código fonte visa incluir todas as instruções criadas pelo pessoal de projeto e processadas para código de máquina por meio de pré-processadores, compiladores e/ou montadores. Na contagem excluem-se comentários e *softwares* utilitários prontos, mas são incluídos comandos de linguagens de controle de *job*[4] , declarações de formato, e declarações de dados. Como a contagem é por linhas, se tivermos uma linha contendo duas ou mais declarações de dados, ou uma declaração de dados que ocupe cinco linhas, não importa, devemos contá-las como uma linha e cinco linhas, respectivamente. Só devemos incluir na contagem de linhas de código os *softwares* de suporte que são desenvolvidos e não são entregues (por exemplo, programas de teste), se estes são desenvolvidos com os mesmos cuidados dados ao *software* entregue, isto é, tiverem revisão, planos de teste, documentação ,etc.

O esforço de desenvolvimento é a principal grandeza de nosso interesse; ele é dado pelo número de pessoas que trabalharam no projeto vezes o tempo que se dedicaram a este trabalho e será representado de agora em diante neste trabalho pela letra "*E*". As unidades usadas para exprimir esforço são pessoas-dia,

---

[4] do termo em inglês "job control languages" que denota as linguagens de controle de compilação e processamento em computadores multi-usuários.

pessoas-hora, pessoas-mês e pessoas-ano. Tomando como base a definição de que uma pessoa-mês correspode a 152 horas de trabalho (este valor é tirado da experiência prática e já considera feriados, férias e faltas devido a doenças), teremos os seguintes critérios de conversão entre unidades [BOEB1] :

PESSOAS-HORA = PESSOAS-MÊS X 152
PESSOAS-DIA = PESSOAS-MÊS X 19
PESSOAS-ANO = PESSOAS-MÊS / 12

A outra grandeza de fundamental interesse é o tempo de desenvolvimento; ela é tipicamente expressa em meses ou anos nos modelos de estimativas. De aqui em diante este trabalho usará as letras "td" para representar esta grandeza. Apesar de não apresentar grandes dificuldades para sua compreensão, esta grandeza guarda relações complexas com a grandeza esforço; a Seção 2.2.3 se dedica a este tema.

O custo de desenvolvimento é expresso em alguma unidade monetária; sua componente fundamental são os gastos com pessoal e porisso o custo guarda uma relação estreita com o esforço de desenvolvimento. O ítem que se segue é dedicado exclusivamente à discussão da grandeza custo, sua relação com o esforço e a razão de a maioria dos métodos procurar estimar esforço e não o custo de desenvolvimento.

### 2.2.2 Custo x Esforço de Desenvolvimento

A maioria dos métodos evita calcular os custos em dólares, ou no caso brasileiro em cruzeiros, devido à grande variação, nas organizações, do que é considerado e incluído como custos (encargos sociais, aluguel de escritório, participações, etc), e devido ao fato da grandeza pessoas-mês ser muito mais estável que qualquer grandeza monetária, já que esta está sujeita à inflação e flutuações de mercado. No Brasil o segundo aspecto citado é especialmente grave devido à atual instabilidade de todas as unidades monetárias; portanto, estimar esforço é muito mais lógico do que estimar custos. Entretanto, apesar de usados frequentemente de forma intercambiável nos textos sobre estimativas de custo e de estarem intimamente relacionados, o custo e o esforço de desenvolvimento não estão sempre relacionados por uma função

simples. O esforço é, como vimos, expresso em pessoas-mês do pessoal técnico necessário para realizar o projeto, e não é fácil transformar isto em custo. O pessoal técnico irá envolver programadores, analistas, líderes de projeto e gerência diretamente associada ao projeto. A tarefa de converter o esforço deste pessoal em custo monetário envolverá o cálculo do salário médio por unidade de tempo e a multiplicação deste valor pelo esforço total (salário médio por mês X pessoas-mês, por exemplo). O problema é determinar o salário médio, já que ele envolve o trabalho relativo de cada tipo de profissional participante e a taxa média de trabalho técnico por dia, considerando-se folgas, feriados, trabalho em atividades administrativas, etc. Para fazer esta determinação deve-se encontrar o melhor compromisso entre simplicidade e precisão; para isso, a melhor abordagem é determinar um salário médio para cada fase principal do projeto. A Figura 2.1 mostra as participações relativas do pessoal técnico envolvido de acordo com a fase do projeto [PRE87].

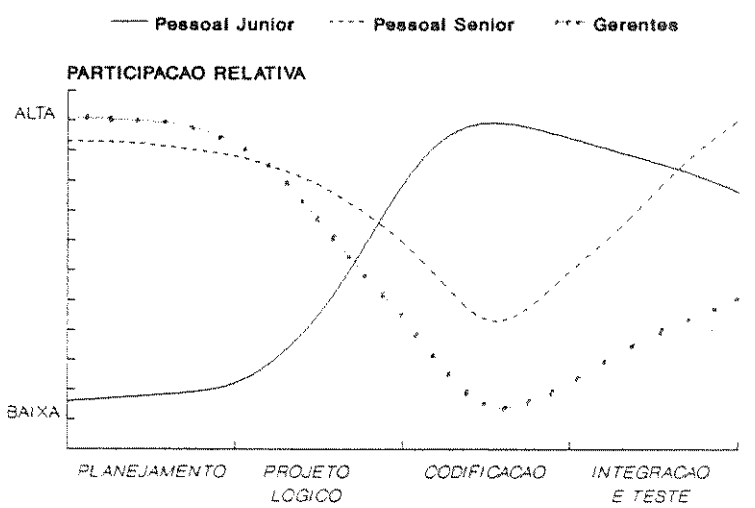


*Figura 2.1* Participação Relativa do Pessoal Técnico por Fase do Projeto

### 2.2.3 Distribuição de Esforço por Tempo de Desenvolvimento

Considerar que o esforço de desenvolvimento é simplesmente o produto do número de pessoas envolvidas pelo tempo de desenvolvimento de um projeto é o que os engenheiros de *software* chamam de um "mito de *software*". O mito está no fato de se pensar que o tempo pode ser arbitrariamente estabelecido pelo gerente desde que este forneça a força de trabalho necessária para perfazer determinado esforço neste tempo, como mostra a Figura 2.2 .

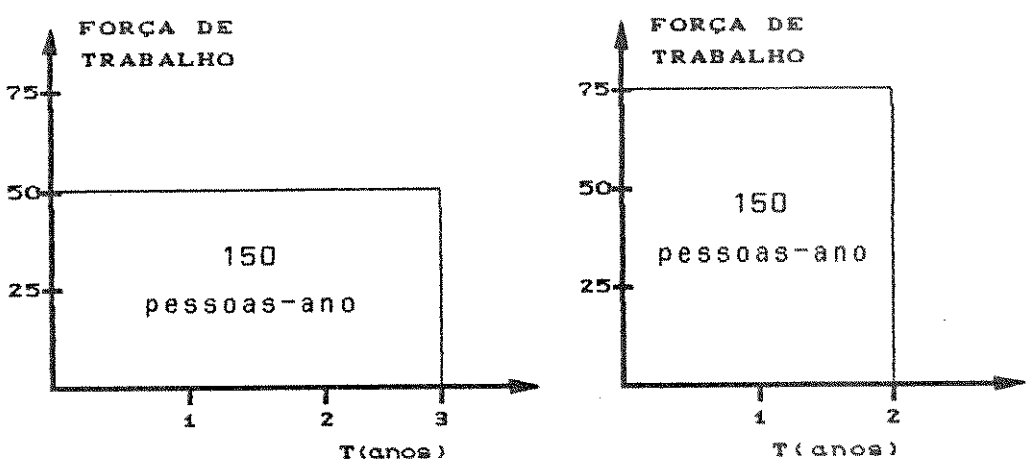


*Figura 2.2* Mito do Compromisso "Esforço X Tempo"

Acontece que tempo e força de trabalho não são arbitrariamente intercambiáveis pois a produtividade depende da força de trabalho empregada (vide discussão na seção 2.3.1). Os modelos de estimativas consideram este fato e pressupõem alguma curva padrão de força de trabalho por tempo de desenvolvimento. Estas curvas têm tipicamente um formato onde a força de trabalho começa com um valor baixo no início do projeto, cresce à medida que ele se desenvolve, atinge o topo perto do final do desenvolvimento para então ir decaindo à medida que o ciclo de vida do *software* prossegue. A seguir apresentamos (Figura 2.3) três curvas de força de trabalho por tempo de desenvolvimento bastante conhecidas: a primeira é a distribuição de Rayleigh-Norden, a segunda é a adotada por Boehm para seu modelo de estimativa, o COCOMO, e a terceira foi proposta por Parr como uma alternativa para curva a de Rayleigh [PAR80].

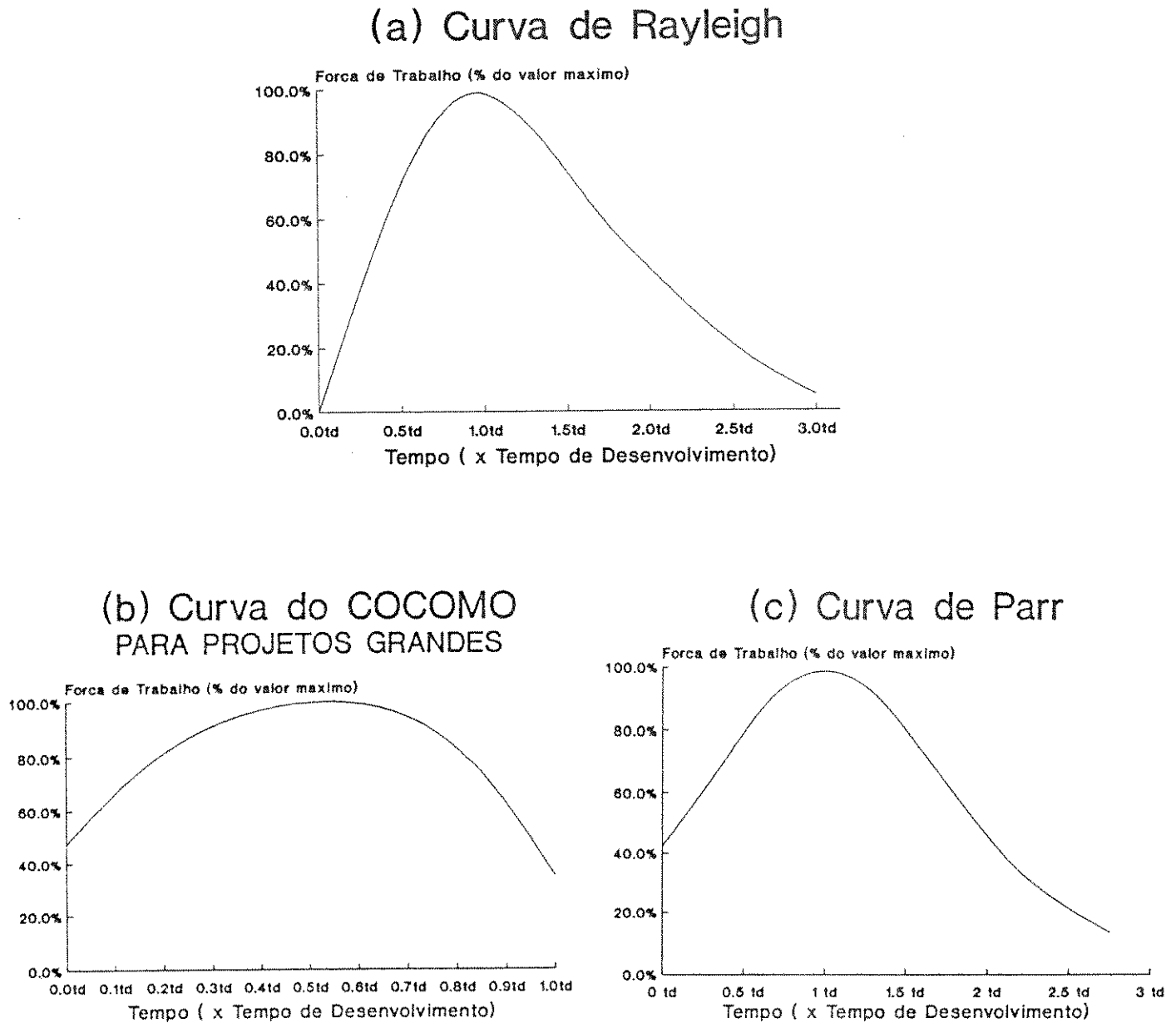


*Figura 2.3* Distribuições de Força de Trabalho por Tempo

A primeira curva surgiu do trabalho de Norden [NOR63] que afirmava que a distribuição de Rayleigh fornecia uma boa aproximação para a distribuição da força de trabalho em muitos tipos de atividades relacionadas à pesquisa e desenvolvimento. Este resultado foi estendido por Putnam [PUT78], que sugeriu a aplicação deste tipo de curva a projetos de *software*, criando um modelo de estimativa a partir de sua equação; o Capítulo 3 deste

trabalho analisa mais detidamente esta curva e explica o significado de cada um dos termos da equação da curva de Rayleigh-Norden apresentada abaixo:

$$\text{FORÇA DE TRABALHO} = E.\left(\frac{t}{t_d^2}\right).e^{-(t^2/2t_d^2)} \qquad \text{(eq. 2.1)}$$

A curva proposta por Boehm foi derivada de regressão estatística sobre dados a respeito de vários projetos de desenvolvimento de *software* e porisso não apresenta equação; entretanto, no seu trabalho [BOE81] Boehm aproxima a curva de Rayleigh à sua curva para um determinado projeto e conclui que elas só possuem apreciáveis diferenças na cauda da curva de Rayleigh (fase de manutenção do projeto).

A curva de Parr foi desenvolvida por seu autor para cobrir uma das grandes falhas da curva de Rayleigh que é considerar que a força de trabalho inicial de projeto é zero. A sua equação é mostrada a seguir:

$$\text{FORÇA DE TRABALHO} = \alpha.E.A.e^{-\alpha\gamma t}/(1+Ae^{\alpha\gamma t})^{1+1/\gamma} \qquad \text{(eq. 2.2)}$$

onde: t é o tempo em semanas
E é o esforço de desenvolvimento em homens-hora
A é o fator de deslocamento horizontal
$\alpha$ é o fator de normalização de tempo
$\gamma$ é o índice de estruturação do projeto

O trabalho de Parr não foi aplicado em modelos de estimativas devido à dificuldade de determinar todos os índices e fatores presentes nas suas fórmulas; uma discussão sobre o assunto pode ser encontrada em [BAS81].

As curvas mostradas têm um intervalo de variação para um mesmo projeto, estabelecendo-se a possibilidade de várias soluções de compromisso de $td$ x $E$. No trabalho de Boehm, ele considera que a variação máxima é uma compressão de 25% no tempo de desenvolvimento estimado; para calcular o esforço correspondente ao tempo de desenvolvimento comprimido se aplica uma fórmula que relaciona $td$ X E. No trabalho de Putnam é calculado um gradiente de dificuldade que estabelece o limite inferior do tempo de desenvolvimento, e uma equação obtida da curva de Rayleigh é usada para determinar o esforço a partir do tempo de desenvolvimento escolhido.

## 2.3 ASPECTOS QUE DEVEM SER CONHECIDOS ANTES DE SE ESTIMAR

Para se obter estimativas confiáveis de custo de *software* não basta apenas colocar números em uma fórmula ou seguir um algoritmo de cálculo: é fundamental que se compreendam alguns aspectos da natureza do desenvolvimento do *software* bem como das limitações intrínsecas ao processo de estimativa. A seguir, discutimos alguns aspectos das incertezas das grandezas que se deseja estimar e tecemos algums comentários quanto a suposições adotadas nos modelos de estimativas.

### 2.3.1 Número e Organização do Pessoal de Desenvolvimento

Este aspecto foi tratado pioneiramente por Brooks em seu conhecido livro "*The Mythical Man-Month*[5]" [BR075]; ele observou que à medida que um time de programação cresce, o número de intercomunicações entre os programadores aumenta para realizar a coordenação apropriada na construção dos sub-módulos do sistema. A consequência deste efeito é que a produtividade individual diminui e o esforço total aumenta quando aumentamos o número total de pessoas envolvidas em um projeto.

> "*... aumentar o número de pessoas em um projeto de software atrasado causa ainda maior atraso.*"
>
> *FREDERICK BROOKS* [BR074],[BR075]

Matematicamente falando, se o projeto for organizado de forma que cada membro de um time de $\mathbb{P}$ programadores tiver que comunicar-se com cada um dos outros $(\mathbb{P}-1)$ programadores, teremos então $\mathbb{P}(\mathbb{P}-1)/2$ caminhos de comunicação entre eles; se, por outro lado, cada membro se comunicar apenas com um único supevisor geral teremos $\mathbb{P}$ caminhos de comunicação entre eles. Entre os dois extremos, um com $\Phi(\mathbb{P})$ e outro com $\Phi(\mathbb{P}^2)$, haverá organizações cuja

---

[5] do inglês, "O Mítico Homem-Mês" (esta expressão refere-se aos mitos sobre produtividade do programador já que homem-mês é a mais conhecida unidade para medir esforço de programação)

gerência irá conduzir a estruturas com caminhos de comunicação com $\Theta(\mathbb{P}^{\varepsilon})$, $1 \leq \varepsilon \leq 2$. Tomando como exemplo um caso onde o projeto é dividido em $\mathbb{K}$ sub-grupos de trabalho com o mesmo número de pessoas, sendo que, dentro de cada sub-grupo a comunicação é livre (cada um se comunica com todos os outros) mas entre dois grupos existe apenas um caminho de comunicação, o número total de caminhos de comunicação, $C(\mathbb{P})$, seria dado por:

$$C(\mathbb{P}) = \frac{\mathbb{K}}{2} \cdot \frac{\mathbb{P}}{\mathbb{K}} \cdot \left( \frac{\mathbb{P}}{\mathbb{K}} - 1 \right) + \mathbb{K} \qquad \text{(eq. 2.3)}$$

Se supomos que para cada caminho de comunicação necessário o programador tem uma perda de produtividade, vê-se que à medida que $\mathbb{P}$ aumenta a produtividade média diminui. Conte e outros [CON85] mostraram que existe um ponto de máximo na curva de número total de linhas produzidas por quantidade de pessoal utilizado e apresentam alguns cálculos para determinar $\mathbb{P}$ ótimo para máxima produção (note que máxima produção não implica em máxima produtividade, pois ,como já se afirmou, esta decresce à medida que $\mathbb{P}$ aumenta).

Da discussão anterior nota-se que a produtividade dos programadores e, consequentemente, o custo e tempo de desenvolvimento, irá depender diretamente do número e da organização das pessoas envolvidas no projeto. Em projetos onde se pretende aplicar os modelos de estimativa que iremos discutir supõe-se que eles serão decompostos em módulos e o pessoal envolvido organizado em pequenos grupos de trabalho que independentemente irão desenvolver estes módulos do sistema. O nível de organização destes grupos de desenvolvimento refletirão então o nível de decomposição do sistema. Pressupõe-se que estes grupos envolvidos no projeto não serão substancialmente alterados durante o desenvolvimento.

### 2.3.2 Aplicação de Bons Princípios de Projeto

Só podemos pensar em estimar o esforço de desenvolvimento de um projeto de *software* que cumpra um conjunto mínimo de princípios que garanta seu desenvolvimento normal comparável à média dos projetos que vêm sendo historicamente desenvolvidos. Para isto é necessário que um projeto cumpra o que chamamos de bons princípios de desenvolvimento. Por bons princípios entende-se : programação estruturada; decomposição do projeto em módulos; estabelecimento de requisitos; inspeção dos requisitos estabelecidos ,do projeto funcional e do código; etc.

Como exemplo da importância destes princípios nos custos totais do software basta citar uma frase de BOEHM " Encontrar e resolver problemas de um programa após entregá-lo é cem vezes mais caro do que encontrá-los e resolvê-los durante as fases de análise de requisitos e projeto funcional " [BOE87].

Os princípios são fundamentais para uma boa estimativa; alguns são considerados implicitamente e outros explicitamente nos modelos de estimativas. Os princípios considerados implicitamente sempre devem estar presentes nos projetos em que se pretende utilizar os modelos de estimativa. Os outros princípios a serem considerados entrarão explicitamente em algum momento no processo de estimativa.

O processo de desenvolvimento de *software* que se pretende estimar deve ter os seguintes princípios básicos implícitos :

a - definição e validação cuidadosa dos requisitos do projeto por um número relativamente pequeno de pessoas, antes do início do projeto do sistema.

b - definição e validação cuidadosa do projeto do sistema, por uma equipe um pouco maior (mas ainda pequena), antes do trabalho de projeto detalhado e codificação.

c - o projeto detalhado, a codificação e o teste de unidade é feito por um grupo grande de programadores com organização rígida das tarefas a serem feitas.

d - a integração e teste de cada parte será baseada em uma quantidade significativa de programação de testes, para realizar uma pré-eliminação da maior parte das falhas internas dos sub-módulos a serem integrados.

e - boa parte do esforço de documentação (o rascunho do

manual do usuário, por exemplo) é feita o mais rapidamente possível, de forma a facilitar a realimentação entre usuários e projetistas sobre a natureza operacional do produto.

### 2.3.3 Influência da Qualificação do Pessoal

Um estudo de Walston e Felix [WAL77] indica variações na produtividade de 3,11 vezes de acordo com a experiência e qualificação geral do pessoal e, ainda, variações de 19,00 vezes devido à experiência específica do pessoal com a linguagem de programação, o computador de desenvolvimento e a aplicação. Considerando que as grandezas fossem totalmente independentes, teríamos no total uma variação de 58,9 vezes na produtividade devido aos aspectos de qualificação e experiência pessoal.

Em modelos de estimativa como o COCOMO (vide Cap. 3 deste trabalho) a influência média do pessoal de projeto nos custos são considerados pelo que o modelo chama de atributos de pessoal [BOE81]. Nesse modelo os chamados atributos de pessoal podem alterar os custos estimados em 10,53 vezes, sendo que 2,52 vezes devido à experiência do pessoal e 4,18 vezes devido à capacidade total das pessoas envolvidas no projeto.

Apesar de considerado um índice médio de qualificação, deve-se ressaltar que a qualificação individual dos programadores não é considerada pelos modelos ;muitas experiências controladas mostram uma grande variação de desempenho de um indivíduo para outro. A seguir estão alguns limites de variação do desempenho de programadores trabalhando em problemas idênticos, segundo alguns quantificadores típicos [FER81]:

| | |
|---|---|
| Tempo de codificação | 25:1 |
| Tempo de remoção de erros | 26:1 |
| Tamanho do programa | 5:1 |

*"Faça tudo ao seu alcance para conseguir as melhores pessoas para trabalhar no seu projeto". BARRY BOEHM* [BOE87]

A despeito destas tremendas variações, os modelos tendem a tratar da mesma forma todos os programadores. Os erros intrínsecos

dessa abordagem devem ser sobrepujados pelas médias de produtividade e pelos fatores de correção (calculados de acordo com a qualificação média do pessoal) usados pelos modelos. Desta discussão se conclui que a variação individual da produtividade dos programadores é um aspecto que introduz incerteza em uma estimativa e que a aplicação dos modelos a projetos, onde o número de programadores seja tão pequeno que a presença de pessoas sub ou super qualificadas altere significativamente a produtividade global, deve ser cuidadosa.

### 2.3.4 Incertezas Associadas às Estimativas

A estimativa será uma suposição do tempo de desenvolvimento e esforço para desenvolver um sistema; apesar de quantificada através de análises e estudos estatísticos, ela será sempre uma suposição [FER81].

A estimativa de custo e esforço em *software* nunca será uma ciência exata; muitas variáveis (humanas, técnicas, ambientais ,etc) podem afetar o custo final de um *software*. Deve-se notar que não se pode estimar sem riscos; por melhor que seja o modelo de estimativa de custo não há meios de se compensar a nossa falta de compreensão sobre o software a ser feito. Boehm ([BOE81],[BOE84]) argumenta que na fase de estudos de viabilidade do sistema a precisão com que a estimativa pode ser feita é tal que um fator de erro de 4 vezes para mais ou para menos no seu valor pode ocorrer. O nível de incerteza é ilustrado na Figura 2.4.

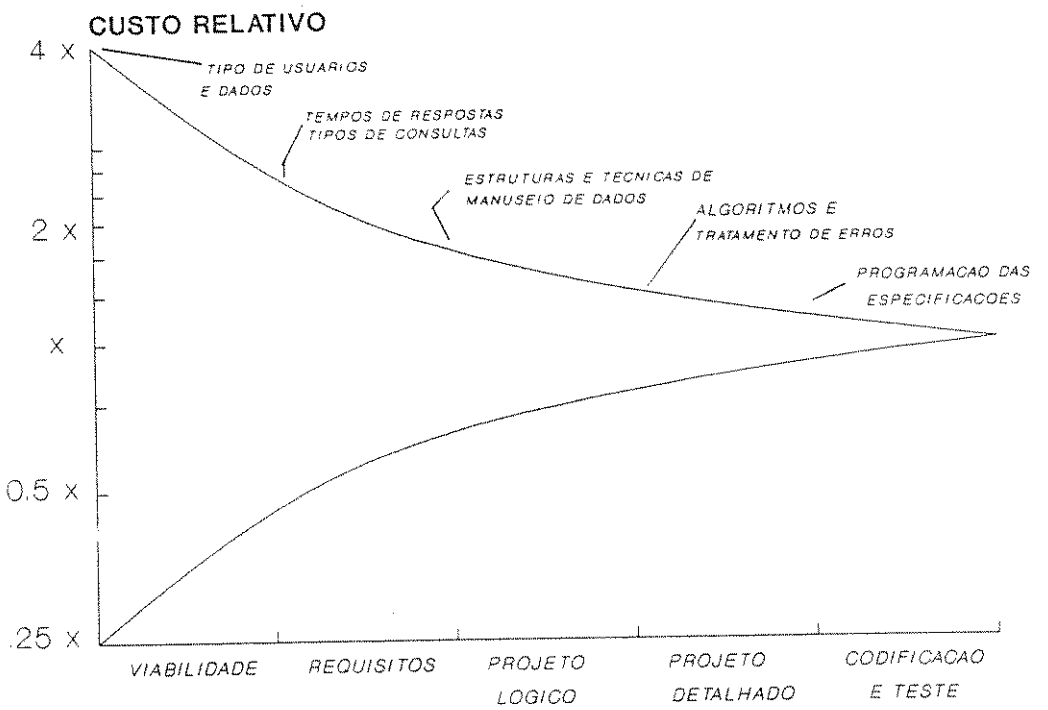


*Figura 2.4* **Incerteza da Estimativa por fase do Projeto**

O gráfico anterior mostra as curvas de variação das estimativas de custos por fase do software, e as principais fontes de incerteza, em um sistema de interface homem-máquina, são destacadas sobre a curva de acordo com a fase mostrada no eixo das ordenadas. Estas curvas limites foram determinadas subjetivamente e pretendem ser válidas para pelo menos 80% dos casos. Estas curvas ilustram bem o fato de que a incerteza da estimativa diminui à medida que o projeto avança, como afirma Pressman [PRE87] :

> *"Se você atrasar a estimativa até o fim do projeto, ela será 100% precisa"*

Como esta alternativa é inviável resta ao usuário dos modelos de estimativa reconhecer suas limitações e aplicá-los com consciência e comedimento. A Seção 2.7 explica os passos que devem ser seguidos para realizar estimativas de custo de *software*.

## 2.4 PRINCIPAIS TÉCNICAS DE ESTIMATIVA

As principais técnicas de estimativa foram classificadas por Boehm, no seu conhecido livro *"SOFTWARE ENGINEERING ECONOMICS"* [BOE81] (resumido depois em artigo, [BOE84]), em:

A - MODELOS ALGORITMICOS: são modelos que fornecem um ou mais algorítmos que produzem a estimativa de custo do software como uma função das variáveis que, segundo o criador do modelo, mais influenciam o custo do projeto.

B - JULGAMENTO DE PERITO: esta técnica consiste em consultar um ou mais peritos (talvez usando um mecanismo de obtenção de consenso sobre as opiniões diversas). Para realizar as estimativas os peritos se baseiam na sua experiência em projetos e sistemas semelhantes, na intuição, e em analogia.

C - ANALOGIA: esta técnica consiste em estimar por analogia com um ou mais projetos concluidos que sejam semelhantes ao novo projeto que se deseja realizar.

D - PARKINSON: Esta técnica consiste simplesmente em igualar a estimativa de custo aos recursos disponíveis; ela se baseia no princípio de Parkinson *"o trabalho se expande para preencher o espaço disponível"*.

E - PREÇO PARA GANHAR: Esta técnica consiste simplesmente em realizar uma estimativa de custo necessária para se obter o projeto (ou um prazo necessário para se por um produto no mercado).

F - *"TOP-DOWN*[6]*"*: Esta técnica consiste em realizar uma estimativa total de custo a partir das propriedades globais do produto (*software*). O custo total é então repartido entre seus vários componentes.

---

[6] do inglês, de cima para baixo

G - "*BOTTOM-UP*[7]": Nesta técnica cada componente do projeto é estimada separadamente, e os resultados são agregados para produzir uma estimativa total de esforço.

*Tabela 2.1* Pontos Fortes e Fracos das Principais Técnicas de Estimativa

| | PONTOS FORTES | PONTOS FRACOS |
|---|---|---|
| MODELOS ALGORITMICOS | - objetivo, repetitivo, fórmulas analisáveis<br>- eficientes e bons para análises de sensitividade<br>- calibrado objetivamente pela experiência | - entradas subjetivas<br>- não considera circunstâncias excepcionais<br>- calibrado para o passado e não para o futuro |
| JULGAMENTO DE PERITO | - considera circunstâncias excepcionais<br>- é realizada com interação com as realidades do projeto | - polarizado e incompleto |
| ANALOGIA | - baseado em experiências representativas | - a representatividade das experiências pode não ser razoável |
| *TOP-DOWN* | - foca o nível de sistema (as atividades de integração, documentação e gerência são consideradas)<br>- é eficiente | - pouco detalhado<br>- pouco estável |
| *BOTTOM-UP* | - é mais detalhada<br>- é mais estável<br>- promove o envolvimento das pessoas que estão trabalhando no projeto | - pode ignorar custos a nível de sistema (de integração e testes)<br>- requer maior esforço |

As técnicas de PREÇO PARA GANHAR e PARKINSON são inaceitáveis pois não produzem resultados satisfatórios; entretanto, sua

[7] do inglês, de baixo para cima

citação neste trabalho evidencia que elas já foram usadas por muitos gerentes na tomada de decisões. A Tabela 2.1 apresenta os pontos fortes e fracos de cada uma das técnicas restantes.

Da Tabela 2.1 pode-se tirar as seguintes conclusões :

- Nenhuma das alternativas é melhor que as outras em todos os aspectos.
- Os pontos fortes e pontos fracos de algumas técnicas são complementares (notadamente MODELOS ALGORITMICOS x JULGAMENTO DE PERITO e *TOP-DOWN* x *BOTTOM-UP*).
- MODELOS ALGORITMICOS, ANALOGIA e JULGAMENTO DE PERITO podem ser aplicados em procedimentos *TOP-DOWN OU BOTTOM-UP* de estimativa.

Desta discussão se conclui que se deve combinar as técnicas citadas, comparar resultados e interagir onde eles diferem. Segundo Boehm [BOE81], uma combinação efetiva destas técnicas seria:

- Uma estimativa *TOP-DOWN* usando JULGAMENTO de mais de um PERITO, usando ANALOGIA quando um projeto comparável ao anterior estiver disponível.

- Uma estimativa *BOTTOM-UP* usando MODELOS ALGORITMICOS com as entradas e estimativas a nível de componentes realizadas pelo próprio pessoal que irá fazer o desenvolvimento.

- Comparação e interação entre as duas estimativas anteriores.

O *SISTEMA DE AUXÍLIO AO PLANEJAMENTO DE DESENVOLVIMENTO DE SOFTWARE* apresentado por este trabalho procurará reunir as abordagens acima. No Capítulo 4 desta tese apresentaremos este sistema e discutiremos sua aplicabilidade sob a ótica das idéias discutidas nesta seção.

## 2.5 MODELOS ALGORÍTMICOS DE ESTIMATIVA

Os modelos algorítmicos fornecem um ou mais algoritmos matemáticos que produzem estimativas de custo de *software* como função de uma série de variáveis consideradas como os fatores de maior influência no processo de desenvolvimento e manutenção do *software*. Os modelos algorítmicos são a melhor solução quando se quer automatizar o processo de estimativa de custo de *software*. O *SISTEMA DE AUXÍLIO AO PLANEJAMENTO DE DESENVOLVIMENTO DE SOFTWARE*, mostrado neste trabalho, tem como ponto de partida estes modelos, pois eles apresentam uma série de vantagens que são fundamentais na criação de um sistema deste tipo:

- Eles são objetivos e não são influenciáveis por fatores como desejo de vencer concorrências, desejo de satisfação pessoal, ou desgosto pelo projeto.

- Eles são repetíveis: pode-se realizar, para um dado projeto, uma estimativa hoje e outra daqui a uma semana que será obtido o mesmo resultdo.

- Eles são eficientes e capazes de suportar uma família de estimativas ou análises de sensitividade.

- Eles são objetivamente calibrados a partir de experiências anteriores.

Existem vários tipos de modelos algorítmicos; portanto é interessante classificá-los em categorias. A seguir apresentamos três das classificações feitas por conhecidos autores da área de engenharia de *software*.

Basili [BAS80] classificou os modelos em:

- *estáticos monovariáveis:* são os modelos onde as grandezas são estimadas através de fórmulas, onde elas são dependentes de uma outra variável, por exemplo:

$$Y = a\ X^b$$

- *estáticos multivariáveis:* são modelos onde as grandezas dependem de várias variáveis, por exemplo:

$$Y = a_1X_1 + a_2X_2 + a_3X_3 + \ldots + a_nX_n$$

- *dinâmicos multivariáveis:* são modelos onde as grandezas dependem da variável tempo permitindo, desta forma, que seus valores instantâneos possam ser estimados para todo o ciclo de vida do software, por exemplo:

$$Y = X_1.t.\exp(X_2.t)$$

- *modelos teóricos:* são modelos cujas fórmulas são baseadas em considerações teóricas sobre os padrões que os processos de criação e manutenção de *software* devem seguir. As considerações teóricas determinam diversos formatos de equações; portanto, esta classificação é aplicável junto com as outras três acima.

Conte, Dunsmore e Shen [CON85], classificaram os modelos algorítmicos em:

- *modelos histórico-experimentais:* são os modelos que se baseiam em dados históricos de projetos anteriores acompanhados; são tipicamente representados por modelos que usam matrizes de custo que relacionam produtividade (ou dificuldade) versus atividades e tarefas.

- *modelos estatísticos:* são modelos que usam análise através de regressão estatística de projetos prontos para derivar fórmulas de estimativas para os projetos futuros. Eles podem ser sub-divididos em lineares e não lineares, como mostram os exemplos abaixo:

$$Y = a_o + \sum_{i=1}^{n} a_i X_i \quad , \text{ linear}$$

$$Y = a + b.X^c \quad , \text{ não linear}$$

- *modelos teóricos:* estes modelos têm a mesma classificação dada por Basili.

- *modelos compostos:* estes modelos se caracterizam pela composição da abordagem algorítmica com o julgamento de especialistas. O julgamento especialista é aplicado na determinação de características do projeto, do produto e do pessoal, que serão utilizadas no cálculo de coeficientes de ajuste para as grandezas determinadas pela abordagem algorítmica.

Boehm [BOE81] classificou os modelos em:

- *modelos lineares:* são os modelos estatísticos lineares de [CON85].

- *modelos multiplicativos:* são os modelos não lineares da forma:

$$Y = a_0 . a_1^{X1} . a_2^{X2} \ \ldots \ a_n^{Xn}$$

- *modelos analíticos:* são todos os modelos não lineares diferentes da forma acima, por exemplo:

$$Y = \frac{\eta_1 . N_2 . N . \log_2 \eta}{2 . S . \eta_2}$$

- *modelos tabulares:* são modelos que usam tabelas, ou matrizes de custo, que relacionam os vários componentes do *software* com um custo por linha de código , ou com um multiplicador de ajuste de custo, de acordo com a classificação de cada um destes componentes em categorias de dificuldade e tipo. Podem ser incluídos na classificação de modelo histórico-experimental de [CON85].

- *modelos compostos:* classificação idêntica à de [CON85]

Como se pode notar, as classificações acima são bastante semelhantes, e as três foram citadas porque são, em certo grau, complementares. Dentro destas classificações podemos realizar as seguintes observações:

-os primeiros modelos algorítmicos eram lineares, estáticos e multivariáveis ; com fórmulas do tipo : $Y = a_1 X_1 + \ \ldots \ + a_n X_n$

( [NEL66], [FAR65] ) ; entretanto eles não conseguiam boas previsões evidenciando as muitas interações não lineares entre as variáveis presentes no desenvolvimento de *software*.

- os modelos não lineares, estáticos monovariáveis e multiplicativos, ou seja, com fórmulas do tipo: $Y=a.X^b$ , estão entre os mais divulgados na literatura ([WAL77], [HER77], [BAI81],[BOE81]) ; eles geralmente calculam esforço de desenvolvimento a partir do número de linhas de código (LOC) com uma fórmula do tipo mostrado acima.

- entre outros tipos de modelos não lineares podemos citar:o de Halstead ([HAL77], [SCH78]) que, como já comentado usa uma métrica difícil de se estimar nas fases iniciais de projeto; e o de Putnam que é também um modelo teórico, sendo bastante divulgado na literatura ([PUT78a], [PUT78b], [PUT79a], [PUT79b], [PUT79c], [PUT80]).

- os modelos tabulares (histórico-experimentais) são fáceis de entender, implementar e modificar; todavia, eles são de uso complicado dada a presença de grande número de variáveis que são usadas nas tabelas ou matrizes (este tipo de modelo tem pouca sensibilidade quando se usam poucas variáveis). O grande número de variáveis e tabelas torna o modelo difícil de calibrar, pois isto implica a necessidade de grandes bases de dados sobre projetos completados e este é um dos problemas críticos neste campo de estudo. O mais conhecido modelo tabular é o de Wolverton [WOL74].

