PUC

Série: Monografias em Ciência da Computação
Nº 15/81

UM ESTUDO COMPARATIVO DE METODOLOGIAS DE PROGRAMAÇÃO

por

Dimas A. de Andrade
Antonio F. Veras
Rubens N. Melo

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453
RIO DE JANEIRO – BRASIL

PUC/RJ - Departamento de Informática

Série: Monografias em Ciência da Computação
Nº 15/81

Editor: Marco Antonio Casanova

Dezembro, 1981

UM ESTUDO COMPARATIVO DE METODOLOGIAS DE PROGRAMAÇÃO*

por

Dimas A. de Andrade
Antonio F. Veras
Rubens N. Melo

* Trabalho patrocinado em parte pela FINEP

Resumo

Este trabalho apresenta de maneira resumida as principais metodologias de projeto e implementação de programas recentemente propostas na literatura.

Os métodos são também analisados e comparados com base em alguns aspectos desejáveis para uma boa metodologia.

Palavras-chaves:

Engenharia de software, Metodologia de Programação, Programação Estruturada, Projeto Estruturado.

Abstract

This work makes a brief presentation of the main Program Design and Implementation techniques which have been recently proposed in the literature.

The methods are also analysed and compared based on some desirable aspects for a good methodology.

Keywords:

Software Engineering, Programming Methodology, Structured Programming, Structured Design, Composite Design.

INDICE

Capítulos	Pág.
1. INTRODUÇÃO	1
2. CARACTERÍSTICAS DE UMA BOA METODOLOGIA	4
3. TÉCNICAS TRADICIONAIS	
3.1 - FLUXOGRAMA (FLOWCHARTING)	
3.1.1 - Apresentação	11
3.1.2 - Exemplo	14
3.1.3 - Considerações	18
3.2 - TABELAS DE DECISÃO	
3.2.1 - Apresentação	20
3.2.2 - Exemplo	24
3.2.3 - Considerações	25
4. HISTÓRIA E FUNDAMENTOS DA PROGRAMAÇÃO ESTRUTURADA	28
5. TÉCNICAS DE ESTRUTURAÇÃO	
5.1 - NASSI-SHNEIDERMAN CHARTS	
5.1.1 - Apresentação	32
5.1.2 - Exemplo	35
5.1.3 - Considerações	37
5.2 - MÉTODO DE JACKSON	
5.2.1 - Apresentação	39
5.2.2 - Exemplo	44
5.2.3 - Considerações	50
5.3 - LÓGICA DE CONSTRUÇÃO DE PROGRAMAS	
5.3.1 - Apresentação	52
5.3.2 - Exemplo	60
5.3.3 - Considerações	68
5.4 - HIPO	
5.4.1 - Apresentação	69
5.4.2 - Exemplo	76
5.4.3 - Considerações	79
5.5 - COMPOSITE DESIGN (ou STRUCTURED DESIGN)	
5.5.1 - Apresentação	83
5.5.2 - Exemplo	90
5.5.3 - Considerações	93
6. CONCLUSÕES E CONSIDERAÇÕES FINAIS	95

1 - INTRODUÇÃO

Em todas as áreas da atividade humana - na matemática, engenharia, etc... - a preocupação com a definição de métodos de trabalho, onde sejam estabelecidos uma sucessão de passos a serem executados para se alcançar determinados resultados, sempre foi uma constante. Como exemplo mais conhecidos podemos citar a Regra de Kramer para inversão de matrizes e o método PERT/CPM para acompanhamento do desenvolvimento de projetos.

Com objetivos semelhantes, na área de processamento de dados, diversos pesquisadores e empresas vêm desenvolvendo metodologias de trabalho para apoio à atividade de programação. Historicamente foram desenvolvidos inicialmente os Fluxogramas, a seguir as Tabelas de Decisão e, mais recentemente, métodos tais como o de Jackson e o Structured Design, que têm por base os fundamentos da programação estruturada.

Como é natural, sendo o processamento de dados uma área relativamente nova, mesmo com todos os esforços que ultimamente tem sido realizados na busca de melhores métodos de trabalho, ainda não se conseguiu formular um método geral que englobe todos os aspectos e nuanças relacionadas com a atividade de programação. Assim, encontram-se metodologias cujo objetivo principal é a completa documentação do programa e outras onde a preocupação central é a estruturação de sua lógica. Além disso, observa-se que algumas metodologias são mais próprias às situações onde as transformações sofridas pelos dados é dominante - por exemplo, aplicações de natureza científica -, enquanto outras são dirigidas para a solução de problemas onde a estrutura dos dados de entrada e saída caracteriza o problema - por exemplo, aplicações de natureza comercial cujos dados e relatórios possuem, normalmente, estruturas hierárquicas.

Assim sendo; a adoção de uma determinada metodologia em um Centro de Processamento de Dados dependerá das metas que se tem em mente, bem como da natureza das aplicações desenvolvidas nesse Centro.

Nesse contexto, o objetivo do presente trabalho é o de congregar informações sobre metodologias de programação visando facilitar aos profissionais da área de processamento de dados a avaliação e seleção da ferramenta mais adequada às suas metas e aplicações. Para tanto, serão descritas as principais metodologias propostas até o momento, acompanhadas de comentários sobre suas potencialidades e restrições, que terão por base certos aspectos relacionados com a atividade de programação que, a nosso ver, devem ser considerados quando se busca uma ferramenta de apoio a essa atividade.

Objetivando tornar mais compreensíveis as considerações que serão aqui tecidas, dedicamos o Capítulo 2 à discussão dos pontos tomados como referência para a elaboração das mesmas.

Ao se estudar as diversas metodologias em conjunto nota-se que, de certo modo, elas podem ser encaradas como decorrentes de um processo em evolução, onde uma metodologia mais recente corresponde a uma anterior incorporando novos conceitos e/ou eliminando deficiências. Assim, não só por razões históricas, mas principalmente por acreditarmos que desta forma estaremos evidenciando o que seja uma melhor metodologia, destinamos o Capítulo 3 à apresentação do FLUXOGRAMA (Flowcharting) tradicional, e das TABELAS DE DECISÃO. Estas duas técnicas caracterizam a primeira preocupação dentro da área de processamento de dados, que foi a de documentar o fluxo/condições de processamento.

O Capítulo 4 concerne à história e fundamentos do que se de

nominou PROGRAMAÇÃO ESTRUTURADA, base das metodologias propostas na década passada que, acima da documentação do fluxo de processamento, têm como objetivo o "design" do programa. Estas metodologias aqui apresentadas por NASSI - SHNEIDERMAN CHARTS, MÉTODO DE JACKSON, LÓGICA DE CONSTRUÇÃO DE PROGRAMAS, HIPO E COMPOSITE DESIGN, são o assunto do Capítulo 5.

O Capítulo 6 contém um resumo dos fatos evidenciados para cada metodologia analisada e nosso pensamento quanto a utilização das mesmas.

Salientamos não ser nosso objetivo ensinar o uso das diferentes metodologias aqui apontadas, mas simplesmente, caracterizá-las. Para a aprendizagem destas metodologias haverá necessidade de consulta às referências mencionadas ao longo do texto.

2 - CARACTERÍSTICAS DE UMA BOA METODOLOGIA

A nosso ver, uma boa metodologia é aquela que atua de forma positiva no processo de desenvolvimento de programas, favorecendo o registro de conhecimentos alcançados a cada etapa de aprofundamento no problema e a comunicação destes conhecimentos entre os elementos participantes da sua solução, promovendo o aumento da produtividade da atividade de programação e melhorando a qualidade dos programas elaborados.

Outras características importantes de uma boa metodologia são a praticabilidade, a generalidade e a existência de sistemas automatizados que apoiem a sua utilização.

Para melhor fixá-los, já que são a base das considerações elaboradas para cada metodologia apresentada, a seguir discutiremos cada um dos pontos acima referidos.

REGISTRO DE SOLUÇÕES E COMUNICAÇÃO

A concepção de um programa se dá através de um processo de refinamentos sucessivos do conhecimento de sua estrutura, onde inicialmente se busca uma visão geral do problema e, em seguida, se procura visualizar, cada vez com mais detalhes, suas partes e interligações. Estas partes e interligações são então transformadas numa sequência de comandos, através dos quais podemos nos comunicar com o computador.

Quando a aplicação a ser desenvolvida é simples, a fase de concepção pode ser realizada num período de tempo curto, permitindo que a estrutura detalhada do programa seja memorizada pelo programador. Porém, à medida que a aplicação torna-se mais complexa isto não é mais possível e torna-se necessário que resultados intermediários desse processo mental - os diversos níveis de entendimento - venham a ser registrados de forma sistemática. Isto devido a própria limitação do ser humano quanto à

capacidade de memorização e/ou também por participarem diferentes pessoas nas diversas fases de desenvolvimento de um programa (projeto, codificação e testes).

Além de permitir a preservação e comunicação dos conhecimentos alcançados, deve-se considerar que a documentação produzida é o único meio viável para se acompanhar e avaliar o desenvolvimento dos trabalhos.

Face o exposto, uma boa metodologia é aquela que represente um processo de análise de problemas através de refinamentos sucessivos, acompanhando assim o modo humano de agir, e promova a documentação dos resultados alcançados na evolução da concepção do programa de modo a garantir preservação e a comunicação dos conhecimentos adquiridos, e o acompanhamento e avaliação dos trabalhos desenvolvidos.

Por outro lado, a experiência tem demonstrado que a transmissão de informações através de textos nem sempre é a maneira mais adequada de fazê-la. Frases podem receber interpretações diferentes, dependendo da pessoa que estiver analisando o texto, e normalmente perde-se a visão geral do problema quanto mais detalhes são colocados. Além do mais, existe a dificuldade adicional de homogeneização dos documentos devido aos diferentes estilos de escrita.

Em decorrência disto, uma boa metodologia de programação deve fazer uso intensivo de formas gráficas como meio de registro das soluções alcançadas, pois o seu uso possibilita uma comunicação mais eficiente bem como também agiliza o trabalho de documentação em função da capacidade de concisão dos gráficos. O uso de textos deve ficar restrito a complementações, como forma de esclarecimento para certos detalhes.

Como justificativa para esta afirmação apresentamos na fi

gura abaixo, nas duas formas discutidas, a interdependência de um conjunto de atividades, sendo que como notação gráfica adotamos aquela definida para o método CPM. Como pode ser observado, o gráfico permite a visualização do problema de uma forma mais imediata e concisa que o texto.

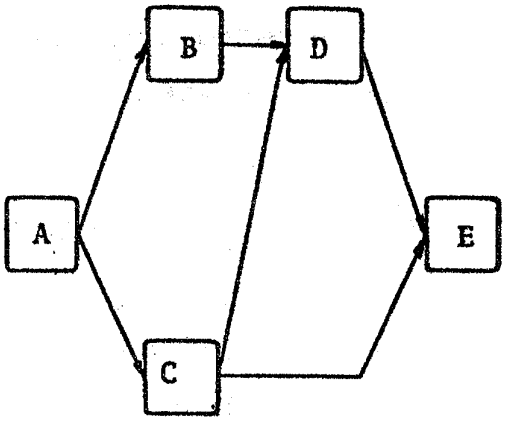
Descrição Textual	Descrição Gráfica
Início do trabalho, atividade A. O início das atividades B e C dependem do término da atividade A e são independentes entre si. O início da atividade D depende do término de B e C. E é a última atividade e só pode ser iniciada após o término de C e D.	