- os modelos compostos incorporam combinação de vários dos modelos citados acima para realizar estimativas. Eles são os modelos realmente utilizados na atualidade. Os modelos apresentados por Putnam [PUT80] e Boehm [BOE81], por exemplo, são na realidade modelos compostos onde se usam equações analíticas, regressão estatística e julgamento perito para realizar as estimativas.

No Capítulo 3 são discutidos os principais modelos e as ferramentas automatizadas de estimativas de custo de *software*, e procurar-se-á enquadrá-los nas categorias apresentadas até aqui.

## 2.6 AVALIAÇÃO DE ESTIMATIVAS

Nesta seção serão discutidos primeiramente quais critérios são importantes na avaliação dos modelos de estimativa; a seguir apresentamos algumas medidas importantes que podem ser realizadas para esta avaliação; finalizando discutimos a validade de cada uma delas, bem como o intervalo de valores que elas devem ter para que um modelo seja considerado bom.

### 2.6.1 Critérios que Devem ser Usados na Avaliação

Para discutir os modelos devem-se estabelecer critérios para julgar seus méritos e fraquezas. A seguir apresentamos os critérios que Conte, Shen e Dunsmore [CON85] consideram úteis para este julgamento:

Critério 1 - Validade

- O modelo fornece boas estimativas pelo menos na base de dados usada na sua validação?
- O modelo fornece o grau de confiabilidade das estimativas ?
- O modelo é aplicável ao projeto sob consideração ?

Critério 2 - Objetividade

- As estimativas são baseados em medidas e dados que são reproduzíveis (e podem ser determinados) ?
- As estimativas dependem de fatores subjetivos que podem variar significativamente de acordo com o usuário do modelo ?

Critério 3 - Facilidade de uso

- Os dados necessários ao modelo são fáceis de obter ?
- São necessários muitos dados ?
- O esforço necessário para obter os dados para aplicação do modelo é excessivamente alto ?
- A informação necessária à aplicação do modelo está

disponível cedo no ciclo de vida do *software* ?

...Critério 4 - Sensitividade

- Uma pequena mudança em um dos parâmetros de entrada pode conduzir a uma mudança grande na estimativa do modelo ?

...Critério 5 - Transportabilidade

- O modelo é independente de dados locais que não podem ser obtidos em ambientes de desenvolvimento diferentes ?

### 2.6.2 Medidas para Avaliação dos Modelos de Estimativa

Mostraremos nesta seção as principais medidas para avaliar as estimativas dos modelos; esta avaliação se encaixa no critério *validade* discutido acima. Neste ítem será usada a letra "E" para representar o esforço real de desenvolvimento de *software* e "$\bar{E}$" para representar o esforço estimado (vide Seção 2.2.1) .

a - Erro Relativo - "ER"

Dados: um projeto de *software* que necessitou de $E$ pessoas-mês para se completar e um modelo de estimativa que previu um esforço $\bar{E}$ para realizar este projeto , define-se o erro relativo $ER$ desta estimativa como:

$$ER = \frac{E - \bar{E}}{E} \qquad \text{(eq. 2.4)}$$

b - Magnitude do Erro Relativo - "MER"

Define-se magnitude do erro relativo como:

$$MER = |ER| = \left|\frac{E - \bar{E}}{E}\right| \qquad \text{(eq. 2.5)}$$

c - Magnitude Média do Erro Relativo - "$\overline{MER}$"

Se um modelo de estimativa for usado em $n$ projetos, $P_1, P_2, P_3, \ldots, P_n$ , então a magnitude média do erro relativo, será dada por:

$$\overline{MER} = \frac{1}{n} \cdot \sum_{i=1}^{n} MER_i$$ , onde $MER_i$ é a magnitude do erro relativo do i-ésimo projeto (eq. 2.6)

Se $\overline{MER}$ é pequeno, então o modelo produz na média um bom cunjunto de previsões.

d - Previsão Média a Nível $r$ - "PREV(r)"

Dados $n$ projetos para os quais se realizaram estimativas com um determinado modelo; se em $K$ destes $n$ projetos tivemos MER $\leq r$ , dizemos então que a previsão média do modelo a nível $r$ é igual a $K/n$ ; ou seja:

$$PREV(r) = K/n \qquad (eq. 2.7)$$

Por exemplo, se PREV(0,25)=0,77 , então 77% dos valores estimados cairam dentro da faixa : *valor real* ± 25%

e - Erro Quadrático Médio Relativo - "$\overline{EQR}$"

Dados $n$ projetos, define-se:

o esforço médio nestes $n$ projetos como:

$$\overline{E} = \frac{1}{n} \cdot \sum_{i=1}^{n} E_i \qquad (eq. 2.8)$$

o erro quadrático relativo como:

$$EQR = \left[ \frac{1}{n} \cdot \sum_{i=1}^{n} (E_i - \bar{E}_i)^2 \right]^{1/2} \qquad \text{(eq. 2.9)}$$

e o erro quadrático médio relativo como:

$$\overline{EQR} = EQR/\bar{E} \qquad \text{(eq. 2.10)}$$

### 2.6.3 Como Avaliar os Modelos com Estas Medidas

Das medidas de avaliação de modelos as principais são $\overline{EQR}$, PRED(r) e $\overline{MRE}$. A seguir se discute quais são os valores aceitáveis destas medidas em um modelo aplicado para estimativa de um determinado conjunto de projetos.

a - $\overline{MER}$ - para um modelo de estimativa considera-se $\overline{MER} \leq 0,20$ como um valor aceitável na previsão de esforço de desenvolvimento. É importante notar que $\overline{MER}$ pequeno não implica que todas as estimativas foram boas, mas sim, que a maioria foi.

b - PREV(r) - o critério considerado aceitável para a previsão a nível $r$ em um modelo é $PREV(0,25) \geq 0,75$ . Note que não há limite para as MRE dos 25% outros projetos.

c - $\overline{EQR}$ - o critério considerado aceitável é $\overline{EQR} \leq 0,25$

Em [CON85] discute-se o valor de cada medida aplicando um modelo a oito conjuntos de dados que contêm ao todo 187 projetos concluídos catalogados; o trabalho conclui que $\overline{EQR} \leq 0,25$ é o critério mais difícil de ser cumprido, particularmente quando os

modelos são aplicados a conjuntos de projetos de características heterogêneas. As outras duas medidas , PREV(r) e $\overline{MER}$, apresentam um comportamento semelhante entre si, avaliando as estimativas dos modelos em aceitáveis ou não (segundo os critérios estabelecidos acima) para os mesmos conjuntos de dados. O artigo acaba por concluir que a satisfação simultânea dos critérios $\overline{MER} \leq 0,20$ e $PRED(0,25) \geq 0,75$ é uma boa forma para classificar um modelo como bom. No nosso trabalho acrescentaremos ainda o critério $PREV(0,50) \geq 0,90$ para evitar que o modelo permita em muitos projetos estimativas por demais discrepantes da média.

É importante ressaltar que os métodos atuais mal conseguem cumprir estes critérios quando aplicados por especialistas a projetos semelhantes a aqueles que foram usados para calibrá-los.

## 2.7 REALIZANDO ESTIMATIVAS

O processo de realização de estimativas de esforço e tempo de desenvolvimento de projetos de sistemas de *software* deve ser encarado como um miniprojeto que deve ser planejado, revisado e acompanhado. Boehm [BOE81] dividiu este processo em sete passos básicos:

1 - Estabeler os objetivos
2 - Planejar quais dados e recursos são necessários
3 - Estabelecer os requisitos do *software*
4 - Detalhar o projeto o máximo possível
5 - Aplicar diferentes técnicas de estimativa
6 - Comparar e interagir estimativas
7 - Acompanhar estimativa X projeto real

As seções 2.7.1 a 2.7.7 discutem cada um desses passos.

### 2.7.1 Estabelecer os Objetivos

Em estimativas de custos uma grande quantidade de esforço pode ser desperdiçada para se obter informações e realizar estimativas em ítens que não têm relevância no custo final do projeto. Desta forma, é extremamente importante estabelecer os objetivos da estimativa de custo de *software* como o primeiro passo do processo; estes objetivos definirão então o nível de detalhe e o esforço requerido para realizar os outros passos da estimativa.

O principal fator que comandará o estabelecimento dos objetivos da estimativa será a fase do projeto; ela está diretamente relacionada com o nosso conhecimento do *software* cujos custos queremos estimar. A Figura 2.4 na seção 2.3.4 ilustra a precisão com que a estimativa de custo de *software* pode ser feita de acordo com a fase do projeto. Nesta figura pode ver visualizado que ao final das especificações de requisitos o nosso grau de incerteza intrínseco sobre o custo do projeto é de 1.5 vezes e, ao final das especificações do projeto é de 1.25 vezes. A conclusão que se tira destas incertezas é que não adianta querer realizar estimativas com detalhes e precisão a nível de final de

especificacões de projeto, se ainda estamos no começo das especificações de requisitos. Outra conclusão importante é que pode-se combinar estimativas de componentes do *software* em diferentes fases de desenvolvimento desde que suas incertezas tenham magnitudes semelhantes. O grau da incerteza é fundamental na determinação de quais componentes estimados são realmente críticos e devem ser considerados naquela fase do projeto; por exemplo: se um dos módulos do *software* tem custo estimado de 1000 unidades monetárias e está na fase de estudos de viabilidade seus custos podem variar de até 3 vezes ,ou seja de 333 a 3000 u.m.; frente a um módulo cujo custo que se estima em 200 u.m. e está na fase de final de projeto detalhado, o primeiro módulo é muito mais crítico e porisso determinará toda a incerteza da estimativa.

Outro ponto importante é a participação relativa de uma estimativa na tomada de decisão. Considere o exemplo onde se quer decidir entre comprar ou desenvolver um conjunto de softwares: se é sabido de antemão que alguns destes softwares irão ter custos da mesma ordem de magnitude quer sejam desenvolvidos ou comprados, a princípio será perda de tempo estimar os custos destes *softwares* para a tomada de decisão, já que sua influência nela será mínima.

É fundamental, ainda, realizar estimativas pessimistas e otimistas (principalmente em tomada de decisões) e fornecer para cada uma delas o grau de incerteza associado. Estes limites são muito importantes pois permitem visualizar-se o risco associado às decisões que serão tomadas.

Os pontos principais da discussão desta seção podem ser resumidos em:

- Estabeleça os objetivos da estimativa para as informações necessárias para as tomadas de decisão.
- Determine os objetivos de precisão da estimativa a partir dos vários componentes estimados do sistema.
- Reexamine os objetivos das estimativas à medida que o processo de desenvolvimento prossegue e modifique-os sempre que necessário.

### 2.7.2 Planejar Quais Dados e Recursos são Necessários

No começo desta seção afirmou-se que a estimativa de custo de *software* deve ser encarada como um mini-projeto. Como um mini-projeto a estimativa de custo de *software* deve ter um projeto; abaixo mostramos um esboço de um plano simples e de caráter genérico que se aplica muito bem a estimativas de custos:

1 - *Finalidade. Porquê esta estimativa está sendo feita ?*
2 - *Produtos e Cronogramas. O que está sendo fornecido e quando será fornecido ?*
3 - *Responsabilidades. Quem é responsável por cada produto ? Onde eles farão o trabalho (geográficamente e organizacionalmente) ?*
4 - *Procedimentos. Como o trabalho será feito ? Quais ferramentas e técnicas de estimativa serão usadas ?*
5 - *Recursos Necessários. Quanto de dados, tempo, dinheiro e esforço é necessário para realizar a estimativa ?*
6 - *Considerações. Sob quais condições as estimativas podem ser consideradas válidas?*

### 2.7.3 Estabelecer os Requisitos do Software

Se não se sabe que produto está se construindo, certamente não poder-se-á estimar bem o custo de construi-lo. Isto implica que as especificações do *software* devem ser estabelecidas logo que possível. A melhor forma de se saber o quão estimável é o custo de um *software* especificado é saber-se o quão testável ele é [BOEB1]. Uma especificação é testável se pode-se definir uma clara rotina de testes (do tipo passa ou falha) para determinar se o *software* desenvolvido irá satisfazer as especificações. É fundamental que estas especificações não sejam vagas ou ambíguas; por exemplo: especificações como "As respostas para requisições de serviços das estações A, B e C devem ser respondidas em tempo real", devem ser substituídas por especificações do tipo "As respostas para requisições de serviços das estações A, B e C devem ser respondidas em menos de 12, 18 e 25 segundos, respectivamente".

Pode-se argumentar que estas especificações exigem muito esforço, mas tal esforço é valioso pelos seguintes motivos:

- Ele teria que ser empregado na fase de testes de qualquer forma.
- Fazendo este trabalho cedo, eventualmente eliminam-se desgastes posteriores em controvérsias e desilusões.
- E, como argumenta-se aqui, pode-se fazer estimativas mais precisas e, consequentemente, planejar melhor o desenvolvimento.

Nos casos em que for impossível assegurar que os requisitos do *software* sejam testáveis, mais esforço será necessário para se realizar estimativas. Nestes casos deve-se documentar todas as considerações assumidas antes de se estimar; em particular se elas forem pessimistas ou otimistas.

### 2.7.4 Detalhar o Projeto o Máximo Possível

Em geral quanto mais detalhes são considerados em uma estimativa , mais precisa ela será, por três razões principais:

1 - Quanto mais detalhes de projeto são explorados melhor se compreende o *software* a ser desenvolvido; consequentemente menor é a incerteza da estimativa, Figura 2.4.

2 - Quanto mais pedaços de *software* são estimados mais reduzida é a variância da estimativa total. Isto se deve ao fato de que os erros tendem a ser compensados.

3 - Quanto mais se pensa sobre as funções que o *software* deve realizar , menos provável se torna o esquecimento de computar o custo de algum componente do sistema.

É importante notar que no *detalhar o máximo possível* sejam considerados os objetivos do *software* discutidos na Seção 2.7.1, ou seja, deve-se considerar os objetivos da estimativa para que não se desperdice esforço em detalhes não relevantes.

### 2.7.5 Aplicar Diferentes Técnicas de Estimativas

Este passo consiste em aplicar várias técnicas de estimativas e combinar seus resultados de forma a evitar suas fraquezas e aproveitar suas qualidades. A Seção 2.4 discute as principais técnicas de estimativas e apresenta uma sugestão de como combinar estas técnicas, porisso não rediscutiremos estes ítens aqui.

### 2.7.6 Comparar e Interagir Estimativas

Um dos principais motivos de se usar diversas técnicas independentes de estimativa de custo é que esta abordagem permite a análise das diferentes componentes do custo e suas influências nas diferenças das estimativas. Desta forma, se uma estimativa *Bottom-up* e uma *Top-down* fornecem valores diferentes pode-se analisar as razões das diferenças, identificar os principais componentes de custo de cada uma das técnicas, estabelecer as diferenças e reestimar o custo do *software*. Como exemplo, considere que uma estimativa *Bottom-up* que forneça um custo de 3 milhões de unidades monetárias e uma outra estimativa *Top-down* que forneça um custo de 5 milhões de unidades monetárias para o mesmo projeto; ao analisar em detalhe as duas estimativas pode-se concluir que aquela que usou a técnica *Bottom-up* não considerou atividades a nível de sistema, tais como, integração, gerência de configuração e controle de qualidade; por outro lado, aquela que usou a técnica *Top-down* não considerou alguns componentes do *software* que a técnica *Bottom-up* considerou por ser mais detalhada; desta forma poder-se-ia concluir que a estimativa mais realista seria 6 milhões de unidades monetárias ao invés de um valor de compromisso entre 4 e 5 milhões.

Outros dois motivos importantes para comparar e interagir diferentes técnicas de estimação são os fenômenos do "otimismo / pessimismo" e dos "componentes principais". Eles são discutidos a seguir:

- Pessoas envolvidas em um projeto tendem a realizar estimativas muito otimistas ou pessimistas conforme sua personalidade, suas funções ou seus incentivos dentro desse projeto. Por exemplo, um gerente de proposta comercial tende a ser muito mais otimista que um gerente de projeto.

- Um *software* normalmente contém componentes que são responsáveis pela maioria do custo (80% dos custos estão contidos em 20% dos componentes); isto faz com que as estimativas destes componentes devam ser repetidas e interagidas com um detalhe maior. Nestas reestimativas é importante considerar que estes componentes precisam ser especialmente reclassificados no aspecto complexidade (um ítem que é considerado em todos os modelos de estimativas que serão usados neste trabalho); isto porque as pessoas tendem a igualar a complexidade global com a complexidade das pequenas partes mais difíceis do componente, fazendo com que alguns dos grandes componentes do *software* sejam classificados erroneamente no ítem complexidade.

Por causa destes fenômenos é importante fazer com que certos componentes do *software* sejam estimados mais de uma vez por pessoas diferentes.

### 2.7.7 Acompanhar Estimativa X Projeto Real

Uma vez iniciado o projeto é fundamental obter os dados reais de seus custos e andamento para compará-los com as estimativas. São várias as razões para isso:

1 - As grandezas de entrada em modelos de estimativa são incertas (estimativas de tamanho, caracterização do projeto, etc). Se durante o projeto aparecem diferenças entre o custo estimado e o real, e estas diferenças podem ser explicadas por um conhecimento mais preciso das grandezas de entrada, então devemos atualizar a estimativa de custo com os valores mais precisos destas grandezas.

2 - As técnicas de estimativas de custo de *software* estão ainda em estágios primários de desenvolvimento. Para a evolução destas técnicas será necessário comparar valores reais com os estimados e usar os resultados para aprimorá-las.

3 - Muitas técnicas de projeto não são compatíveis com certos modelos de estimativas (ou vice-versa); por exemplo, o

desenvolvimento incremental. Neste caso, tanto a interação de curto prazo da gerência com as estimativas para obtenção de resultados razoáveis, como as interações a longo prazo visando adaptação dos modelos a este tipo de projeto, são fundamentais.

4 - Os projetos de *software* muitas vezes são voláteis, isto é, componentes são adicionados, retirados, alterados ou combinados durante o desenvolvimento. Assim, os gerentes de projeto precisam identificar as mudanças e rever as estimativas sempre que necessário.

5 - *Software* é um campo em franca evolução; como os modelos de estimativa são calibrados através de projetos passados , é fundamental acompanhar projetos no presente para constantemente incorporar novas tendências e características do desenvolvimento do *software* aos modelos de estimativas.

# CAPÍTULO 3

# PRINCIPAIS MODELOS DE ESTIMATIVA

Este capítulo é dedicado aos modelos de estimativa. Os modelos de estimativa formam a base deste trabalho e, porisso, devem ser bem conhecidos antes que se façam considerações mais profundas sobre a arquitetura do sistema que se pretende implementar.

Este capítulo pretende apresentar os modelos de estimativa, discutir o seu uso, discutir suas implementações, e mostrar quais modelos foram selecionados para serem inicialmente implementados no SAPDES. O capítulo está dividido em quatro seções principais:

- A primeira parte mostra a evolução dos modelos de estimativa através da apresentação em ordem cronológica dos pricipais modelos de estimativas surgidos a partir de 1965. Cada modelo apresentado é criticado e comparado com os anteriores para ressaltar os pontos onde ocorreu evolução.

- Na segunda parte discutem-se os critérios de seleção dos modelos de estimativa a serem inicialmente implantados no SAPDES e apresentam-se os modelos selecionados.

- Na terceira parte discutem-se algumas ferramentas de estimativa de custo de *software* famosas. Procura-se principalmente mostrar como elas se basearam nos modelos discutidos para serem implementadas.

- Finalmente na quarta parte discute-se como estabeler a contagem das métricas básicas usadas pelos modelos (*function points* e LOC). Ressalta-se a importância destas métricas para a realização de boas estimativas e mostram-se algumas técnicas para realizar as contagens destas métricas.

## 3.1 PRINCIPAIS MODELOS NA LITERATURA

Desde os primeiros dias do estudo do *software* as pessoas tentam desenvolver modelos algorítmicos de estimativa de custo. As primeiras tentativas eram bastante simplistas, por exemplo: "Em um grande projeto cada programador irá produzir uma instrução por homem-hora (aproximadamente 150 instruções por homem-mês)" [BOE84]. Em 1964, a Força Aérea Americana contratou uma empresa (a *System Development Corporation*) para desenvolver um projeto histórico no campo da estimativa de custo de *software*; este empreendimento coletou dados de 169 projetos de *software* e analisou estatisticamente a influência de 104 atributos dos projetos nos seus custos finais. Este estudo resultou em um dos primeiros modelos de estimativa de custo de *software*, o modelo de custo da *SDC* [NEL66]. Desde então já foram apresentados mais de 20 destes modelos. A seguir listamos os modelos em ordem cronológica de publicação e discutimos brevemente os principais deles:

### 3.1.1 Modelo da SDC

Este modelo usava 14 parâmetros (obtidos a partir dos 104 atributos de projetos considerados originalmente) para determinar através de uma equação linear o esforço de construção do *software*. O esforço era calculado através da fórmula 3.1 dada na página seguinte. Quando aplicado à sua própria base de dados de 169 projetos, este modelo forneceu um valor médio de 40 pessoas-mês e um desvio padrão de 62 pessoas-mês, um resultado que evidencia as grandes discrepâncias entre o valor estimado e o real. O modelo é também não intuitivo pois se todos índices tiverem contagem zero o esforço estimado será -33 homens-mês. A conclusão que se tira é que existem muitas não linearidades no processo de desenvolvimento de *software* para que modelos lineares funcionem bem.

$$\overline{E} = -33.63 + 9.15x_1 + 10.73x_2 + 0.51x_3 + 0.46x_4 + 0.40x_5 + 7.28x_6 - 21.45x_7 + 13.53x_8 + 12.35x_9 + 58.82x_{10} + 30.61x_{11} + 29.55x_{12} + 0.54x_{13} - 25.20x_{14}$$

(eq. 3.1)

onde: $x_1$ = falta de requisitos,
este índice é classificado numa escala de 0 a 2

$x_2$ = estabilidade do projeto,
este índice é classificado numa escala de 0 a 3

$x_3$ = percentagem de instruções matemáticas,
este índice é dado no valor real em percentagem

$x_4$ = percentagem de instruções de I/O[1],
dado no valor real em percentagem

$x_5$ = número de subprogramas,
dado no valor real

$x_6$ = linguagem de programação,
este índice é classificado numa escala de 0 a 1

$x_7$ = aplicação comercial,
este índice é classificado numa escala de 0 a 1

$x_8$ = programa independente,
este índice é classificado numa escala de 0 a 1

$x_9$ = primeiro programa no computador,
este índice é 0 ou 1 conforme seja verdadeiro ou falso

$x_{10}$ = desenvolvimento paralelo de *hardware*,
este índice é 0 ou 1

$x_{11}$ = usado equipamento de acesso aleatório,
este índice é 0 ou 1

$x_{12}$ = computador de desenvolvimento diferente do computador de destino,
este índice é 0 ou 1

$x_{13}$ = número de erros pessoais,
valor médio do número de erros por pessoa

$x_{14}$ = desenvolvimento por organização militar,
este índice é classificado numa escala de 0 a 1

---

1 instruções de entrada e saída, do inglês, input/output

### 3.1.2 Modelo de Farr-Zagorski

Este modelo também é linear, foi publicado na mesma época do modelo anterior [FAR65], e utiliza 13 parâmetros para o cálculo do esforço. Ele pode aplicar três equações diferentes a partir de uma classificação a priori da complexidade do projeto. Os parâmetros que mais têm influência nessas equações são o número de instruções entregues, tipos de documentação para uso interno e externo, e o número de palavras na base de dados.

A seguir apresentamos uma destas equações:

$$\overline{E} = -188 + 2,86x_1 + 2,3x_2 + 33x_3 - 17x_4 + 10x_5 + x_6 \qquad (eq\ 3.2)$$

onde: $x_1$ = número de instruções , em milhares
$x_2$ = número de milhas viajadas , em milhares
$x_3$ = número de tipos de documentos entregues
$x_4$ = experiência dos programadores , em anos
$x_5$ = número de terminais de vídeo
$x_6$ = percentagem de novas instruções , em notação decimal

Note que, como a equação 3.2 tem uma constante negativa grande (-188), este modelo fornece estimativas completamente errôneas para *softwares* de pequeno e médio porte.

### 3.1.3 Modelo de Aron

Em seu trabalho [ARO69] Aron observou que a demanda por força de trabalho na construção de grandes sistemas aumenta gradualmente, alcança um pico e então decai até zero. O pico é alcançado em torno da fase de teste. Esta descrição corresponde às curvas de ciclo de vida de *software* apresentadas depois em outros modelos (vide dicussão na Seção 2.2.3).

Aron discutiu quatro métodos para estimativa de custo de *sofware*: os métodos da experiência, das restrições, das unidades de trabalho e o quantitativo. O método da experiência corresponde a aplicação de julgamento perito e analogia para se estimar o custo do projeto (vide Seção 2.4). O método das restrições é equivalente a uma suposição educada, o gerente concorda em fazer o projeto sob dadas restrições , desta forma este método se assemelha aos de custo para vencer e *parkinson*, discutidos na Seção 2.4 . No método das unidades de trabalho, o projeto é decomposto em unidades menores que são estimadas a partir de experiências prévias com unidades semelhantes: este método é portanto uma combinação das abordagens *bottom-up* com a analogia e julgamento perito. O método que realmente nos interessa é o quantitativo, ele é bastante simples. O projeto é dividido em pequenas tarefas e o tamanho destas tarefas é estimado em linhas de código. Cada uma destas tarefas é também classificada em fácil, média ou difícil, baseada nas suas interações e dependências de outras tarefas. A seguir assume-se valores de produtividade de 500 , 250 , e 125 linhas de código por pessoa-mês para as tarefas fáceis, médias ou difíceis, respectivamente. Para calcular o esforço total basta então se calcular o esforço de cada tarefa, multiplicado-se o tamanho desta pela produtividade, e somar os sub-totais. Note que além de bastante simplista (podem existir vários outros índices de produtividade), este modelo usa apenas uma abordagem *bottom-up* desconsiderando todos os esforços a nível de sistema (integração, teste de conjunto, gerência, etc).

### 3.1.4 Modelo de Wolverton / TRW

Este modelo [WOL74] assume que o esforço total de se produzir um programa é linearmente proporcional ao número de linhas de código a serem produzidas. O modelo usa uma matriz histórica de custo por instrução, organizada por categoria dos módulos do *software* (controle, I/O, pré/pós-processador, algoritmo, gerência de dados e tempo crítico) e pelo grau de dificuldade destes módulos (módulo velho: fácil, médio, difícil ; módulo novo: fácil, médio, difícil).

Para se estimar identificam-se os módulos que compõem o *software* de acordo com as categorias acima; a seguir classifica-se o grau de dificuldade de cada módulo, e na matriz de custo encontra-se o custo de cada instrução destes módulos. Com o custo por instrução de um módulo basta estimar-se o número de instruções deste módulo e encontrar-se o custo total do módulo por multiplicação. O custo total do sistema será obtido fazendo-se o somatório dos custos de todos os módulos. A seguir é mostrado um exemplo da matriz de custo e de seu uso:

*Tabela 3.1* Matriz de Custo de Wolverton

| CATEGORIA DO MODULO | NÍVEL DE DIFICULDADE | | | | | |
|---|---|---|---|---|---|---|
| | velho fácil 1 | velho médio 2 | velho difícil 3 | novo fácil 4 | novo médio 5 | novo difícil 6 |
| 1 controle | 21 | 27 | 30 | 33 | 40 | 49 |
| 2 I/O | 17 | 24 | 27 | 28 | 35 | 43 |
| 3 pré/pos processador | 16 | 23 | 26 | 28 | 34 | 42 |
| 4 algoritmo | 15 | 20 | 22 | 25 | 30 | 35 |
| 5 gerência de dados | 24 | 31 | 35 | 37 | 46 | 57 |
| 6 tempo crítico | 75 | 75 | 75 | 75 | 75 | 75 |

Suponha que um dado módulo $k$ tenha categoria $i(k)$ e dificuldade $j(k)$ ; se o seu tamanho estimado é $L(k)$ linhas de

código, então o custo deste módulo é:

$$C(k) = L(k).C_{i(k),j(k)} \qquad \text{(eq. 3.3)}$$

onde: $C_{i(k),j(k)}$ é o custo por instrução obtido da matriz acima

O custo total é dado por:

$$CUSTO\ TOTAL = \sum_{\text{módulos}} C(k) \qquad \text{(eq. 3.4)}$$

Este modelo, apesar de sua grande objetividade, tem alguns problemas graves:

a - o primeiro deles é não considerar os custos a nível de sistema.

b - o segundo é que a matriz de custo deve ser continuamente revisada para refletir os custos atuais.

c - o terceiro, e mais grave, é que se quisermos levar em conta os muitos outros fatores que influenciam no custo (capacidade do pessoal de desenvolvimento, uso de práticas modernas de programação, etc) seremos forçados a aumentar brutalmente o tamanho da matriz de custo, tornando-a virtualmente impossível de calibrar e manusear. Por exemplo, se acrescentamos cinco novos fatores de influência que sejam também classificados em seis categorias, teríamos necessidade de $6^7$, ou 279936, números na matriz de custo.

### 3.1.5 Modelo de DOTY

Eeste modelo [HER77] foi resultado de extensa análise dos dados disponíveis na época sobre projetos, inclusive muitos dos dados da *SDC*. Ele estima força de trabalho, custo e tempo de desenvolvimento de um projeto de *software*. O tamanho do sistema proposto é estimado por analogia com outros projetos. Este modelo observa que o mesmo tempo é necessário para escrever um dado número de instruções em uma linguagem de alto nível ou de baixo nível. A diferença está no fato de que com uma linguagem de alto nível mais coisas são ditas (ou seja, são necessárias menos instruções para tratar o mesmo problema) e se ganha em clareza e facilidade de manutenção.

Este modelo pode ser dividido em vários sub-modelos de formas similares, cada um deles desenvolvido para uma área específica de aplicação. Um exemplo de uma destas formas (desenvolvida para aplicações gerais) é:

$$E = 5{,}288(kLOC)^{1{,}047} \quad , \text{para } kLOC \geq 10 \qquad (eq.\ 3.5)$$

$$E = 2{,}060(kLOC)^{1{,}047}\left(\prod_{j=1}^{14} f_j\right) \quad , \text{para } kLOC < 10 \qquad (eq.\ 3.6)$$

onde: kLOC significa milhares de linhas de código
E significa esforço em homens-mês
$f_j$ são fatores de ajustes do esforço estimado

Entre os modelos mostrados até agora este é o primeiro que apresenta características dos modelos atuais. O uso de fatores de ajuste torna o modelo composto, ou seja, ele inclui, além do próprio modelo algorítmico, a necessidade de julgamento perito nas respostas para classificação dos fatores de ajuste. É importante notar que os fatores de ajuste só são usados em *softwares* de pequeno porte, evidenciando que as estimativas para estes projetos são sujeitas a maiores influências de fatores externos (em projetos maiores valores médios de produtividade tendem a ser mais estáveis); entretanto o método é falho quando considera que estes fatores também não influenciam significativamente projetos de

maior porte; em particular, a utilização destas duas fórmulas cria um ponto de instabilidade já que a curva de esforço estimado é descontínua quando o tamanho do projeto é 10 kLOC.

Os valores $f_j$ são determinados a partir da classificação de 14 fatores que influenciam o custo; estes fatores (ou variáveis) de influência estão listados na Tabela 3.2 .

*Tabela 3.2* Fatores de Ajuste de Esforço Estimado nos Modelos do DOTY

| *fator* | $f_j$ | sim | não |
|---|---|---|---|
| saída em vídeo especial | $f_1$ | 1,11 | 1,00 |
| definição detalhada dos requisitos de operação | $f_2$ | 1,00 | 1,11 |
| mudanças nos requisitos de operação | $f_3$ | 1,05 | 1,00 |
| operação em tempo real | $f_4$ | 1,33 | 1,00 |
| restrições de memória principal | $f_5$ | 1,43 | 1,00 |
| restrições de tempo de processamento | $f_6$ | 1,33 | 1,00 |
| primeiro *software* desenvolvido na UCP | $f_7$ | 1,92 | 1,00 |
| desenvolvimento concorrente de *hardware* | $f_8$ | 1,82 | 1,00 |
| desenvolvimento em tempo compartilhado | $f_9$ | 0,83 | 1,00 |
| usuário não está habituado com a máquina | $f_{10}$ | 1,43 | 1,00 |
| desenvolvimento no local de funcionamento da máquina | $f_{11}$ | 1,39 | 1,00 |
| comp. de desenvolvimento diferente do computador de destino do sistema | $f_{12}$ | 1,25 | 1,00 |
| desenvolvimento em mais de um local | $f_{13}$ | 1,25 | 1,00 |
| acesso do programador ao computador limitado | $f_{14}$ | 1,00 | 0,90 |

### 3.1.6 Modelo de Walston & Felix

Este trabalho [WAL77] é mais conhecido não pelo modelo de estimativa que apresenta mas pelo estudo sobre a influência de vários fatores na produtividade final do software. Este trabalho inicialmente apresenta uma série de fórmulas não lineares de uma variável (vide Seção 2.5) que relacionam importantes grandezas do processo de desenvolvimento de *software*:

$$E = 5{,}2.kLOC^{0{,}91} \qquad (eq.\ 3.7)$$

$$D = 49.kLOC^{1{,}01} \qquad (eq.\ 3.8)$$

$$td = 4{,}1.kLOC^{0{,}36} \qquad (eq.\ 3.9)$$

$$td = 2{,}47.E^{0{,}35} \qquad (eq.\ 3.10)$$

$$S = 0{,}54.E^{0{,}6} \qquad (eq.\ 3.11)$$

$$C = 1{,}84.kLOC^{0{,}96} \qquad (eq.\ 3.12)$$

$$C = 1{,}1.E^{0{,}81} \qquad (eq.\ 3.13)$$

onde: E = esforço total em pessoas-mês
kLOC = milhares de linhas de código fonte entregues
D = páginas de documentação
M = duração em meses
S = número médio de pessoas no grupo
C = custo de computação

A primeira das fórmulas acima pode ser usada para estimativa, contudo ela não fornece resultados satisfatórios nem na sua própria base de dados de validação; isto se deve à heterogeneidade dos projetos catalogados, com produtividades variando de 30 a 500 LOC por pessoa-mês. Como as formulas sozinhas não eram suficientes para realizar boas estimativas, era necessário identificar quais eram as variáveis que mais tinham influenciado na produtividade ao final dos projetos.

Walston e Felix começaram com 68 variáveis, mas concluiram que deveriam restringir os estudos a 29 delas, pois eram as mais significativas. Foi pedido que os gerentes de projetos classificassem, para um dado projeto, se a presença de cada uma destas variáveis era acima do normal, normal ou abaixo do normal. Fazendo a média das classificações de cada variável obteve-se a produti-

vidade média em cada uma das três classificações das 29 variáveis. Baseados nestes valores Walston & Felix desenvolveram uma metodologia para estimativa da produtividade, em linhas de código por pessoas-mês, de um projeto (com a estimativa de produtividade e do tamanho do projeto em linhas de código pode-se calcular o esforço esperado). A metodologia baseia-se nas variações entre a produtividade média, máxima e mínima de cada variável, aqui representada por $\Delta$ . A partir dos $\Delta i$ de cada uma das $i$ variáveis estima-se o índice de produtividade:

$$I = \sum_{i=1}^{29} W_i \cdot x_i \qquad \text{(eq. 3.14)}$$

onde: $W_i = \frac{1}{2} LOG(\Delta_i)$ , da i-ésima variável (eq. 3.15)

$x_i = 1$
$= 0$
$= -1$, conforme o gerente classifique que no seu projeto aquela variável indica produtividade aumentada, normal ou decrescida

Com o valor de $I$ pode-se encontrar a produtividade através da fórmula:

$$LOG(\text{produtividade}) = a + bI \qquad \text{(eq. 3.16)}$$

onde: $a$ e $b$ são constantes determinadas, mas que propositalmente não são fornecidas por Walston e Felix

Note que a fórmula acima é não linear de uma variável (muito semelhante às descritas anteriormente):

$$\text{produtividade} = 10^a \cdot \left(10^I\right)^b = c \cdot IE^b \qquad \text{(eq. 3.17)}$$

onde: $c=10^a$ e $IE=10^I$

Na páginas seguintes é apresentada uma tabela com as 29 variáveis, seus valores médios de produtividade conforme sua classificação e os valores de variação desta produtividade ($\Delta$). Os valores de produtividade e variação de produtividade são dados em linhas de código por pessoa-mês.