Figura 2.1 - Descrições alternativas de um conjunto de atividades.

PRODUTIVIDADE

Num ambiente de produção de grandes projetos tem-se de 5 a 10 instruções codificadas por homem/dia {02}. Assim sendo, e considerando-se que este número de instruções não demanda mais do que poucos minutos para serem codificados, a maior parte do tempo de um programador é dedicada a depuração de programas e recodificação de instruções.

Por isso, uma boa metodologia é aquela que promove a diminuição de erros na programação e tornem os programas mais fáceis de serem depurados.

Um outro ponto importante para o incremento da produtividade diz respeito a organização e metodização do trabalho de programação.

Em muitas atividades observa-se que a produtividade é fun

ção entre outras variáveis, de quão padronizada (rotineira) é a tarefa a ser executada. Como exemplo, tomemos a tarefa de digitação: quanto mais padronizado for o formulário e o processo adotado para digitá-lo maior será a produtividade alcançada. A razão disto está no fato de que com a padronização diminui-se o universo de decisões a serem tomadas, proporcionando a automatização dos atos e consequentemente o aumento de velocidade.

Não existem razões para que tais princípios não possam ser aplicado no desenvolvimento de programas. Embora os problemas a serem resolvidos não sejam exatamente iguais (como podem ser os formulários do exemplo dado) observa-se que estes podem ser agrupados em classes de problemas com determinadas características, cujas soluções apresentam estruturas semelhantes. Isto tanto é verdade que pode-se desenvolver, por exemplo, uma estrutura genérica para problemas de atualização de cadastro sequenciais ("balance-line").

Visto que pode-se sempre identificar uma certa semelhança entre problemas aparentemente distintos, resta observar a constância no processo de resolução. Neste sentido, uma boa metodologia é aquela que sistematiza o processo de análise do problema.

Ressaltamos que esta sistematização não tem reflexos apenas na produtividade, mas também no planejamento e controle dos trabalhos de gerência. Se estão bem definidas as diversas etapas do processo de desenvolvimento, mais facilmente pode-se estimar prazos alocar recursos e controlar os resultados obtidos.

Uma decorrência importante do fato de se levar em consideração a semelhança entre problemas é o desenvolvimento e divulgação de estruturas padrões, a serem adaptadas para problemas particulares - tais como o "balance line" - o que aumenta a produ

tividade em geral e induz ao aprimoramento contínuo de estruturas de programas.

QUALIDADE DO PROGRAMA

A principal qualidade de um programa, certamente, é a de executar as funções especificadas pelo usuário. Entretanto, isto pode ser alcançado a partir de soluções bastante distintas e, por este motivo, outras propriedades devem ser consideradas quando se tem como objetivo programas de boa qualidade.

Segundo Enos e Van Tilburg {10} estas outras propriedades seriam:

- Testabilidade: o desempenho do programa pode ser avaliado contra os requisitos do usuário de maneira quantitativa;
- Manutenibilidade: o programa pode ser entendido pelos responsáveis pela atividade de manutenção, sendo fácil de modificar e de testar quando da necessidade de inclusão de novos requisitos, correção de deficiências ou erros;
- Eficiência: o programa executa tarefas úteis ao usuário sem excessivo disperdício de recursos;
- Inteligibilidade: o usuário pode facilmente entender o funcionamento do produto e as relações entre estes e outros produtos e componentes do sistema;
- Adaptabilidade: as características do projeto do programa incluem legibilidade, independência de dispositivos, estruturação e "self-containment", e pode facilmente ser transferido para um equipamento similar.

Além destas propriedades, tem-se aquela denominada Robustez e que se refere a habilidade do programa em operar adequadamente mesmo quando ocorrerem erros de uso, ou condições de operações adversas.

Da análise destas propriedades observa-se que a Eficiência,

Inteligibilidade, Robustez e parte da Adaptabilidade - no que concerne a independência de equipamentos - correspondem à funcionalidade do programa, significando obediência a especificações ou inclusão de requisitos visando sua segurança, seu desempenho, etc... Independentemente da origem, estas propriedades se referem a "o que" o programa deverá conter.

Já a Testabilidade, Manutenibilidade e demais características relacionadas com Adaptabilidade são propriedades dependentes da filosofia de desenvolvimento do programa, isto é, de "como" este é elaborado.

Assim sendo, no contexto de qualidade, uma boa metodologia é aquela que tem como objetivo a clareza do projeto ("design") e uma documentação que garanta a legibilidade e compreensibilidade do programa, e que também inclua características (tais como modularidade) que facilitem sua alteração. Por outro lado, deve considerar a incorporação de requisitos para atender as demais propriedades.

PRATICABILIDADE

Entendido o termo praticabilidade como sinônimo de "fácil aprendizagem e uso", então este ponto, ao contrário dos anteriores, se refere muito mais a uma característica intrínsica da metodologia do que a seu relacionamento com o processo de desenvolvimento de programas. Porém, não se pode deixar de considerá-lo na análise de uma metodologia, dado que a existência destas particularidades é fator preponderante para a aceitação e utilização de uma nova ferramenta de trabalho.

A realidade de nossos centros de processamento é a de que a grande maioria dos profissionais, principalmente na atividade de programação, possuem uma formação matemática limitada. Desta forma, torna-se bastante difícil a implementação de métodos que

exijam formação mais sofisticada - por exemplo, métodos de prova de correção de programas, existindo grande probabilidade que qualquer tentativa nesse sentido venha a resultar em fracasso.

Assim sendo, uma boa metodologia é aquela que não requeira pré-requisitos de formação, e que, de preferência, seja constituida de aplicações de conceitos e regras compreensíveis através do bom senso.

Quanto à facilidade de uso, uma boa metodologia é aquela que apresenta uma nomenclatura simples e não implica na utilização de todo um "aparato" para a descrição do programa - infinidade de gabaritos e formulários especiais - já que estes detalhes constituem um desestímulo à sua aplicação.

GENERALIDADE

A utilização de várias metodologias em um Centro de Processamento de Dados traz consigo sérias incoveniências, principalmente quanto à dificuldade de padronização da documentação e treinamento dos profissionais.

Neste sentido, então, uma boa metodologia é aquela que pode ser aplicada ao maior número possível de problemas diferentes.

SISTEMA DE SUPORTE

A elaboração de documentação pelo processo manual sempre é lenta e ocupa grande parte do tempo do pessoal envolvido com o desenvolvimento de programas. Por outro lado, a validação de soluções encontradas é um trabalho bastante complicado, quando não impossível de ser realizado a mão.

Assim sendo, uma boa metodologia é aquela que esteja suportada por um sistema automatizado que contribua na elaboração da documentação e na validação dos resultados alcançados ao longo do projeto do programa.

3 — TÉCNICAS TRADICIONAIS

Este capítulo será dedicado à apresentação de técnicas que constituiram os primeiros esforços para o estabelecimento de métodos de trabalho de auxílio à atividade de programação.

Ainda que estas técnicas não possam ser caracterizadas como verdadeiras metodologias de programação, a apreciação das mesmas nos ajudará a compreender melhor a revolução ocorrida em meados dos anos 60, quando foi proposto o que se denominou Programação Estruturada. Este fato, muito mais do que razões históricas, foi o que determinou a elaboração deste capítulo.

Selecionamos para nossa discussão os clássicos Fluxogramas e as Tabelas de Decisão, por serem as técnicas mais significativas.

3.1 - FLUXOGRAMA (FLOWCHARTING)

3.1.1 - Apresentação

Fluxograma é um diagrama que combina certos símbolos e frases abreviadas para descrever uma sequência de operações, tanto de um sistema de processamento de dados, como de um programa. Estes símbolos incluem figuras geométricas tais como retângulos, círculos, triângulos e losângulos, os quais são conectados por linhas indicando direção ou sequência. Cada símbolo tem um significado - por ex. um retângulo representa um processo e um losângulo é um ponto de decisão - e algumas palavras colocadas no interior do símbolo definem a operação que ele representa.

O Fluxograma foi a primeira técnica desenvolvida com o propósito de representar a solução de um problema de manipulação de dados, tendo sido seus fundamentos estabelecidos por John Von Neumann {06} e seus associados na Universidade de Princeton, ao final dos anos 40. Embora nos detalhes o fluxograma atualmente em uso difira daquele por eles definido, o espírito e a

filosofia permanecem os mesmos, sendo esta uma das poucas ferramentas disponíveis para programação onde um certo grau de padronização foi alcançado.

Face às diversas convenções que foram surgindo ao longo do tempo - como consequência de alterações introduzidas pelos fabricantes de equipamento, por usuário ou grupos de usuários - durante os anos 60, esforços foram realizados pela American Standards Association (ANSI) e a International Standards Organization (ISO) no desenvolvimento de um padrão para fluxogramas. As informações contidas neste trabalho correspondem ao padrão proposto pela ANSI em 1970, no documento denominado "Standard Flowchart Symbols and their use in Information Processing" {01}. Além deste documento, para um estudo mais profundo de fluxogramas, indicamos o artigo escrito por Ned Chapin {06} e o escrito por Henry C. Lucas {18}.

O PADRÃO ANSI

A proposta da ANSI consiste, em primeiro lugar, de uma série de símbolos e regras de uso, divididos em três grandes grupos: BÁSICO, ADICIONAL e ESPECIALIZADO. Fluxogramas completos podem ser desenhados usando-se somente os grupos básico e adicional, sendo opcional os símbolos especializados. Caso estes últimos venham a ser utilizados, devem ser de maneira consistente com a padronização.

Na figura 3.1 tem-se a representação dos símbolos básicos e adicionais e de parte dos símbolos especializados. Salientamos que os símbolos adicionais consistem, basicamente, de facilidades para tornar o fluxograma mais visível ou para indicar a continuação do fluxo de processamento quando o fluxograma não couber em uma única página. Já os símbolos especializados servem para aumentar a legibilidade do fluxograma, pois caracteri

zam os meios de armazenagem de dados utilizados e os tipos de processamento realizados.

Além da especificação dos símbolos a serem utilizados na representação de sistemas e programas, são estabelecidas convenções para, por exemplo, uso de referências cruzadas, cruzamento de linhas de fluxo, uso múltiplo de símbolos, representação de pontos de decisão com grande número de opções, formato básico de fluxogramas e dimensões dos símbolos, linhas de fluxo e setas.