*Tabela 3.3* Variáveis Selecionadas por Walston & Felix e sua Influência na Produtividade

| *variável* | *produtividades médias obtidas* | | | Δ |
|---|---|---|---|---|
| complexidade de interface com o cliente | < normal<br>500 | normal<br>295 | > normal<br>124 | 376 |
| particpaação do usuário na definição dos req. | nenhuma<br>491 | alguma<br>267 | muita<br>205 | 286 |
| mudanças no projeto do sistema pelo cliente | poucas<br>297 | | muitas<br>196 | 101 |
| experiência do cliente na área do projeto | nenhuma<br>318 | alguma<br>340 | muita<br>206 | 112 |
| experiência/qualificação geral do pessoal | baixa<br>132 | média<br>257 | alta<br>410 | 278 |
| % dos programadores fazendo o desenvolv. que participaram do proj. funcional | < 25%<br>153 | 25-50%<br>242 | > 50%<br>391 | 238 |
| exp. prévia com o computador/SO | mínima<br>146 | média<br>270 | extensa<br>312 | 166 |
| exp. prévia com a linguagem de programação | mínima<br>122 | média<br>225 | extensa<br>385 | 263 |
| exp. prévia com aplicação de tamanho/complexidade semelhante | mínima<br>146 | média<br>221 | extensa<br>410 | 264 |
| relação tamanho médio do grupo por tempo(meses) | < 0,5<br>305 | 0,5-0,9<br>310 | > 0,9<br>173 | 132 |
| desenvolvimento concorrente de hardware | não<br>297 | | sim<br>177 | 120 |
| acessos ao comp. de desenv. abertos sob requisição especial | 0%<br>226 | 1-25%<br>274 | > 25%<br>357 | 131 |
| acessos negados ao comp. de desenvolvimento | 0-10%<br>303 | 11-85%<br>251 | > 85%<br>170 | 133 |
| ambiente de segurança para o comp. e 25% dos prog. e dados | não<br>289 | | sim<br>156 | 133 |

*Tabela 3.3 (continuação)*

| *variável* | *produtividades* | *médias* | *obtidas* | Δ |
|---|---|---|---|---|
| % de programação estruturada | 0-33%<br>220 | 34-66%<br>--- | > 66%<br>339 | 119 |
| inspeção de projeto/código | 0-33%<br>220 | 34-66%<br>300 | > 66%<br>339 | 119 |
| desenvolvimento top-down | 0-33%<br>196 | 34-66%<br>237 | > 66%<br>321 | 125 |
| uso de chefes de times de programação | 0-33%<br>219 | 34-66%<br>--- | > 66%<br>408 | 189 |
| complexidade do código desenvolvido | < média<br>314 | | > média<br>185 | 129 |
| compl. de processamento da aplicação | < média<br>349 | média<br>299 | > média<br>209 | 129 |
| compl. do fluxo do programa | < média<br>289 | média<br>299 | > média<br>209 | 80 |
| restrições no projeto do sistema | mínimas<br>293 | médias<br>286 | severas<br>166 | 107 |
| restrições de armazenamento pricipal | mínimas<br>391 | médias<br>277 | severas<br>193 | 198 |
| restrições de tempo de processamento | mínimas<br>303 | médias<br>317 | severas<br>171 | 132 |
| código para tempo-real, operações interativas ou com muitas restrições de tempo | < 10%<br>279 | 10-40%<br>337 | > 40%<br>203 | 76 |
| percentagem do código a ser entregue | 0-90%<br>159 | 91-99%<br>327 | 100%<br>265 | 106 |
| código não matemático e de formatação de entrada/saída | 0-33%<br>188 | 34-66%<br>311 | 67-100%<br>267 | 79 |
| número de classes de itens na BD por 1000 LOC | 0-15<br>334 | 16-80<br>243 | > 80<br>193 | 141 |
| número de pag. de documentação entregues por 1000 LOC entregues | 0-32<br>320 | 33-88<br>252 | > 88<br>195 | 125 |

O modelo mostrado apresenta algumas falhas que devem ser discutidas. A primeira delas é que considerável julgamento pessoal é necessário para se estabelecer se uma variavel está presente ou não no projeto. A segunda é que o uso de valores discretos para $x_i$ provoca grandes variações de $I$ para apenas três classificações de cada variável considerada. A terceira, e mais grave, é que muitas das variáveis consideradas estão altamente correlacionadas entre si; isto resulta no superdimensionamento das influências de algumas variáveis agrupadas. Entretanto, o estudo de Walston e Felix foi uma grande contribuição na identificação de variáveis que influenciam a produtividade do desenvolvimento de *software*, assim como no estabelecimento da magnitude do efeito destas variáveis.

### 3.1.7 O Modelo PRICE-S

O PRICE[2] [FRE79] é um modelo disponível comercialmente (através da companhia RCA). Ele foi originalmente desenvolvido para estimar sistemas embutidos[3] (mas especificamente sistemas militares aeroespaciais), mas ele foi aperfeiçoado e estendido, e agora é utilizável para estimativas de sistemas de aplicações comerciais. Como ele é um produto comercial não existe informações sobre o seu funcionamento. Ele realiza uma estimativa *top-down* e fornece custo e cronograma a partir de tamanho, tipo e dificuldade do projeto proposto.

*Tabela 3.4* Entradas do Modelo PRICE-S

| *variável* | *descrição* | *explicação* |
|---|---|---|
| instruções | número de instruções entregues | |
| código novo | % de código novo | valor entre 0 e 1 |
| projeto novo | % de projeto novo | valor entre 0 e 1 |
| aplicação | tipo da aplicação | valor entre 0,8 e 11 (vide tabela 3.5) |
| utilização | % da capacidade de memória e do tempo de relógio de hardware usados | valor entre 0 e 1 |
| plataforma | ambiente operacional planejado para o software | valor entre 0,6 e 2,5 (vide tabela 3.6) |
| índice de produtividade | é uma variável derivada empiricamente para calcular a produtividade | |
| força de trabalho | número médio de pessoal envolvido no projeto em todo seu ciclo de vida | |
| fração de tempo | é fração de tempo média dedicada ao software | |
| complexidade | descreve o efeito relativo de fatores complicadores do projeto, ex: familiaridade com o produto, projeto conjunto de hardware/software | (vide tabela 3.7) |

[2] sigla em inglês da expressão, Revisão Programada de Informação para Orçamento e Avaliação.

[3] do termo em inglês, EMBEDDED

As entradas deste modelo são mostradas na Tabela 3.4 . As Tabelas 3.5 a 3.7 explicam como classificar as variáveis aplicação, plataforma e complexidade, usadas como entrada do modelo PRICE-S.

A Tabela 3.5 apresenta os valores de entrada de acordo com o tipo de aplicação.

*Tabela 3.5* Valores Entrados para cada Aplicação

| tipo da aplicação | valor de entrada |
|---|---|
| sistema operacional | 10,95 |
| operações interativas | 10,95 |
| controle e comando em tempo real | 8,46 |
| comunicação *on-line* | 6,16 |
| operações com banco de dados | 4,10 |
| operações com *strings* | 2,31 |
| operações matemáticas | 0,86 |

A Tabela 3.6 apresenta os valores da variável plataforma de acordo com as classificações do ambiente em que o sistema desenvolvido irá funcionar.

*Tabela 3.6* Valores Entrados para cada Plataforma

| Ambiente Operacional | Plataforma |
|---|---|
| *software* interno da empresa | 0,6-0,8 |
| *software* externo à empresa | 1,0 |
| aplicação militar em terra | 1,2 |
| aplicação militar móvel (navio ou carros de combate) | 1,4 |
| aviação comercial | 1,7 |
| aviação militar | 1,8 |
| aplic. espacial não tripulada | 2,0 |
| aplicação espacial tripulada | 2,5 |

A Tabela 3.7 ajuda a determinar o valor da variável complexidade. Primeiramente, classifica-se o pessoal e a sua familiaridade com o produto a ser desenvolvido. Em segundo lugar, considera-se quais dos fatores complicantes listados estão presentes no projeto. Finalmente, somam-se os coeficientes de ajuste mostrados caso as características e fatores complicantes específicos estejam presentes.

*Tabela 3.7* Classificação da Complexidade no PRICE-S

| Classificação do Pessoal | Ajuste da Complexidade |
|---|---|
| - Excepcional, entre os melhores da industria | -0,2 |
| - Larga experiência, e alguns talentos excepcionais | -0,1 |
| - Capacidade normal, pessoal experiente | 0 |
| - Grupo com pessoas experientes e novatos | +0,1 |
| - Muitas pessoas inexperientes no grupo | +0,2 |
| **Familiaridade com o Produto** | |
| - Muito alta, repetição de trabalho anterior | -0,2 |
| - Familiar com o tipo de projeto | -0,1 |
| - Projeto novo normal, na linha de trabalho | 0 |
| - Nova linha de trabalho para o grupo | +0,2 |
| **Fatores Complicantes** | |
| - Linguagem de programação desconhecida | +0,1 |
| - Processador desconhecido | +0,1 |
| - Linguagem nova | +0,2 a +0,3 |
| - Hardware novo | +0,2 a +0,3 |
| - Desenvolvimento em mais de um lugar | +0,2 |
| - Projeto multinacional | +0,4 |
| - Hardware desenvolvido em paralelo ou muitas mudanças nos requisitos | +0,2 a +0,3 |
| - Linguagem de máquina | +0,2 a +0,3 |

O modelo precisa ser calibrado para ambientes de desenvolvimento específicos: isto é feito através do fornecimento da descrição de diversos projetos prontos ao modelo PRICE, com estas descrições o modelo calcula a variável *índice de produtividade* para aquele ambiente específico de desenvolvimento: o cálculo é feito através da aplicação do modelo PRICE de forma inversa, ou do modelo ECIRP como o denominaram seus criadores.

Apesar das fórmulas e metodologias de cálculo não serem disponíveis para este modelo, pode-se notar, a partir de suas entradas e seu procedimento de calibração, que ele é muito semelhante aos modelos classificados como compostos, como os de Walston & Felix, Boehm e principalmente de Putnam (vide Seção 3.2.2). Este modelo está instalado em computadores da RCA e é utilizado mediante contrato de aluguel. A despeito dos preços do aluguel este modelo é usado por grandes empresas americanas [CUE87] como:

- *Boeing*
- *General Dynamics*
- *US Department of Defense*
- *IBM*
- *General Electric*
- *Texas Instruments*

### 3.1.8 Modelo de Bailey-Basili

Este modelo [BAI81] procurava derivar uma metodologia de estimativa que fosse utilizável para um determinado local de desenvolvimento de *software*. Desta forma assumiu como convenção que as constantes de qualquer fórmula de estimativa eram dependentes do ambiente e do pessoal de desenvolvimento. Desta maneira fórmulas validadas em bases de dados sobre projetos locais eram as que mais aproximadamente refletiam as características de um dado ambiente de desenvolvimento.

A partir de uma base de dados com 19 projetos mais ou menos homogêneos catalogados (a maioria dos projetos eram de *softwares* científicos e programados em FORTRAN), o modelo derivou por de regressão a fórmula:

$$\overline{E} = 3{,}5 + 0{,}73.kLOC^{1{,}16} \qquad (eq.\ 3.18)$$

Note que o esforço é igual a 3,5 homens-mês para zero linhas de código programadas; segundo Bailey-Basili esta constante deve ser interpretada como o esforço inicial de compreensão do projeto antes do começo da codificação. A fórmula derivada apresentou bons resultados na base de dados usada.

A equação 3.18 considera apenas o tamanho do *software* em linhas de código; Bailey e Basili estudaram ainda as variáveis que mais influenciam no custo do *software* e derivaram uma fórmula de ajuste no esforço baseando-se nestas variáveis. A fórmula usada é dada por:

$$\overline{E}ajustado = \overline{E}calculado.ER$$

$$ER = -0{,}036.(METOD) + 0{,}009.(CMPLX) + 0{,}80 \qquad (eq.\ 3.19)$$

onde: ER é a relação de erro
METOD são os atributos relativos à metodologia
CMPLX são os atributos relativos à complexidade

As contagens dos atributos relativos à metodologia, à complexidade e à experiência (não considerada na fórmula acima), são feitas com a soma total das classificações das variáveis associadas a cada um destes três atributos. Cada variável é classificada com um valor de 0 até 5 de acordo com o grau da

presença destas no projeto que se pretende desenvolver. As variáveis estão listadas a seguir, em um total de 21, das quais 9 variáveis são relativas ao atributo metodologia, 7 ao atributo complexidade e 5 ao atributo experiência. Note que o valor máximo de METOD é 45 (pois existem 9 variáveis associadas a este) e o de CMPLX é 35 (pois existem 7 variáveis associadas a este).

*Tabela 3.8* Variáveis de Ajuste de Custo

| | |
|---|---|
| Atr. Metodologia: | Uso de diagramas de árvores |
| | Projeto top-down |
| | Documentação formal |
| | Uso de times de programação |
| | Uso de planos formais de testes |
| | Treinamento formal |
| | Uso de formalismos de projeto |
| | Inspeção de código |
| | Desenvolvimento modular |
| Atr. Complexidade: | Complexidade da aplicação |
| | Complex. da interação com o cliente |
| | Complexidade do fluxo do programa |
| | Complexidade da comunicação interna |
| | Complexidade da base de dados |
| | Complexidade da comunicação externa |
| | Mudanças de proj. requisitadas pelo cliente |
| Atr. Experiência: | Qualificações dos programadores |
| | Experiência geral do grupo |
| | Exp. do programador com o computador |
| | Exp. do programador com a aplicação |
| | Exp. do prog. com a ling. de programação |

O método acima fornece bons resultados na sua base de dados, o erro padrão é de ±25% para $\overline{E}_{calculado}$ e de ±15% para $\overline{E}_{ajustado}$. O modelo apresenta algumas falhas; a mais importante delas é que as próprias variáveis poderiam ser usadas para calcular *ER*, já que agrupá-las por atributos faz com que variáveis claramente distintas, mas que pertencem ao mesmo atributo, tenham as mesmas contribuições para o cálculo do esforço dentro do modelo.

### 3.1.9 Modelo de Alocação de Recursos de Putnam

Este é um modelo de base teórica que se baseia na suposição de que a força de trabalho necessária para o desenvolvimento e manutenção de um *software* de grande porte segue o padrão das curvas de ciclo de vida de Norden [NOR63]; mais especificamente, segue a fórmula de Rayleigh dada pela equação [PUT78a] :

$$\dot{Y} = K \cdot \frac{t}{t_d^2} \cdot exp\left[-t^2/(2.t_d^2)\right] \qquad (eq.\ 3.20)$$

onde: $\dot{Y}$ é o numero de pessoas trabalhando em qualquer instante t

K é a área sob a curva e representa o esforço total durante todo o ciclo de vida do software

$t_d$ é o instante de pico da curva e corresponde aproximadamente ao tempo de desenvolvimento para softwares de grande porte

Integrando a equação acima no intevalo (0,∞] obtém-se K, o esforço total de desenvolvimento e manutenção do *software*. Integrando-se a equação no intervalo (0,t] obtém-se o esforço acumulado até o instante t :

$$E(t) = \frac{K}{2}\left[1 - exp\left[-t^2/(2.t_d^2)\right]\right] \qquad (eq.\ 3.21)$$

A Figura 3.9 apresenta o gráfico das duas equações mostradas acima.

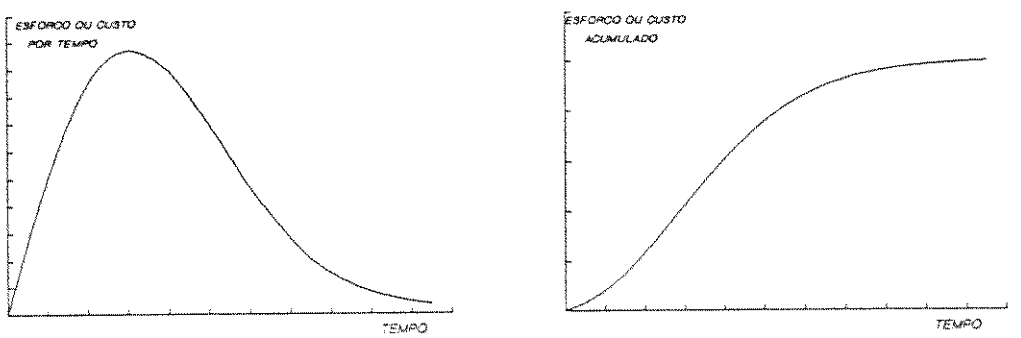


*Figura 3.1* Curvas de Força de Trabalho e Esforço Acumulado

Na teoria desenvolvida por Putnam a grandeza fator de dificuldade tem um papel fundamental; ela é definida como:

$$D = \frac{K}{t_d^2} \qquad \text{(eq. 3.22)}$$

Esta grandeza é assim denominada pois Putnam observou que em sistemas relativamente fáceis de desenvolver $D$ era pequeno, enquanto em sistemas relativamente difíceis de desenvolver, $D$ era grande. Putnam então formulou a hipótese de que o fator de dificuldade e a produtividade deveriam guardar uma relação do tipo:

$$P = cte \cdot D^{\beta} \qquad \text{(eq. 3.23)}$$

onde: P é a produtividade em LOC por esforço de desenv.
$\beta$ é uma constante a se determinar

Usando uma base de dados de projetos de *software* das forças armadas americanas Putnam chegou à fórmula:

$$P = cte \cdot D^{-2/3} \qquad \text{(eq. 3.24)}$$

Considerando que $t_d$ corresponda muito aproximadamente ao tempo de desenvolvimento temos, da eq. 3.21, que o esforço de desenvolvimento é:

$$E(t_d) = 0.3935\ K \qquad \text{(eq. 3.25)}$$

Como, P = LOC/E(t), podemos encontrar uma equação do *software* que relaciona esforço, tempo de desenvolvimento e tamanho do *software*, combinando as equações 3.22 , 3.21 e 3.24:

$$LOC = cte_1 \cdot K^{1/3} \cdot t_d^{4/3} \quad \text{ou} \quad K = cte_2 \cdot LOC^3 / t_d^4 \qquad \text{(eq. 3.26)}$$

Com a equação acima pode-se calcular o esforço de desenvolvimento:

$$\overline{E} = 0.3935\ K = 0.3935 \cdot LOC^3 / t_d^4 \cdot 1/cte_1^3 \qquad \text{(eq. 3.27)}$$

A constante $cte_1$ foi chamada por Putnam de fator de tecnologia e deve ser determinada para cada projeto a partir da estimativa do grau de presença no projeto de várias variáveis, que afetam a produtividade, ou a partir de dados históricos sobre projetos desenvolvidos em ambientes semelhantes. Desta forma o fator de tecnologia (de agora em diante chamado de $C_K$) faz o ajuste do esforço estimado de acordo com o ambiente de desenvolvimento e as características do projeto pretendido. A metodologia para o cálculo de $C_K$ a partir das variáveis que influenciam o processo de desenvolvimento não é fornecida; no entanto estas variáveis são bastante semelhantes àquelas usadas para ajuste do esforço em outros modelos. Putnam propõe que $C_K$ assuma 20 valores discretos variando de 610 a 57.314 de acordo com o projeto (considerando $K$ medido em pessoas-ano e $t$ em anos).

Outro ponto importante é a necessidade de se estabelecer o tempo de desenvolvimento. Na maioria dos modelos apresentados até aqui, o esforço de desenvolvimento é calculado a partir de alguma métrica e de algumas variáveis de ajuste, e $t_d$ é calculado a partir do valor deste esforço de desenvolvimento. No modelo proposto por Putnam o esforço é determinado a partir de $t_d$. O valor de $t_d$ deve ser fornecido pelo usuário do modelo. Para estabelecer uma faixa de valores possíveis de $t_d$, Putnam acrescentou a sua teoria a restrição:

$$\nabla D = K/t_d^3 \qquad \text{(eq. 3.28)}$$

Esta restrição estabelece que $\nabla D$ assume valores discretos correspondentes aproximadamente à magnitude do gradiente de dificuldade do sistema a ser desenvolvido; desta forma, $\nabla D$ determina na fórmula acima o valor de tempo de desenvolvimento mínimo. Fisicamente falando, o valor $\nabla D$ determina o valor do tempo mínimo possível de desenvolvimento de um projeto de acordo com a dificuldade do projeto a ser desenvolvido. Para encontrar $t_d$ mínimo, e o correspondente esforço (máximo) de desenvolvimento, basta resolver o sistema de equações não lineares formado pelas equações 3.27 e 3.28.

Putnam propõe que $\nabla D$ assuma seis valores discretos que variam de 7,3 até 89,0 ; por exemplo, em um sistema com grande complexidade e código totalmente novo, $\nabla D=7,3$ , enquanto em um sistema relativamente simples e com bastante código reutilizado, $\nabla D=27$.

Além da restrição física estabelecida pelo gradiente de dificuldade do projeto, o usuário pode estabelecer suas próprias restrições aos valores de $t_d$ e $E$ , como, por exemplo: custo máximo suportável, tempo de desenvolvimento máximo, máxima quantidade de pessoal disponível. Estas restrições combinadas fornecerão um valor máximo e mínimo possível para $t_d$ e, consequentemente, para o esforço. Dentro destes limites de variação o usuário escolhe o $t_d$ desejado e determina o esforço correspondente, ou vice-versa, usando a equação do *software* (eq. 3.26).

Putnam propôs ainda estudos de riscos de variação dos valores estimados usando simulação. Este processo consiste em fazer variar, segundo uma distribuição estatística, os valores usados para encontrar $t_d$ e o esforço. Abstemo-nos de comentar mais profundamente este tema aqui, ressaltando que esta facilidade foi colocada do modelo de Putnam implementado no SAPDES e, que ela será discutida no capítulo 5.

É importante notar que, na equação do *software*, o esforço varia inversamente com a quarta potência do tempo de desenvolvimento em um projeto com tamanho e ambiente de desenvolvimento definido, como pode-se ver abaixo.

$$K = \left( \frac{1}{t_d} \right)^4 . \left( \frac{LOC}{C_k} \right)^3 \qquad \textit{(eq. 3.29)}$$

Se, por exemplo, o tempo de desenvolvimento fosse de dois anos e se se quisesse reduzí-lo para 1 ano e meio , haveria um acréscimo de 216% no esforço estimado pela fórmula (o valor de K para 1,5 anos seria 3,16 vezes o valor de K para 2 anos). Esta penanalidade tão severa no valor de $K$, e consequentemente de $E$, por uma redução de 25% no valor de $t_d$, não está presente em outros modelos e é bastante criticada na literatura.

Abaixo mostramos outras críticas feitas a este modelo:

- ele concentra a influência de todas as variáveis do projeto, inclusive complexidade e atributos do pessoal de desenvolvimento, no fator de tecnologia Ck.

- validações independentes da relação que, segundo Putnam, $D$ guarda com a produtividade, falharam [CON85]: desta forma a própria equação do *software*, proposta por Putnam, pode não ser válida.

- O modelo funciona bem em projetos muito grandes mas super-estima, gravemente, o esforço em projetos médios e pequenos.

- Só um usuário experimentado pode usar de forma efetiva este modelo, devido pricipalmente à forte influência de $t_d$ no esforço total estimado.

Pelos motivos citados acima, este modelo é difícil de utilizar e de implementar. A implementação feita no SAPDES não permite ainda estimativa de Ck pois Putnam não mostra como fazê-lo; desta forma, Ck pode ser determinado apenas a partir de projetos anteriores. Outro ponto crítico é como encontrar $\nabla D$, para calcular $t_d$ mínimo. Na implementação do SAPDES $\nabla D$ assume apenas quatro valores discretos de acordo com respostas à perguntas feitas pelo SAPDES ao usuário.

### 3.1.10 Modelo Construtivo de Custo (COCOMO)

O COCOMO (*Cost Constructive Model*) ([BOE81] e [BOE84]) é o mais conhecido, completo e documentado de todos os modelos de estimativa de custo de *software*. Ele fornece fórmulas para determinar cronograma de desenvolvimento, esforço total de desenvolvimento, esforço por fase e atividades do projeto, e esforço de manutenção do sistema a ser desenvolvido.

O COCOMO existe em três níveis de precisão: básico, intermediário e detalhado. A aplicabilidade de cada uma destas versões depende basicamente da fase em que projeto se encontra. A estimativa de esforço de desenvolvimento é realizada com equações do tipo:

$$\bar{E} = a_i \cdot LOC^{b_i} \cdot c(x) \qquad (eq.\ 3.30)$$

onde: $a_i$ e $b_i$ são constantes para um dado modo de desenvolvimento

$c(x)$ é um coeficiente de ajuste determinado a partir do grau de presença no projeto de 15 variáveis de influência

Boehm identificou no seu modelo três modos de desenvolvimento e determinou os valores de $b_i$, para cada modo de desenvolvimento, e de $a_i$, para cada nível de precisão e modo de desenvolvimento. A tabela abaixo mostra os valores de $a_i$ e $b_i$ para os três modos de desenvolvimento no COCOMO básico e intermediário.

*Tabela 3.9* Coeficientes das Fórmulas do COCOMO

| MODO DE DESENVOLVIMENTO | BÁSICO | | INTERMEDIÁRIO | |
|---|---|---|---|---|
| | $a_i$ | $b_i$ | $a_i$ | $b_i$ |
| ORGÂNICO | 2,4 | 1,05 | 3,2 | 1,05 |
| SEMI-DESCONECTADO | 3,0 | 1,12 | 3,0 | 1,12 |
| EMBUTIDO | 3,6 | 1,20 | 2,8 | 1,20 |

Projetos desenvolvidos no modo orgânico são caracterizados por serem relativamente pequenos em tamanho, requererem pouca inovação, terem requisitos flexíveis e serem desenvolvidos internamente à própria empresa que deseja o sistema. Os projetos no modo embutido são relativamente grandes, têm que operar sob

fortes restrições, têm requisitos rígidos, apresentam grande inovação, e têm *interface* complexa com o cliente e/ou *hardware*. Projetos desenvolvidos no modo semi-desconectado localizam-se, aproximadamente, entre os modos orgânico e embutido. A Tabela 3.10 apresenta alguns critérios para classificação do modo de desenvolvimento.

No COCOMO básico o valor de c(x) na equação 3.30 é sempre igual a um, qualquer que seja o grau de presença das variáveis que influenciam no projeto (nesta seção usaremos, de agora em diante, o termo '*cost drivers*[4]' para nos referirmos a estas variáveis). A Tabela 3.11 lista os *cost drivers* selecionados por Boehm em seu modelo e agrupa-os em quatro categorias. No COCOMO intermediário c(x) assume vários valores a depender dos *cost drivers*. A fórmula a seguir mostra como calcular c(x) a partir dos *cost drivers*.

$$c(x) = \prod_{j=1}^{15} f_j \qquad \textit{(eq. 3.31)}$$

onde: $f_j$ é o fator de ajuste do esforço estimado de acordo com o grau de presença do *cost driver j* no projeto

Na Tabela 3.12 são mostrados os fatores de ajuste de esforço de acordo com a classificação de cada um dos *cost drivers*.

Os valores dos fatores de ajuste são atribuídos de acordo com a presença do *cost driver* no projeto; esta presença é classificada em:

- muito baixa - MB
- baixa ..... - B
- normal .... - N
- alta ..... - A
- muito alta - MA
- extra alta - EA

Os critérios da classificação da presença de cada *cost driver* no projeto não serão discutidos aqui, eles podem ser encontrados em detalhes em [BOEB1].

---

[4] este termo vem do inglês e é usado por Boehm para referir-se às variáveis que, segundo ele, mais influenciam o processo de desenvolvimento e manutenção de software.

*Tabela 3.10* Critérios de Classificação do Modo de Desenvolvimento

| CARACTERÍSTICAS | MODO DE DESENVOLVIMENTO | | |
|---|---|---|---|
| | orgânico | semi-desconectado | embutido |
| compreensão organizacional dos objetivos do produto | COMPLETA | CONSIDERÁVEL | GERAL |
| experiência com sistemas semelhantes | EXTENSA | CONSIDERÁVEL | MODERADA |
| necessidade de concordância com os requisitos pré-estabelecidos | BÁSICA | CONSIDERÁVEL | COMPLETA |
| necessidade de concordância com as interfaces externas especificadas | BÁSICA | CONSIDERÁVEL | COMPLETA |
| desenvolvimento concorrente de hardware e procedimentos operacionais novos | ALGUM | MODERADO | EXTENSO |
| necessidade de algoritmos e arquiteturas inovadoras | MÍNIMA | ALGUMA | CONSIDERÁVEL |
| prêmio para término mais rápido | | | |
| intervalo de tamanho do produto | < 50 kDSI | < 300 kDSI | todos os tamanhos |
| EXEMPLOS | processamento de dados batch<br>modelos científicos<br>modelos comerciais<br>sistema operacional familiar<br>compilador familiar<br>controle de produção simples | sistema de processamento de transações<br>sistema operacional novo<br>sistema de gerência de banco de dados<br>controle de produção ambicioso<br>sistema simples de controle/comando | sistema de processamento de transações grande e complexo<br>sistema operacional complexo e grande<br>aviônica<br>sistema ambicioso de controle/comando |

*Tabela 3.11* Variáveis que Influenciam no Custo

| VARIÁVEIS QUE INFLUENCIAM NO CUSTO *(COST DRIVERS)* | | |
|---|---|---|
| Atributos do Produto | Confiabilidade requerida do software | RELY |
| | Tamanho do banco de dados | DATA |
| | Complexidade do produto | CPLX |
| Atrib. do Computador | Restrições de tempo de execução | TIME |
| | Rest. de armazenamento principal | STOR |
| | Volatilidade da máquina virtual | VIRT |
| | Tempo de *'TURNAROUND*[5]*'* do computador | TURN |
| Atributos de Pessoal | Competência dos analistas | ACAP |
| | Experiência com a aplicação | AEXP |
| | Competência dos programadores | PCAP |
| | Experiência com a máquina virtual | VEXP |
| | Exp.com a linguagem de programação | LEXP |
| Atributos de Projeto | Práticas modernas de programação | MODP |
| | Uso de ferramentas de software | TOOL |
| | Cronograma de desenv. requerido | SCED |

*Tabela 3.12* Valores das Variáveis de Ajuste de Esforço

| FATORES DE AJUSTE DO ESFORÇO | | | | | | |
|---|---|---|---|---|---|---|
| COST DRIVERS | PRESENÇA DA VARIÁVEL NA APLICAÇÃO | | | | | |
| | MB | B | N | A | MA | EA |
| *RELY* | 0,75 | 0,88 | 1,00 | 1,15 | 1,40 | --- |
| *DATA* | --- | 0,94 | 1,00 | 1,08 | 1,16 | --- |
| *CPLX* | 0,70 | 0,85 | 1,00 | 1,15 | 1,30 | 1,65 |
| *TIME* | --- | --- | 1,00 | 1,11 | 1,30 | 1,66 |
| *STOR* | --- | --- | 1,00 | 1,06 | 1,21 | 1,56 |
| *VIRT* | --- | 0,87 | 1,00 | 1,15 | 1,30 | --- |
| *TURN* | --- | 0,87 | 1,00 | 1,07 | 1,15 | --- |
| *ACAP* | 1,46 | 1,19 | 1,00 | 0,86 | 0,71 | --- |
| *AEXP* | 1,29 | 1,13 | 1,00 | 0,91 | 0,82 | --- |
| *PCAP* | 1,42 | 1,17 | 1,00 | 0,86 | 0,70 | --- |
| *VEXP* | 1,21 | 1,10 | 1,00 | 0,90 | --- | --- |
| *LEXP* | 1,14 | 1,07 | 1,00 | 0,95 | --- | --- |
| *MODP* | 1,24 | 1,10 | 1,00 | 0,91 | 0,82 | --- |
| *TOOL* | 1,24 | 1,10 | 1,00 | 0,91 | 0,83 | --- |
| *SCED* | 1,23 | 1,08 | 1,00 | 1,04 | 1,10 | --- |

[5] tempo transcorrido entre a submissão de um job e a recepção da resposta

Com o que foi mostrado até o momento já se pode estimar o esforço de desenvolvimento do *software*. Para determinar o tempo de desenvolvimento deste *software* o COCOMO usa as fórmulas mostradas a seguir:

$$td = 2{,}5 \cdot \overline{E}^{\,0{,}38} \quad \text{, no modo orgânico} \qquad (eq.\ 3.32)$$

$$td = 2{,}5 \cdot \overline{E}^{\,0{,}35} \quad \text{, no modo semi-desconectado} \qquad (3.33)$$

$$td = 2{,}5 \cdot \overline{E}^{\,0{,}32} \quad \text{, no modo embutido} \qquad (eq.\ 3.34)$$

O COCOMO permite, ainda, a determinação do cronograma de desenvolvimento por fase do projeto, a partir do tempo de desenvolvimento determinado, e da distribuição do esforço por estas fases; isto é feito através de tabelas onde a distribuição por fase depende do tamanho do sistema, em linhas de código, e do modo de desenvolvimento determinados. O COCOMO fornece também tabelas para determinação dos esforços por atividades dentro de cada fase do projeto. Todas estas tabelas podem ser encontradas em [BOE81] ou [MEN90].

As equações do COCOMO foram derivadas de uma base de dados com 63 projetos desenvolvidos entre 1964 e 1979 pela empresa americana *TRW Systems*. Estes projetos foram escritos em diversas linguagens de programação, entre elas: ASSEMBLY, FORTRAN, COBOL, PL/I, e Jovial. Os projetos variam de tamanho: de 2000 até 1.000.000 LOC; e de tipo de aplicação: científica, comercial, de controle, de supervisão, etc. Por todos estes motivos o autor não utilizou regressão diretamente sobre a base de dados; em vez disto, Boehm utilizou sua experiência em outros modelos de estimativa, e opiniões de gerentes de software experientes, para chegar a parâmetros iniciais do modelo baseado-se em um sub-conjunto da base de dados de projetos. Os parâmetros iniciais foram refinados e calibrados com os outros projetos da base de dados. O COCOMO apresenta excelente performance nesta base de dados, principalmente considerando o quanto ela é heterogênea. A boa performance deste modelo, assim como de outros, depende, todavia, de calibração ([KEM87] , [RUB85] e [KIT85]). Os trabalhos

que tratam sobre calibração de modelos são pouco difundidos na literatura; entretanto, podemos citar [MIY85] que trata especificamente de calibração e adaptação do COCOMO.

Apesar das críticas, o COCOMO é o mais completo e documentado modelo de estimativa de custo de *software* disponível atualmente. Boehm dedica boa parte das 791 páginas do seu livro [BOE81] a apresentar e discutir o COCOMO. A consequência desta disponibilidade de material fez com que este modelo se tornasse o mais discutido na literatura e o que é mais implementado de forma automatizada em computadores (vide Seção 3.3).

### 3.1.11 Modelo de Análise por Function Points

O modelo de análise por *functions points*, como o nome indica, usa uma métrica básica diferente de linhas de código para estimar o esforço de desenvolvimento de *software*. Esta métrica foi apresentada por Albrecht em [ALB79], e sua grande vantagem é que ela pode ser obtida cedo no processo de desenvolvimento de *software* , pois pode ser derivada já na discussão dos requisitos do *software* com os usuários ou clientes. Ela tenta determinar a quantidade de funções a serem fornecidas pelo sistema que se vai construir. Para fazer isto desenvolveu-se a tese de que a contagem e classificação das principais estruturas de dados que são usadas ,lidas e fornecidas pelo *software* quantificam esta funcionalidade. O modelo de estimativa de esforço com os *function points* assume que esta quantificação pode ser usada como métrica para medir produtividade e, desta forma, guarda estreitas relações com o esforço de desenvolvimento. A contagem dos *function points* é representada por FP. As relações entre FP (function points) e o esforço de desenvolvimento foram derivadas por regressão a partir de projetos prontos e publicadas em [ALB83]; a seguir apresentamos algumas das fórmulas para o cálculo do esforço de desenvolvimento a partir de FP.

$$\overline{E} = 54.(FP) - 13390 \qquad (eq.\ 3.35)$$

$$\overline{E} = 10,75.(FP/2).\log_2(FP/2) - 8300 \qquad (eq.\ 3.36)$$

$$\overline{E} = 4,89.(FP.\log_2 FP) - 8762 \qquad (eq.\ 3.37)$$

Simplificadamente FP é calculado como:

$$\begin{aligned} FP = {} & (\text{NÚMERO DE ENTRADAS} \times 4) + \\ & (\text{NÚMERO DE ARQUIVOS MESTRES} \times 10) + \\ & (\text{NÚMERO DE PERGUNTAS} \times 4) + \\ & (\text{NÚMERO DE SAÍDAS} \times 5) \end{aligned} \qquad (eq.\ 3.38)$$

Neste modelo, a contagem dos *functions points* é o ponto fundamental; a Seção 3.4.2 dedica-se exclusivamente à contagem dos *function points*, e uma referência mais detalhada sobre o assunto pode ser obtida em [ALB83].

Um ponto que deve ser ressaltado neste modelo é que as variáveis que influenciam o processo de desenvolvimento ,e que em outros modelos têm sua influência considerada sempre após a aplicação das fórmulas de estimativa, são consideradas neste modelo durante a própria contagem de FP, ou seja, antes da aplicação de qualquer fórmula para determinar esforço de desenvolvimento.

A contagem dos FP é realizada em três fases e está quase toda descrita na seção 3.4.2 , que mostra o processo de contagem até a obtenção dos FC ou *function points* não ajustados; para se encontrar os FP é necessário ajustar-se FC a partir das variáveis que influenciam o processo de desenvolvimento. O processo de ajuste de FC para encontrar os FP é mostrado a seguir para ressaltar a semelhança entre as variáveis de ajuste usadas neste modelo e as variáveis usadas pelos outros modelos. As variáveis usadas neste modelo são chamadas por Albrecht de características gerais da aplicação, e são representadas por DI. Elas ajustam os FC em ±35% conforme a fórmula abaixo:

$$\text{Fator de ajuste} = 0{,}65 + \left\{0{,}01 \times \left\{\sum_{i=1}^{14} DI_i\right\}\right\} \qquad (eq.\ 3.39)$$

$DI_i$ é o grau de influência da i-ésima característica geral da aplicação, $0 \leq DI \leq 5$.

Onde:

DI=0 ,se não há influência ou presença da característica na aplicação.

DI=1 ,se há influência ou presença pouco significativa da característica.

DI=2 ,se há influência ou presença moderada da característica.

DI=3 ,se há influência ou presença mediana da característica.

DI=4 ,se há influência ou presença significativa da característica.

DI=5 ,se há forte influência ou presença da característica.

As características gerais da aplicação são quatorze e estão listadas a seguir :

a - os dados e informações de controle usadas pela aplicação são enviados ou recebidos usando facilidades de comunicação. Terminais conectados localmente à unidade de controle são considerados facilidades de comunicação.

b - funções e/ou dados distribuidos são características da aplicação.

c - há objetivos de desempenho que influenciam o tempo de resposta, *throughput*, projeto, desenvolvimento, instalação e suporte exigidos da aplicação.

d - o usuário irá executar a aplicação em um equipamento que será muito requisitado.

e - a taxa de transações será alta, e ela influenciará o projeto, desenvolvimento, instalação, e suporte da aplicação.

f - entrada de dados *on-line*[σ] e funções de controle serão fornecidas pela aplicação.

g - as funções *on-line* visarão a eficiência do usuário final.

h - a aplicação permitirá a atualização *on-line* dos arquivos lógicos internos.

i - processamento complexo será uma característica da aplicação. Por exemplo :
- muitas interações de controle e pontos de decisão.
- extensas equações lógicas e matemáticas.
- muito processamento de exceção resultante de transações incompletas que devem ser processadas novamente.

---

σ do termo em inglês, em linha, por meio de terminais dedicados aos usuários.

j - a aplicação e o código a serem gerados serão especificamente projetados e suportados para serem reutilizáveis.

k - fácil conversão e instalação serão algumas das características da aplicação. Um plano de instalação e conversão será fornecido e aprovado durante a fase de teste do sistema.

l - facilidade operacional será uma característica do sistema. Procedimentos de partida, *back-up*[7], e restabelecimento serão fornecidas e testadas durante a fase de teste do sistema. A aplicação visará minimizar a necessidade de atividades manuais tais como, manuseio de papel, troca de fitas e discos, e intervenções operacionas.

m - a aplicação será desenvolvida e suportada para ser instalada em diversos locais por diversas organizações.

n - a aplicação será concebida para facilitar mudanças:

- capacidade de questionamento será fornecida.
- informações sujeitas a mudanças serão agrupadas em tabelas atualizáveis pelo usuário.

O modelo apresentado é bastante simples e tem sofrido críticas; todavia seu uso no SAPDES deveu-se ao desejo de se ter um modelo que usasse uma métrica diferente de linhas de código. Modelos mais complexos usando *FUNCTION POINTS* existem [RUB83], mas não são de domínio público.

[7] do termo inglês, cópia de segurança

### 3.1.12 Modelo de Programação Cooperativa - COPMO

Este modelo foi desenvolvido por Thebaut [THE84] visando explicitamente considerar ,nas estimativas de esforço, a influência do tamanho do grupo de desenvolvimento. Neste modelo a estimativa de esforço baseia-se em duas variáveis, o número de linhas de código fonte em milhares (kLOC), e o tamanho médio do grupo de desenvolvimento (aqui simbolizado por $\overline{P}$). O modelo é aditivo e não linear; a fórmula de estimativa de esforço é mostrada abaixo:

$$\overline{E} = Ep(kLOC) + Ec(\overline{P}) \qquad (eq.\ 3.40)$$

onde: Ep é o esforço das pessoas em trabalhos independentes (por exemplo, programação de módulos do sistema)

Ec é o esforço de coordenação do processo de desenvolvimento (por exemplo, atividades de integração, testes e gerência)

Para encontrar Ep e Ec foram sugeridas as seguintes fórmulas:

$$Ep = a + b.kLOC \qquad e \qquad Ec = c.\overline{P}^{d} \qquad (eq.\ 3.41)$$

o que fornece:

$$\overline{E} = a + b.kLOC + c.\overline{P}^{d} \qquad (eq.\ 3.42)$$

Na determinação dos coeficientes $(a, b, c, d)$ Thebaut usou regressão em dois passos. Primeiramente ele localizou os projetos onde $\overline{P} \cong 1$, ou seja, projetos onde não foi necessário nenhum esforço de coordenação já que, em média, envolveu só uma pessoa, e com estes projetos ele determinou $(a, b)$ a partir da fórmula:

$$E = a + b.kLOC \quad , \text{ com } \overline{P} \cong 1 \qquad (eq.\ 3.43)$$

Com os valores $(a, b)$, o segundo passo foi determinar os valores $(c, d)$; para isto selecionaram-se projetos catalogados na base de dados com $\overline{P} \geq 2$ e aplicou-se regressão a este conjunto de

projetos usando a equação abaixo:

$$E - (a + b.kLOC) = c.\overline{P}^{d} \qquad (eq.\ 3.44)$$

note que a regressão determinou $(c, d)$ , a partir dos valores conhecidos $(E, a, b, \overline{P}, kLOC)$

O modelo então pode ser escrito como:

$$\overline{E} = \begin{cases} a + b.S & , \quad \overline{P} \leq 1 \\ a + b.S + c.\overline{P}^{d} & , \quad \overline{P} > 1 \end{cases} \qquad (eq.\ 3.45)$$

Esta abordagem foi aplicada a 6 diferentes bases de dados [CON85]; na base de dados usada para validar o modelo COCOMO (vide seção 3.2.3) [BOE81], a fórmula obtida foi:

$$\overline{E} = 3,42 + 1,31.kLOC + 2,59\ .\overline{P}^{\,1,60} \qquad (eq.\ 3.46)$$

Deste estudo foi concluido que :

a - A constante $a$ torna-se insignificante para projetos de médio e grande porte.

b - As constantes $(a, b, c)$ diferem bastante de uma base de dados para outra, mostrando que são altamente dependentes do ambiente de desenvolvimento.

c - A constante $d$ mantém-se quase constante com um valor variando em torno de 1,5 , ela pode ser interpretada como a influência do número de caminhos de comunicação entre os programadores no esforço total.

Conte,Dunsmore e Shen resolveram então apresentar um COPMO generalizado levando em conta as seguintes considerações:

a - Para que o modelo tenha caráter geral, as constantes deverão ser definidas para conjuntos de projetos com complexidade semelhantes; desta forma obtêm-se diversas fórmulas que têm caráter geral dentro das classes de complexidades de projeto definidas.

b - A melhor maneira de se classificar a complexidade dos projetos catalogados nas base de dados foi verificar suas produtividades médias.

c - As constantes eram determinadas por regressão por mínimos quadrados; todavia, o objetivo da calibração não deveria ser minimizar a somatória dos quadrados dos erros, mas sim as medidas básicas: *Previsão Média a Nível r*, PRED(r); e *Magnitude Média do Erro Relativo*, MER. Vide seção 2.6.2 .

d - Para corrigir o problema exposto acima, as constantes foram determinadas com o objetivo de maximizar PRED(r) de cada categoria de complexidade.

e - A constante $a$ foi eliminada da fórmula devido a sua pouca influência.

f - A constante $d$ pouco variava em torno de 1,5 , logo não dependia da complexidade; desta forma só seria necessário determinar as constantes ($b$, $c$).

Partindo destas considerações, separaram-se os projetos catalogados em 10 categorias de complexidade e para cada uma delas encontraram-se os coeficientes ($b$, $c$) que maximizavam PRED(r), para a fórmula abaixo:

$$\overline{E} = bi \cdot kLOC + ci \cdot \overline{P}^{1,5} \qquad (eq.\ 3.47)$$

onde: o índice $i$ indica que esta fórmula é aplicável à i-ésima categoria de complexidade

A tabela a seguir mostra as dez classes de complexidade, a produtividade típica de cada classe, e os valores de $b_i$ e $c_i$ para cada uma delas.

*Tabela 3.13* Classes de Complexidade no COPMO

| Classe de Complex. | Produtividade | $b_i$ | $c_i$ |
|---|---|---|---|
| 1 | 0 - 85 | 4,7 | 3,6 |
| 2 | 86 - 200 | 3,0 | 2,0 |
| 3 | 201 - 300 | 2,4 | 1,5 |
| 4 | 301 - 400 | 2,2 | 1,2 |
| 5 | 401 - 500 | 1,8 | 1,0 |
| 6 | 501 - 600 | 1,6 | 0,9 |
| 7 | 601 - 700 | 1,2 | 0,8 |
| 8 | 701 - 800 | 1,1 | 0,7 |
| 9 | 801 - 1000 | 0,9 | 0,5 |
| 10 | 1000 - $\infty$ | 0,8 | 0,4 |

Para aplicar o modelo é necessário estimar-se KLOC, $\overline{P}$, e a classe de complexidade. A estimativa de $\overline{P}$ pode ser feita de várias formas; [CON85] apresenta uma metodologia como parte do modelo, que não será abordada para não alongarmos mais a discussão. A classe de complexidade pode ser estimada usando as variáveis de ajuste de esforço apresentadas pelos outros modelos; [CON85] mostra um exemplo usando as variáveis do COCOMO.

O modelo apresentou boa performance nas bases de dados usadas na sua validação, como ele é relativamente novo não existem ferramentas comerciais que utilizem esta abordagem de estimativa; entretanto, sua utilização tem boas perspectivas, pois ele é um modelo composto desenvolvido sobre uma base teórica que procura explicitamente abordar a influência do esforço de coordenação de pessoal nas suas fórmulas.

## 3.2 MODELOS DE ESTIMATIVAS IMPLEMENTADOS NO SAPDES

O SAPDES é um sistema criado para abrigar diversos modelos de estimativas de custo e cronograma de desenvolvimento de *software*; para o escopo do trabalho de tese procurou-se, todavia, limitar inicialmente o número de modelos de estimativas programados no SAPDES; os motivos para isto foram:

a - Para um primeiro protótipo é necessário que existam apenas alguns modelos para que se torne mais fácil alterar o SAPDES.

b - A contínua implementação de modelos de estimativa poderia acarretar o alongamento do trabalho e a perda de foco em outros pontos importantes na concepção inicial do sistema.

Com o intuito de cumprir as restrições acima e, ao mesmo tempo, testar a generalidade do SAPDES em suportar novos modelos de estimativa, decidiu-se por implementar modelos cujas concepções fossem diferentes. Concluiu-se que implementar três modelos seria o bastante para atingir nossos objetivos e, ao mesmo tempo, agilizar o desenvolvimento do primeiro protótipo. Decidido que seriam três modelos restava decidir quais seriam eles; consideraram-se os seguintes critérios de escolha:

- Disponibilidade de material sobre os modelos: os modelos são muitas vezes comerciais e, frequentemente, partes de suas fórmulas não são fornecidas. É fundamental que os modelos escolhidos possam bem compreendidos, não só com o intuito de facilitar sua implementação, mas também para posterior programação de critérios de seleção e aplicação dos modelos durante uma consulta ao SAPDES. Desta forma era importante que os modelos selecionados tivessem literatura ampla não só sobre sua implementação, mas também sobre experiências de sua utilização e calibração.

- Os modelos escolhidos deveriam ser diferentes: muitos

modelos são diferentes nas fórmulas e variáveis mas não na forma; na verdade, como vimos, a maioria dos modelos usa fórmulas não lineares de uma variável (geralmente LOC) derivada por regressão estatística. Se quiséssemos generalidade no SAPDES seria importante escolhermos modelos diferentes na forma.

Dentro dos critérios anteriores foram selecionados os seguintes modelos:

COCOMO: por ser o mais difundido modelo na literatura.

Modelo de Alocação de Recursos de PUTNAM: por ser também bastante difundido, e por apresentar uma forma estrutural bastante diversa do modelo do COCOMO.

Modelo de ANÁLISE POR *FUNCTION POINTS*: por ser um dos poucos modelos difundidos na literatura que usa uma métrica básica que não é linhas de código.

Estes modelos foram apresentados e discutidos nas Seções 3.1.9 , 3.1.10 e 3.1.11 deste capítulo.

## 3.3 FERRAMENTAS PARA ESTIMATIVA DE CUSTO DE SOFTWARE

Esta seção se dedica à discussão de ferramentas automatizadas para estimativa de custo de *software*. O objetivo aqui será apresentar algumas das ferramentas que existem no mercado mundial, compará-las entre si, e situar o SAPDES neste universo.

A primeira destas ferramentas já foi apresentada na Seção 3.1; ela é o PRICE-S da RCA (parte da General Electric Company). Esta ferramenta está instalada em computadores da própria companhia e é utilizada mediante um contrato de aluguel. Este modelo é usado por grandes empresas americanas. PRICE é a sigla para *"Programmed Review of Information for Costing and Evaluation"*[8]. O uso do modelo faz-se através de terminais ligados por *modem* a computadores *time-sharing* da RCA PRICE Systems, onde o modelo está instalado. A saída desta ferramenta consiste de uma estimativa do esforço (em pessoas-mês), do tempo e do cronograma de desenvolvimento requerido para o projeto. Estas saídas são fornecidas quase que imediatamente após o fornecimento das variáveis de entradas mostradas na seção 3.1. A ferramenta fornece um módulo para que ela seja calibrada para ambientes de desenvolvimento específicos. Isto é feito através do fornecimento da descrição de diversos projetos prontos ao módulo ECIRP. Com esta descrição o ECIRP calcula a variável *índice de produtividade* para aquele ambiente específico de desenvolvimento. O cálculo é feito através da aplicação do modelo usado no PRICE de forma inversa, em uma abordagem semelhante à determinação do fator de tecnologia a partir de projetos antigos no modelo de Putnam.

As ferramentas JS-1, JS-2 e JS-3 baseiam-se no modelo de Jensen ([JEN81], [JEN83a] e [JEN83b]) da Hughes Aircraft Company. O modelo usa equações e variáveis de ajuste de forma similar aos modelos do DOTY e de Boehm para determinar o esforço de desenvolvimento. Para determinar a distribuição de esforço o modelo usa a curva de Rayleigh com uma relação entre $t_d$ e $\overline{E}$ mais conservadora do que a usada por Putnam. Além das saídas normais a ferramenta fornece taxa de gastos de recursos, distribuições de probabilidade de variação do custo e cronograma estimados, etc.

---

[8] Revisão Programada de Informação para Orçamento e Avaliação

Uma outra ferramenta que se baseia no modelo de Jensen [JEN81] é o SYSTEM-3 da Computer Economics Company. Nesta ferramenta o usuário necessita entrar com três valores (provável, otimista e pessimista) para cada uma das muitas variáveis que este modelo usa. Os valores são usados para derivar o intervalo de incerteza associado a cada variável. Além das entradas usuais nos modelos, a ferramenta pergunta por: restrições de desenvolvimento (relação pessoas por ano, máximo número de pessoas e máximo esforço), em uma abordagem semelhante ao modelo de Putnam (vide Seção 3.1.9); impacto de reutilização e reconstrução (percentagem de mudança necessária no projeto, percentagem de esforço de implementação necessário e percentagem de esforço de teste necessário), em uma contagem semelhante ao número equivalente de linhas de código apresentado por Boehm (vide seção 3.4.3); e fatores financeiros (taxa de inflação, salário médio do pessoal de desenvolvimento, etc). Como saída o System-3 fornece: o esforço e tempo de desenvolvimento por fase e atividade de projeto; divisão detalhada de custos por fase, atividade e categoria de pessoal; horas de trabalho por categoria de pessoal; análise de riscos de esforço e cronogramas programados; etc.

O modelo do COCOMO serviu como base para criação de uma série de ferramentas. O WICOMO foi feito em linguagem PASCAL para rodar em microcomputadores no Wang Institute; ela é dirigida por comandos e produz estimativas de esforço de desenvolvimento em pessoas-mês, tempo de desenvolvimento em meses, número médio de pessoas requeridas por mês, produtividade em LOC por mês, custo por mês, distribuição de esforço e tempo por atividades, etc. O BYL, desenvolvida pelo Gordon Group , e o GECOMO da GEC Software , são exemplos de outras ferramentas baseadas no COCOMO. Todas elas fornecem os mesmos tipos de saídas do WICOMO.

O SLIM é a ferramenta desenvolvida por Putnam para aplicação de seu modelo. Ela é propriedade da Quantitative Software Management Company. Como vimos, o modelo de Putnam baseia-se na curva de Raleigh. Para usar o modelo é necessário estimar-se o número de LOC , a constante de tecnologia Ck e o gradiente de dificuldade do projeto. A estimativa do números de LOC é o tema da Seção 3.4.1. No SLIM esta estimativa é feita através da técnica PERT. Para encontrar o valor da constante de tecnologia e do gradiente de dificuldade do projeto o SLIM oferece ao usuário duas opções. Na primeira opção o usuário responde a 22 perguntas e o

SLIM recomenda, a partir das respostas, valores de Ck e $\nabla D$ . Na segunda opção, Ck e $\nabla D$ são determinados a partir de dados sobre projetos completados em ambientes de programação semelhantes. Esta ferramenta realiza, baseando-se na distribuição BETA de probabilidade e no desvio padrão das variáveis de entradas, análises de riscos e sensitividade dos valores estimados. Entre as saídas do SLIM estão: estimativa de esforço e tempo por fase e atividades do projeto, produtividade, confiabilidade (taxa de erros esperadas), melhor compromisso de $td$ x E , estimativa de documentação a ser produzida, análise de risco e sensitividade.

O ESTIMACS é uma ferramenta desenvolvida por Howard Rubin do Hunter College [RUB83] e comercializada pela Management Computer Services (MACS). O modelo usado nela não é de domínio publico; todavia, um dos pontos interessantes deste modelo é que ele não usa LOC como métrica basica. Sua métrica básica é uma contagem semelhante aos *Function Points*. O ESTIMACS faz cerca de 25 perguntas incluindo cinco sobre a contagem de: número de saídas, número de entradas, número de arquivos lógicos, número de consultas, número das principais funções. Estas variáveis de entrada, muito semelhantes às usadas na contagem dos *function points*, são as que têm influência fundamental no custo estimado pelo modelo implementado no ESTIMACS [KEM87]. Baseando-se nas variáveis de entrada, o ESTIMACS gera um conjunto de saídas, até mesmo usando recursos gráficos, do esforço e tempo de desenvolvimento por fase e atividades do projeto, da produtividade (em *function points* por esforço), do esforço de manutenção, dos gastos financeiros X $td$, análise de risco (médio e por categoria de atividade), sensitividade, etc.

O SPQR/20 é baseado no modelo de Jones [JON86] e é propriedade da Productivity Research Company. Esta ferramenta tem várias variáveis de entrada, normalmente obtidas através de questões de múltipla escolha usando menus. Ela fornece as saídas básicas fornecidas por outras ferramentas e mais: quantidade total de documentação, eficiência acumulada de remoção de erros no projeto, número de defeitos na entrega do projeto, periodo de estabilização e confiabilidade do projeto, número de defeitos por 1000 LOC.

Além das ferramentas comerciais algumas grandes empresas desenvolvem ferramentas de estimativa de custo para uso próprio, geralmente já calibradas especificamente para o ambiente de

desenvolvimento presente na companhia; é o caso, por exemplo, da Texas Instruments e AT&T [LEH88].

É importante notar que a maioria das ferramentas apresentadas baseia-se nos trabalhos de Putnam, Boehm e Albrecht. Os modelos e as metodologias de estimativas nelas implementadas usam combinações, variações e evoluções dos modelos e metodologias mostrados em [PUT80], [BOE81] e [ALB83]. Algumas ferramentas combinam a metodologia de cálculo do COCOMO com a distribuição de esforço de Raleigh; outras, usam LOC como métrica básica mas fornecem como opção o uso dos *function points* para contagem do tamanho do projeto (neste caso os *function points* são usados para estimar o número de linhas de código fonte para que se possa utilizar o modelo implementado).

Existem alguns trabalhos importantes de comparação entre modelos ou ferramentas, entre eles podemos citar: [JEN84] que compara o modelo de JS-2 com o COCOMO; [KIT85] que compara o SLIM com o COCOMO; [RUB85] que compara o JS-2, o SLIM, o GECOMO e o ESTIMACS; [MAR88] que compara o BYL, WICOMO, SYSTEM-3 e o SPQR/20. Todos os trabalhos são unânimes em afirmar que os modelos devem ser calibrados para o ambiente específico de desenvolvimento antes de serem utilizados. Um usuário deve estar consciente das limitações das ferramentas e ser suficientemente experiente sobre os conceitos usados nos modelos, principalmente as definições da métrica e variáveis usadas como dados de entrada nas ferramentas. Martin [MAR88] afirma que um usuário necessita modelar vários projetos antes de ganhar a experiência necessária para realmente usufruir das ferramentas de estimativas de custo de *software*.

## 3.4 MÉTRICAS PARA ESTIMATIVA DE TAMANHO DO SOFTWARE

Em todos os modelos discutidos as estimativas de custo são realizadas a partir de alguma estimativa de tamanho do *software*, quase sempre linhas de código. Esta estimativa é fundamental pois em modelos com fórmulas do tipo $E=a.kLOC^b$, $b>1$, um erro na estimativa do tamanho do *software* implicará automaticamente um erro ainda maior na estimativa de esforço; portanto a estimativa correta do tamanho do *software* é uma necessidade fundamental para o bom funcionamento de um modelo, em verdade esta estimativa é uma das componentes mais críticas de qualquer dos modelos apresentados. O problema é realizar estimativas confiáveis do tamanho do *software* ainda numa fase inicial de projeto. Este ítem se dedica à discussão deste tema; apresenta algumas técnicas para estimativa e discute quais problemas geralmente aparecem com estes tipos de estimativas.

### 3.4.1 Contagem das Linhas de Código

Para estimar tamanho de *software* pode-se usar diversas grandezas, ou métricas (vide seção 2.1); quase todos os modelos que discutimos usam linhas de código como métrica (apenas um usa outra métrica, os *function points*). Linhas de código é a métrica mais usada pois nenhuma outra métrica apresenta, ainda, superioridade sobre ela, ou seja, nenhuma apresenta melhor correlação com o esforço de desenvolvimento [BOE84].

Historicamente gerentes de projetos têm se baseado em julgamento perito e analogia para estimar o tamanho de software; no entanto os resultados têm sido ruins; os três principais fatores para isto é que eles têm referências incompletas de experiências prévias, eles geralmente não estão familiarizados com todas as componentes do projeto e, principalmente, eles são geralmente otimistas e supõem que todo o pessoal de desenvolvimento têm capacidade semelhante à sua.

A técnica PERT de estimativa de tamanho sugerida em [PUT80] é mais objetiva do que a técnica acima. Ela consiste em que os

analistas que irão trabalhar no projeto estimem três valores de tamanho para cada um dos principais módulos que irão compor o sistema a ser desenvolvido: o menor número possível de linhas de código, o número mais provável de linhas de código e o maior número de linhas de código que cada módulo deverá ter:

$a$ é o valor otimista
$m$ é o valor provável
$b$ é o valor pessimista de linhas de código

Com estes valores pode-se estimar o total esperado de linhas de código, bem como a magnitude da incerteza associada à estimativa; isto é feito através das fórmulas:

$$LOC = \frac{(a + 4m + b)}{6} \quad \text{, é o valor esperado} \qquad (eq.\ 3.48)$$

$$\sigma = \frac{(b - a)}{6} \quad \text{, é o desvio padrão} \qquad (eq.\ 3.49)$$

As fórmulas acima são obtidas a partir da distribuição estatística assumida pela análise PERT.

De acordo com a fase do projeto os analistas poderão dividir o projeto em um número cada vez maior de módulos e estimar separadamente $a$, $b$ e $m$ para cada módulo. Fazendo isto o desvio padrão da estimativa irá diminuir à medida que a especificação funcional do projeto ficar mais detalhada. As fórmulas abaixo indicam como encontrar o valor total do tamanho do software e do seu desvio padrão a partir das estimativas parciais:

$$LOC_{total} = \sum LOC_i \qquad (eq.\ 3.50)$$

$$\sigma_{total} = \sqrt{\Sigma \sigma_i^2} \qquad (eq.\ 3.51)$$

onde: $LOC_i$ e $\sigma_i$ é o número de linhas de código e o desvio padrão do i-ésimo módulo que compõe o sistema

Esta abordagem vem de encontro a um dos passos de estimativas que é detalhar o projeto o maximo possível (vide Seção 2.7.4), pois se usarmos as fórmulas acima veremos que quanto mais módulos

são estimados menor é o erro relativo total (desvio padrão dividido pelo valor estimado).

Apesar de mais adequada que o julgamento perito, este tipo de estimativa de tamanho do *software* ainda não é totalmente satisfatória; Boehm [BOE81] faz várias críticas a esta abordagem; a maior falha desta técnica é que ela, tal qual o julgamento perito, é altamente subjetiva, e o problema da estimativa otimista continua presente, apesar de que menos intensamente que no julgamento perito.

Uma outra forma de se estimar o tamanho do *software* em linhas de código é procurar usar outras métricas para calcular o número de linhas de código. Esta outra métrica deve estar disponível cedo no processo de desenvolvimento e deve utilizar alguma fórmula que relacione linhas de código a ela. Existem vários trabalhos que fazem este tipo de relação; entre eles os trabalhos de Curtis, Sheppard e Milliman ( [CUR79a] e [CUR79b] ) que relacionam a métrica de complexidade de Halstead [HAL79] e de McCabe [MCC76] com LOC, e o trabalho de Albrecht e Gaffney [ALB83] que relacionam *function points* com LOC. Por exemplo, a contagem do número de *functins points* (vide próxima seção) está relacionada com o número de LOC pelas fórmulas abaixo para as linguagens COBOL e PL/I [ALB83] :

$$LOC = 118{,}7 \cdot F - 6400 \quad , \text{COBOL} \qquad (eq.\ 3.52)$$

$$LOC = 73{,}1 \cdot F - 4600 \quad , \text{PL/I} \qquad (eq.\ 3.53)$$

Como já foi discutido, as medidas de complexidade não estão disponíveis cedo no processo de desenvolvimento do *software*; entretanto, em um estudo apresentado em [CON85] , é mostrada uma relação bastante simples entre a métrica de número de operandos de uma linguagem $\eta_2$ e LOC ,como é mostrado abaixo:

$$LOC = 3{,}61 \cdot \eta_2 \qquad (eq.\ 3.54)$$

Note que as duas métricas citadas acima estão disponíveis bastante cedo no processo de desenvolvimento do *software*, pois $F$ é a contagem do número de pontos de interação externa (*interfaces*) e

de arquivos lógicos, e $\eta_2$ é a contagem das estruturas de dados, do *software* a ser desenvolvido. Apesar disto, estas metodologias de medida de LOC ainda não são confiáveis, pois os resultados apresentados foram obtidos em ambientes restritos, no caso de Albrecht as fórmulas foram obtidas a partir de *software* de pequeno e médio porte para aplicações comerciais; no caso de Conte a fórmula foi obtida a partir de experimento em ambiente universitário com *softwares* de pequeno porte. Esta área é no entanto promissora tanto para estimativa de tamanho, como para a própria estimativa de custo de *software*.

### 3.4.2 Contagem dos Function Points

A contagem dos *functions points* é realizada em três fases:

FASE 1: Classifica-se e conta-se a presença dos cinco tipos de funções listadas abaixo no *software*:

1-Entrada externa
2-Saída externa
3-Arquivo lógico interno
4-Arquivo de interface externa
5-Consulta ("*inquiry*") externa

FASE 2: Multiplicam-se as funções acima por um coeficiente de ajuste de acordo com as suas complexidades. A Tabela 3.14 mostra os valores de ajuste.

Se somarmos os valores calculados nesta fase obtemos os

functions points não ajustados ou FC.

$$FC = \sum_{j=1}^{5} \left( \sum_{i=1}^{3} n_{ji} . c_{ji} \right) \qquad \text{(eq. 3.55)}$$

onde : $n_{ji}$ é o número de funções do tipo j com complexidade do tipo i

$c_{ji}$ é o fator de multiplicação das funções $n_{ji}$

FASE 3 : Ajusta-se a contagem dos *function points* , em ±35% , de acordo com a complexidade do processamento a ser feito. A fórmula abaixo ilustra o processo:

$$FP = FC \times CAP \qquad \text{(eq. 3.56)}$$

, onde CAP é o coeficiente de ajuste da complexidade de processamento: $0,65 < CAP < 1,35$

$$CAP = 0,65 + (0,01 \times PC) \qquad \text{(eq. 3.57)}$$

,onde PC é o grau de influência da complexidade.

*Tabela 3.14* Fatores de Ajuste da Complexidade das Funções

| FUNÇÃO | COEFICIENTE DE AJUSTE | | |
|---|---|---|---|
| | simples(1) | média(2) | complexa(3) |
| Entrada externa (1) | 3 | 4 | 6 |
| Saída externa (2) | 4 | 5 | 7 |
| Arq. lógico interno (3) | 7 | 10 | 15 |
| Arq. interface externa (4) | 5 | 7 | 10 |
| Pergunta externa (5) | 3 | 4 | 6 |

O FC é encontrado através da fórmula:

$$FC = \sum_{i=1}^{14} DI_i \qquad \text{(eq. 3.58)}$$

, onde DI é o grau de influência da i-ésima variável que influencia o processo de desenvolvimento (estas variáveis são discutidas na Seção 3.2.2).

Para finalizarmos a discussão sobre *function points* deve-se ressaltar que esta medida é independente de tecnologia e é realizada contando-se funções externas ou facilmente visualizáveis; isto faz que seja uma medida menos vaga que a contagem de linhas de código, principalmente nas fases inicias do desenvolvimento do software [DRU85]. Outro ponto importante é que já na contagem considera-se a influência de variáveis de ajuste que, nos modelos de estimativa que usam LOC, só são consideradas a posteriori; é difícil dizer se esta abordagem é vantajosa pois não existem trabalhos sobre o assunto. A contagem dos *function points* é mais trabalhosa já que necessita da classificação não só das variáveis de ajuste mas também da complexidade das funções contadas (em [DRU85] são considerados cinco ao invés de três categorias de complexidade); entretanto, quando executado nas fases iniciais do *software*, este esforço é bom para o processo de estimativa pois certamente ajuda a compreender melhor o produto a ser desenvolvido. Já existem ferramentas de estimativas comerciais que usam os *function points* nos seus modelos [RUB83]; todavia o uso desta métrica ainda está sob discussão e sofre críticas [KNA86] e [SYM88].

### 3.4.3 Contagem de Tamanho em Software Reaproveitado

A maioria dos modelos pressupõe que todas as linhas de código do software a ser projetado são completamente desenvolvidas; entretanto, é comum a produção de *softwares* que consistem de uma parte nova e uma parte já desenvolvida anteriormente e adaptada para este novo produto. É claro que as linhas de código reaproveitadas deverão ser contadas pois é necessário esforço para adaptá-las ao novo ambiente.

Os efeitos de linhas de código reaproveitadas sobre o esforço de desenvolvimento são manipulados pelo COCOMO através do cálculo do número equivalente de linhas de código (ELOC), que é usado no lugar de LOC nas equações de estimativa do COCOMO. Para calcular o ELOC é necessário que se estimem as seguintes grandezas de adaptação:

- ALOC ou LOC adaptadas - é o número estimado de LOC adaptadas de softwares existentes, para formar um novo produto.

- PM ou percentagem modificada do projeto - é a percentagem do projeto do software que é modificada de forma a se adaptar aos novos objetivos e ambiente da aplicação sendo desenvolvida.

-CM ou percentagem de código modificada - é a percentagem de código do software que é modificada de forma a se adaptar aos novos objetivos e ambiente.

- IM ou percentagem de integração requerida para o software modificado - É a percentagem de esforço necessário para integrar o software adaptado ao produto total e, também, o esforço extra necessário no teste do produto resultante devido aos softwares adaptados.

A seguir são mostradas as equações para calcular FAA (o fator de ajustamento da adaptação), e com ele, o EDSI (número

equivalente de instruções fontes entregues).

$$FAA = 0,40\ (PM) + 0,30\ (CM) + 0,30\ (IM) \qquad (eq.\ 3.59)$$

$$EDSI = (ALOC) \times (AAF\ /\ 100) \qquad (eq.\ 3.60)$$

PM, CM e IM deverão ser estimados através da experiência dos projetistas, e ALOC será uma estimativa semelhante à estimativa de LOC, só que agora com dados mais concretos, pois o projetista já possui o software a ser adaptado.

Como exemplo ilustrativo suponha que se quer calcular o esforço para se converter um programa de análise de circuitos eletrônicos de 50 KDSI em FORTRAN de um UNIVAC 1110 para um IBM 3033. Para esta situação teríamos:

- PM = 0, pois nenhuma mudança no projeto do programa seria necessária.
- CM = 15, talvez 15% das linhas seriam mudadas devido às diferenças do compilador, das interfaces com o sistema operacional, da linguagem de controle de *job*, etc.
- IM = 5, pois uma pequena quantidade de esforço seria requerida para integrar as mudanças acima.

Assim obteríamos :

FAA = 0,4 ( 0 ) + 0,3 ( 15 ) + 0,3 ( 5 ) = 6

ELOC = 50000 ( 6/100 ) = 3000 linhas

A abordagem acima é válida apenas para linhas de código. Para a métrica de *function points* não existe nenhum trabalho publicado sobre contagem de tamanho de *software* reaproveitado; nada impede, entretanto, que uma contagem semelhante à acima possa ser desenvolvida. A única grandeza que deveria ser revisada seria CM, pois seria necessário desenvolver uma metodologia para estimar a percentagem de *function points* modificados.

Infelizmente não existem trabalhos que discutem a eficiência da técnica de contagem de LOC adaptadas; porisso ela deve ser usada com cautela.

# CAPÍTULO 4

# O SISTEMA DE AUXÍLIO AO PLANEJAMENTO DO DESENVOLVIMENTO DE SOFTWARE

# S A P D E S

No Capítulo 2 discutiu-se o que é estimativa de custo de *software* e quais as formas de realizar e avaliar esta estimativa. No Capítulo 3 discutiram-se os principais modelos de estimativa de custo de *software* e algumas das ferramentas que implementam estes modelos. Neste capítulo será apresentado o SAPDES e como a engenharia de conhecimento é aplicada neste, ou seja, este capítulo trata de dois temas relacionados, mas distintos:

- Primeiramente apresenta-se a concepção inicial do Sistema de Auxílio ao Planejamento do Desenvolvimento de Software - o SAPDES. Explicam-se os critérios de projeto do SAPDES e sua estrutura funcional é mostrada. Descreve-se como será realizada uma estimativa no SAPDES, como os dados estimados são apresentados e como ele poderá calibrar modelos de estimativa. Finalmente compara-se a concepção funcional do SAPDES com a de outras ferramentas de estimativa de custo de *software*.

- Na segunda parte, discute-se o papel da engenharia de conhecimento no SAPDES e apresenta-se este assunto sob a ótica do sistema que foi desenvolvido. São mostradas as características do SBC[1] que foi desenvolvido e discute-se como o conhecimento pode ser representado e usado sob a forma de regras e *frames*. Discute-se ainda a ligação do SBC com programas convencionais e como o SBC poderá processar conhecimento incerto.

---

[1] sistema baseado em conhecimento

O *SISTEMA DE AUXÍLIO AO PLANEJAMENTO DE DESENVOLVIMENTO DE SOFTWARE* (SAPDES) é um sistema de estimativa de esforço, custo, tempo e cronograma de desenvolvimento de *software*. O SAPDES incorpora um conjunto de ferramentas para auxiliar o usuário a realizar estas tarefas de forma automática. Esta seção se dedica à discussão conceitual de como o SAPDES realizará esta função e quais requisitos ele deverá cumprir para realizá-la de forma eficiente.

### 4.1.1 Requisitos Exigidos do SAPDES

Como mostrou-se no Capítulo 2, estimar os indicadores quantitativos chaves de um projeto de *software* não consiste simplesmente em aplicar um modelo algorítmico de estimativa e confiar cegamente nos resultados obtidos. A necessidade de encarar a estimativa como um mini-projeto e dividi-la em vários passos é discutida na Seção 2.7 deste trabalho.

Os três primeiros passos de uma estimativa (ESTABELECER OS OBJETIVOS, PLANEJAR QUAIS DADOS E RECURSOS SÃO NECESSÁRIOS , e ESTABELECER OS REQUISITOS DO *SOFTWARE*) não são realizados através do SAPDES. A única contribuição do sistema a estes passos, é que, pelo fato de ser uma ferramenta automática, o SAPDES facilita aos usuários planejar os dados que serão necessários para estimativa; na realidade, um dos seus módulos recomenda quais modelos algorítmicos são mais aplicáveis ao problema, usando, entre outras informações, a capacidade do usuário em obter e fornecer informações para determinados modelos (vide Seção 5.3.1). Nada impede, entretanto, que no futuro a ferramenta venha a englobar módulos específicos para auxiliar nestes três passos. Estes módulos deveriam ser baseados em conhecimento e deveriam realizar

o papel de conselheiro perito[2] para auxiliar o usuário do sistema na realização destas tarefas.

A concepção do SAPDES mostrada nesta dissertação pretende auxiliar na realização dos quatro últimos passos de uma estimativa:

- Detalhar o Projeto o Máximo Possível
- Aplicar Diferentes Técnicas de Estimativa
- Comparar e Interagir Estimativas
- Acompanhar Estimativa X Projeto Real

Para cumprir o primeiro destes passos ("Detalhar o Projeto o Máximo Possível") a solução concebida foi a criação de um módulo de descrição do *software*, onde o sistema a ser desenvolvido possa já ser descrito segundo as métricas que serão utilizadas futuramente nas técnicas de estimativa. Este módulo realiza uma descrição detalhada do sistema através de decomposição com descrição verbal e o cômputo das métricas usadas pelos modelos para cada um dos componentes do sistema.

A solução concebida para realizar o segundo passo ("Aplicar Diferentes Técnicas de Estimativa") de forma automatizada foi integrar no SAPDES diversos modelos algorítmicos compostos (vide Seção 2.5). Os modelos compostos se caracterizam pela composição da abordagem algorítmica com o julgamento perito. O SAPDES pretende que estes modelos possam ser aplicados de forma *top-down* e *bottom-up*. Desta forma, o SAPDES integrará as quatro principais técnicas de estimativa citadas na Seção 2.4; são elas: *modelos algorítmicos*, *julgamento perito*, *top-down*, *bottom-up*. No protótipo inicial o *julgamento perito* ainda será executado pelo usuário pois este, para aplicar os modelos compostos, terá que responder a perguntas que exigem conhecimento perito (caracterização do *software*, seu modo e ambiente de desenvolvimento). As implementações destes modelos permitirão, entretanto, que conselheiros peritos possam ser integrados ao sistema visando a exigir no futuro um conhecimento muito menor do usuário para a aplicação destes

---

[2] normalmente usa-se o termo em inglês "expert advisor"

modelos. É importante notar que, além de permitir a utilização de várias técnicas de estimativas, o SAPDES permitirá a utilização de vários modelos compostos de estimativa.