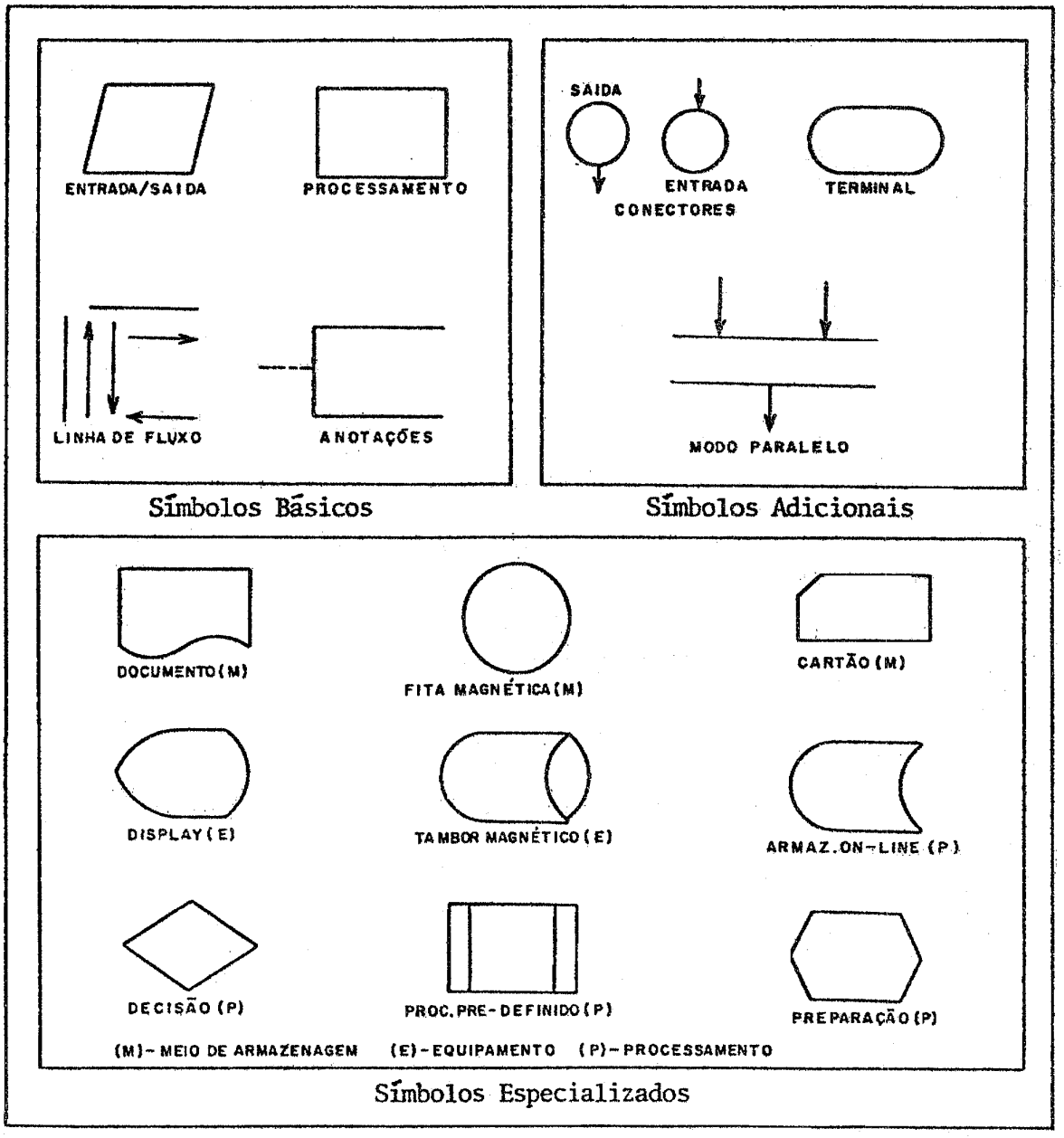


Figura 3.1 - Simbologia para desenho de Fluxogramas

3.1.2 - Exemplos

Como exemplo de aplicação do padrão ANSI, apresentamos a seguir um fluxograma que modela o problema abaixo definido:

a) Enunciado

Desenvolver um programa que emita uma listagem contendo dados de Estoque e Movimentação de Produtos a partir de 2 (dois) arquivos, um contendo a identificação e o estoque antigo do produto e outro contendo as movimentações ocorridas. A partir do estoque antigo e das movimentações deverá ser calculado o estoque atual.

b) Arquivos de Entrada

Cadastro:

Nº PRODUTO	NOME PRODUTO	QUANTIDADE

Movimento:

Nº PRODUTO	Nº DOCUMENTO	QUANTIDADE	CÓDIGO

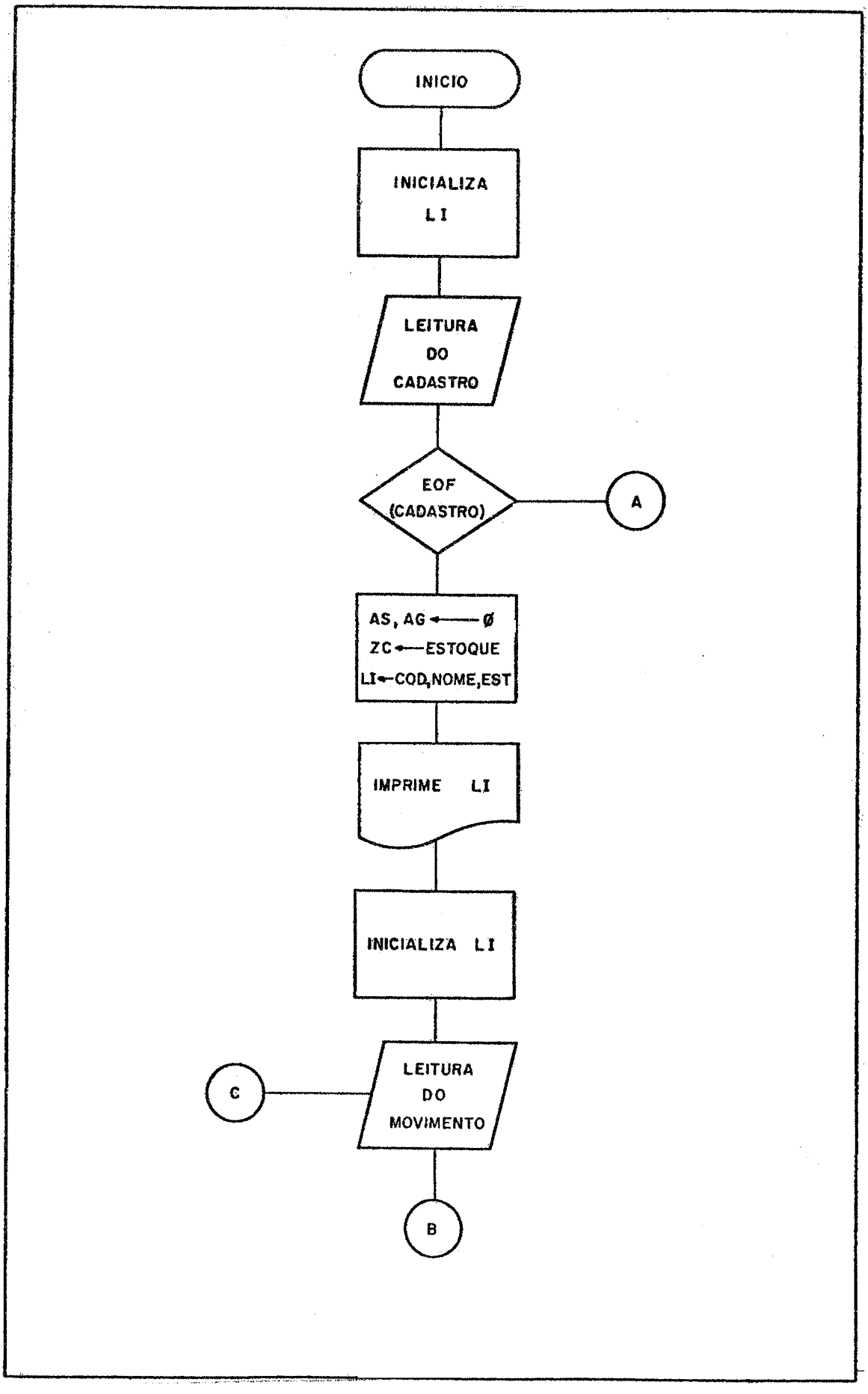
onde Código pode assumir (S=Saída) ou (E=Entrada)

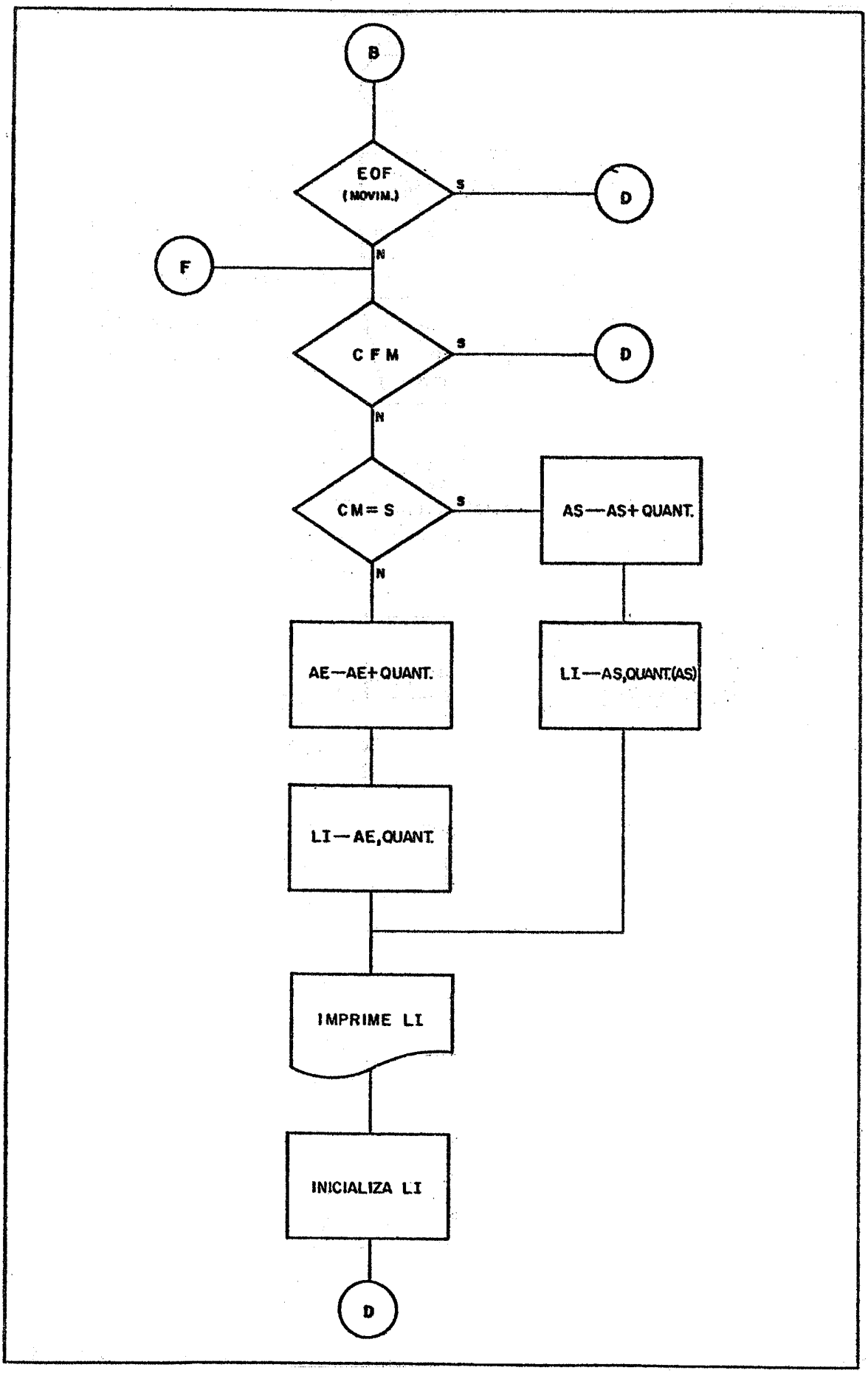
c) Arquivo de Saída (Relatório)