O terceiro passo ("Comparar e Interagir Estimativas") deverá ser realizado pela criação de módulos que auxiliem o usuário na compreensão dos resultados fornecidos pelas diversas técnicas e modelos de estimativa, e na interpretação das diferenças encontradas nestas estimativas (vide discussão na Seção 2.7.6). Com estas análises o usuário poderá optar entre fazer novas estimativas com abordagens diferentes, aceitar resultados de um dado modelo ou técnica de estimativa, ou compor os resultados das estimativas já feitas para gerar novos resultados (em uma abordagem mais ambiciosa pretende-se que o SAPDES auxilie o usuário também nestas operações).

O quarto passo ("Acompanhar Estimativa X Projeto Real") deve ser cumprido pela criação de uma base de dados de estimativas realizadas e de facilidades para reavaliação destas estimativas pelo SAPDES, sempre que dados mais precisos sobre o projeto puderem ser fornecidos.

Para realizar uma boa estimativa é necessário, além da aplicação dos quatro passos básicos citados acima, que o usuário consiga usar eficientemente os modelos e que estes modelos sejam aplicáveis ao projeto que se está estimando.

Para auxiliar o usuário inexperiente prevê-se, além da criação de telas de ajuda durante uma consulta, a criação de módulos capazes de aconselhar e facilitar o uso de cada modelo implementado no SAPDES. Pretende-se que, para utilizar um modelo, o usuário inexperiente tenha que responder a um conjunto maior de perguntas, porém com perguntas mais simples.

A implementação do SAPDES aponta para a necessidade da criação de um módulo que auxilie na escolha dos modelos e técnicas mais apropriadas para realizar estimativas corretas, considerando o tipo do projeto e os conhecimentos do usuário.

É fundamental que o SAPDES forneça facilidades de calibração de cada modelo implementado ao ambiente de desenvolvimento de uma empresa. Para isto é necessário que se possa criar uma base de dados sobre projetos concluídos por uma empresa e que esta base

possa ser usada pelo SAPDES para calibrar os modelos para o ambiente de desenvolvimento desta empresa.

A discussão realizada nesta seção estabelece um conjunto de requisitos de projeto para o SAPDES. Este conjunto de requisitos de projeto podem ser resumidos em:

- Eficiente interface com o usuário

- Capacidade de integração modular de modelos de estimativa

- Capacidade de escolha (ou orientação de escolha) dos modelos e técnicas apropriadas ao problema do usuário, usando para isto um diálogo inteligente com este usuário.

- Facilidade de utilização dos modelos e técnicas, possivelmente com uma interface amistosa e inteligente.

- Capacidade de validação de estimativas através da comparação dos resultados fornecidos por diferentes modelos e técnicas de estimativa.

- Capacidade de recalibração de modelos de estimativa a novos ambientes de desenvolvimento.

As seções que se seguem apresentam a estrutura funcional capaz de permitir ao SAPDES cumprir as exigências acima.

### 4.1.2 Estrutura Funcional do SAPDES

A figura abaixo apresenta a estrutura lógica que se deseja do SAPDES.

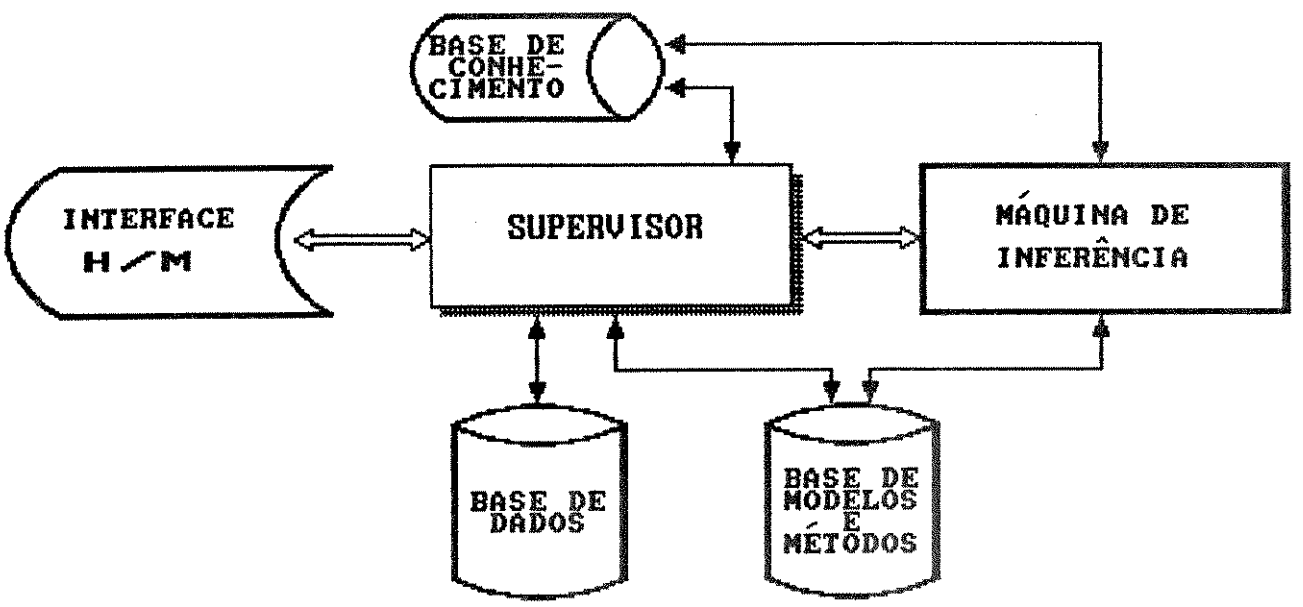


*Figura 4.1* Estrutura funcional do SAPDES

A estrutura funcional do SAPDES , mostrada na Figura 4.1 , deverá realizar as seguintes operações:

*a* - Fornecer ao usuário estimativas de custo, esforço e tempo de desenvolvimento, a partir do uso de diversos modelos e técnicas de estimativa (vide Seção 4.1.2).

*b* - Fornecer ao usuário facilidades para refazer estimativas de projeto conforme este evolua (vide Seção 4.1.2).

*c* – Permitir a verificação de dados de estimativas, ou de projetos completados, contidos na base de dados (vide Seção 4.1.3)

*d* – Permitir a descrição e caracterização de projetos prontos, para que se possa criar no sistema uma base de dados históricos (vide Seção 4.1.3)

*e* – Fornecer facilidades para recalibração dos modelos implementados para que eles se adaptem melhor a ambientes específicos de desenvolvimento (vide Seção 4.1.4)

Como é mostrado no esquema da Figura 4.1 , a estrutura funcional adequada ao SAPDES é baseada em conhecimento. Sistemas baseados em conhecimento são o tema da Seção 4.2 , onde se faz uma breve discussão sobre o assunto e se realçam os pontos do SAPDES onde conhecimento é utilizado.

É importante ressaltar a modularidade da abordagem, pois para se integrar um novo modelo ao sistema será necessário, apenas, colocá-lo na base de modelos e alimentar a base de conhecimento com regras sobre seu escopo de aplicação e metodologias de recalibração.

#### 4.1.2.1 Como Uma Estimativa é Realizada no SAPDES

A Figura 4.1 ajuda a compreensão do funcionamento do SAPDES. Através dessa figura pode-se entender como o sistema realiza as estimativas através dos seguintes passos:

1 - O sistema inicia o diálogo com o usuário através da interface homem-máquina. As primeiras perguntas procuram obter informações para que o mecanismo de inferência possa escolher os modelos adequados ao problema. Para realizar esta inferência o sistema usará o conhecimento sobre o escopo de aplicação de cada modelo contido na sua base de conhecimento.

2- Escolhido o modelo, o módulo supervisor deverá buscá-lo na base de modelos e técnicas, e passar a utilizá-lo. Para aplicar os modelos o usuário deverá fornecer dados necessários a estes através da interface homem-máquina.

3 - O modelo utilizado fornecerá estimativas que deverão ser colocadas na base de dados.

4 - Se houver mais algum modelo selecionado o supervisor repetirá os Passos 2 e 3.

5 - Terminadas todas as estimativas, o supervisor irá fornecer dados ao mecanismo de inferência para que este valide as estimativas a partir de regras sobre o assunto na base de conhecimento. Os dados fornecidos pelo supervisor, para esta validação serão: as grandezas estimadas e seus valores, os modelos utilizados, as características do projeto (já obtida no Passo 1 para escolha dos modelos) e, se disponível, valores históricos de produtividade de projetos desenvolvidos no mesmo ambiente (este valores já devem estar na base de dados do sistema).

6 - O sistema apresenta as estimativas ao usuário e/ou armazena-as na base de dados.

Para acompanhar projetos em andamento o SAPDES deve fornecer ao usuário do sistema facilidades para refazer estimativas, conforme os dados de entrada dos modelos de estimativa fiquem mais precisos. Para isto, o sistema deverá reaproveitar os dados fornecidos pelo usuário na estimativa anterior, e perguntar quais deles serão modificados nesta nova estimativa. A Figura 4.2 ilustra como o SAPDES auxilia o usuário na realização de reestimativas.

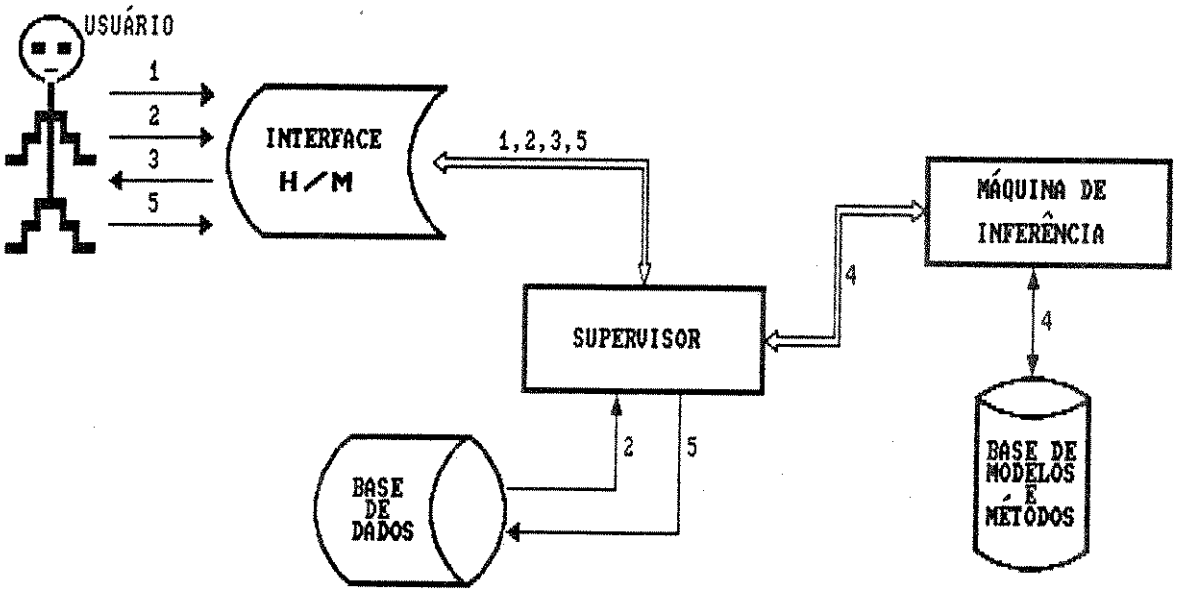


*Figura 4.2* **Reestimativa**

Conforme ilustrado na Figura 4.2 , o processo de reestimativa segue os seguintes passos:

1 - O usuário avisa ao SAPDES que deseja revisar uma estimativa.

2 - O SAPDES obtém o nome da estimativa com o usuário, e busca os dados sobre ela na base de dados.

3 - O SAPDES pergunta quais dos dados de entrada da estimativa o usuário deseja alterar.

4 - Com os dados alterados a estimativa é refeita e apresentada ao usuário.

5 - Caso o usuário deseje, a nova estimativa é quardada como a versão atual (a estimativa antiga é mantida ou apagada da base de dados conforme a vontade do usuário).

#### 4.1.2.2 Apresentação e Coleta de Dados

Após qualquer estimativa o SAPDES deverá apresentar os dados estimados ao usuário do sistema. A apresentação poderá ser em vídeo ou impressora (sob a forma de relatório). Na apresentação em vídeo a *interface* deve ser bastante amigável, preferencialmente usando menus e janelas. Na apresentação em impressora, os dados devem ser mostrados na forma de relatórios organizados em ítens para facilitar a leitura.

Os dados apresentados ao usuário deverão ser os valores estimados pelos modelos, os dados determinados pelo SAPDES para validação dos modelos e, finalmente, os valores estimados pelo SAPDES a partir desta validação.

O usuário pode desejar verificar os dados de uma estimativa contida na base de dados. Neste caso, o supervisor buscará na base de dados a estimativa desejada e realizará a apresentação conforme discutido nos parágrafos anteriores.

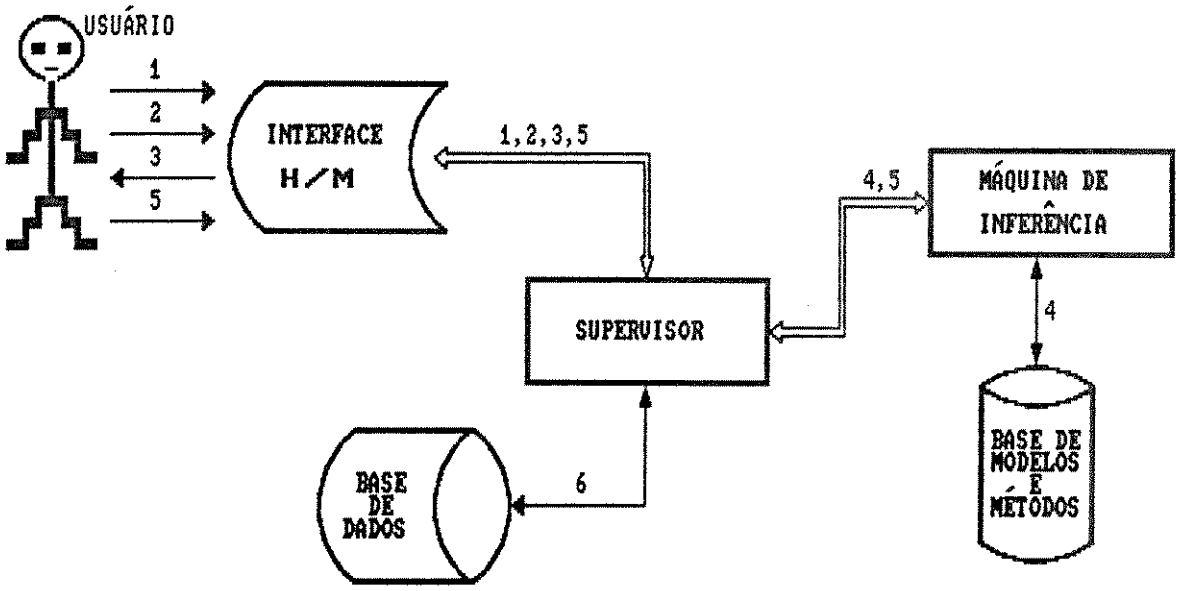


*Figura 4.3* Coleta de dados

Para realizar a calibração de modelos é fundamental que existam dados sobre projetos já concluídos na base de dados; desta forma, o SAPDES deve fornecer facilidades para se incluir descrições de projetos já concluídos na base de dados; o esquema mostrado na Figura 4.3 ilustra esta operação.

Na Figura 4.3 o processo de coleta de dados é descrito pelos seguintes passos:

1 - O usuário avisa ao sistema que deseja inserir a descrição de um projeto pronto na base de dados do sistema.

2 - O SAPDES obtém dados de caráter genérico sobre o projeto que vai ser descrito.

3 - O SAPDES pergunta ao usuário quais modelos de estimativa serão usados para descrever o projeto.

4- Escolhido o modelo, o módulo supervisor deverá buscá-lo na base de modelos e técnicas, e passar a utilizá-lo. Para isto o usuário deverá fornecer os dados necessários aos modelos através da interface homem-máquina.

5 - O modelo é aplicado como se fosse uma estimativa, só que, ao invés de estimar grandezas como esforço ou tempo de desenvolvimento, ele pergunta ao usuário sobre os valores reais destas grandezas no projeto.

6 - Os Passos 4 e 5 são repetidos até que todos os modelos selecionados tenham sido aplicados e os dados desejados coletados.

7 - O módulo supervisor guarda todos os valores obtidos na parte da base de dados reservada a projetos históricos e a operação é encerrada.

É importante notar que a descrição de projetos completados é coletada já segundo a ótica dos diversos modelos de estimativa; desta maneira o processo de calibração, discutido na Seção 4.1.2.3 , será bastante facilitado.

### 4.1.2.3 Análise de Dados e Calibração de Modelos

Segundo vários autores que utilizaram modelos de estimativa de custo de *sofware* ( [CUE87] , [KEM87] , [KIT85] e [RUB85] ), a calibração do modelo ao ambiente real de desenvolvimento é fundamental para o seu perfeito funcionamento.

O processo de calibração de modelos não é simples pois não consiste simplesmente em aplicar regressão a um conjunto de dados sobre projetos em mão e encontrar novos coeficientes para as fórmulas usadas nos modelos. Antes de tudo, é desejável fazer-se uma pré-seleção heurística de um sub-conjunto dos projetos concluídos contidos na base de dados, evitando que projetos heterogêneos ao grupo sejam considerados [BOE81]. Em segundo lugar, os processos de regressão usados pelos autores dos modelos variam muito: muitos linearizam as equações e fazem regressão com mínimos quadrados; outros minimizam o erro padrão da estimativa usando regressão não linear [BAI81]; outros, ainda, fazem considerações teóricas para realizar regressões por partes nas fórmulas que propõem [CON85].

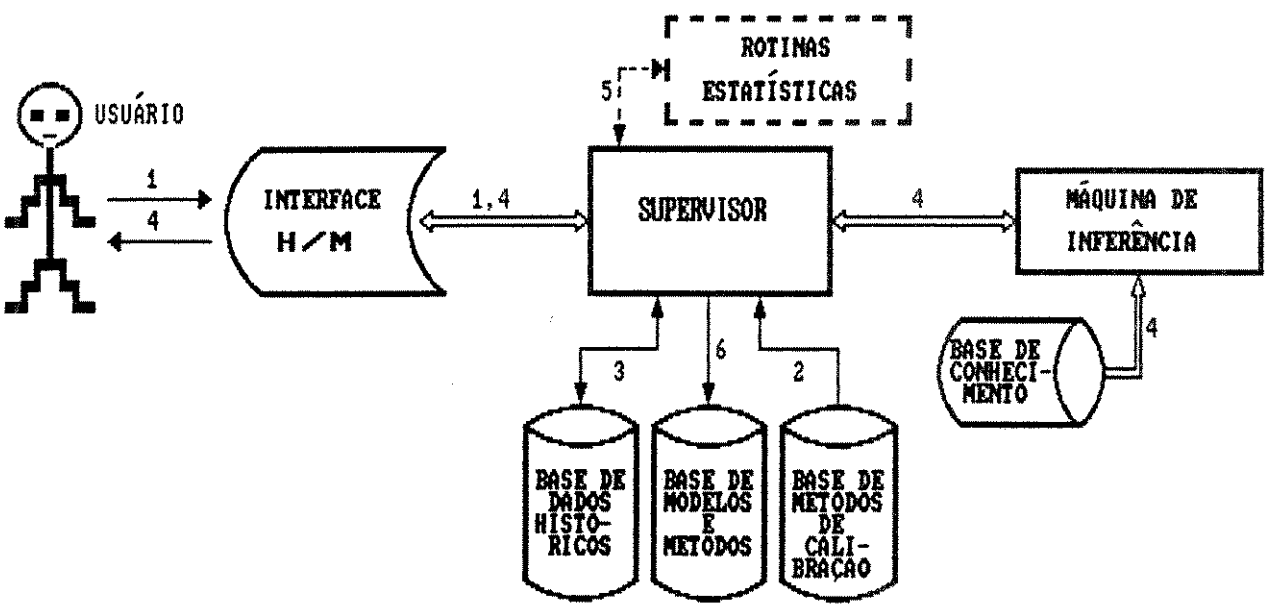


*Figura 4.4* Calibração de modelos

Para suportar os mais diversos tipos de calibração o SAPDES deve ter uma estrutura própria para estes objetivos, a Figura 4.1 está reescrita na Figura 4.4 de forma a evidenciar como o SAPDES pode suportar esta estrutura. O funcionamento do SAPDES durante a calibração de um modelo seguirá os passos abaixo:

1 - O usuário informa ao sistema que deseja calibrar um determinado modelo de estimativa.

2 - O módulo supervisor obtém junto à base de métodos de calibração a metodologia de calibração do modelo escolhido.

3 - O supervisor seleciona na base de dados os projetos prontos que foram inseridos usando o modelo de estimativa que se deseja calibrar.

4 - Dos projetos históricos existentes apenas um sub-conjunto significativo e homogêneo deve ser selecionado para a calibração. A seleção do sub-conjunto de projetos históricos é feita conjuntamente pelo SAPDES e usuário.

5 - O método de calibração do modelo é aplicado sobre o conjunto selecionado de projetos. Este método de calibração provavelmente terá que fazer uso de uma série de rotinas estatísticas ativadas pelo módulo supervisor.

6 - As novas fórmulas obtidas para o modelo de estimativa deverão ser agregadas a este na base de modelos e métodos de estimativa.

É importante notar que um método de calibração é desenvolvido especificamente para um determinado modelo, pois cada método fará uso da estrutura do modelo para realizar a calibração, assim como da forma de seu armazenamento na base de modelos e estimativas para alterar os coeficientes de suas fórmulas.

### 4.1.3 Comparando o SAPDES com outras Ferramentas

O SAPDES foi concebido para ser uma ferramenta de nível mais elevado do que as discutidas na Seção 3.3. Ele objetiva ser um sistema mais fácil de utilizar, mais eficiente nas estimativas, e mais adaptável à evolução natural dos modelos e técnicas de estimativas. Estas metas determinam as principais diferenças entre o SAPDES e outras ferramentas de estimativa de custo. A estrutura funcional mostrada na Seção 4.1.2 pretende:

- *Tornar o SAPDES mais adaptável ao ambiente do cliente*; este ponto consiste em fornecer facilidades de calibração da ferramenta ao ambiente de desenvolvimento do *software*. Ele é cumprido por algumas outras ferramentas.

- *Tornar o SAPDES adaptável ao problema do usuário*; a concepção do SAPDES pretende que ele integre vários modelos de estimativa e selecione aqueles que mais se adaptam ao problema do usuário. Este ponto é parcialmente cumprido por algumas poucas ferramentas; a ferramenta BYL [MAR88], por exemplo, permite que se use como métrica básica LOC ou *function points* conforme o desejo do usuário. Entretanto, o SAPDES é bem mais abrangente já que, além de poder conter uma grande gama de modelos diferentes, ele facilita a contínua evolução do módulo de seleção de modelos.

- *Facilitar o uso do SAPDES por usuários inexperientes*; o SAPDES englobará módulos baseados em conhecimento que ajudam o usuário na escolha de modelos, na aplicação dos modelos escolhidos ,e na interpretação e validação dos resultados obtidos. Neste ponto ele tem uma posição bastante diferenciada das outras ferramentas.

- *Facilitar a implementação de novos modelos no SAPDES*; o SAPDES foi concebido de forma que se possa acrescentar a ele novos modelos de estimativa. Ele tem uma concepção modular que permite que novos modelos sejam facilmente integrados à sua estrutura. Esta concepção contrasta com as outras

ferramentas que implementam um modelo ou uma combinação de modelos numa estrutura única.

- *Facilitar a avaliação de modelos usando o SAPDES*: está prevista no SAPDES a criação automática de bases históricas de projetos. Estas bases, associadas a módulos de análise estatística, permitiriam que comparações entre valores reais ocorridos e valores estimados por um determinado método possam ser estudadas com o intuito de avaliar o desempenho dos modelos sob diversas óticas.

A concepção funcional do SAPDES é potencialmente mais poderosa que a das ferramentas da Seção 3.3; todavia, é importante ressaltar que ao contrário das ferramentas citadas, o SAPDES é um protótipo em estágio ainda muito inicial que necessita ser utilizado em situações reais.

Um ponto importante é que, dada a sua facilidade de englobar novos modelos, o SAPDES poderá ser uma ferramenta de estudos de modelos e métricas de estimativa de esforço, tempo e custo de desenvolvimento de *software*. Desta forma o SAPDES poderá extrapolar o seu objetivo inicial, que é estimar estas grandezas, para se tornar uma ferramenta de estudo de modelos e técnicas de estimativas.

## 4.2 A ENGENHARIA DE CONHECIMENTO E O SAPDES

A engenharia de conhecimento tem papel fundamental no SAPDES esta seção dedica-se à discussão do assunto.

A engenharia de conhecimento diferencia-se basicamente do processamento de dados convencional por manipular conhecimento em vez de dados [WAT86]. A tabela abaixo apresenta as diferenças entre as duas abordagens:

*Tabela 4.1* Diferenças Entre o Processamento de Dados Convencional e A Engenharia de Conhecimento

| *Processamento de Dados* | *Engenharia de Conhecimento* |
|---|---|
| representa e utiliza dados | representa e utiliza conhecimento |
| usa algoritmos | usa heurística |
| processos repetitivos | processos inferenciais |
| manipula bases de dados | manipula bases de conhecimento |

O interesse que se tem dentro da engenharia de conhecimento para se construir o SAPDES está restrito à área de solução perita de problemas específicos relacionados à aplicação de modelos de estimativa de custo de *software*. Este tipo de aplicação está muito próximo do que se classifica como sistema especialista. Este tipo de sistema tem sua construção e uso representados na Figura 4.5a. Os sistemas especialistas são sistemas baseados em conhecimento que possuem conhecimento especializado objetivando resolver problemas dentro de um domínio de conhecimento. Em [WAT86] e [HAY83] pode-se obter informações adicionais sobre o assunto.

O SAPDES, todavia, não é somente um sistema especialista. A engenharia de conhecimento é usada apenas em uma parte deste sistema. O SAPDES combina o processamento convencional com o processamento heurístico sobre a seleção e o uso de cada um dos modelos de estimativa nele implementados. A Figura 4.5b representa o processo de contrução e uso do SAPDES.

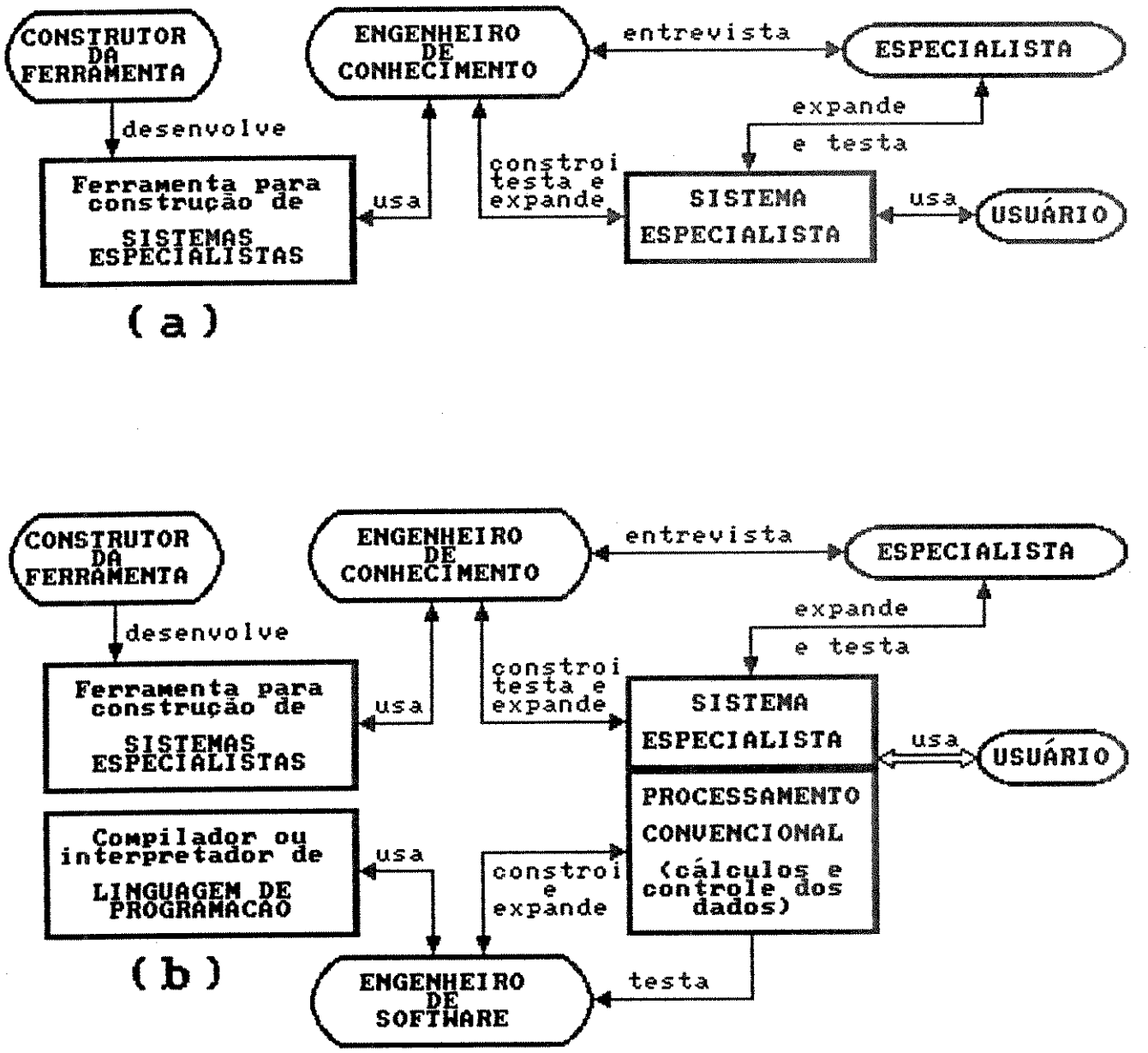


*Figura 4.5* (a)Construção e uso de um sistema especialista
(b)Construção e uso do SAPDES

A seção que se segue discute o papel da engenharia de conhecimento no SAPDES.

### 4.2.1 O Papel da Engenharia de Conhecimento no SAPDES

Na descrição do comportamento funcional do SAPDES, feita na Seção 4.1 , vê-se que a necessidade de uso de processamento heurístico é uma constante. A engenharia de conhecimento é necessária e apropriada nas seguintes partes do sistema:

- Durante a escolha dos modelos que mais se adaptam ao problema do usuário, deve-se usar conhecimento sobre o escopo de aplicação de modelo e sobre a capacidade do usuário em fornecer os dados necessários aos modelos.

- Durante o uso dos modelos pode-se usar conhecimento para ajudar e aconselhar o usuário inexperiente na realização desta operação.

- Durante a validação dos valores estimados, discrepâncias entre grandezas equivalentes estimadas por modelos e técnicas diversas podem ser interpretadas usando-se: o conhecimento obtido na escolha dos modelos que mais se aplicam ao problema, e o conhecimento sobre como se obter valores de compromisso a partir de valores estimados por técnicas diferentes (vide Seção 2.7.6).

- Durante o processo de calibração de modelos, pode-se usar conhecimento para selecionar quais projetos históricos contidos na base de dados formam um conjunto homogêneo e representativo do ambiente de desenvolvimento para o qual desejamos calibrar os modelos.

Na seção que se segue são apresentadas as características da ferramenta escolhida para o desenvolvimento do SAPDES. Nela se discute como o conhecimento é representado e utilizado, e como programas convencionais podem ser ligados ao sistema baseado em conhecimento que vamos desenvolver.

### 4.2.2 A Ferramenta de Desenvolvimento Escolhida

Como foi discutido na Seção 4.2.1 o sistema proposto aponta para o uso de sistemas baseados em conhecimento devido à necessidade de processamento heurístico na:

- Seleção dos modelos a serem aplicados
- Aplicação dos modelos por usuários inexperientes
- Seleção e validação de resultados
- Escolha de dados para calibração dos modelos

O uso de programas convencionais de computador é todavia inevitável devido a:

- Uso de cálculos matemáticos e tabelas de valores pelos modelos de estimativa.
- Necessidade de criar-se no SAPDES um sistema próprio de gerência de dados e estimativas.
- Necessidade de criação de módulos específicos de aquisição e apresentação de dados (módulos da *interface* homem/máquina).

A dualidade das duas abordagens acima indicava para a escolha de uma ferramenta que permitisse o desenvolvimento de um sistema baseado em conhecimento com capacidade de ativação de rotinas desenvolvidas de forma tradicional durante o processo de consulta; desta forma seria possível construir um sistema onde se pudesse manipular conhecimento e aplicar algoritmos convencionais sem perda de funcionalidade e desempenho.

Outra característica exigida da ferramenta de desenvolvimento era a estruturação modular do conhecimento de forma que:

- os processos de acrescentar e calibrar os modelos de estimativa no SAPDES fossem facilmente realizados.

- a alteração das regras de manipulação dos modelos e suas estimativas fosse simples.

A partir das exigências citadas, selecionou-se uma ferramenta de desenvolvimento de sistemas baseados em conhecimento[3] que permitiu o cumprimento destas exigências. Essa ferramenta possui as seguintes características:

- permite a criação de sistemas baseados em regras com possibilidade de encadeamento para trás e para frente, processamento de incerteza do projetista e usuário, processamento de meta-regras (regras para controle de regras) e *interface* amigável.

- permite que rotinas em LISP sejam facilmente integradas à base de conhecimento e ligadas a regras de forma a serem ativadas durante o processo de consulta.

- permite que conjuntos de regras sejam estruturados em *frames*, fazendo com que a estrutura global da base de conhecimento possa ser organizada em segmentos lógicos diferentes mas relacionados, facilitando a modularização dos modelos de estimativa e das estruturas de controle no SAPDES.

A seguir discutiremos os sistemas baseados em conhecimento sob a ótica do tipo de sistema que a ferramenta selecionada desenvolve. As próximas sub-seções descrevem como funcionam os sistemas baseados em conhecimento desenvolvidos com a ferramenta usada para construir o SAPDES.

---

[3] este tipo de ferramenta é frequentemente chamada de "SHELL" pois forma um ambiente que objetiva, não só fornecer um conjunto de facilidades para desenvolvimento de sistemas baseados em conhecimento, mas também suportar o sistema construido, oferecendo ao usuário uma série de recursos de questionamento e reavaliação do processo dedutivo realizado por este sistema.

### 4.2.2.1 Estruturas Básicas

A ferramenta de desenvolvimento escolhida constrói sistemas baseados em conhecimento com três estruturas básicas: *frames*, *regras e parâmetros*. Elas são usadas para organizar e controlar o uso das informações. Cada uma destas estruturas tem propriedades que definem suas características:

- O *frame* é uma estrutura criada para organizar uma coleção de diversos tipos de dados sobre um campo de conhecimento. Esta coleção inclui informações sobre domínio, estrutura e operação desta base de conhecimento. O *frame* é usado para agrupar parâmetros (fatos) e regras relacionados a um problema que se deseja resolver. Uma base de conhecimento pode ter mais de um *frame* com o intuito de dividir sua complexidade em estruturas lógicas menores e mais simples.

- Os parâmetros são partes individuais de conhecimento; no nosso sistema, por exemplo, *EXPERIÊNCIA_DOS_PROGRAMADORES* pode ser um parâmetro, e *ALTA* pode ser o valor atribuído a este parâmetro.

- As regras estabelecem relacionamentos entre parâmetros e conclusões sobre eles. A forma sintática de uma regra é uma declaração *se-então (if-then)*. A regra abaixo é um exemplo de uma declaração relacionada ao custo de codificação na desenvolvimento de *software*:

```
IF EXPERIÊNCIA_DOS_PROGRAMADORES = ALTA AND
   LINGUAGEM_DE_PROGRAMAÇÃO = FAMILIAR THEN
   PRODUTIVIDADE_NA_CODIFICAÇÃO = ALTA
```

Existem ainda outras estruturas muito importantes ao sistema; elas são as propriedades. As propriedades descrevem o comportamento dos *frames*, *regras*, *parâmetros* e outros objetos usados no sistema. Cada um destes objetos tem seu próprio conjunto de propriedades para descrever este comportamento.

#### 4.2.2.2 Como Funciona o SBC[4]

As estruturas descritas na Seção 4.2.2.1 devem ser organizadas de forma a produzir conclusões a partir de uma consulta do usuário à base de conhecimento montada por elas (no nosso caso estas conclusões são, em última instância, a estimativa de tempo e custo de desenvolvimento de um *software*). Para que o sistema funcione, o conhecimento organizado nestas estruturas devem formar um fluxo lógico que, durante uma consulta, conduza a uma conclusão. Para isto ocorrer, é necessário que se estabeleça pelo menos um *parâmetro meta*. A meta é um parâmetro cujo valor será a conclusão alcançada pela consulta ao sistema. Quando uma consulta ocorre, o objetivo do sistema é encontrar um valor para a meta; a busca pela meta é o ítem que comanda uma consulta. As metas são uma das propriedades dos *frames*. As primeiras metas são estabelecidas quando o *frame raiz* do sistema é instanciado.

##### 4.2.2.2.1 Como os Frames são Utilizados

Um frame é uma coleção de informações. Uma parte desta informação define a área do problema em forma de regras e parâmetros; a outra parte da informação define como operar este conhecimento com propriedades como metas, dados iniciais necessários e outros componentes básicos desta estrutura. Uma base de conhecimento contém pelo menos um *frame*, o *frame raiz*. Quando a base de conhecimento é grande e/ou complexa, podem-se criar frames adicionais. Estes frames são organizados de forma hierárquica. Para se compreender como funciona dinamicamente uma base de conhecimento com frames dois conceitos são importantes:

- A herança. Os frames contêm regras e parâmetros próprios.

---

[4] Sistema Baseado em Conhecimento

Eles são organizados em árvore com pais e filhos. Um frame filho herda os parâmetros e seus valores (fatos) de seus frames ancestrais. De forma inversa, os frames pais têm acesso às regras contidas nos frames descendentes.