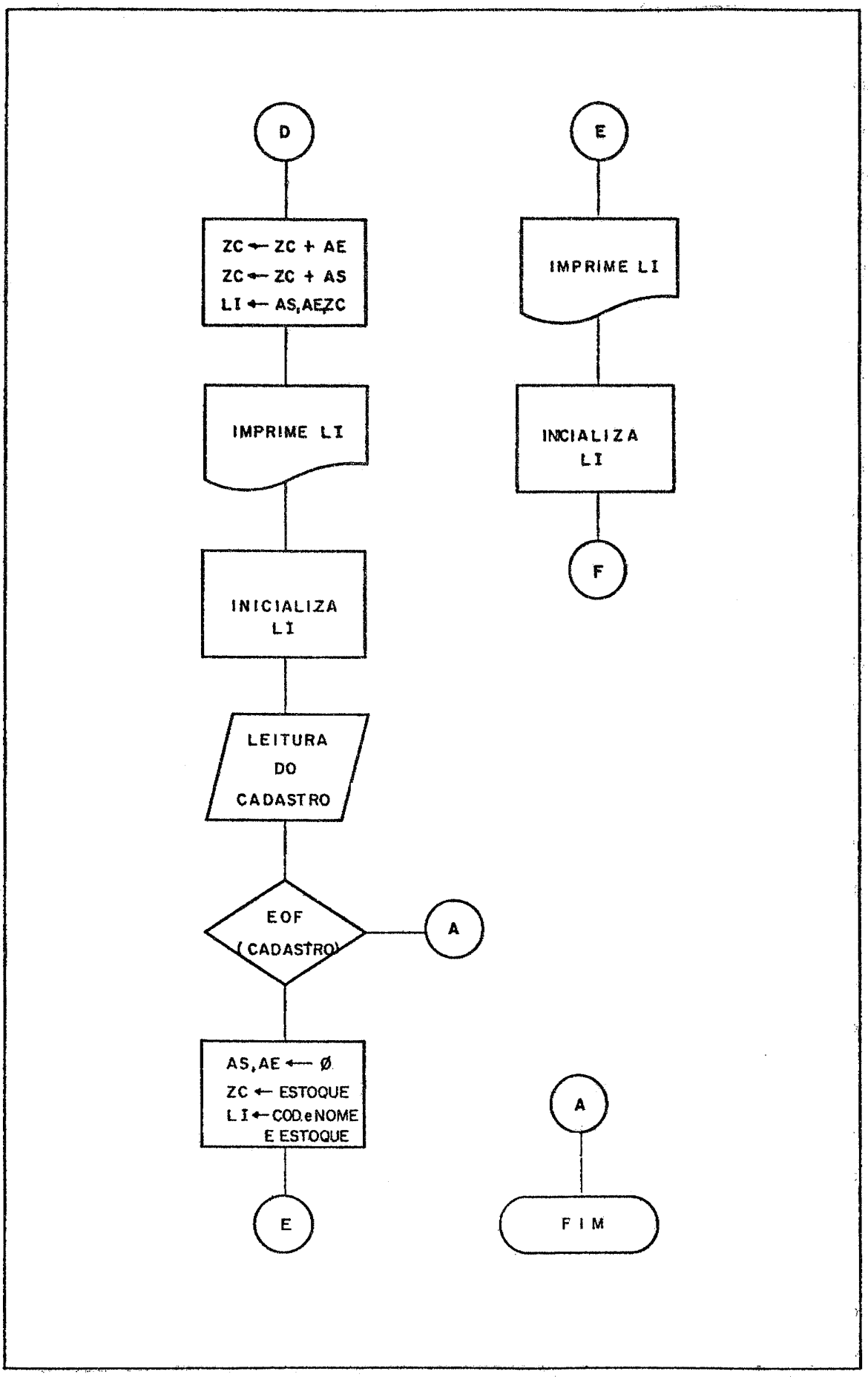
Nº PRODUTO	NOME PRODUTO		ESTOQUE ANTERIOR
	Nº DOCUMENTO	ENTRADA	
	Nº DOCUMENTO	SAÍDA	
		TOTAL ENT	TOTAL SAÍDA	ESTOQUE ATUAL
Nº PRODUTO	NOME PRODUTO		ESTOQUE ANTERIOR

d) Abreviações usadas no fluxograma

LI - Linha de Impressão
ZC - Zona de Cálculo
AE - Total de Entradas
AS - Total de Saídas
C - Código do Produto (cadastro)
M - Código do Produto (movimento)
CM - Código do Movimento


INICIO
INICIALIZA
L I
LEITURA
DO
CADASTRO
EOF
(CADASTRO)
A
AS, AG ← Ø
ZC ← ESTOQUE
LI ← COD,NOME,EST
IMPRIME LI
INICIALIZA LI
C
LEITURA
DO
MOVIMENTO
B


B
EOF
(MOVIM.)
S
D
N
F
C F M
S
D
N
CM = S
S
AS — AS + QUANT.
N
AE — AE + QUANT.
LI — AS,QUANT.(AS)
LI — AE,QUANT.
IMPRIME LI
INICIALIZA LI
D


D
ZC ← ZC + AE
ZC ← ZC + AS
LI ← AS,AE,ZC
IMPRIME LI
INICIALIZA LI
LEITURA DO CADASTRO
EOF (CADASTRO)
A
AS,AE ← Ø
ZC ← ESTOQUE
LI ← COD.e NOME E ESTOQUE
E
E
IMPRIME LI
INCIALIZA LI
F
A
FIM

3.1.3 - Considerações

O que podemos deduzir do padrão para fluxogramas proposto pela ANSI é que esta ferramenta consiste apenas de um conjunto de símbolos e regras a serem observados quando da diagramação de procedimentos ou fluxo de processamentos. Não está contida nesta ferramenta uma orientação para a análise do problema apresentado.

É, essencialmente, uma ferramenta de documentação que, se utilizando de formas gráficas, favorece a comunicação da solução encontrada. Entretanto, isto só é verdade para programas simples, que podem ser representados em uma, ou poucas páginas, e que não possuem muitos pontos de decisão. Caso contrário, a legibilidade do fluxograma fica comprometida.

No tocante a documentação através desta técnica, vale considerar a crítica feita por Frederick P. Brooks {04}, que defende o uso de programas auto-documentados. A colocação deste autor é, se cada símbolo no fluxograma corresponde a uma instrução em uma linguagem de alto-nível, então não tem sentido ser desenhado um fluxograma antes do início de sua codificação. Também diz que esta técnica teve sentido apenas quando introduzida por Von Neumann "pois os símbolos e seus conteúdos serviam como uma linguagem de alto-nível agrupando as impenetráveis declarações em linguagem de máquina".

Posto estas considerações gerais, analisemos agora os fluxogramas com respeito a cada uma das características definidas no capítulo 2.

a) Registro de Soluções e Comunicação

Como já foi dito, o padrão ANSI não contém em si um processo de análise de problemas através de refinamentos sucessivos, não constituindo, portanto, um auxílio efetivo ao proces

so de desenvolvimento de programas. Por outro lado, ao não estabelecer etapas a serem percorridas e os resultados a serem alcançados em cada uma delas, dificulta o acompanhamento e controle dos trabalhos.

Embora seja uma ferramenta essencialmente gráfica, sua comunicação é prejudicada em função do excesso de detalhes que normalmente um diagrama contém. Naturalmente, poder-se-ia elaborar diagramas resumidos, contendo apenas as principais estruturas, tornando-os mais claros. Contudo, isto é muito dependente da capacidade do programador e mesmo do seu "bom gosto". Além do mais, estes resumos não são contemplados no padrão ANSI.

b) Produtividade

A facilidade com que os desvios (GO TO's) podem ser representados em um fluxograma permite a construção das lógicas mais absurdas, o que tem reflexos negativos tanto na depuração como na manutenção de programas.

Esta técnica não induz os programadores a comportamentos padrões para a abordagem de problemas e não favorece a visualização de estruturas semelhantes. Com a experiência, os programadores "sentem" esta semelhança e passam a utilizar as formas concebidas anteriormente na elaboração de seus novos fluxogramas. Entretanto, o destaque de semelhanças não é uma propriedade do fluxograma.

c) Qualidade

O uso desta ferramenta não contribui para a qualidade do programa, já que não interfere quanto a clareza do seu projeto, e não necessariamente leva a documentações que garantam a legibilidade e compreensibilidade do programa. Não enfatiza a modularidade, o que também dificulta a sua manutenção e a inclusão de requisitos visando sua eficiência.

d) Praticabilidade

Por consistir apenas de um conjunto de símbolos e regras, o padrão ANSI é bastante fácil de ser aprendido e de ser utilizado, principalmente, quando se usa gabaritos e formulários padronizados.

Entretanto, por ser muito detalhado, os programadores apresentam certa resistência no seu uso, preferindo partir para uma codificação direta - observando então o já salientado por Brooks sobre a equivalência entre símbolos e comandos da linguagem. Este mesmo detalhamento torna a manutenção dos diagramas bastante trabalhosa.

e) Generalidade e Sistemas de Suporte

Pelas mesmas razões que o tornam prático, o fluxograma é uma ferramenta genérica, podendo ser aplicado a qualquer classe de problema.

Não existem sistemas automatizados que dêm suporte a sua utilização, sendo encontrados no mercado apenas alguns sistemas que, a partir do código do programa, geram o fluxograma correspondente. Além de não serem muito claros, os fluxogramas gerados significam um trabalho de documentação "a posteriori", o que não tem muito sentido.

3.2 - TABELAS DE DECISÃO

3.2.1 - Apresentação

Como já foi dito, o Fluxograma tradicional é uma ferramenta limitada, cujo poder de comunicação fica restrito à representação de programas simples, com poucos pontos de decisão. A sua aplicação na representação de uma sequência grande de comandos, com uma série de desvios, resulta em diagramas bastante complexos em função do aparecimento de muitas ramificações. Neste caso a clareza deixa de existir, ficando difícil a visualiza

ção da estrutura global do programa, e mesmo o acompanhamento do diagrama com o intuito de se verificar a sua completeza. Desta forma, métodos alternativos se fazem necessários quando da representação de processos que contenham muitos pontos de decisão.

Uma das maneiras mais simples e convenientes de se sumarizar um número de situações "if-then" é a Tabela de Decisão, que mostra a primeira vista a ação ou ações a serem tomadas se uma dada condição ou conjunto de condições ocorrem. Não há limites com relação ao número de condições e regras, que podem ser representadas em uma tabela, porém a prática demonstra que várias pequenas tabelas são preferíveis a uma única grande tabela, sendo a comunicação função inversa do seu tamanho. Mesmo assim, as tabelas de decisão complexas, são mais fáceis de serem lidas que os fluxogramas equivalentes.