- A instanciação. Quando um frame é ativado durante um processo de consulta à base de conhecimento diz-se que ele foi instanciado. Os frames e regras contidas na base de conhecimento são estruturas estáticas. Só quando instanciados tornam-se estruturas dinâmicas. Desta forma, um processo de consulta consiste de instanciação de frames e regras em busca de alguns *parâmetros meta*.

Quando uma sessão de consulta começa, o frame raiz é instanciado automaticamente. Os parâmetros meta do frame raiz passam a ser neste momento as metas da consulta. Para encontrar o valor destes parâmetros-meta o sistema usa regras num processo de encadeamento para trás, descrito na próxima seção. Os sub-frames (frames filhos do frame raiz) são instanciados quando, no curso normal de uma consulta (encadeamento para trás), suas regras são necessárias ou quando eles são explicitamente referenciados na parte de conclusão de uma regra instanciada e disparada em um frame ancestral. Quando um sub-frame é instanciado, seu ancestral é posto em espera e as metas deste novo sub-frame (se existirem) passam a ser as metas da consulta. Só após as metas deste sub-frame serem resolvidas o sistema buscará as regras úteis no processo de encadeamento do frame ancestral e retornará a ele. Este processo prossegue até o sistema encontrar os valores dos parâmetros-meta do frame raiz ou concluir que estes valores não podem ser encontrados a partir das regras e frames contidos na base de conhecimento; neste ponto a consulta se encerra e suas conclusões são apresentadas ao usuário do sistema.

#### 4.2.2.2.2 Aplicando Regras (encadeamento para trás)

Durante uma consulta, o sistema procura uma regra que atribua um valor para o parâmetro-meta, ou seja, uma regra que contenha o parâmetro-meta na sua parte de conclusão (*then*). Para conseguir aplicar a regra é necessário que o sistema obtenha os valores dos parâmetros que estão na parte de condição da regra (*if*). A fim de obter o valor destes parâmetros o sistema procura outras regras que os contenham na parte de conclusão. Desta forma, um ciclo de busca é estabelecido. Quando a parte condicional da regra é satisfeita a regra é disparada e sua parte de conclusão é aplicada estabelecendo valores para os parâmetros que estão ali. Caso se chegue a uma situação onde não há mais regras a se tentar, pois existem parâmetros cujos valores não podem ser estabelecidos pelas regras ativas, o sistema verifica que parâmetro é necessário e tem a propriedade de poder ter seu valor solicitado ao usuário. O sistema então questionará o usuário sobre o valor deste parâmetro e tentará disparar a regra na qual o parâmetro é condição necessária (*if*). Este processo é chamado de encadeamento para trás e termina quando se encerra a possibilidade de aplicação de regras para atribuição de valores ao parâmetro-meta (a cadeia de fatos esteja completa). A sequência algorítmica da descrição textual acima é mostrada abaixo:

1 - O sistema procura por uma meta na lista de *parâmetros-meta*; se a encontrou cria uma lista de parâmetros procurados, coloca-a como primeiro elemento desta lista, faz dela o *parâmetro-atual* e executa o passo seguinte; se não a encontrou, encerra a consulta.

2 - O sistema procura uma regra ativa que estabeleça um valor para o *parâmetro-atual*; se a encontrou continua; se não, vai ao Passo 7.

3 - O sistema tenta aplicar a regra encontrada verificando sua parte condicional; se alguns dos valores já encontrados para os parâmetros da parte condicional a tornam falsa, esta regra é marcada como inativa (não utilizável) e volta-se ao Passo 2; caso contrário, continua-se.

4 - Caso a regra não possa ser disparada porque algum parâmetro da sua parte condicional não tem valor estabelecido, este parâmetro vai para a lista de parâmetros procurados, passa a ser o *parâmetro-atual* e o sistema volta ao Passo 2. Caso contrário, continua-se.

5 - A regra é disparada e todos os parâmetros de sua parte de conclusão recebem os valores explicitados; se o *parâmetro-atual* não é um *parâmetro-meta*, ele é retirado da lista de parâmetros procurados, o parâmetro anterior nesta lista é recuperado, passa a ser o *parâmetro-atual* e volta-se ao Passo 2 ; se o *parâmetro-atual* é um *parâmetro-meta*, continua-se.

6 - O usuário é informado que um *parâmetro-meta* foi encontrado, e o sistema volta ao Passo 1.

7 - O sistema verifica se o valor de *parâmetro-atual* pode ser perguntado ao usuário; se não, vai ao Passo 10; se pode, continua-se.

8 - O sistema pergunta ao usuário o valor do *parâmetro-atual*; caso este parâmetro pertença à lista de *parâmetros-meta* o sistema vai ao Passo 6; caso contrário, continua-se.

9 - O sistema estabelece o valor do *parâmetro-atual* e retira-o da lista de parâmetros procurados, pega o parâmetro anterior nesta lista e volta ao Passo 2.

10 - O sistema verifica se o *parâmetro-atual* pertence à lista de *parâmetros-meta*; se pertence, ele informa ao usuário que aquele *parâmetro-meta* não pode ser encontrado, retira-o da lista de *parâmetros-meta* e volta ao Passo 1; se não pertence, retira-o da lista de parâmetros procurados, marca todas as regras que o possui na parte condicional (*if*) como inativas (não utilizáveis), pega o parâmetro anterior da lista de parâmetros procurados e volta ao Passo 2.

#### 4.2.2.2.3 Aplicando Regras (encadeamento para frente)

No encadeamento para frente uma regra é tentada toda vez que um parâmetro de sua parte condicional recebe seu valor. Uma base de conhecimento que usasse exclusivamente encadeamento para frente não teria parâmetros meta. Usualmente este tipo de processamento é usado como complemento ao encadeamento para trás. Em sistemas que usam encadeamento para frente e para trás, as regras onde o encadeamento para frente deve ser tentado recebem a propriedade especial de regra antecedente. Este tipo de encadeamento é usado pelos seguintes motivos:

- criar informação ainda não requerida no encadeamento para trás com a previsão que esta informação irá encurtar o processo de busca da meta.

- encurtar o caminho para uma meta quando determinadas condições de um problema geram um caminho lógico bem definido para se encontrar esta meta; por exemplo, pode-se criar uma regra do tipo: *IF* FATO_A *AND* FATO_B *AND* FATO_C *THEN* SOLUÇÃO_1

- evitar perguntar o valor de um grande número de parâmetros que não serão úteis futuramente; por exemplo, se o parâmetro *TIPO_DO_SOFTWARE_A_SER_DESENVOLVIDO* recebe o valor *TEMPO_REAL*, uma regra antecedente pode estabelecer valores para vários outros parâmetros evitando que o sistema pergunte sobre parâmetros relevantes apenas a outros tipos de *software*.

#### 4.2.2.2.4 OUTRAS CARACTERÍSTICAS

Uma característica importante da ferramenta escolhida é realizar processamento de incerteza. Pode-se associar a parâmetros um fator de certeza quanto ao seu valor; isto facilita a criação de ambientes onde o conhecimento incerto ou duvidoso está presente. A associação de incerteza ao valor de um parâmetro se dá quando ele recebe este valor; isto ocorre em duas ocasiões:

- *Quando o usuário fornece o valor do parâmetro em resposta a uma pergunta do sistema.* Neste caso, o coeficiente de certeza deve ser fornecido junto com a resposta do usuário (para que a pergunta permita que um coeficiente de certeza seja associado ao valor do parâmetro fornecido pelo usuário, uma propriedade especial deve ser atribuída a este parâmetro).

- *Quando o parâmetro recebe o valor por disparo de uma regra.* Neste caso, o fator de certeza do valor associado ao parâmetro é estabelecido pelo processamento dos coeficientes de certeza da parte condicional e de conclusão da regra. A incerteza da parte condicional da regra é calculada de acordo com os fatores de certeza de cada parâmetro contido nesta parte. O fator de certeza da parte de conclusão está explicitamente declarado na regra.

Outra característica importante da ferramenta de desenvolvimento escolhida é a de permitir que programas em linguagens de programação convencionais possam ser ativados durante um processo de consulta à base de conhecimento. Esta operação é especialmente simples com a linguagem LISP, já que a ferramenta de desenvolvimento executa sobre uma máquina virtual LISP.

Até agora falamos em regras do tipo,

*IF condição THEN conclusão* ,

ou seja, dado um número de condições então estabeleça um conjunto

# CAPÍTULO 5

# IMPLEMENTAÇÃO DO SAPDES

No capítulo anterior discutiu-se a concepção funcional do SAPDES e as características da ferramenta selecionada para o seu desenvolvimento. Neste capítulo apresenta-se como a estrutura funcional mostrada na Seção 4.1 (Fig. 4.1) pode ser implementada com a ferramenta de desenvolvimento mostrada na Seção 4.2.2 .

Neste capítulo mostram-se inicialmente os principais módulos do SAPDES e discute-se sua organização no sistema. A seguir cada um destes módulos é discutido em detalhes, as suas funcões são descritas e um exemplo de seu funcionamento é mostrado. Os módulos apresentados são: implementação dos modelos de estimativa, seleção dos modelos de estimativa, coleta de dados iniciais, gerência de dados, seleção de resultados e apresentação de resultados.

## 5.1 A ARQUITETURA IMPLEMENTADA

Uma vez escolhida a ferramenta de desenvolvimento que seria utilizada, a arquitetura do SAPDES mostrada na Figura 4.1, foi reestruturada para a forma mostrada na figura a seguir.

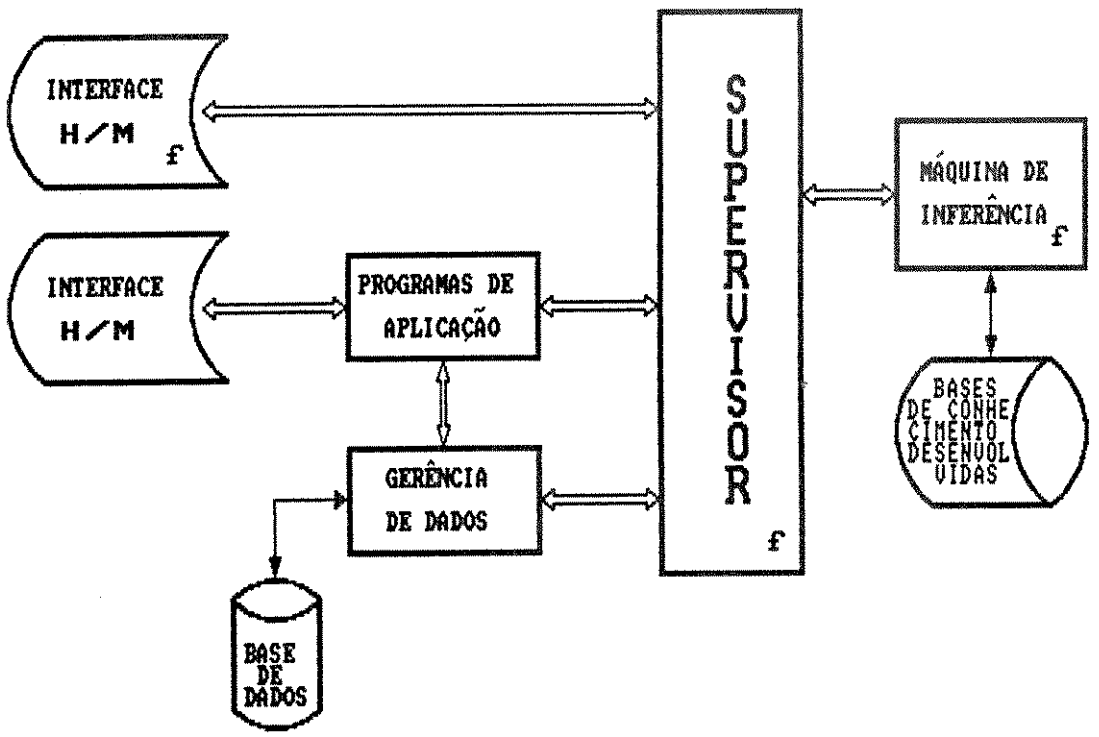


*Figura 5.1* Arquitetura implementada do SAPDES

O esquema acima mostra o sistema como está sendo implementado. Os blocos com a letra *F* representam funções já fornecidas pela ferramenta de desenvolvimento. A filosofia de funcionamento desta arquitetura é a mesma discutida no Capítulo 4; as diferenças entre as Figuras 4.1 e 5.1 podem ser explicadas da seguinte forma:

1 - O bloco base de modelos e métodos da Figura 4.1 foi englobado pela base de conhecimento pois, na realidade, esta é

composta de regras organizadas em *frames* que contêm o conhecimento inserido no sistema (inclusive os modelos e as técnicas de estimativas). Os modelos são implementados na base de conhecimento em *frames* separados.

2 - O gerenciador de dados acrescentado ao esquema é o responsável pela organização e tratamento para armazenagem dos dados estimados.

3 - O bloco 'programas de aplicação' contém programas em LISP ativados a partir do supervisor. Estes programas realizam funções não adequadas a sistemas baseados em conhecimento (por exemplo, cálculos algébricos associados a cada modelo de estimativa).

4 - A interface homem/máquina foi dividida em duas, explicitando que: uma parte desta interface já existia pois é feita automaticamente pelo mecanismo de inferência/supervisor baseado no conteúdo da base de conhecimento; uma outra parte foi desenvolvida visando criar facilidades de consulta às bases de dados (estimativas e projetos históricos) e facilidades para descrição inicial do *software*.

## 5.2 ESTRUTURA DA BC[1] e MÓDULOS DE PROCESSAMENTO CONSTRUÍDOS

A Figura 5.2 contém a estrutura da base de conhecimento e programas que compõem, ou comporão (em tracejado)[2], o SAPDES. Os módulos com a letra *L* são desenvolvidos em LISP e os com letra *F* são *frames* da base de conhecimento. A estrutura mostra como estes módulos estão organizados entre si.

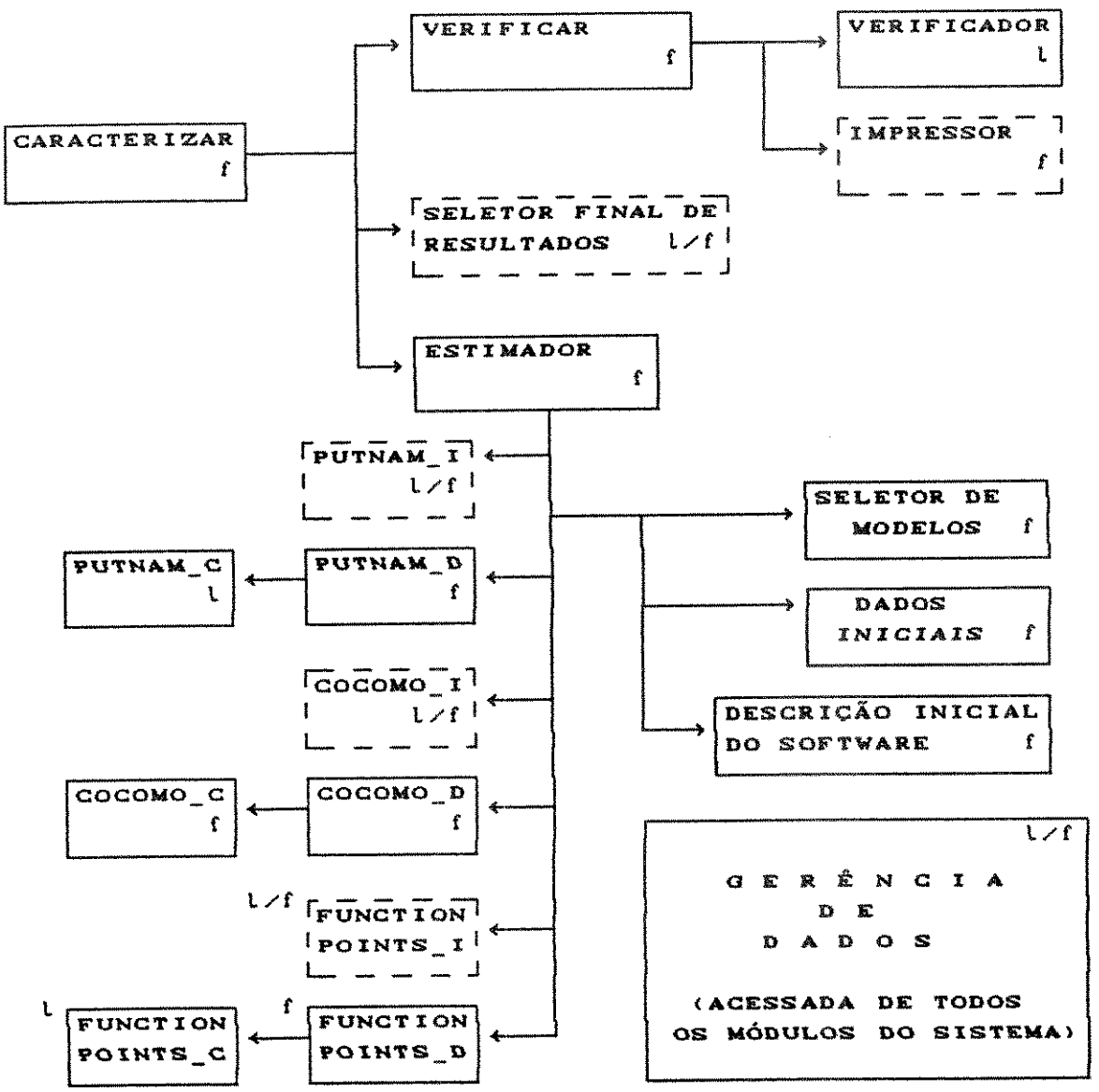


*Figura 5.2* Módulos que Compõem o SAPDES

---

[1] Base de Conhecimento

[2] Os módulos em tracejado são citados pois já têm seu comportamento funcional bem definido.

Os blocos mostrados na figura anterior são descritos resumidamente a seguir:

- Frame CARACTERIZAR: é o *frame* raiz do sistema; sua função é apenas verificar a operação desejada pelo usuário e sequenciar esta operação. Por exemplo, no caso do usuário querer realizar uma estimativa este *frame* vai ativar os *frames* SELETOR_DE_MODELOS , ESTIMADOR , SELETOR_FINAL_DE_RESULTADOS e finalmente VERIFICADOR.

- Frame SELETOR_DE_MODELOS: este *frame* tem como objetivo selecionar os modelos que melhor se aplicam à estimativa de um determinado projeto. Para isto ele considera as características do projeto , a capacidade do usuário de fornecer dados aos modelos e o desejo do usuário em usar ou não determinados modelos.

- Frame ESTIMADOR: sua função é sequenciar a aplicação de modelos selecionados; sua importância está no fato de viabilizar o uso de diferentes estratégias de estimativas onde os dados gerados por um modelo podem ser utilizados por outros.

- Frame DADOS_INICIAIS: Sua função é obter alguns dados iniciais comuns aos modelos e outros necessários ao início da aplicação de determinados modelos.

- Programa DESCRIÇÃO_INICIAL_DO_SOFTWARE: é um programa que fornece uma interface amigável para descrição o mais detalhada possível do projeto. Está descrição é feita para encontrar o valor das métricas básicas usadas pelos modelos de estimativa (LOC no caso do COCOMO e do modelo de Putnam, e *function points* no caso do modelo de Albrecht).

- Frame *modelo*_D: é o *frame* que usa conhecimento para determinar as variáveis de influência de projeto usadas pelo *modelo* (vide Seção 3.2). Estas variáveis são geralmente relacionadas com as seguintes características do projeto: tipo do *software* a ser desenvolvido, pessoal que atuará no projeto, e ambiente e facilidades de desenvolvimento.

- Programa *modelo_C*: baseando-se nos dados obtidos e deduzidos pelo *frame modelo_D*, este programa realiza os cálculos para determinar os valores que queremos estimar.

- Frame *modelo_I*: este *frame* é ativado no caso em que se queira descrever os dados obtidos em um projeto pronto sob o prisma de determinado modelo. Ele perguntará sobre as variáveis de influência de projeto e os valores reais de esforço, tD, cronograma de desenvolvimento, etc; permitindo colher os dados para avaliar futuramente os modelos.

- Frame SELETOR_FINAL_DE_RESULTADOS: seleciona resultados das estimativas de diversos modelos e cria uma única estimativa do sistema; para isto considera quais resultados de quais modelos são mais relevantes.

- Frame VERIFICAR: é ativado para apresentação de dados da base de dados ou da consulta sendo feita naquele momento.

- Programa VERIFICADOR: é uma interface H/M amigável destinada a apresentação de dados no vídeo. É ativada pelo *frame* VERIFICAR.

- Programa IMPRESSOR: apresenta os resultados de uma consulta ao SAPDES sob a forma de relatório impresso.

- Programa GERÊNCIA_DE_DADOS: controla os dados fornecidos e guardados pelo SAPDES, fazendo formatação, armazenamento e fornecimento destes dados quando requisitado por outros programas. Realiza, também, gravação, leitura e remoção dos dados em disco conforme requisitado.

### 5.3.1 Seleção dos Modelos a Serem Aplicados

Como já discutido, a seleção dos modelos é feita no *frame* SELECIONA_MODELO; a seleção usa regras que consideram a aplicabilidade do modelo a um projeto, a capacidade do usuário de aplicar o modelo, e o que o modelo pode fornecer ao usuário; cada um destes critérios são explicados abaixo:

- A aplicabilidade do modelo a um projeto é determinada considerando-se a adequação do modelo àquele tipo de projeto, ou seja, se o projeto a ser desenvolvido está dentro do universo de projetos ao qual o modelo se aplica. Neste escopo são importantes critérios como porte, aplicação e nível tecnológico do projeto para o qual os modelos foram construídos.

- A capacidade do usuário em aplicar um modelo determina se o usuário tem conhecimento suficiente para responder às questões que serão necessárias para aplicação do modelo, ou se uma determinada abordagem é mais fácil para ele.

- O que o modelo pode fornecer ao usuário deve ser considerado porque alguns modelos fornecem muitas outras informações além de esforço, tempo e cronograma de desenvolvimento. Como foi mostrado, alguns modelos fornecem considerações sobre documentação a ser produzida, distribuição de carga de trabalho por atividade e fase do projeto, análises de riscos associados, etc.

A atividade de seleção de modelos é baseada fundamentalmente no diálogo com o usuário. Ao usuário é dada a opção de escolher os modelos sem intervenção do sistema e, mesmo quando o usuário usa *frame* SELECIONA_MODELO como um conselheiro perito[3], os modelos selecionados têm sua aplicação apenas sugerida ao usuário para que este dê, então, a palavra final quanto aos modelos que serão utilizados em uma consulta ao SAPDES.

Os modelos selecionados para aplicação em uma estimativa são colocados como valor de um parâmetro chamado QUAIS_METODOS; este será usado no *frame* ESTIMADOR que é o responsável pela ativação dos modelos.

Como durante o seu funcionamento o *frame* SELECIONA_MODELO objetiva dar valor ao parâmetro QUAIS_METODOS, este parâmetro é o *parâmetro-meta* deste *frame* (vide Seção 4.2.2.2.1).

Na implementação atual do SAPDES o parâmetro QUAIS_METODOS pode assumir os valores COCOMO , PUTNAM e/ou FUNCTION-POINTS com diferentes fatores de certeza.

Nas Figuras 5.3 e 5.4 são mostrados dois diagramas representando parte do processo de seleção dos modelos. As figuras buscam apenas ilustrar como o processo de seleção dos modelos é feito, não sendo uma representação exata do processo implementado. Nas figuras são mostradas: questões apresentadas de alguma forma ao usuário (diagramas retangulares), e fatos que o sistema recebe como resposta às questões formuladas ou deduz a partir destas respostas (diagramas ovais). A dedução de fatos é mostrada de cima para baixo nos diagramas, isto é, os fatos procurados estão na parte de baixo dos diagramas. As perguntas mostradas nos diagramas retangulares não são necessariamente feitas diretamente ao usuário; o SAPDES pode fazer uma série de perguntas mais simples que permirá a ele concluir a respostas às perguntas mais complexas.

A Figura 5.3 mostra simplificadamente o processo decisório para determinar se o usuário tem capacidade de aplicar o COCOMO.

---

[3] usualmente usa-se o termo em inglês, expert advisor

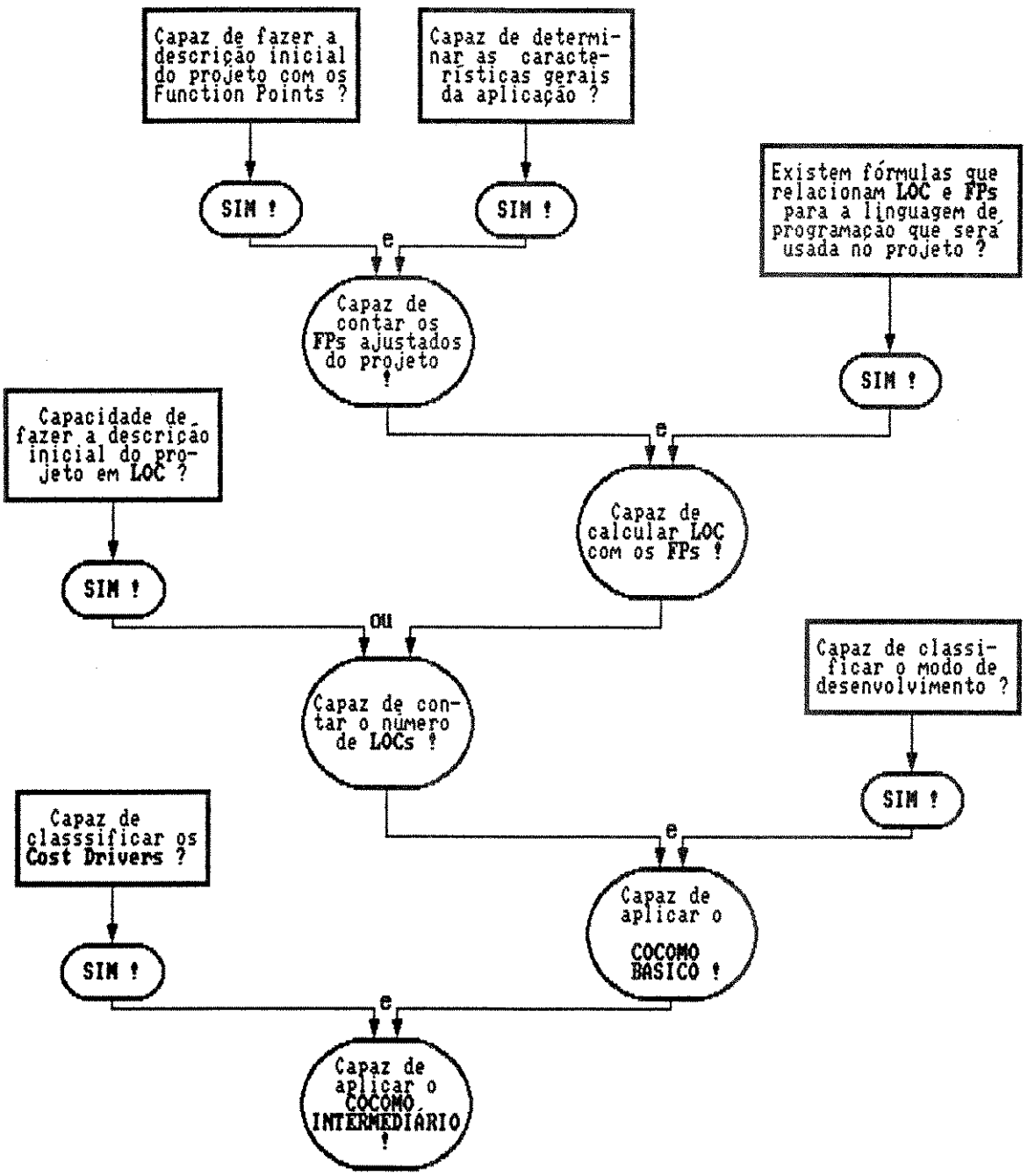


*Figura 5.3* Processo decisório para determinar a Capacidade do Usuário de Aplicar o COCOMO

A Figura 5.4 mostra simplificadamente o processo decisório para determinar a aplicabilidade do modelo de PUTNAM a um projeto. Nesta figura estão associados fatores de certeza a alguns fatos. Estes fatores permitem que o processo decisório para se encontrar o fato 'Aplicabilidade do Modelo de Putnam ao Projeto' seja reali-

zado com índices de incerteza. Na Figura 5.4 o fato acima é deduzido a partir da composição ponderada de outros três fatos: 'Tamanho Típico de Projeto Para o Modelo de Putnam', 'Classe de Aplicação Típica do Modelo', e 'Localidade Típica de Projeto Para o Modelo de Putnam'.

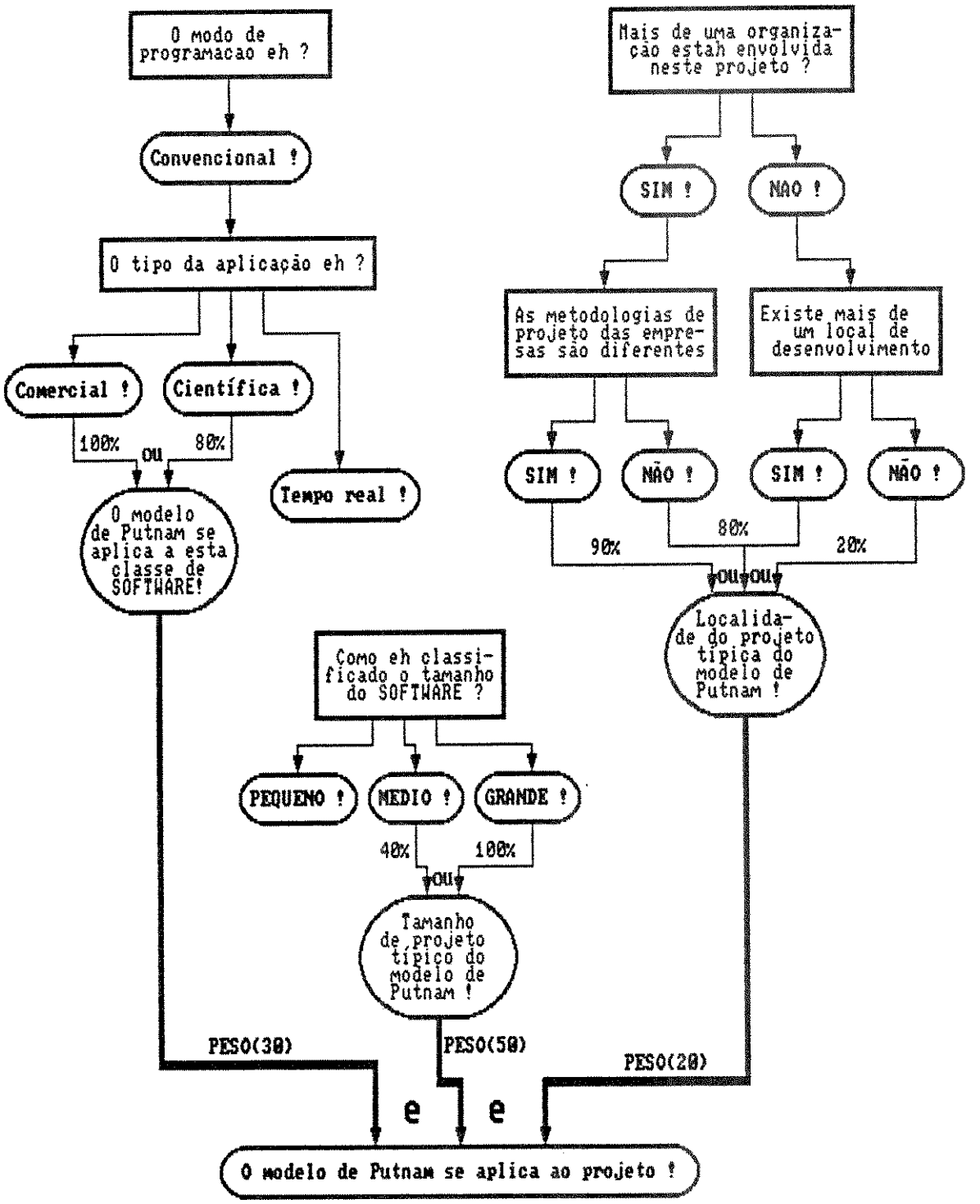


*Figura 5.4* Processo decisório para determinar a Aplicabilidade do Modelo de Putnam a um Projeto

O *frame* SELECIONA_MODELOS deverá ser alterado sempre que:

- um novo modelo for acrescentado ao sistema: esta alteração visará acrescentar regras ao sistema para que se possa aconselhar aplicação do modelo a determinados projetos.

- um modelo for calibrado a determinado ambiente: esta alteração visará alterar o critério de aplicabilidade do modelo, já que esta aplicabilidade também depende dos tipos e de homogeneidade dos projetos históricos usados na sua calibração.

As Figuras 5.5(a,b,c) são exemplos de algumas das telas que solicitam informações do usuário durante uma instanciação do frame SELECIONA_MODELOS; a Figura 5.5(d) é a tela que apresenta ao usuário os modelos de estimativas recomendados.

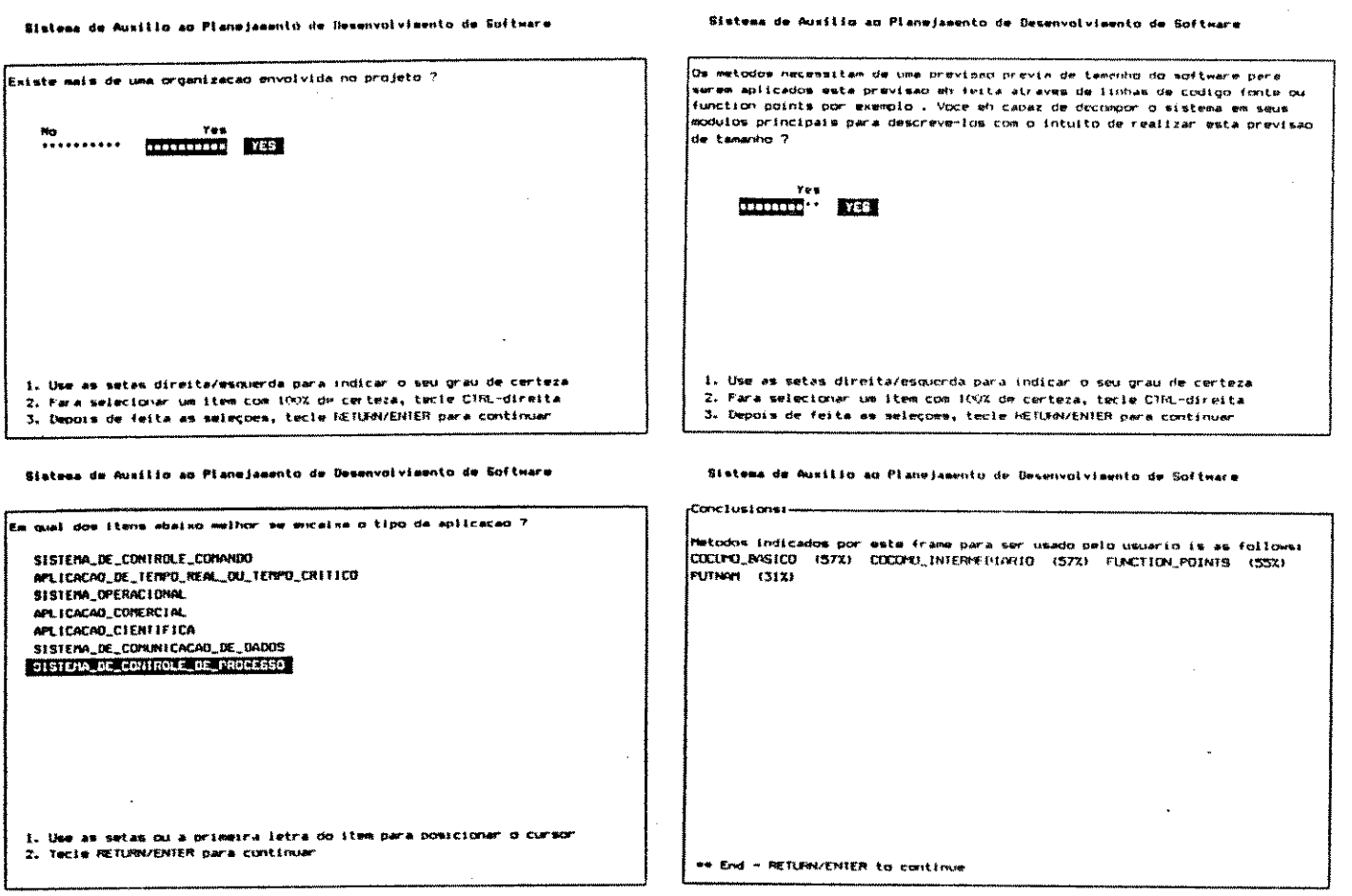


*Figura 5.5 (a,b,c)* **Telas Solicitando Dados ao Usuário**
*(d)* **Modelos Indicados Pelo Frame**

### 5.3.2 Coleta de Dados Iniciais

Na Seção 2.7.4 do Capítulo 2 discutimos um dos principais passos para se estimar corretamente, qual seja, *detalhar o projeto o máximo possível*. Na Seção 3.4 do Capítulo 3 discutimos as métricas usadas para estimativa de tamanho do *software* e sua importância para o correto funcionamento dos modelos de estimativa de custo. O detalhamento do projeto e a estimativa do seu tamanho são atividades relacionadas; no SAPDES um módulo é dedicado a auxiliar o usuário na realização destas atividades. O programa DESCRIÇÃO_INICIAL_DO_SOFTWARE tem o objetivo de auxiliar o usuário a realizar uma descrição *top-down* do projeto. Este programa irá facilitar a decomposição do projeto para sua descrição e para estimativa do seu tamanho em LOC ou *function points*. Durante a decomposição, o usuário deverá descrever cada sub-módulo do sistema. A decomposição deve ir até o nível desejado pelo usuário. Durante a decomposição o usuário realiza a estimativa de tamanho dos sub-módulos do sistema. A estimativa de tamanho pode ser:

- Em LOC: neste caso quando o módulo descrito é um módulo folha (não se decompõe mais), o usuário deve estimar o valor máximo, mínimo e provável de linhas de código que aquele módulo deverá ter; quando o módulo descrito se decompõe em outros, o usuário deve estimar o valor máximo, mínimo e provável de linhas de código que aquele módulo deverá ter para realizar o processamento não considerado nos seus sub-módulos.