FORMATO BÁSICO DE TABELAS DE DECISÃO

Uma tabela de decisão é a representação tabular de relações lógicas entre condições e ações de tal maneira que a manipulação de dados fique explicada.

O formato básico de uma Tabela de Decisão, apresentado na figura 3.2, consiste de seis seções, a saber:

CABEÇALHO: contém de informações gerais que permitem a identificação de tabela de decisão;

DECLARAÇÃO DE CONDIÇÕES: lista condições que devem ser consideradas na situação de decisão;

REGISTRO DE CONDIÇÕES: onde são indicadas as combinações possíveis de condições e que formam cada regra;

DECLARAÇÃO DE AÇÃO: lista as ações a serem tomadas como resultado de condições satisfeitas em cada regra;

REGISTRO DE AÇÕES: especifica as ações a serem executadas como

resultado do cumprimento das condições em cada regra;

OBSERVAÇÕES: contém, quando couberem, comentários a respeito do conteúdo da tabela.

Algumas informações adicionais que podem ser encontradas em uma Tabela de Decisão são os identificadores de condições, ações e regras, que servem como códigos de identificação de tais dados.

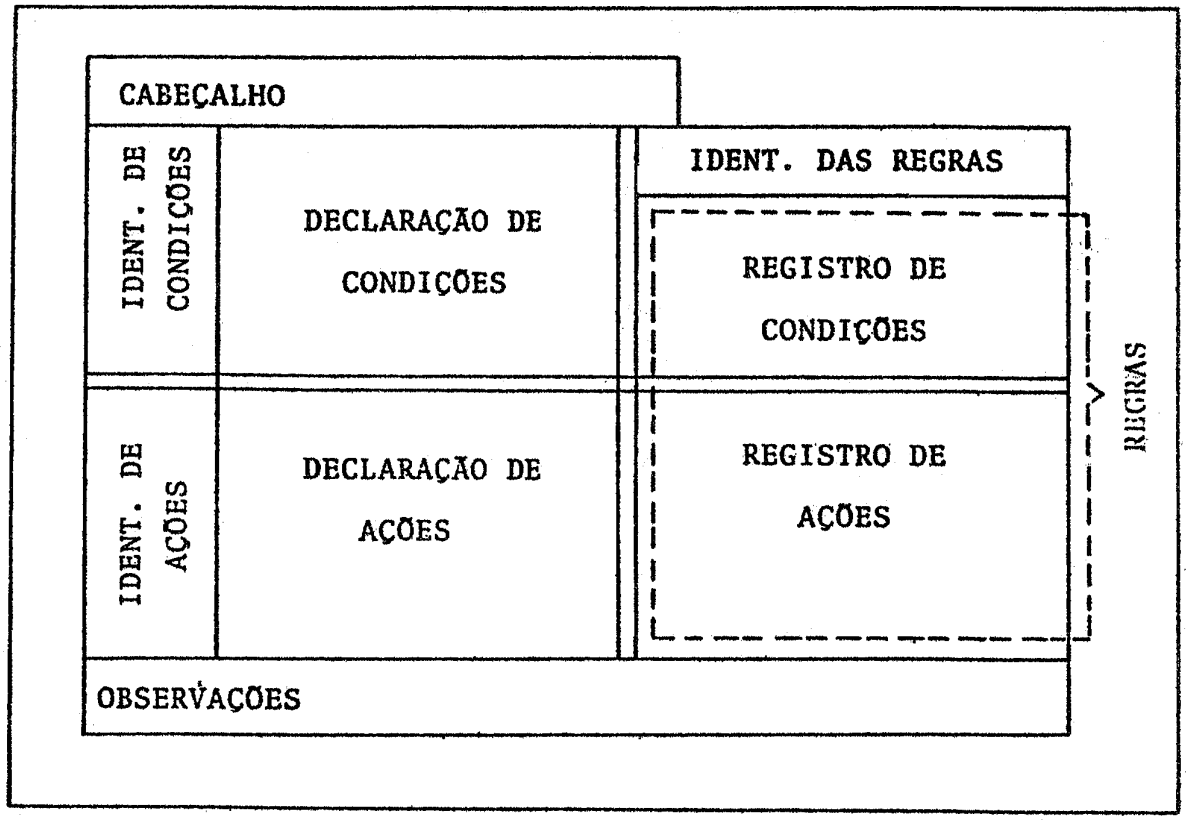


Figura 3.2 - Formato Básico de Tabelas de Decisão

Um exemplo de tabela de decisão é apresentado na figura 3.3, e corresponde ao procedimento de um "carioca" ao se levantar pela manhã, cuja descrição seria a que segue:

"Se o dia corresponder a um sábado, domingo ou feriado e a previsão do tempo é de sol, então tomar café e ir para a praia, caso contrário voltar para cama. Se dia normal, independente da previsão do tempo, tomar café e ir para o trabalho".

PROCEDIMENTO DE UM CARIOCA AO SE LEVANTAR PELA MANHÃ

		R1	R2	R3
C1	DIA NORMAL	S	N	N
C2	TEMPO BOM	-	S	N
A1	TOMAR CAFÉ	X	X	-
A2	IR PARA O TRABALHO	X	-	-
A3	IR PARA A PRAIA	-	X	-
A4	VOLTAR PARA CAMA	-	-	X

OBSERV.: (S)- SIM (X)- AÇÃO A SER TOMADA
(N)- NÃO (-)- IRRELEVANTE OU NÃO SE APLICA

Figura 3.3 - Exemplo de Tabela de Decisão

O formato vertical aqui exemplificado é o mais comum. Contudo, encontra-se também tabelas de decisão no formato horizontal, onde as regras trocam de posição com as condições/ações. Por outro lado existem variações quanto a formulação das regras, o que define o tipo de tabela de decisão. Estas e outras variações podem ser apreciadas no artigo de Fergus {11}.

REPRESENTAÇÃO DE PROGRAMAS

A representação de um programa através de uma tabela de decisão é feita obedecendo as mesmas regras de construção já apresentadas. É importante observar que ao contrário do fluxograma, onde as operações são ordenadas no tempo, a tabela de decisão constitui uma representação simultânea. Tal característica é que a torna mais compacta e mais flexível.

Para indicar a sequência de execução das ações adota-se o uso de indicações numéricas em lugar de X's. Estes e outros recursos para melhorar a legibilidade em Tabelas de Decisão são encontrados no livro de Hartman {12}.

3.2.2 - Exemplo

A seguir mostramos a Tabela de Decisão que modela o problema definido no item 3.1.2.

TABELA 1

	CONDIÇÕES	REGRAS 01	02	03	04	05	06
01	INÍCIO DO PROGRAMA	S	N	N	N	N	N
02	FIM DO ARQUIVO CADASTRO	-	S	N	N	N	N
03	FIM DO ARQUIVO MOVIMENTO	-	-	S	N	N	N
04	COD.PROD.(C) ≠ COD.PROD.(MOV.)	-	-	-	S	N	N
05	COD.MOVIMENTO = "E"	-	-	-	-	S	N
06	COD.MOVIMENTO = "S"	-	-	-	-	-	S
	A Ç Õ E S						
01	AS. AE ← Ø	3		6	6		
02	ZC ← ESTOQUE	4		7	7		
03	INICIALIZAR LI	1					
04	AE ← AE + QUANT					1	
05	AS ← AS + QUANT						1
06	ZC ← ZC + AE			1	1		
07	ZC ← ZC - AS			2	2		
08	LI ← COD., NOME, ESTOQUE	5					
09	LI ← Nº DOCUM., QUANT(AS)						2
10	LI ← Nº DOCUM., QUANT(AE)					2	
11	LI ← AS, AE, ZC (EST.ATUAL)			3	3		
12	IMPRIMIR LI E INICIALIZAR LI	6		4	4	3	3
13	LER ARQ. CADASTRO	2		5	5		
14	LER ARQ. MOVIMENTO	7				4	4
15	FINALIZA PROGRAMA		1				
16	GO TO TABELA 1	8		8	8	5	5

OBSERVAÇÕES:
S - Sim
N - Não
\- - Irrelevante
1-9 - Ação a ser tomada

3.2.3 - Considerações

Assim como o fluxograma, a Tabela de Decisão não traz consigo, intrinsicamente, uma filosofia de desenvolvimento de programas através de refinamentos sucessivos, constituindo apenas uma forma alternativa, mais prática, para se representar processos, ou partes de processos, que envolvam várias condições e ações.

A seguir apresentamos algumas vantagens das Tabelas de Decisão sobre os fluxogramas, apontadas em [11]:

- São mais fáceis de serem construídas, revisadas, lidas e mantidas;
- Condições e ações são apresentadas de uma maneira clara e ordenada;
- Aplicações envolvendo interações complexas de variáveis são documentadas mais claramente do que com fluxogramas;
- Como o teste de condições é apresentado separado das ações a serem tomadas, é fácil seguir o curso exato para a conclusão da tabela e checar sua completeza. Esta separação entre condições e ações também favorece a modularização do programa.
- Facilitam a comunicação entre as várias pessoas envolvidas no projeto e codificação do programa, promovendo uma melhor definição do problema.

Dadas estas informações gerais a respeito de Tabelas de Decisão, vejamos agora as considerações específicas, tomando por base as características estabelecidas para uma boa metodologia.

a) Registro de Soluções e Comunicação

Não constituindo um processo de refinamentos sucessivos para desenvolvimento de programas, as Tabelas de Decisão

não atuam de forma contundente para a solução do problema. Entretanto, vale observar que elas representam um nível de solução mais alto do que aquele proporcionado pela aplicação do fluxograma, que resulta em descrições gráficas dos comandos de um programa. Uma Tabela de Decisão não fornece o fluxo de processamento, mas apenas indica condições e ações que o programa deverá incorporar.

Ainda que não atue firmemente no processo de desenvolvimento de um programa, a Tabela de Decisão favorece a compreensão do problema pois promove o registro sistemático dos vários processos que deverão ser modelados.

Podendo ser entendida como um nível de conhecimento no processo de desenvolvimento de programas, a utilização desta técnica, em combinação com fluxograma, possibilitaria um ponto intermediário para acompanhamento e controle dos trabalhos.

Quanto ao aspecto de comunicação, as Tabelas de Decisão são mais claras e concisas que os fluxogramas, pertimindo uma visualização mais rápida do processo sendo analisado.

b) Produtividade

A codificação de programas a partir das tabelas de decisão não é tão direta quanto de um fluxograma, o que pode proporcionar o aparecimento de programas com uma lógica obscura, difíceis de serem depurados e mantidos. Uma tabela clara não necessariamente implica num programa claro, sendo isto função do profissional que o estiver codificando.

O uso costumeiro da Tabela de Decisão cria um comportamento padrão para a abordagem de problemas, porém não incrementa a visualização de estruturas semelhantes.

c) Qualidade

Promovendo a modularização dos programas, uma condi

ção que facilita a manutenção, esta técnica influi positivamente no que tange a qualidade final dos mesmos. Por outro lado, as Tabelas de Decisão podem constituir uma ferramenta de auxílio para a compreensão e legibilidade dos programas fontes.

Além desses aspectos devemos considerar que as Tabelas de Decisão permitem verificar a ocorrência de todas as condições do programa o que redunda em produtos de melhor qualidade.

d) Praticabilidade

É uma técnica fácil de ser aprendida e utilizada, podendo-se tornar seu uso mais agradável com a definição de formulários padronizados.

e) Generalidade

As Tabelas de Decisão podem ser aplicadas em quaisquer problemas. Contudo, vale observar que ocorrem situações onde sua utilização não seria tão eficiente, valendo mais a pena o desenho de um fluxograma. Estas situações seriam aquelas onde são poucas as condições consideradas no problema.

f) Sistemas de Suporte

Ao contrário do fluxograma, esta técnica é suportada por uma série de sistemas automatizados que visam tanto a verificação da completeza das tabelas, como também a geração de códigos de programa a partir delas. Exemplos destes sistemas são o TABSOL, o LOGTAB e o FORTAB [11].

4 - HISTÓRIA E FUNDAMENTOS DA PROGRAMAÇÃO ESTRUTURADA

Os primeiros anos da década de 60 foram marcados pelo que se denominou "software crisis", resultante da constatação da existência de custos crescentes no desenvolvimento de programas e uma taxa custo/qualidade relativamente alta nos sistemas desenvolvidos.

O aumento dos custos era consequência, basicamente, da baixa produtividade apresentada pelos programadores que passavam a maioria de seu tempo na depuração de erros nos programas.

Por sua vez, a baixa qualidade dos sistemas desenvolvidos, que podia ser traduzida pelos programas sem as propriedades mencionadas no capítulo 2, era decorrente tanto da má especificação do problema, devido a falta de ferramentas adequadas à análise de problemas complexos, como também ao desenvolvimento de projetos e textos de programas pobres, que dificultavam sua compreensão.

Tal situação, no fundo, era devida ao uso de métodos de trabalho já ultrapassados para as novas exigências que surgiam naquele momento: aplicações mais complexas a serem desenvolvidas em menor espaço de tempo. Esta situação passou então a requerer uma revisão dos métodos utilizados no desenvolvimento de sistemas. O "hardware" e o "software" oferecido pelos fabricantes começavam a propiciar o desenvolvimento de novas classes de aplicações, enquanto os métodos de trabalho disponíveis permaneciam os mesmos da época dos equipamentos à válvula.

Uma resposta ao problema foi apresentada por Dijkstra, em 1965, em seu artigo "Programming Considered as a Human Activity" [07], quando defendeu a elaboração de programas de forma estruturada e sugeriu a eliminação do comando GO TO das linguagens de programação. Este trabalho foi o marco inicial de uma nova

era na atividade de programação, e que se denominou Programação Estruturada.

Em 1966 um grande passo para a programação estruturada foi dado por C. Bohm e G. Jacopine {03} ao demonstrarem que a lógica de qualquer programa pode ser expressa a partir de três estruturas de controle básicas. Estas estruturas, apresentadas na figura 4.1, incluem: (1) um mecanismo de sequência, (2) um mecanismo de seleção e (3) um mecanismo de repetição.

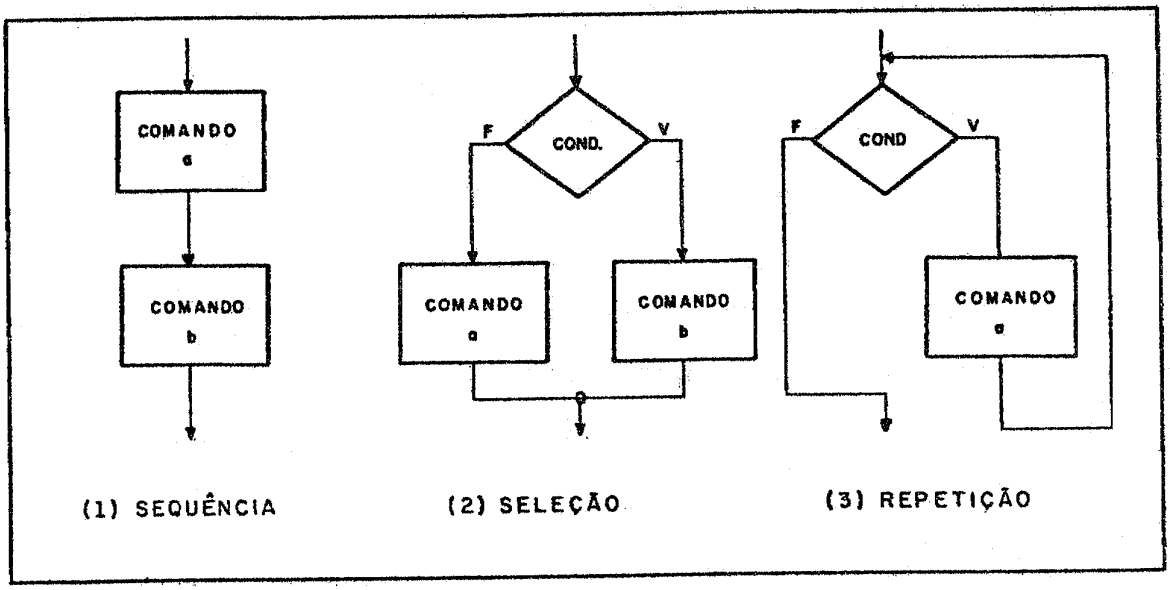


Figura 4.1 - Estruturas de Controle Básicas.

Um impulso maior, entretanto, foi dado pelo próprio Dijkstra, com a sua famosa carta "GO TO Statement Considered Harmful", publicada em 1968 {08}. Nesta publicação ele salientava que, na codificação de programas, deveria ocorrer uma correspondência entre o texto do programa e o processo que este programa descrevia, ou seja, dever-se-ia poder ler o programa de cima para baixo, e que o uso descuidado de comandos de desvio (GO TO) tornava impossível esta correspondência. O impacto destas afirmações foi tão significativo que "programação estruturada" passou a ser interpretada como sendo "programação sem o uso de GO TO"; os objetivos reais da programação estruturada, que eram a efi

ciência na programação, a legibilidade e a correção dos programas, ficaram relegados a um segundo plano.

É importante a observação desta controvérsia em torno do uso ou não do comando GO TO. Naturalmente, o uso irrestrito deste comando poderá levar a programas ilegíveis e portanto de difícil manutenção. Porém, nem sempre isto é verdade, sendo que em muitas vezes o uso do mesmo se faz necessário para aumentar a clareza do programa. Exemplos desta natureza, onde o impedimento do uso de GO TO levam a estruturas complexas, devido à necessária inclusão de várias chaves, foram apresentadas por Knuth {16} e M.E. Hopkins {14}. Assim, a presença ou ausência de comandos GO TO em um programa não fornecem uma boa medida para a sua performance. Na verdade, a rara ocorrência de comandos GO TO, a presença das estruturas de controle e as identificações aninhadas de segmentos de programas são apenas reflexos deste estilo de programação, e não a própria programação estruturada.

O princípio fundamental da programação estruturada é o processo denominado "Stepwise Refinement" (refinamentos sucessivos) estabelecido por Niklaus Wirth em 1971 {31}. Melhorando um processo originalmente concebido por Dijkstra {09}, ele definiu o procedimento básico para o desenvolvimento de um programa de forma estruturada. Ele sugeriu que a elaboração de um programa deveria consistir de uma sequência de passos de refinamento, sendo que cada passo deveria corresponder a divisão de uma dada tarefa em um certo número de sub-tarefas. Este refinamento na descrição do problema deveria ser acompanhado, em paralelo, por um refinamento na descrição dos dados, os quais constituem o meio de comunicação entre as sub-tarefas.

Assim, "programação estruturada deve ser entendida como a aplicação de métodos básicos de decomposição de problemas esta

belecendo uma estrutura hierárquica e, deste modo, administrável, à semelhança de métodos de decomposição comuns na engenharia, física, química e ciências biológicas" {10}. Como consequência, uma definição para programação estruturada poderia ser aquela dada por Wirth {32}:

"Programação estruturada é a formulação de programa como estruturas hierárquicas encaixantes de comandos e objetos de computação".

Tal definição, por si só, não determina os instrumentos ou construção de controles lógicos que devem ser usados para se elaborar um programa estruturado, nem limita as técnicas de formulação de estruturas disponíveis para o programador, permitindo afirmar que, antes de mais nada, programação estruturada é uma filosofia, ou melhor, uma maneira de proceder com relação a elaboração de programas. Isto tanto é verdade que hoje existem vários caminhos para o desenvolvimento de programas estruturados e que são representados pelas diferentes metodologias de programação propostas na década de 70, das quais algumas possamos apresentar a seguir.

5 - TÉCNICAS DE ESTRUTURAÇÃO

Neste capítulo serão apresentadas as técnicas de programação: NASSI-SHNEIDERMAN CHARTS, MÉTODO DE JACKSON, LÓGICA DE CONSTRUÇÃO DE PROGRAMAS-LCP, HIPO e COMPOSITE DESIGN, propostas nos anos 70, e que trazem na sua formulação os fundamentos da Programação Estruturada. Como poderá ser observado, ao contrário do Fluxograma tradicional e Tabela de Decisão, que, no fundo, significam somente uma forma de documentação, estas técnicas permitem que se fale de metodologias de programação, já que agem no modo de pensar e condionam as ações dos programadores.

Como não poderia deixar de ser, devido à dimensão deste trabalho, as técnicas aqui discutidas representam apenas um subconjunto das formulações feitas nos últimos anos. Entretanto, em nosso ponto de vista, elas são bastante significativas, o que pode ser medido pelos inúmeros artigos, livros e seminários dedicados a uma ou mais destas técnicas.

Por outro lado, com suas abordagens distintas para o mesmo problema, elas caracterizam bem os diversos caminhos que podem ser seguidos na elaboração de programas estruturados.

5.1 - NASSI-SHNEIDERMAN CHARTS

5.1.1 - Apresentação

Apesar dos conceitos de programação estruturada poderem ser conciliados com o uso de fluxogramas convencionais, nota-se que o uso desta técnica pode dificultar ou até mesmo prejudicar o desenvolvimento de programas estruturados.

A dificuldade aparece, no sentido de algumas estruturas de controle - por ex. as de iteração - não tem tradução direta na simbologia padrão do fluxograma e devem ser constituídas a partir de adaptações. Este fato introduz cuidados e tarefas adicionais, além de dificultar a própria compreensão do diagrama.

Por outro lado, a facilidade com que os desvios podem ser representados nos fluxogramas - permitindo a transferência arbitrária de controle - prejudica o desenvolvimento de programas estruturados. Como já foi dito várias vezes ao longo deste trabalho, o uso irrestrito de GO TO's tanto compromete a validação do programa como a sua depuração, otimização e posterior manutenção. Além disso, o fluxograma tradicional não traz consigo o processo de análise de problemas através de refinamentos sucessivos de sua estrutura.

Com base nestes pressupostos, em 1973, I. Nassi e B. Shneiderman [22] propuseram uma nova técnica para construção de diagramas de fluxo de processamento, mais própria aos fundamentos da programação estruturada, e que leva em consideração os recursos das novas linguagens de programação. Um método bastante semelhante ao que será discutido aqui foi desenvolvido e proposto por Ned Chapin [05].