- Em *function points*: neste caso deverão ser descritos todos os arquivos de *interface* externa, entradas, saídas, arquivos lógicos internos e consultas externas. Cada uma destas funções podem ser descritas textualmente e devem ser classificadas pelo usuário em simples, média ou complexa. Deve-se ter cuidado para não se descrever duas vezes a mesma função em módulos diferentes pois neste caso haverá dupla contagem dos FP's associados a esta função.

A Figura 5.6 representa como o projeto do *software* é descrito

em módulos. Para cada módulo deve-se fornecer no mínimo as seguintes informações:

1. Nome do módulo
2. Finalidade do módulo
3. Sub-módulos associados ao módulo (decomposição)
4. Estimativa de valor otimista, pessimista e provável de LOC para o módulo, e/ou, descrição dos arquivos de *interface* externa, entradas, saídas ,arquivo lógicos internos e perguntas externas do módulo.

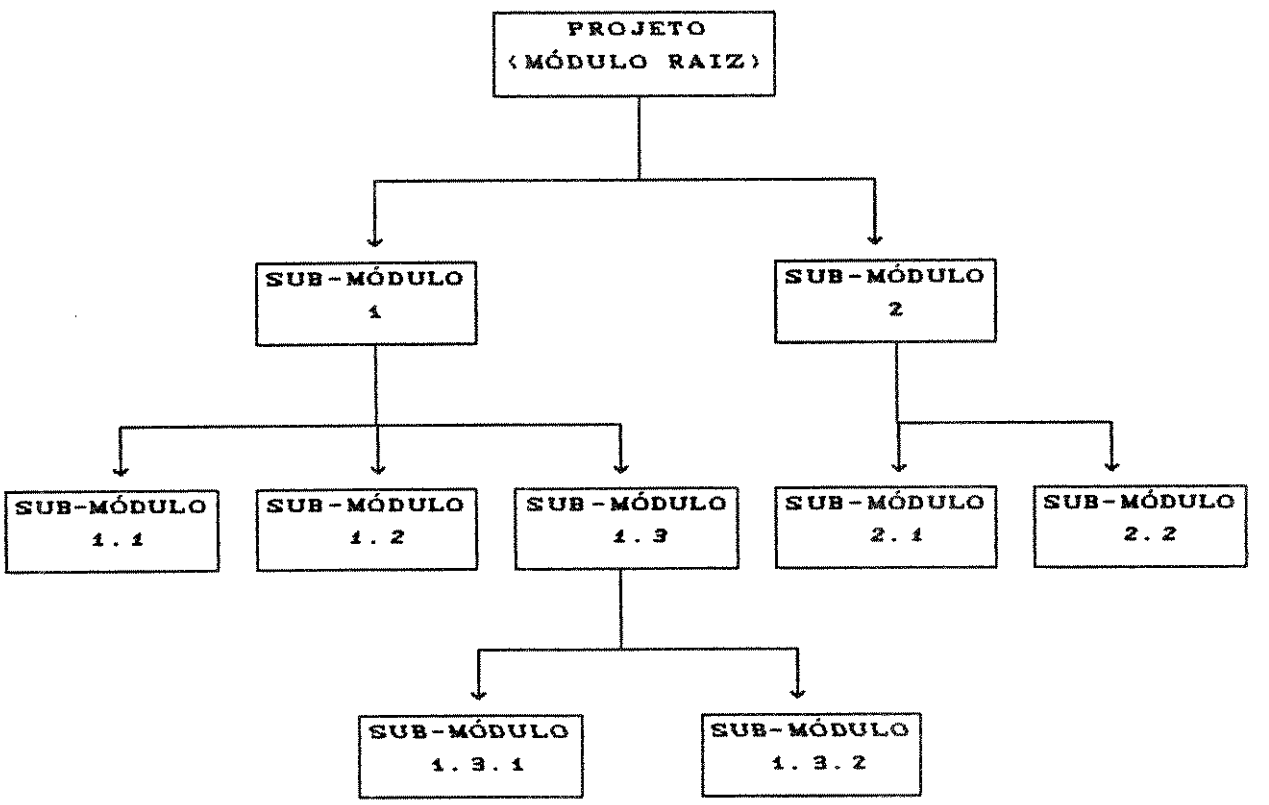


*Figura 5.6* Descrição inicial do projeto em módulos

Toda a operação de descrição inicial do projeto é realizada com uma interface amigável que permite a visualização gráfica da decomposição. O programa verifica se a especificação está completa e automaticamente pede os dados que faltam. Este programa fornece

ainda um conjunto de rotinas básicas de alteração e apresentação da descrição do projeto.

A figura 5.7 apresenta em sequência cronológica algumas das telas de diálogo com o usuário durante a descrição inicial do *software*.

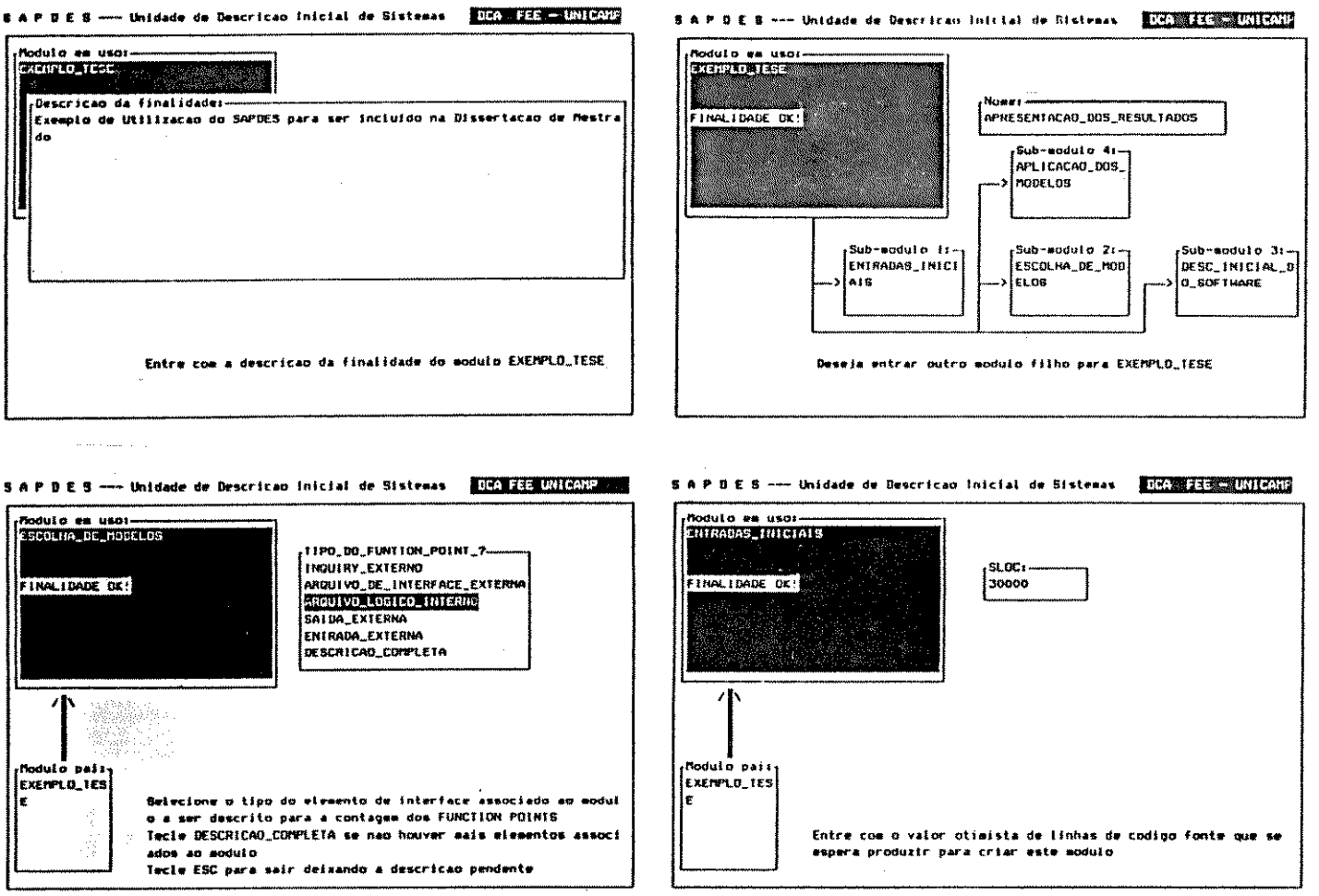


*Figura 5.7(a,b,c,d)* **Telas Usadas Durante a Descrição Inicial do Projeto**

Após a descrição, o cômputo total do tamanho do projeto em LOC e/ou *function points* é realizado e passado para os *frames modelo_D*.

Os outros dados básicos do projeto e da estimativa são coletados no *frame* DADOS_INICIAIS. Entende-se por dados básicos, os dados que se necessita conhecer em toda estimativa, tais como: fase do projeto, tipo do projeto, nome do usuário do sistema, etc. Este *frame* é responsável pela correta ativação do programa DESCRIÇÃO_INICIAL_DO_SOFTWARE para que este use *function points* ou LOC, conforme os modelos de estimativa escolhidos pelo usuário.

### 5.3.3 Como os Modelos são Implementados

Na Figura 5.2 foram mostradas a estrutura dos programas e a base de conhecimento no SAPDES. Nesta estrutura cada modelo foi dividido em três módulos:

- *modelo*_D
- *modelo*_C
- *modelo*_I

Os dois primeiros módulos são usados para estimativa e o último para coleta de resultados de projetos prontos. A função destes módulos é descrita a seguir:

- *modelo*_D : é o módulo responsável pela coleta de dados para aplicação dos modelos; este processo consiste no uso dos dados fornecidos pelo usuário do SAPDES de forma heurística para determinar as variáveis de influência no processo de desenvolvimento do *software*. Em outras palavras, este módulo é responsável pela determinação do coeficiente de tecnologia e gradiente de dificuldade no modelo de PUTNAM, pela determinação do modo de desenvolvimento e *COST DRIVERS* no COCOMO, e pelas características da aplicação no modelo de análise por *Function Points*.

- *modelo*_C : é o modulo responsável por programas que realizam todas as funções de processamento de dados que não são realizadas pelo módulo acima, notadamente cálculos e processamento matemático. Os programas contidos neste módulo são ativados pelo módulo *modelo*_D ou pelo *frame* ESTIMADOR e suas funções são totalmente dependentes dos modelos que ele irá aplicar.

- *modelo*_I : sua função é coletar dados com o intuito de criar uma base histórica para avaliação e calibração dos modelos. Para isto, este módulo realiza a função combinada dos outros dois módulos, determina as variáveis de influência

no processo de desenvolvimento do *software* como no módulo *modelo_D* e coleta os valores reais das grandezas que seriam estimadas no módulo *modelo_C*. Este módulo é aplicável sempre que o usuário selecionar no *frame* CARACTERIZAR a opção INSERIR_PROJETO_PRONTO.

Os três módulos acima são totalmente dependentes dos modelos. Sua separação permite a criação e inserção de outros modelos no sistema de forma simples. O módulo *modelo_C* é particularmente funcional pois ele pode conter qualquer tipo de rotina computacional útil a um modelo, e o sistema permite que esta rotina possa ser chamada de qualquer *frame* ou programa. O módulo *modelo_D*, além da função de coleta de dados, é fundamental na organização do sistema por ser em última instância o responsável pela ativação do modelo de estimativa. Quando se quer ativar o modelo do COCOMO, por exemplo, ativa-se o *frame COCOMO_D*. Todo o processo de aplicação do modelo, obtenção e envio dos resultados ao módulo de gerência de dados, é feito sob a coordenação deste módulo.

### 5.3.3.1 Implementação do COCOMO

O Modelo Construtivo de Custo está implementado em dois módulos: COCOMO_D e COCOMO_C . A aplicação do modelo segue as etapas abaixo:

- Estimativa do número de linhas de código do *software* a ser desenvolvido, realizada pelo módulo DESCRIÇÃO_INICIAL_DO_SOFTWARE (vide Seção 5.3.2).

- Determinação do modo de desenvolvimento do *software*, realizada pelo *frame* COCOMO_D.

- Determinação dos coeficientes de ajuste de esforço realizado pelo *frame* COCOMO_D.

- Cálculo do esforço e do tempo de desenvolvimento usando as fórmulas mostradas na Equação 3.43 e na Tabela 3.10 , realizado pelo módulo COCOMO_C.

- Cálculo da distribuição de esforço por tempo e atividade de desenvolvimento, realizado pelo módulo COCOMO_C.

A determinação do modo de desenvolvimento é feita a partir de diálogo com o usuário do sistema para a determinação das características de classificação mostradas na Tabela 3.11. A Figura 5.8 mostra a solicitação de dados sobre a experiência do usuários com projetos semelhantes.

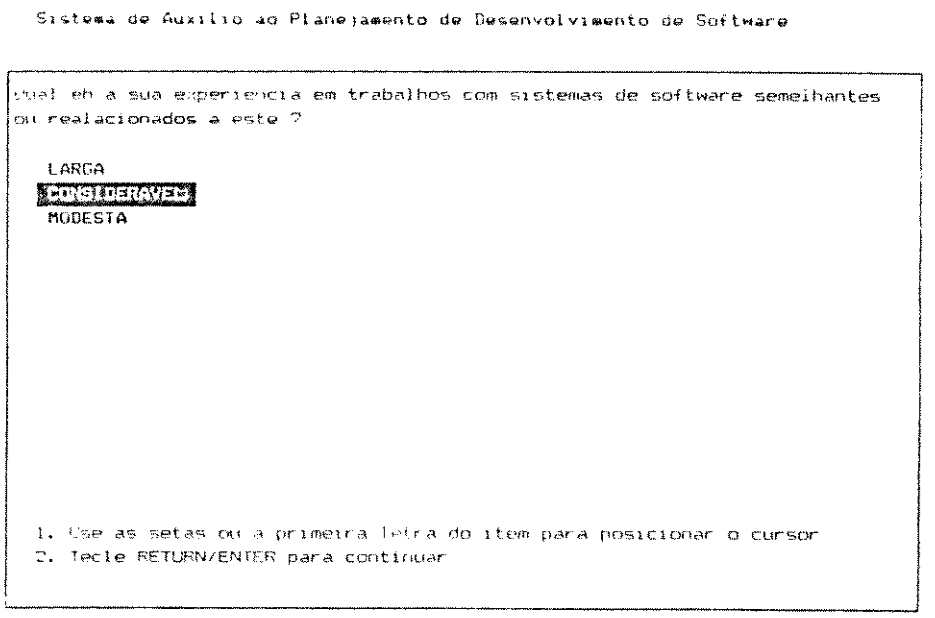


*Figura 5.8* **Tela Solicitando Dados Para Determinação do Modo de Desenvolvimento do COCOMO**

A determinação do coeficiente de ajuste de esforço é feita a partir de diálogo com o usuário para a determinação dos valores dos *Cost Drivers* mostrados nas Tabelas 3.12 e 3.13. A Figura 5.9 mostra a solicitação da classificação do *Cost Driver* 'Uso de Práticas Modernas de Programação' (MODP) pelo SAPDES.

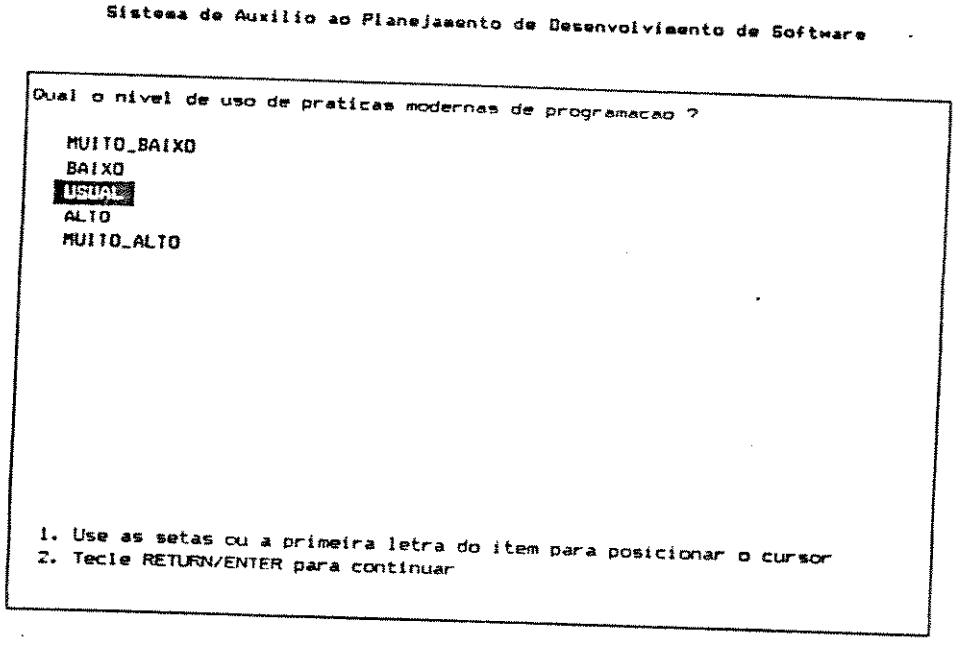


*Figura 5.9* **Tela Solicitando Classificação do COST DRIVER 'Uso de Práticas Modernas de Programação'**

A distribuição de esforço por tempo e por atividade de desenvolvimento é determinada através de tabelas que mostram a distribuição típica do esforço por tempo (ou atividade) de acordo com o tamanho e o modo de desenvolvimento do *software*. Quase sempre, necessita-se interpolar os valores nas tabelas para se determinar a distribuição de esforço. Estas tabelas podem ser encontradas em [BOE81]; no SAPDES elas estão armazenadas no arquivo COCOMO.TAB para facilitar alterações em casos de futuras calibrações do modelo a novos ambientes.

Todos os resultados relevantes são gravados, durante a aplicação do modelo, através do sistema de gerência de dados.

### 5.3.3.2 Implementação do Modelo de Análise por FP

Este modelo está implementado de forma bastante simples; a sua aplicação é feita seguindo os passos abaixo:

- Contagem dos *function points* do *software* a partir da descrição hierárquica feita no módulo DESCRIÇÃO_INICIAL_DO_SOFTWARE (vide Seção 5.3.2).

- Determinação do grau de presença no projeto do *software* das características gerais da aplicação (vide Seção 3.2.1), e determinação do fator de ajuste dos *function points* pela fórmula 3.3.2. Estas operações são realizadas pelo *frame* FUNCTION_D.

- Aplicação das equações 3.28 , 3.29 e 3.30 para determinação do esforço de desenvolvimento do *software*, realizada pelo *frame* FUNCTION_D.

A contagem dos *function points* é mais complexa que a contagem de LOC's (vide Seção 3.4.2); por este motivo a determinação do tamanho do *software* é mais complexa neste modelo.

As outras partes da aplicação deste modelo são bastante simples já que nesta implementação ele estima apenas o esforço de desenvolvimento.

A determinação do grau de presença no projeto do *software* das características gerais da aplicação é determinado através de diálogo com o usuário. A Figura 5.10 mostra a tela usada pelo SAPDES para determinar o grau de presença no projeto da característica geral da aplicação 'Código Projetado para Reutilização'.

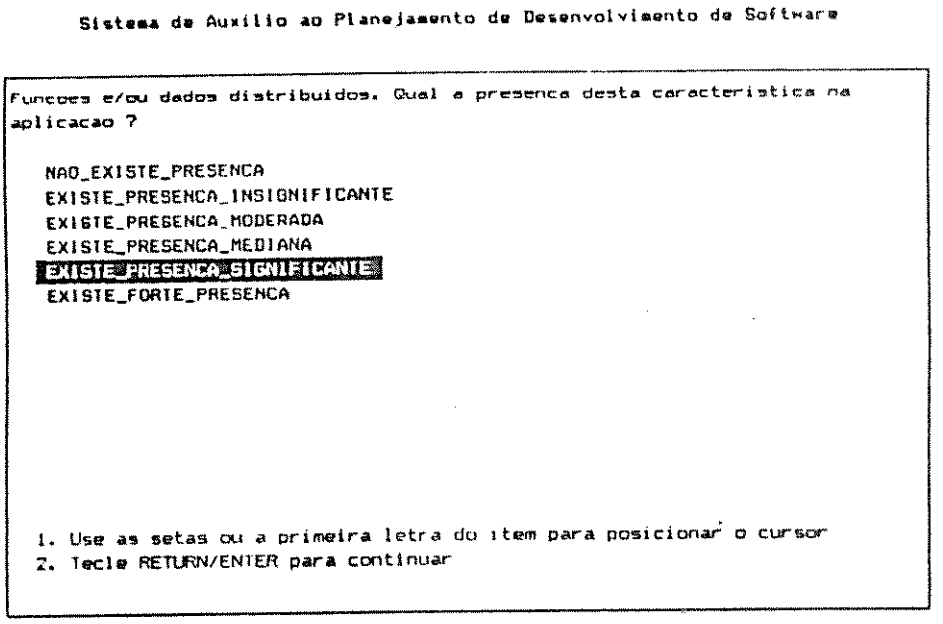


*Figura 5.10* **Tela Solicitando Classificação de uma das Características Gerais da Aplicação dos FPs**

Como os cálculos matemáticos reduziram-se a aplicar a equação 3.32 para encontrar os *function points* ajustados e as equações 3.28 , 3.29 e 3.30 para determinar o esforço de desenvolvimento, não foi necessário criar um módulo FUNCTION_C. A simplicidade dos cálculos permitiu que neste modelo eles fossem realizados por funções matemáticas agregadas diretamente ao *frame* FUNCTION_D.

A gravação dos dados relevantes é feita de forma semelhante à feita no modelo anterior.

### 5.3.3.3 Implementação do Modelo de Alocação de Recursos

O modelo de Putnam está implementado em dois módulos; são eles: PUTNAM_D e PUTNAM_C. A aplicação do modelo é feita nas seguintes etapas:

- Estimativa do número de linhas de código do *software* a ser desenvolvido, de forma idêntica à estimativa feita para o COCOMO.

- Determinação do coeficiente de tecnologia por diálogo com o usuário (módulo PUTNAM_D) ou a partir de projetos anteriores (módulo PUTNAM_C), Figura 5.11(a).

- Determinação do gradiente de dificuldade $\nabla D$ do sistema a ser desenvolvido a partir de diálogo com o usuário (*frame* PUTNAM_D), Figura 5.11(b).

- Determinação dos valores máximo e mínimo de esforço e tempo de desenvolvimento a partir das restrições estabelecidas pelo modelo e pelo usuário (módulo PUTNAM_C), Figura 5.11(c).

- Determinação de E x $t_d$ para a faixa permitida (entre os valores máximo e mínimo determinados anteriormente), realizada pelo módulo PUTNAM_C, Figura 5.11(d).

- Determinação da distribuição da Força de Trabalho em função de $t_d$, para os valores de E e $t_d$ escolhidos (módulo PUTNAM_C), Figura 5.11(e).

- Análise de riscos de desvio dos valores estimados de E e $t_d$ (módulo PUTNAM_C), Figura 5.11(f).

A)

Sistema de Auxilio ao Planejamento de Desenvolvimento de Software

Como deseja encontrar o valor de CK ?

APARTIR_DE_PROJETOS_ANTIGOS
APARTIR_DE_PERGUNTAS_REALIZADAS_PELO_SISTEMA
FORNECER_DIRETAMENTE_CK

1. Use as setas ou a primeira letra do item para posicionar o cursor
2. Tecle RETURN/ENTER para continuar

B)

Sistema de Auxilio ao Planejamento de Desenvolvimento de Software

Em quais das opcoes abaixo melhor se enquadra seu sistema ?

SISTEMA_NOVO_COM_MUITAS_INTERFACES_COM_OUTROS_SISTEMAS
SISTEMA_NOVO_COM_POUCAS_INTERFACES_COM_OUTROS_SISTEMAS
SISTEMA_CONSTRUIDO_APARTIR_DE_RECODIFICACAO_E_INTEGRACAO
SISTEMAS_COMPOSTO_DE_SUB_SISTEMAS_DESENVOLVIDOS_EM_PARALELO

1. Use as setas ou a primeira letra do item para posicionar o cursor
2. Tecle RETURN/ENTER para continuar

C)

Sistema de Auxilio ao Planejamento de Desenvolvimento de Software

Esforco minimo : 19.83 homens-ano
TD maximo : 2.03 anos
Pessoal minimo : 15.00 homens
TD minimo : 1.92 anos
Esforco maximo : 24.99 homens-ano
Pessoal maximo : 20.00 homens
Esforco maximo(f) : 33.56 homens-ano
Tempo minimo(f) : 1.78 anos

QUAL_O_PROXIMO_PASSO_?
CONFIRMAR_E_CONTINUAR
ALTERAR_TD_MAXIMO
ALTERAR_ESFORCO_MAXIMO
ALTERAR_PESSOAL_MAXIMO
ALTERAR_PESSOAL_MINIMO

Os valores limites do esforco , tempo de desenvolvimento e pessoal estao estabelecidos acima , escolha opcao desejada

D)

Sistema de Auxilio ao Planejamento de Desenvolvimento de Software

TD atual : 2.00 anos
Esforco atual : 21.18 homens-ano
Simulacao : NAO

Valores possiveis do tempo de desenvolvimento X esforco ,TECLE ENTER

TEMPO de DESENVOLVIMENTO X ESFORCO

| Tempo de Desen. | Esforco |
|---|---|
| 1.92 anos | 24.99 homens-ano |
| 1.93 anos | 24.60 homens-ano |
| 1.93 anos | 24.22 homens-ano |
| 1.94 anos | 23.84 homens-ano |
| 1.95 anos | 23.47 homens-ano |
| 1.96 anos | 23.11 homens-ano |
| 1.96 anos | 22.76 homens-ano |
| 1.97 anos | 22.41 homens-ano |
| 1.98 anos | 22.07 homens-ano |
| 1.99 anos | 21.73 homens-ano |
| 1.99 anos | 21.41 homens-ano |
| 2.00 anos | 21.08 homens-ano |
| 2.01 anos | 20.77 homens-ano |
| 2.02 anos | 20.46 homens-ano |
| 2.02 anos | 20.15 homens-ano |
| 2.03 anos | 19.83 homens-ano |

E)

Sistema de Auxilio ao Planejamento de Desenvolvimento de Software

TD atual : 2.00 anos
Esforco atual : 21.18 homens-ano
Simulacao : NAO

DISTRIBUICAO DE ESFORCO para TD escolhido. Tecle ENTER para continuar

Forca_de_trabalho_X_TD :

| DATA | PESSOAL UTILIZADO |
|---|---|
| .13 anos | 1.78 homens |
| .27 anos | 3.54 homens |
| .40 anos | 5.26 homens |
| .53 anos | 6.90 homens |
| .67 anos | 8.45 homens |
| .80 anos | 9.90 homens |
| .93 anos | 11.22 homens |
| 1.07 anos | 12.40 homens |
| 1.20 anos | 13.43 homens |
| 1.33 anos | 14.31 homens |
| 1.47 anos | 15.02 homens |
| 1.60 anos | 15.57 homens |
| 1.73 anos | 15.96 homens |
| 1.87 anos | 16.18 homens |
| 2.00 anos | 16.26 homens |

F)

Sistema de Auxilio ao Planejamento de Desenvolvimento de Software

TD atual : 2.00 anos
Esforco atual : 21.18 homens-ano
Simulacao : SIM

A tabela mostra valores para TD e ESFORCO e a probabilidade de que estes valores NAO sejam superados. Tecle ENTER

Analise de riscos :

| TD em anos | ESFORCO em homens-ano | | | | |
|---|---|---|---|---|---|
| 1.88 | 11.54 | 4 % | / 1.99 | 20.64 | 46 % |
| 1.88 | 12.07 | 5 % | / 2 | 21.18 | 50 % |
| 1.89 | 12.61 | 6 % | / 2.01 | 21.71 | 54 % |
| 1.9 | 13.14 | 7 % | / 2.01 | 22.25 | 58 % |
| 1.9 | 13.68 | 8 % | / 2.02 | 22.78 | 62 % |
| 1.91 | 14.22 | 10 % | / 2.03 | 23.32 | 65 % |
| 1.92 | 14.75 | 12 % | / 2.03 | 23.85 | 69 % |
| 1.93 | 15.29 | 14 % | / 2.04 | 24.39 | 72 % |
| 1.93 | 15.82 | 16 % | / 2.05 | 24.93 | 76 % |
| 1.94 | 16.36 | 19 % | / 2.05 | 25.46 | 79 % |
| 1.95 | 16.89 | 21 % | / 2.06 | 26 | 81 % |
| 1.95 | 17.43 | 24 % | / 2.07 | 26.53 | 84 % |
| 1.96 | 17.96 | 28 % | / 2.07 | 27.07 | 86 % |
| 1.97 | 18.5 | 31 % | / 2.08 | 27.6 | 88 % |
| 1.97 | 19.03 | 35 % | / 2.09 | 28.14 | 90 % |
| 1.98 | 19.57 | 38 % | / 2.1 | 28.67 | 92 % |
| 1.99 | 20.11 | 42 % | / 2.1 | 29.21 | 93 % |

*Figura 5.11(a,b,c,d,e,f)* Telas Apresentadas Durante o uso do Modelo de Alocação de Recursos

Na implementação deste modelo o módulo de cálculos é bem mais complexo que nos outros dois modelos. Cada vez que se deseja encontrar E e $t_d$ é necessário resolvermos um sistema de duas equações não lineares (equações 3.41 e 3.42). Como no modelo de Putnam, $t_d$ e E são encontrados um em função do outro, o módulo PUTNAM_C fornece uma interface interativa onde $t_d$ pode ser encontrado a partir do E fornecido e vice-versa.

Outro ponto complexo dos cálculos no modelo de Putnam é a análise de riscos. Ela é feita por simulação; nela $\nabla D$ (gradiente de dificuldade) e LOC são gerados aleatoriamente segundo uma distribuição de probabilidade normal com o valor médio e desvio padrão determinados em cálculos anteriores. O sistema de equação formado pelas eqs. 3.41 e 3.42 é solucionado pelo menos 1000 vezes para diferentes valores gerados de $\nabla D$ e LOC; desta forma pode-se encontrar o valor médio e desvio padrão de $t_d$ e E. A estes valores são associados uma distribuição normal de probabilidade que, segundo Putnam, descreve o risco de não se terminar o projeto abaixo ou acima de um determinado esforço ou tempo de desenvolvimento. Toda a metodologia destes cálculos pode ser encontrada em [PUT80].

### 5.3.4 Controle dos Dados

No SAPDES são gerados dados pelas regras, no processo de consulta aos *frames*, e pelos programas convencionais, nos cálculos necessários à aplicação dos modelos. Além disto, cada modelo implementado gera um conjunto heterogêneo de dados. Devido a estes problemas, concluiu-se que deveria existir um módulo de controle de dados que fornecesse facilidades para que todos os modelos organizassem seus resultados em um única estrutura que pudesse ser posteriormente consultada de uma forma padrão, independentemente dos dados que contivesse. Esta estrutura de dados deveria ser organizada de forma a facilitar a apresentação das estimativas através de interfaces amigáveis, fornecendo facilidades para que a apresentação fosse comandada pelo próprio modelo através de caracteres de controle incluidos na estrutura de dados. O módulo de gerência de dados é representado na Figura 5.12.

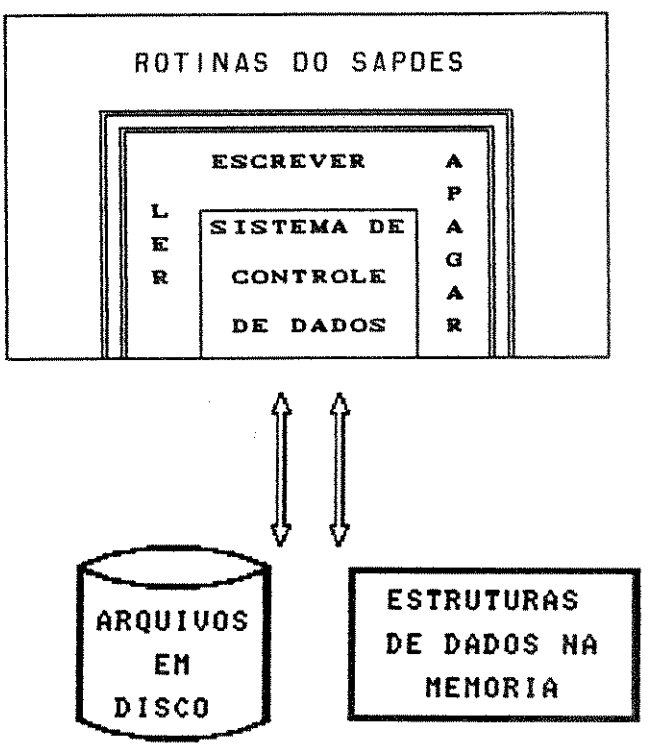


*Figura 5.12* Módulo de gerência de dados

O controle dos dados feito através do módulo de gerência de dados contém rotinas para:

- Ler e escrever a estrutura de dados no disco

- Ler, escrever e apagar dados na estrutura

- Controlar cronologicamente as estimativas feitas para um mesmo projeto

Toda comunicação com o módulo de gerência de dados é feita na forma de mensagens. Estas mensagens ativam as rotinas que realizam as funções listadas acima. Devido ao uso de mensagens, os dados são enviados a este módulo à medida que são estimados. Junto com as mensagens contendo os dados enviados pode-se incluir dados de controle para comandar o futuro processo de apresentação destes dados. Os dados de controle serão usados nos programas VERIFICADOR e IMPRESSOR para organizar o processo de apresentação dos dados. Desta forma, quando um novo modelo for acrescentado ao sistema, os dados a serem apresentados ao cliente e os critérios para sua apresentação devem ser enviados ao módulo de gerência de dados por este modelo assim que ele realize as suas estimativas.

Todos os dados são atualmente armazenados em uma única estrutura não acessível a outras rotinas do sistema. A estrutura de dados gerada após uma estimativa é mostrada na Figura 5.13. Esta estrutura é composta de três sub-estruturas importantes:

- Os dados iniciais como PROJETO, RESPONSÁVEL, DATA e FASE da estimativa.

- Os modelos usados mostrados na lista MODELOS

- As janelas compostas por estruturas de dados que descrevem: os dados a serem apresentados, como eles devem ser apresentados e quais as próximas janelas que podem ser apresentadas ao usuário (janelas filhas).

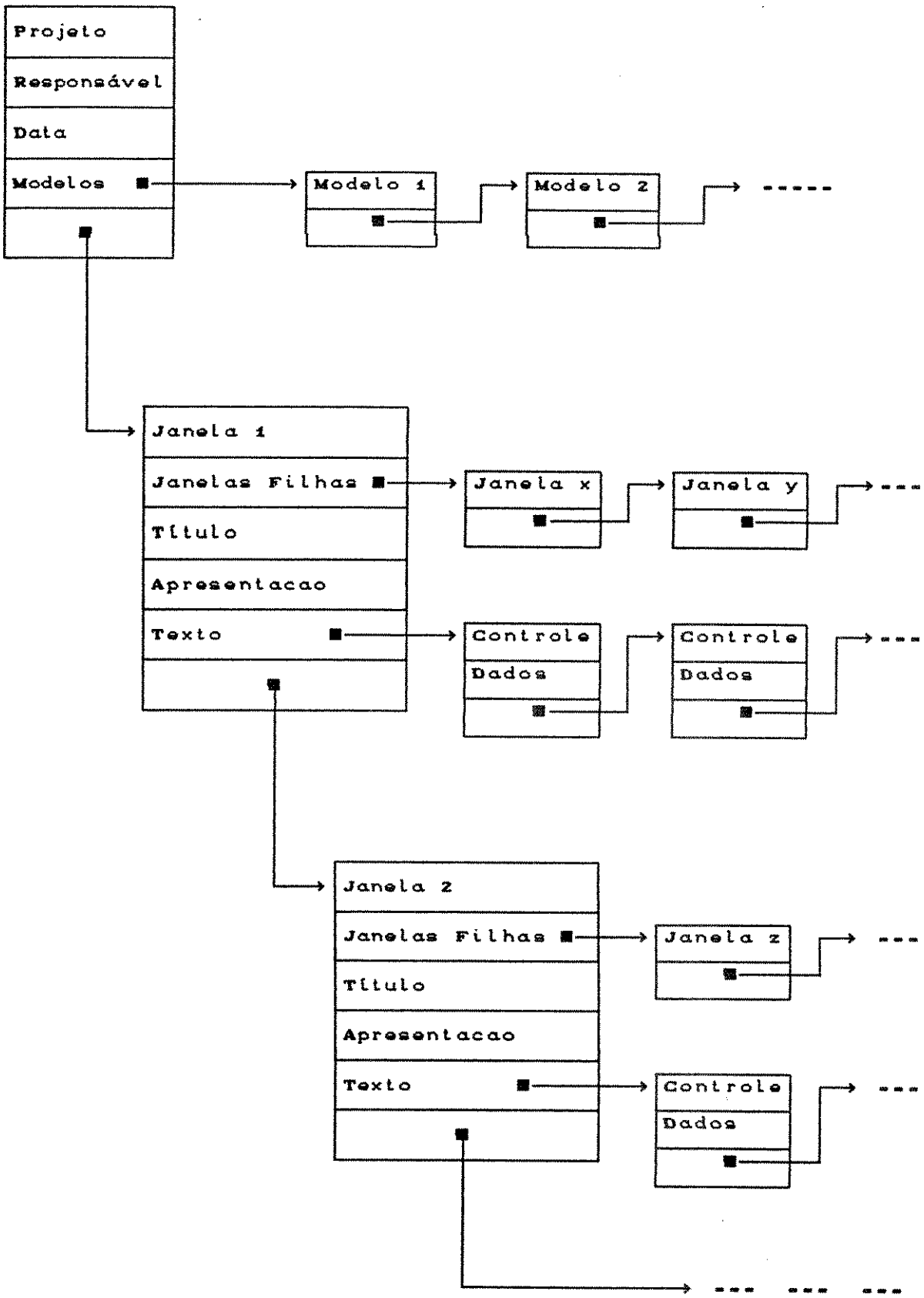


*Figura 5.13* **Estrutura de Dados Montada Durante uma Consulta ao SAPDES**

### 5.3.5 Interação e Seleção de Resultados

A interação e seleção de resultados são aspectos importantes para o funcionamento do sistema. Como vários modelos podem ser aplicados é de se esperar que o sistema trate os resultados gerados pelos modelos para apresentar ao cliente uma estimativa principal. Este processo de tratamento envolve seleção e interação de resultados. Seleção e interação de resultados são necessárias em três situações:

1 - Quando modelos diferentes fornecem resultados complementares.

2 - Quando modelos diferentes fornecem resultados diferentes para grandezas semelhantes.

3 - Quando um mesmo modelo fornece valores diferentes para técnicas *bottom-up* e *top-down*.

O tratamento de resultados deve ser baseado em conhecimento e levar em conta fatores como:

1 - Os resultados a serem apresentados são aqueles verdadeiramente relevantes ao usuário: muitos modelos fornecem uma série de resultados que podem não ser de interesse do usuário.

2 - Resultados de modelos diferentes devem estar coerentes entre si.

3 - Os resultados mais confiáveis são aqueles do modelo cuja aplicação foi mais recomendada no *frame* SELECIONA_MODELO.

O SAPDES poderá tratar resultados dissonantes de três formas:

1 - Considerar que entre dois resultados diferentes um deles é incorreto e ,por consequência, ignorá-lo. Neste caso, o valor considerado correto é fornecido ao usuário.

2 - Considerar que, entre dois resultados diferentes, ambos estão incorretos, pois cada uma das estimativas que geraram estes resultados ignoraram fatores que foram considerados na outra. Neste caso o SAPDES pode sugerir um valor mais realista de estimativa. Como se discutiu na Seção 2.7.6, este valor não é necessariamente um valor de compromisso entre os dois resultados.

3 - Considerar que os dois resultados estão incorretos e sugerir uma nova estimativa com uma abordagem diferente das anteriores.

Note que mesmo em estimativas não dissonantes entre si, o tratamento de resultados é desejável para se selecionar os resultados de maior interesse e conscientizar o usuário, através de diálogo, das incertezas daquela estimativa. A atividade de selecionar os resultados de maior interesse não será mais necessária quando o sistema permitir a seleção de estratégias de estimativas (discutidas na Seção 6.1.4), pois nesta seleção poderá se fazer com que o SAPDES aplique apenas as partes de interesse dos modelos e, consequentemente, gere apenas os resultados de interesse do usuário. É interessante, para propósito de estudo, que as regras de seleção de resultados sejam quardadas junto a eles.

O *frame* SELECIONA_RESULTADOS vai funcionar como um modelo que fornece estimativas próprias. O modo para obtenção destas estimativas é selecionar os resultados obtidos por outros modelos. Quando implementado, este módulo vai fornecer a estimativa característica do SAPDES. Os resultados desta estimativa serão quardados na base de dados do SAPDES como a estimativa do MODELO_SISTEMA. Este módulo não está implementado ainda no SAPDES; como foi discutido neste parágrafo, o seu desenvolvimento equivale qualitativamente ao desenvolvimento de um modelo: por isso, a sua implementação de-

verá ser assunto de futuras pesquisas (vide Seção 6.1.5).

O *frame* SELECIONA_RESULTADOS auxiliará o usuário no sexto passo para realização de uma estimativa, *comparar e interagir estimativas* (vide Seção 2.7.6). Como a seleção de resultados depende dos modelos implementados este *frame* terá que ser continuamente atualizado para que ele possa tratar corretamente os dados de novos modelos e de modelos recalibrados.

### 5.3.6 Apresentação de Resultados

Como vimos da discussão das seções anteriores cada consulta ao SAPDES resultará na criação de resultados referentes a cada um dos modelos aplicados e de resultados do MODELO_SISTEMA (gerados pela seleção e composição dos resultados dos modelos aplicados).

Como é desejável que as rotinas de apresentação sejam aplicáveis a qualquer conjunto de dados gerados pelos modelos, necessita-se que dentro dos próprios resultados existam dados de controle da apresentação de resultados. Os dados de controle visam basicamente organizar a forma de apresentação dos dados gerados pelos modelos. Na Figura 5.13, ao se analisar a sub-estrutura JANELA, pode-se compreender melhor como a apresentação de dados é feita. A sub-estrura JANELA contém os seguintes ítens:

- Um nome que identifica a janela na estrutura de dados.

- Uma tradução do nome para ser usada como título da janela.

- O formato de apresentação, que indica as características da janela (tamanho, atributos, posição, etc) e o tipo de menu de opções que estará associado à janela.

- O texto com as estimativas que serão mostradas ao usuário naquela janela. Neste texto podem ser incluidos caracteres de controle como tabulação, salto de linhas e espaços em branco, etc.

- A lista de janelas filhas que contém a lista das janelas a que se pode ter acesso a partir da janela atual com o uso do menu associado a esta.

A apresentação dos resultados no SAPDES poderá ser feita em terminal de vídeo ou em impressora. Para a apresentação de dados no vídeo a estrutura descrita acima é usada para organizar os dados em telas e janelas. Nesta estrutura estão descritos o tipo das telas e janelas, a ordem em que estas telas e janelas aparecem

, que resultados estão contidos nelas, e como estes resultados estão posicionados nelas. No caso das rotinas de relatório impresso a organização dos dados em janelas e telas é utilizada para se montar o relatório em seções e um índice destas seções. Desta forma os dados que ficariam contidos em uma tela ou janela ficam agrupados em uma determinada seção no relatório impresso.

Os programas de apresentação de resultados preveem também a facilidade de consulta a base de dados de estimativas e de descrição de projetos prontos. Estes dados são lidos do disco pelo sistema de gerência de dados que monta a estrutura de dados da Figura 5.13 e apresenta ao usuário do sistema nas formas descritas acima.

A Figura 5.14 (a,b,c,d) mostra algumas das telas de apresentação de resultados.

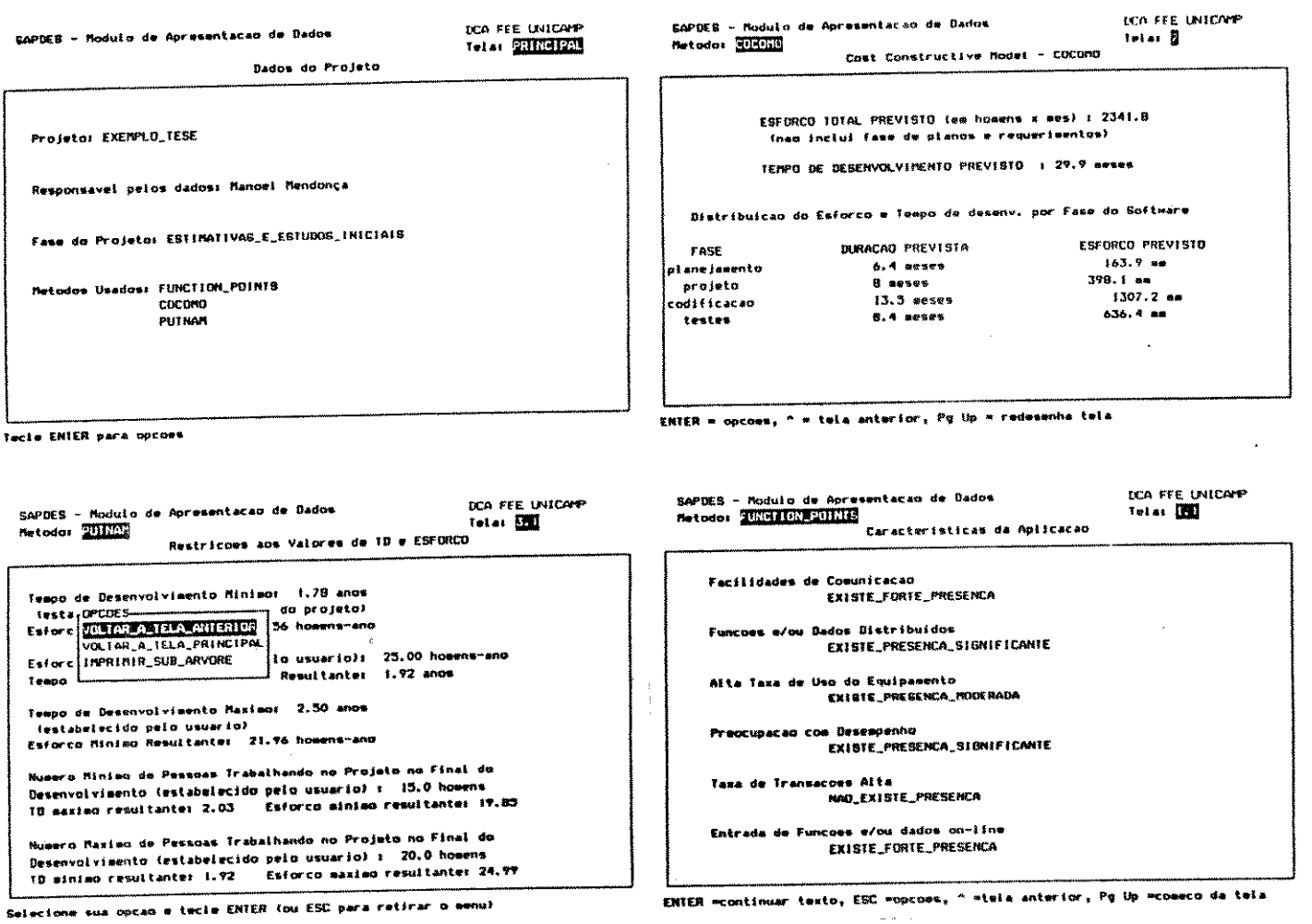


*Figura 5.14* **Telas de Apresentação de Resultados**

# CAPÍTULO 6

# CONCLUSÕES

A engenharia de *software* é um campo de estudos relativamente novo e a estimativa de custo de *software* é uma área bastante empírica dentro deste campo; estes dois fatos indicam que os modelos para estimar custo estão ainda longe do desempenho ótimo. De fato, quando são aplicados por usuários leigos e não estão calibrados para o tipo de projeto que se deseja desenvolver, os resultados dos modelos de estimativa podem ser desastrosos [KEM87] e [KIT85].

Pode-se perguntar porque utilizar modelos de estimativa se seus resultados podem ser incertos; a resposta é que eles certamente serão melhores do que estimativas totalmente empíricas. Os modelos devem ser usados com cuidado, mas seu uso é necessário pois quantidades cada vez maiores de dinheiro estão sendo gastas em sistemas de *software* [BOE88] e [MYE89].

Desenvolver uma ferramenta automática que aplique efetivamente modelos de estimativa de custo, significa resolver o seguinte problema:

> - Os modelos de estimativa de custo de *software* só têm desempenho razoável se aplicados por especialistas e estiverem previamente calibrados a partir de projetos com ambiente de desenvolvimento semelhante ao que se pretende utilizar. Na seção 2.6.3 estabelece-se como bom desempenho de um modelo de estimativa os valores $\overline{MER} \leq 0,20$ , $PREV(0,25) \geq 0,75$ e $PREV(0,50) \geq 0,90$ ; hoje em dia este resultado é dificílimo de ser atingido por um modelo. Uma ferramenta automática de estimativas de custo de *software* deveria auxiliar o usuário não só na aplicação de modelos de estimativa, mas também, fornecer apoio especialista para esta aplicação e facilidades para recalibração dos modelos nela implementados.

A concepção do SAPDES visa solucionar o problema acima. O SAPDES foi desenvolvido objetivando auxiliar o usuário nos quatro últimos e principais passos de uma estimativa em projetos de *software*: detalhar o projeto o máximo possível, aplicar diferentes técnicas de estimativa, comparar e interagir estimativas e, finalmente, acompanhar estimativa X projeto real. Para cumprir o segundo e o terceiro passos satisfatoriamente, decidiu-se que o SAPDES

deveria conter não um, mas vários modelos de estimativa. Montou-se o projeto do SAPDES com o intuito de que ele auxiliasse o usuário não só na aplicação pura e simples de um modelo de estimativa de custo de *software*, mas também na escolha dos modelos mais apropriados ao problema, na aplicação inteligente dos modelos e técnicas de estimativa escolhidas, na comparação e interação das estimativas, no acompanhamento estimativa X projeto real, e, finalmente, na calibração dos modelos aos ambientes específicos de desenvolvimento.

Dois importantes requisitos foram ainda exigidos no projeto do SAPDES:

- Interface amigável com o usuário: boa parte do esforço de desenvolvimento dos módulos de descrição inicial, de aplicação de modelos e de apresentação de dados, foi concentrado no fornecimento de uma interface de fácil manuseio pelo usuário.

- Capacidade de integração modular de modelos de estimativa; esta característica permite que o sistema possa ser modificado e expandido com a frequência necessária para acompanhar a evolução da área de métricas e estimativas em *software*.

Todas as características de projeto mostradas até aqui fazem com que o SAPDES seja superior em concepção às ferramentas de custo de *software* discutidas no Capítulo 3; todavia, a superioridade de concepção não garante a da implementação. A implementação realizada até o momento no SAPDES necessita ser completada:

- Falta criar os módulos de calibração para os modelos implementados (vide Seção 6.1.2).

- Os modelos de estimativas implementados devem ser revisados e completados (vide Seção 6.1.1).

Com a criação dos módulos de calibração, revisão e complemento dos modelos de estimativa, a ferramenta poderá ser usada efetivamente. A sua superioridade deverá se manifestar à medida que se tirar vantagem de sua concepção modular.

A modularidade torna o SAPDES uma ferramenta expansível e adaptável. A expansão e adaptação podem ocorrer não só com a inclusão de novos modelos ao SAPDES, mas principalmente com o acréscimo de conhecimento ao sistema. O acréscimo de conhecimento pode acontecer: no módulo de seleção de modelos, nos módulos que coletam dados para os modelos, e na criação de um módulo de seleção e validação de resultados. O módulo de seleção de resultados, discutido na Seção 6.1.5 , será um módulo experimental que qualitativamente é um modelo de estimativa que funciona fornecendo estimativas a partir da seleção e validação de dados estimados pelos outros modelos implementados no SAPDES.

Dentro de um processo evolutivo diferente o SAPDES poderá ter agregado a si módulos de análise de métricas. Estes módulos (discutidos na Seção 6.1.6) torná-lo-ão um sistema de finalidade mais abrangente pois, além de cumprir seu objetivo inicial que é estimar custo de *software*, o SAPDES se tornará uma ferramenta de auxílio à pesquisa na área de métricas e modelos em engenharia de *software*.

## 6.1 PENDÊNCIAS E DESENVOLVIMENTOS FUTUROS

A Seção 5.3 discutiu em detalhes o SAPDES. Esta seção discute o que falta ser implementado no SAPDES do que foi discutido até aqui, e o que ainda pode ser implementado no SAPDES para que ele possa evoluir no sentido de se tornar uma ferramenta mais poderosa.

### 6.1.1 Pendências nos Modelos de Estimativas

No SAPDES foram implementados três modelos de estimativas, o COCOMO, o modelo de análise por *function points*, e o modelo de alocação de recursos de Putnam.

O COCOMO é o único dos três que atualmente pode ser efetivamente utilizado. Este modelo está implementado nas suas versões básica e intermediária. A versão avançada, que consiste em estimar e aplicar os *cost drivers* em partes para cada um dos módulos do projeto, poderá ser implementada com facilidade a partir das implementações atuais.

O modelo de Putnam ainda não pode ser efetivamente utilizado pois a determinação do fator de tecnologia através de diálogo com o usuário não está implementada. Esta implementação não foi realizada pois, como foi discutido na Seção 3.2.2, a metodologia para determinação de $C_K$ não foi fornecida por Putnam. O fator de tecnologia pode ser determinado a partir de projetos antigos; entretanto, nem sempre o usuário do sistema tem dados quardados sobre projetos anteriores desenvolvidos no mesmo ambiente. Uma técnica para determinação de $C_K$ a partir das características do projeto e do ambiente de desenvolvimento pode ser criada. Os pontos de partida deveriam ser as técnicas usadas em outros modelos para determinação de fatores de ajustamento do esforço calculado, e os intervalos de valores usados por Putnam na sua ferramenta (SLIM). Esta implementação equivaleria a criar parte de um novo modelo e demandaria bastante esforço para validá-la.

O modelo de análise por *function points* pode ser efetivamente

aplicado para estimativas; todavia, ele estima uma única grandeza, o esforço total de desenvolvimento. Este tipo de estimativa é bastante falho já que um gerente de *software* deseja pelo menos poder estimar também o tempo e cronograma de desenvolvimento com o intuito de viabilizar o acompanhamento projeto X dados reais.

A partir da discussão acima conclui-se que as implementações dos três modelos no SAPDES ainda precisam evoluir, especialmente as implementações do modelo de alocação de recursos e do modelo de análise por *function points*.

### 6.1.2 Calibração dos Modelos

A calibração é parte fundamental para o funcionamento adequado de um modelo. No estágio atual de implementação, o SAPDES não tem nenhum módulo de calibração de modelos. Os modelos do SAPDES estão calibrados para as bases de dados nas quais foram desenvolvidos.

Na Seção 4.1.4 foi feita uma discussão sobre como calibrar modelos de estimativa no SAPDES. Foi mostrado que cada modelo necessita de um módulo próprio de calibração. Desenvolver módulos específicos para calibração dos três modelos implementados deverá ser o próximo passo no desenvolvimento do SAPDES.

Os módulos de calibração de modelos serão dependentes da implementação destes modelos no SAPDES. É necessário que as fórmulas dos modelos a serem calibrados sejam colocadas em arquivos acessíveis aos módulos de calibração, e que estes módulos "saibam" como as fórmulas a serem calibradas estão armazenadas neste arquivo para poder modificá-las.

Como se discutiu na Seção 4.1.4 , existem várias metodologias de calibração de modelos; porisso podem existir vários módulos de calibração para um mesmo modelo.

Para desenvolver módulos de calibração para os modelos atualmente implementados no SAPDES é desejável a expansão do módulo de gerência de dados mostrado na Seção 5.3.4, visando permitir melhor manuselo dos dados sobre estimativas e projetos prontos armazenados em arquivo. A idéia é que se ligue o módulo de gerência de

dados do SAPDES a um gerenciador de arquivos. Esta ligação propiciará que se guardem informações sobre as estimativas em um único arquivo e se incluam em cada registro deste arquivo campos contendo informações sobre a fase da estimativa, o tipo do projeto sendo estimado, e os valores das grandezas estimadas. Isto permitirá que as rotinas de calibração possam obter mais racionalmente os dados necessários a seu funcionamento. Estes dados são, não só as informações sobre esforço, cronograma e tempo de desenvolvimento de projetos completados, mas também, dados sobre os tipos de projetos, para que seja feita a seleção dos dados de quais projetos deverão entrar no processo de calibração.

### 6.1.3 Acompanhamento de Projetos em Andamento

Atualmente no SAPDES o acompanhamento de projetos em andamento é feito através de reestimativa. Esta reestimativa consiste em ler a estimativa feita anteriormente para este mesmo projeto, contida em disco, e alterar os dados de entrada conforme novos fatos ocorram durante o desenvolvimento. Esta nova estimativa será armazenada em outro arquivo e permitirá que ao final do projeto o usuário possua os resultados das estimativas feitas durante todo o ciclo de vida daquele projeto.

Com a evolução de módulo de gerência de dados, pode-se usar o gerenciador de arquivos para pegar dados das várias estimativas feitas durante o desenvolvimento do projeto conforme o campo *fase* dos registros de estimativas. Com estes dados é possível apresentarem-se gráficos comparativos de *grandezas estimadas* X *fase do projeto*. Estes gráficos permitirão um melhor acompanhamento de um projeto para comparação entre o que foi planejado e o que foi cumprido.

### 6.1.4 Implementação de Estratégias de Estimativas

Na Seção 5.3.2 discutem-se a seleção de modelos de estimativa; nesta seção mostramos como esta operação pode evoluir no sentido de tornar o SAPDES mais adaptável ao problema do usuário.

Em uma abordagem futura mais complexa prevê-se, não a seleção de modelos, e sim, a seleção de estratégias de estimativa. O termo estratégia de estimativa tem o intuito de significar a combinação de partes de diversos modelos para realizar uma única estimativa. Esta combinação de partes de modelos será realizada conforme o desejo do usuário e o aconselhamento do sistema. Pode-se, por exemplo, usar o modelo dos *function points* para determinar o número de LOC, usar o número de LOC nas equações do COCOMO para estimar esforço e tempo de desenvolvimento, e aplicar a distribuição de Rayleigh para encontrar a distribuição do esforço por tempo de desenvolvimento. Algumas ferramentas, na realidade, já agrupam em um modelo próprio de estimativas a combinação de abordagens apresentadas em outros modelos; é o caso,por exemplo, do JS-2 [JEN83a] , que combina equações semelhantes as usadas no COCOMO com uma distribuição de esforço determinada por uma curva de Rayleigh, ou do ESTIMACS [RUB83], que combina o uso de *function points* com técnicas de determinação de distribuição de esforço e análise de riscos. Um estudo interessante de como agrupar modelos e técnicas de estimativa pode ser visto em [VER87]; este estudo mostra como a análise por *function points* foi combinada com o COCOMO para realizar a estimativa de um projeto de *software* onde será usado linguagem de quarta geração (lembrar que os modelos são calibrados para linguagens de programação convencionais).

Na abordagem descrita a estratégia de estimativa determinada no *frame* SELECIONA_MODELO deverá ser aplicada pelo *frame* ESTIMADOR. Isto significa que cada modelo deverá ser dividido em estruturas funcionais utilizáveis em pedaços logicamente independentes; por exemplo, determinação da métrica básica, estimativa de esforço e tempo de desenvolvimento total, distribuição de esforço por $t_D$, distribuição de esforço por atividade, etc.

### 6.1.5 Seleção de Resultados

Na Seção 5.3.5 discutiu-se o aspecto seleção de resultados. Selecionar os resultados significa compor e/ou modificar as estimativas geradas pelos diversos modelos implementados no SAPDES, a fim de montar uma estimativa melhor. O módulo que fará a seleção de resultados será como um novo modelo cujas entradas serão as estimativas dos outros modelos. Este modelo usa conhecimento sobre os outros modelos e sobre como analisar os seus resultados para fornecer uma nova estimativa, que se espera mais precisa, de custo.

Este módulo não foi implementado nesta versão do SAPDES. Implementar este módulo seria criar um novo modelo (como ressalta a discussão no parágrafo anterior), e o objetivo básico deste trabalho não é criar novos modelos de estimativa de custo de *software*, mas sim criar uma ferramenta capaz de abrigar diversos modelos de estimativa e ajudar o usuário a selecionar os mais aplicáveis ao seu problema. Devido a sua arquitetura o SAPDES suporta esta seleção de resultados; todavia, a implementação deste módulo deve ser tema de estudo cuidadoso, pois como um modelo de estimativa ele deve ter um embasamento teórico e ser cuidadosamente validado. O desenvolvimento deste módulo no SAPDES poderá, desta forma, ser tema de futuras atividades de pesquisa na área de estimativas de projetos de *software*.

### 6.1.6 Análise de Métricas

A estrutura mostrada na Figura 4.3 permite que no SAPDES possam ser implementados métodos de análise de métricas. Da mesma forma que métodos de calibração podem ser colocados na estrutura mostrada na Figura 4.3, métodos de análise de métricas podem ser implementados. Estes métodos de análise de métricas poderão usar a base de dados de projetos prontos do SAPDES para estudos sobre métricas e modelos em engenharia de *software*, numa abordagem semelhante às usadas por Yu, Nejmeh, Dunsmore e Shen [YU88], e por Li e Cheung [LI87]. Nesta abordagem procura-se coletar diversas gran-

dezas medidas em vários projetos de *software* e fornecer uma série de facilidades para análise das relações entre estas grandezas. As principais grandezas que poderiam ser coletadas são: esforço de desenvolvimento, número de programadores, tempo de desenvolvimento, tamanho do *software*, medidas de complexidade, medidas de estrutura de dados, número de defeitos, etc. Entre as facilidades de análise citaríamos desde rotinas estatísticas até analisadores de projetos prontos de *software* para computar automaticamente métricas como: número de linhas de código, tamanho da estrutura de dados, número de operandos e operadores (usa-se esta contagem para computar a complexidade do *software*).

Uma implementação deste tipo demandaria a construção de uma ferramenta completamente nova que poderia ser adicionada ao SAPDES. Esta ferramenta teria uma finalidade fundamentalmente científica, pois estaria completamente voltada ao estudo de métricas na engenharia de *software*.

## 6.2 CONCLUSÕES FINAIS

Pode-se resumir as principais conclusões do trabalho realizado nos seguintes ítens:

- Os modelos de estimativa de custo de *software* estão longe da perfeição. O SAPDES é um passo no sentido de melhor utilizar os modelos existentes. O sistema deverá fornecer melhores estimativas por possuir capacidade de se adaptar a novos ambientes de projeto (por calibração) e por ter capacidade de aplicar mais corretamente modelos de estimativa (por realizar um diálogo inteligente com o usuário).

- O SAPDES, ao contrário de outras ferramentas de estimativas, foi concebido no sentido de privilegiar a evolução e acompanhamento do campo de estudos de estimativas e métricas em engenharia de *software*.

- O SAPDES é uma ferramenta útil na pesquisa por novos modelos de custo de *software* pois ele permite que estes novos modelos possam ser nele implementados e facilita que seu desempenho possa ser comparado aos dos modelos já existentes.

- Caso seja acrescentado ao SAPDES um módulo de análise de métricas, ele poderá servir como uma ferramenta de estudos para a área de engenharia de *software*, colaborando para melhor compreeensão das relações entre grandezas como: tempo de desenvolvimento, esforço de desenvolvimento, características do projeto, características de ambiente de desenvolvimento, características de pessoal de desenvolvimento, etc.

- Algumas partes importantes do SAPDES ainda precisam ser completadas para que ele possa ser usado efetivamente.

- Muito do conhecimento no campo de modelos em engenharia de *software* não é ainda bem compreendido; porisso todas as partes baseadas em conhecimento no SAPDES devem ser consideradas exploratórias (vide discussão em [RAM89]); considerar estas

partes como exploratórias implica na necessidade de validar o sistema em diversas bases de dados sobre projetos prontos e procurar refinar e ajustar constantemente as partes baseadas em conhecimento do SAPDES através de entrevistas com especialistas em métricas e gerência de projetos de *software*.

# BIBLIOGRAFIA

[ALB79] Albrecht,A.J.: "Measuring Application Development Productivity"; Proceedings, IBM Application Development Symposium, pp 83-92; SHARE Inc. and GUIDE International; Monterey, Califórnia, Outubro 1979.

[ALB83] Albrecht,A.J. & Gaffney,J.E.,Jr.: "Software Function, Source Lines of Code, and Development Effort Prediction : A Software Science Validation"; IEEE Trans. Software Eng. SE-9(6), 639-647; 1983.

[ARO69] Aron,J.D.: "Estimating Resources for Large Programming Systems"; NATO Science Committee; Roma, Itália; 1969.

[BAI81] Bailey,J.W. & Basili,V.R.: "A Meta-Model for Software Development Resource Expenditures"; Proc. 5th Int. Conf. Software Eng., 107-116; 1981.

[BAS80] Basili,V.R.: Models and Metrics for Software Management and Engineering, Tutorial on; IEEE Computer Society Press; 1980.

[BAS81] Basili,V.R. & Beane,J.: "Can the Parr Curve Help with Manpower Distribution and Resource Estimation Problems?"; Journal of Systems & Software 2, 59-69; 1981.

[BOE81] Boehm,B.W: Software Engineering Economics; Prentice-Hall, Englewood Cliffs, New Jersey; 1981.

[BOE84] Boehm,B.W: "Software Engineering Economics"; IEEE Trans. Software Eng. SE-10(1), 4-21; 1984.

[BOE87] Boehm,B.W.: "Industrial Software Metrics Top 10 List"; IEEE Software, seção "Quality Time", Vincent Shen (ed.), setembro; 1987.

[BOE88] Boehm.B.W. & Papaccio,P.N.: "Understanding and Controlling Software Costs"; IEEE Trans. Software Eng. SE-14(10); pp 1462-1477 , outubro 1988.

[BRO74] Brooks,F.P.,Jr: "The Mythical Man-Month"; Datamation, dezembro; 1984.

[BRO75] Brooks,F.P.,Jr: The Mythical Man-Month; Addison-Wesley, Reading, Massachusetts; 1975.

[CON85] Conte,S.D.; Dunsmore,H.E. & Shen,V.Y.: "Software Effort Estimation and Productivity"; Advances in Computers, 25, 1-60; Academic Press ed.; 1985.

[COT88] Côté,V.; Bourque,P.; Oligny,S. & Rivard,N.: "Software Metrics: An Overview of Recent Results"; Journal of Systems & Software 8, 121-131; 1988.

[CUE87] Cuelenaere,A.M.E.; van Genuchten,M.J.I.M. & Heemstra,F.J.: "Calibrating a Software Cost Estimation Model: Why and How"; Information and Software Technology, vol. 29 , num. 10 , dezembro 1987.

[CUR79a] Curtis,B.; Sheppard,S.B.; Milliman,P.; Borst,M.A. & Love,T.: "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics"; IEEE Trans. Software Eng. SE-5(2), pp 96-104; 1979.

[CUR79b] Curtis,B.; Sheppard,S.B. & Milliman,P.: "Third Time Charm: Stronger Prediction of Programmer Performance by Software Complexity Metrics"; Proc. 4th Int. Conf. Software Eng. ; pp 356-360, 1979.

[DRU85] Drummond,S.: "Measuring Applications Development Performance"; Datamation; pp 102-108, fevereiro 1985.

[FAR65] Farr,L. & Zagorski,H.J.: "Quantitative Analysis of Programming Cost Factors: A Progress Report"; ICC Symposium Proceedings on Economics Automatic of Data Processing; A.B.Frielink (ed.); North-Holland, Amsterdam; 1965.

[FER81] Ferrentino,A.B.: "Making Software Development Estimates Good"; Datamation, setembro; 1981.

[FRE79] Freiman,F.R. & Park,R.E.: "PRICE Software Model - Version 3 : An Overview"; Proceedings, IEEE-PINY Workshop on Quantitative Software Models; IEEE Catalog Num. TH0067-9 , pp. 32-41; Outubro 1979.

[HAL77] Halstead,M.H.: Elements of Software Science; Elsevier / North-Holland, New York; 1977.

[HER77] Herd,J.R.; Postak,J.N.; Russell,W.E. & Stewart,K.R.: Software Cost Estimation Study - Study Results; Final Tech. Rep. RADC-TR-77-220, Doty Associates, Rockville, Maryland.

[HAY83] Hayes-Roth,F.; Waterman,D.A. & Lenat,D.B.: Building Expert Systems; Addison-Wesley Publishing Company; 1983.

[JEN81] Jensen,R.W.: "A Macro-level Software Development Cost Estimation Methodology"; 14th Asilomar Conference on Circuits, Systems & Computers"; IEEE; New York, NY; 1981.

[JEN83a] Jensen,R.W.: "An Improved Macro-level Software Development Resource Estimation Model"; Proc. 5th ISPA Conf.; pp. 88-92, abril 1983.

[JEN83b] Jensen,R.W. & Lucas,S.: "Sensitivity Analysis of the Jensen Software Model"; Proc. 5th ISPA Conf.; pp. 384-389, abril 1983.

[JEN84] Jensen,R.W.: "A Comparison of the Jensen and COCOMO Schedule and Cost Estimation Models"; Proc. 6th ISPA Conf.; pp. 96-106, maio 1984.

[JON86] Jones,C.: Programming Productivity; McGraw-Hill, New-York; 1986.

[KEM87] Kemmerer,C.F.: "An Empirical Validation of Software Cost Estimation Models"; Communications of the ACM ,vol.30 ,num. 5 , maio 1987.

[KIT85] Kitchenham,B.A. & Taylor,N.R.: "Software Project Development Cost Estimation"; Joournal of Systems & Software ,vol. 5 , 1985.

[KNA86] Knafl,G.J. & Sacks,J.: "Software Development Effort Prediction Based on Function Points"; Proc. 9th Int. Conf. Software Eng.; IEEE; pp 319-325, 1986.

[LEH88] Lehder Jr.,W.E.; Smith,D.P. & Yu,W.D.: "Software Estimation Technology"; AT&T Technical Journal, vol. 67 , num. 4; pp 10-18, junho/agosto 1988.

[LI87] LI,H.F. & Cheung,W.K.: "An Empirical Study of Software Metrics"; IEEE Trans. Software Eng. SE-13(6); junho 1987.

[LUC87] Lucena,C.J.P.: Inteligência Artificial e Engenharia de Software; PUC-RJ / IBM Brasil // Jorge Zahar Editor; 1987.

[MAR83] Martin,E.W.: "Strategy for a DoD Software Initiative"; IEEE Computer Mag., vol 16; pp 52-59 , março 1983.

[MAR88] Martin,R.: "Evaluation of Current Software Costing Tools"; Software Engineering Notes, vol.13 ,num.3 ,pp 49-51; julho 1988.

[MCC76] McCabe,T.J.: "A Complexity Measure"; IEEE Trans. Software Eng. SE-2(4), 308-320; 1976.

[MIY85] Miyazaki,Y. & Mori,K.: "COCOMO Evaluation and Tailoring"; Proceedings, 8th Int. Conf. on Software Eng.; pp 292-299, agosto 1985.

[MYE89] Myers,W.: "Allow Plenty of Time for Large-Scale Software"; IEEE Software Mag., vol.6 , num.4; pp 92-99 , junho 1989.

[NEL66] Nelson,E.A.: Management Handbook for the Estimation of Computer Programming Costs; AD-A648750; Systems Development Corp.; 1966.

[NOR63] Nórden,P.V.: "Useful Tools for Project Management"; Operations Research in Research and Development, cap. 5; B.V.Dean ed.; John Wiley & Sons; 1963.

[PAR80] Parr,F.: "An alternative to the Rayleigh Curve Model for Software Development Effort"; IEEE Trans. Software Eng. SE-6, 291-296; 1980.

[PEN88] Penedo,M.H. & Riddle,W.E.: "Software Engineering Environment Architectures"; IEEE Trans. Software Eng. SE-14(6); pp 689-695 , junho 1988.

[PRE87] Pressman,R.S.: Software Engineering: A Practitioner's Approach; segunda edição; McGraw-Hill ed.; 1987.

[PUT78a] Putnam,L.H.: "A General Empirical Solution to the Macro Software Sizing and Estimating Problem"; IEEE Trans. Software Eng. SE-4(4), 345-361; 1978.

[PUT78b] Putnam,L.H.: "Example of an Early Sizing, Cost and Schedule Estimate for an Application Software System"; Proc. Computer Software and Applications Conf.-COMPSAC; 1978.

[PUT79a] Putnam,L.H. & Fitzsimmons,A.: "Estimating Software Costs - Part 1"; Datamation, setembro, 189-198; 1979

[PUT79b] Putnam,L.H. & Fitzsimmons,A.: "Estimating Software Costs - Part 2"; Datamation, outubro, 171-178; 1979

[PUT79c] Putnam,L.H. & Fitzsimmons,A.: "Estimating Software Costs - Part 3"; Datamation, novembro, 137-140; 1979

[PUT80] Putnam,L.H.: Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers, Tutorial on; IEEE Computer Society Press; IEEE cat. EHO 165-1; 1980.

[RAM89] Ramsey,C.L. & Basili,V.R.: "An Evaluation of Expert Systems for Software Engineering Management"; IEEE Trans. Software Eng., SE-15(6); pp 747-759 , junho 1989.

[RUB83] Rubin,H.A.: "Interactive Macro-estimation of Software Life Cycle Parameters Via Personal Computer: A Technique for Improving Customer/Developer Communication"; Symposium, Application and Assessment of Automated Tools for Software Development; São Francisco, Califórnia, 1983.

[RUB85] Rubin,H.A.: "A Comparison of Cost Estimation Tools"; Proceedings, 8th Int. Conf. on Software Eng.; pp. 174-180, agosto 1985.

[SCH78] Schneider,V.: "Predictions of Software Effort and Project Duration - Four New Formulas"; SIGPLAN Notices 13(6), 49-59; 1978.

[SYM88] Symons,C.R.: "Function Points Analysis: Difficulties and Improvements"; IEEE Trans. Software Eng. SE-14(1); pp 2-11, Janeiro 1988.

[THE84] Thebaut,S.M. & Shen,V.Y.: "An Analytic Resource Model for Large-scale Software Development"; Inf. Process. Manage. 20(1-2), 295-315, 1984.

[VER88] Verner,J.M. & Tate,G.: "Estimating Size and Effort in Fourth-Generation Development"; IEEE Software Magazine, vol.5 , num. 4 ; pp 15-22 , Junho 1988.

[WAL77] Walston,C.E. & Felix,C.P.: "A Method of Programming Measurement and Estimation"; IBM Syst. Journal 16(1), 54-73; 1977.

[WAT86] Waterman,D.A.: A Guide to Expert Systems; Addison-Wesley Publishing Company;1986.

[WOL74] Wolverton,R.W.: "The Cost of Developing Large-Scale Software"; IEEE Trans.Comput. C-23(6), 615-636; 1974.

[YU88] Yu,T.J.; Nejmeh,B.A.; Dunsmore,H.E. & Shen,V.Y.: "SMDC: An Interactive Software Metrics Data Collection and Analysis System"; Journal of Systems & Software, vol.8; Janeiro 1988.