Algumas das principais vantagens desta técnica sobre os fluxogramas tradicionais são:

1. O escopo da iteração, bem como das cláusulas IF-THEN-ELSE, é bem definido e visível;
2. A visualização do escopo das variáveis locais e globais é imediata;
3. Transferências arbitrárias de controle são impossíveis (não existe representação para GO TO's), e estruturas completas podem e devem ser ajustadas em uma única página (não existe conectores de páginas).

NOTAÇÃO

Para viabilizar as vantagens descritas e garantir a construção de diagramas estruturados, Nassi e Shneiderman propuseram uma representação gráfica alternativa para as estruturas de con

trole básicas determinadas por Bohm e Jacopine. Estes símbolos são mostrados na figura 5.1, onde também encontra-se a representação proposta para a estrutura de bloco (BEGIN-END). Observa-se que os símbolos tem a forma de retângulos que podem assumir qualquer dimensão em função da conveniência do programador.

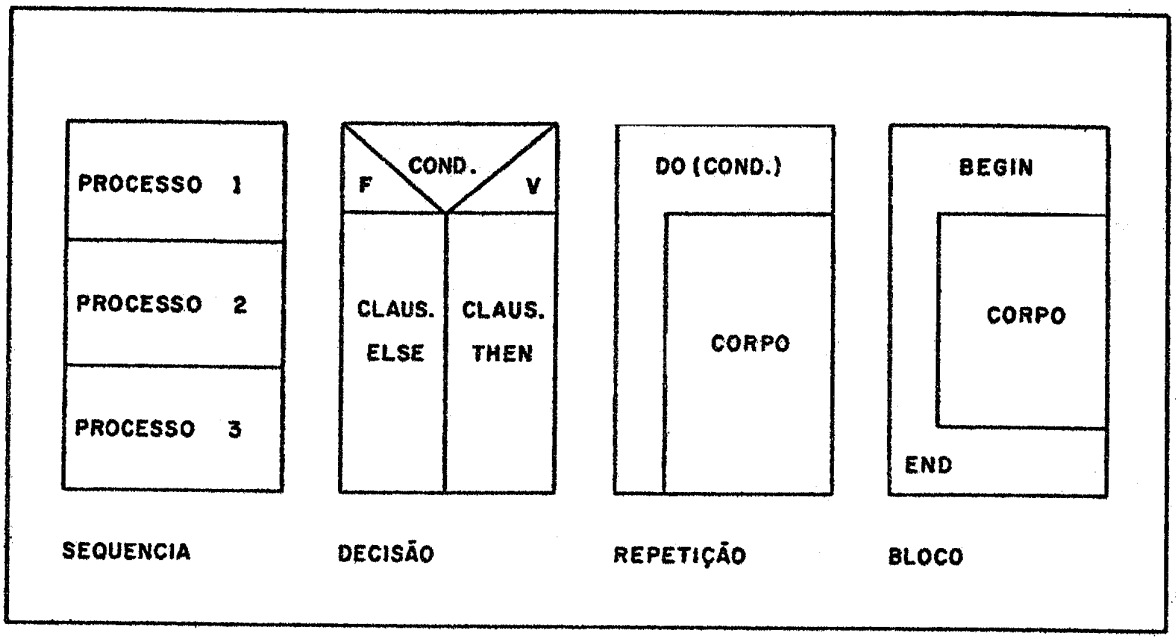


Figura 5.1 - Representação de estruturas de controle básicas.

Assume-se que a sequência de processamento é vertical, de cima para baixo. Retângulos sucessivos são considerados como tendo somente um ponto através do qual o controle passa verticalmente de um retângulo para o próximo.

O símbolo de processo - um retângulo - pode ser usado para representar qualquer comando - designação, leitura/impressão e chamada de sub-rotinas - bem como estruturas mais complexas - por exemplo, uma cláusula IF-THEN-ELSE, um bloco BEGIN-END ou mesmo um trecho de programa representado numa outra folha. Da mesma forma o corpo do bloco BEFIN, ou de uma REPETIÇÃO, pode conter estruturas de qualquer complexidade.

A representação para estruturas específicas, tais como DO UNTIL e DO CASE, contidas em determinadas linguagens também foram propostas por Nassi e Schneiderman, na referência já aponta

da.

DIAGRAMA ESTRUTURADOS E REFINAMENTOS SUCESSIVOS

Uma vantagem adicional do diagrama estruturado sobre os fluxogramas, não evidenciada por Nassi e Shneiderman é o fato de se prestarem ao desenvolvimento de programas por refinamentos sucessivos.

Como já mencionamos no Capítulo 4 o conceito central da análise através de refinamentos sucessivos é o de decompor um procedimento em diferentes subprocedimentos, até a identificação de estruturas onde subdivisões não têm mais sentido.

Na figura 5.2 apresentamos parte do processo de Refinamentos Sucessivos, utilizando-se NASSI-SHNEIDERMAN CHARTS, para o problema definido em 3.1.2

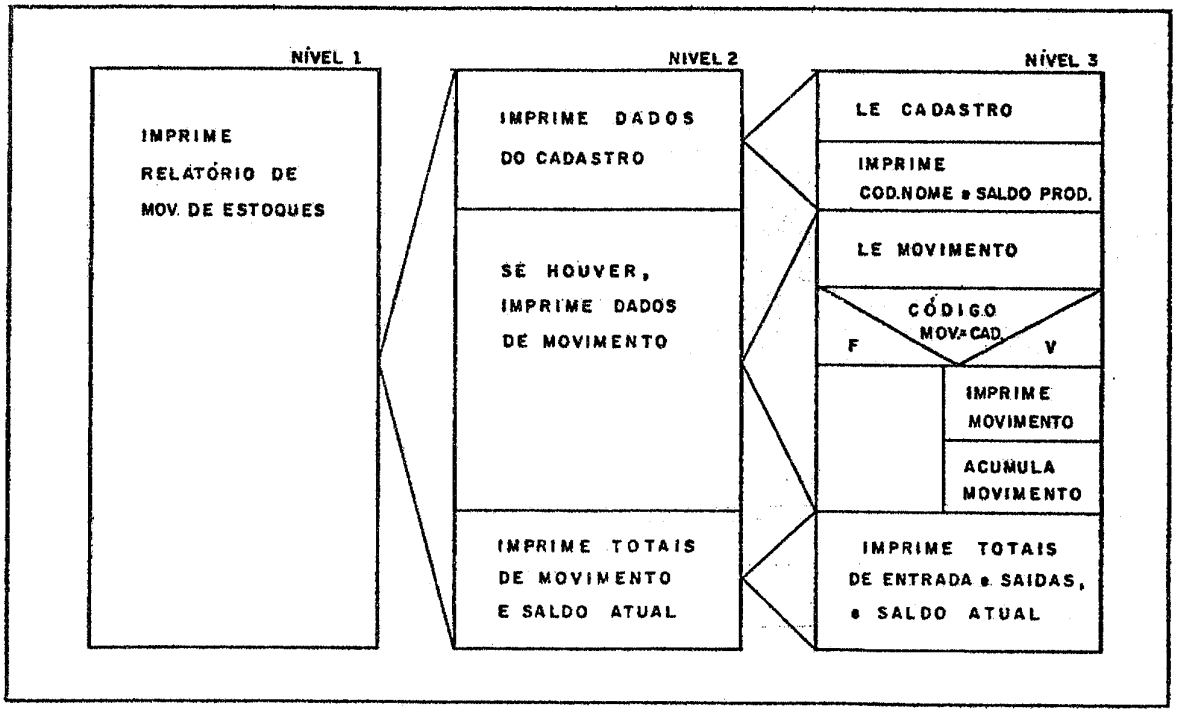
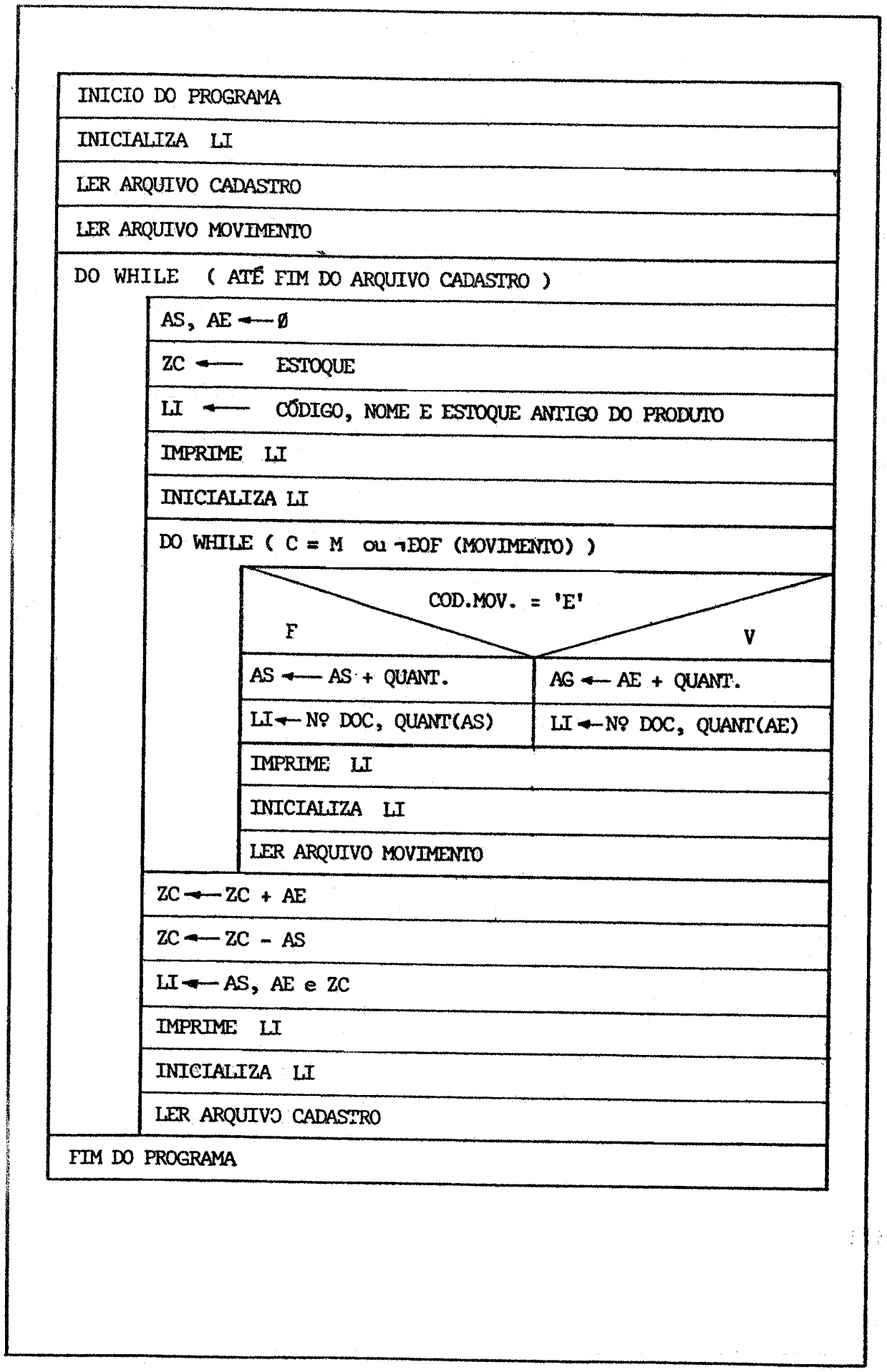


Figura 5.2 - Refinamentos Sucessivos com Nassi-Shneiderman Charts

5.1.2 - Exemplo

A seguir, apresentamos o diagrama detalhado do programa cuja definição encontra-se em 3.1.2.


INICIO DO PROGRAMA
INICIALIZA LI
LER ARQUIVO CADASTRO
LER ARQUIVO MOVIMENTO
DO WHILE (ATÉ FIM DO ARQUIVO CADASTRO)
AS, AE ← Ø
ZC ← ESTOQUE
LI ← CÓDIGO, NOME E ESTOQUE ANTIGO DO PRODUTO
IMPRIME LI
INICIALIZA LI
DO WHILE (C = M ou ¬EOF (MOVIMENTO))
COD.MOV. = 'E'
F
V
AS ← AS + QUANT.
LI ← Nº DOC, QUANT(AS)
AG ← AE + QUANT.
LI ← Nº DOC, QUANT(AE)
IMPRIME LI
INICIALIZA LI
LER ARQUIVO MOVIMENTO
ZC ← ZC + AE
ZC ← ZC - AS
LI ← AS, AE e ZC
IMPRIME LI
INICIALIZA LI
LER ARQUIVO CADASTRO
FIM DO PROGRAMA

5.1.3 - Considerações

De uma maneira geral pode-se dizer que NASSI-SHNEIDERMAN CHARTS consiste de uma forma gráfica alternativa para a representação de fluxos de processamento que, por não possuir notação para comandos de desvios, obriga a elaboração de programas estruturados. Por outro lado, permite a construção de diagramas mais facilmente adaptáveis às linguagens de alto-nível disponíveis; um exemplo disto pode ser analisado no livro de LUCENA e STAA {19}, onde é apresentada uma simbologia voltada para a linguagem COBOL. Além disso, em função do número de processos possíveis de serem desenhados numa única página ser relativamente pequeno (no máximo uns 25) e da inexistência de conectores de páginas, leva o programador a modular seu programa em seções significativas. Outrossim, promove a programação por refinamentos sucessivos, o que a torna mais adequada para o ser humano, ao contrário dos fluxogramas cuja simbologia é voltada para a representação de instruções de máquinas e não de abstrações.

As considerações específicas para esta técnica quanto aos ítens discutidos no capítulo 2, seriam:

a) Registro de Soluções e Comunicação

Vale ressaltar que esta técnica apenas favorece o desenvolvimento de programas por refinamentos sucessivos, não apresentando como propriedade uma orientação para a execução deste processo. Tal fato tanto prejudica o processo de análise do problema, como o acompanhamento e controle de execução dos trabalhos, devido não serem definidos resultados intermediários que caracterizam, efetivamente, um passo dado.

Quanto a comunicação, pode-se dizer que NASSI-SCHNEIDERMAN CHARTS torna mais evidente a solução encontrada para o problema.

b) Produtividade

Sendo a técnica em questão constituída por uma notação que restringe a lógica do programa final - quando bem aplicada sempre levará a programas estruturados - então se tem que estes deverão conter menos erros e serão depurados mais rapidamente e mantidos mais facilmente.

A visualização de estruturas semelhantes não é tão imediata quanto aquela proporcionada por algumas metodologias que veremos a seguir, porém já é bem melhor que no caso das técnicas apresentadas anteriormente. Exemplos de estruturas padronizadas com a utilização desta metodologia podem ser apreciados nas referências {19,27}.

c) Qualidade

Pelas propriedades já expostas, tem-se que a qualidade dos programas serão afetadas positivamente com a aplicação de NASSI-SCHNEIDERMAN CHARTS. Suas características permitem afirmar que, com sua aplicação, ter-se-á projetos de programas mais claros e estruturas finais mais compreensíveis e legíveis. Além disto, tem-se que esta técnica favorece a validação do programa.

Entretanto, vale observar que ela somente induz a uma madularização dos programas, sem estabelecer critérios para adoção deste conceito, tais como em Composite Design. Desta forma, os módulos definidos não necessariamente serão os mais adequados.

d) Praticabilidade

É uma linguagem gráfica de fácil aprendizagem e uso. Não exige gabaritos especiais, sendo que a utilização de folhas padronizadas - por ex. folhas quadriculadas, com os lados dos quadrados medindo 1 cm - pode auxiliar no desenho dos diagramas.

No tocante a este ponto um incoveniente que deve ser apontado é a dificuldade para se padronizar o estilo dos diagramas que, em função da inexistência de gabaritos, é totalmente dependente da pessoa que o está elaborando.

e) Generalidade e Sistemas de Suporte

Esta técnica é genérica podendo ser aplicada, indistintamente, em qualquer classe de problema.

Não existem sistemas automatizados que suportam sua aplicação.

5.2 - MÉTODO DE JACKSON

5.2.1 - Apresentação

Um ponto comum das técnicas discutidas até o momento é que nenhuma delas, efetivamente, apresenta uma proposta para o projeto do programa. As Tabelas de Decisão tratam do registro sistemático das condições e ações envolvidas no problema, e os Fluxogramas e os Nassi-Schneiderman Charts promovem a documentação do fluxo de processamento a ser seguido.

Uma conseqüência disto, principalmente no caso de Fluxogramas e Nassi-Schneiderman Charts, é que os programadores passam a ficar mais preocupados com o fluxo de processamento do que com a estrutura do problema em si.

Baseado neste argumento e na observação de que a estrutura dos dados condiciona a estrutura do programa, M. A. Jackson {15} propôs uma metodologia de desenvolvimento de programas cujo princípio central é: a estrutura do programa deve ser deduzida da estrutura de seus dados.

Na figura 5.8 temos esquematizado o processo de desenvolvimento implícito nesta metodologia e que pode ser traduzido numa sequência de passos que iremos definindo no desenrolar do exemplo.

NOTAÇÃO

Uma notação única a ser utilizada tanto na representação da estrutura dos dados como na estrutura do programa que os manipula constitui a peça principal desta metodologia, pois fundamenta a possibilidade de construção de estruturas de programas a partir da estrutura dos dados.

A notação proposta por Jackson também segue os preceitos da Programação Estruturada e consiste na representação gráfica para:

- Componente elementar - o qual não pode mais ser dividido em outras partes;
- Seqüência - o qual tem duas ou mais partes, sendo que cada uma delas ocorre uma vez, e em ordem;
- Seleção - o qual tem duas ou mais partes, sendo que apenas uma ocorre uma vez;
- Repetição - o qual tem uma parte que ocorre zero ou mais vezes.

Seqüência, Seleção e Repetição formam o grupo denominado "tipos compostos" e suas partes podem ser elementares ou compostas, isto é, elas podem ser quaisquer um dos tipos definidos, sem restrições.

Segue-se a apresentação de cada um desses tipos, a fim de torná-los mais claros.

a) Componente elementar

O componente elementar corresponde às operações elementares na linguagem de programação utilizada na codificação dos programas.

Sua representação gráfica é um quadrado ou retângulo e seriam exemplos de componentes elementares, tomando-se por base a linguagem PL/1, os comandos DCL A DEC FIXED e A=A+B. (fig. 5.3).

NOTAÇÃO	CORRESPONDENCIA	
	DADOS	COMANDOS
	DCL A DEC FIXED;	A= A + B;

Figura 5.3. - Notação e Exemplos de Componente Elementar

b) Sequência

Uma sequência tem duas ou mais partes, todas elas devendo ocorrer em uma determinada ordem, quando da ocorrência da componente.

Exemplos de sequência seriam a declaração de um registro e uma série de comandos em um programa. (Fig. 5.4.).

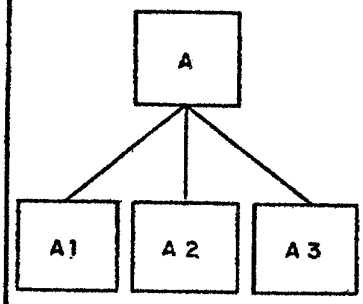
NOTAÇÃO	CORRESPONDENCIAS	
	DADOS	COMANDOS
	DCL 01 A; 02 A1 FIXED (2), 02 A2 FIXED (5), 02 A3 CHAR (5);	A: BEGIN; A1 = A1 ** 2; A2 = A2 * A1; CALL A3 (A2,A1); A: END;

Figura 5.4. - Representação e Exemplos de Sequência

c) Seleção

Uma seleção tem duas ou mais partes, das quais uma, e somente uma, ocorre em cada ocorrência do componente de seleção.

Como exemplos de seleção tem-se declaração de registros com redefinições e comandos do tipo IF-THEN-ELSE ou DO CASE (Fig. 5.5.).

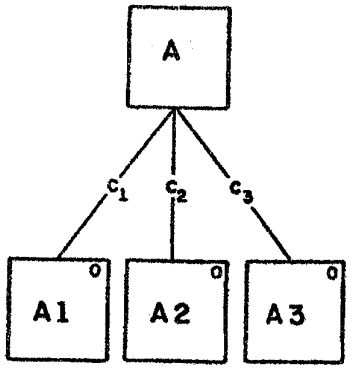
NOTAÇÃO	CORRESPONDÊNCIAS	
	DADOS	COMANDOS
	DCL 01 A, 02 A1....., 02 A2 REDEFINE A1, 02 A3 REDEFINE A1;	A: DO CASE, CASE (C1), CALL A1, CASE (C2), CALL A2, OTHERWISE CALL A3, A: ESAC;

Figura 5.5 - Notação e Exemplos de Seleção

d) Repetição

Uma repetição tem somente uma parte, que deverá ocorrer zero ou mais vezes para cada ocorrência do componente em si.

São exemplos deste tipo composto um arquivo de dados (contendo 0 ou N registros), e comandos do tipo DO WHILE (fig. 5.6).

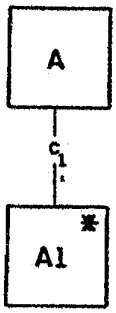
NOTAÇÃO	CORRESPONDÊNCIAS	
	DADOS	COMANDOS
	ARQUIVO SEQUÊNCIAL	A: DO WHILE (C1); CALL A1, A: END;

Figura 5.6 - Notação e Exemplos de Repetição

PSEUDO-CÓDIGO

Além da notação gráfica apresentada, Jackson propõe a utilização de uma pseudo-linguagem para a descrição da estrutura do programa.

Esta pseudo-linguagem permite uma descrição intermediária do programa, antes de sua codificação final, e sua caracterização encontra-se na figura 5.7.

SEQUENCIA	SELEÇÃO	REPETIÇÃO
A: SEQ A1:=A1 * * 2 A2:=A2 * A1 DO A3 A: END	A: SELECT (C1) DO A1 OR(C2) DO A2 OR (C3) DO A3 A: END	A: ITER WHILE (C1) DO A1 A: END

Figura 5.7 - Caracterização do Pseudo-Código

ESQUEMA DE DESENVOLVIMENTO DE PROGRAMAS PELO MÉTODO DE JACKSON

Na figura 5.8 apresentamos o esquema geral de desenvolvimento de programas proposto por Jackson.

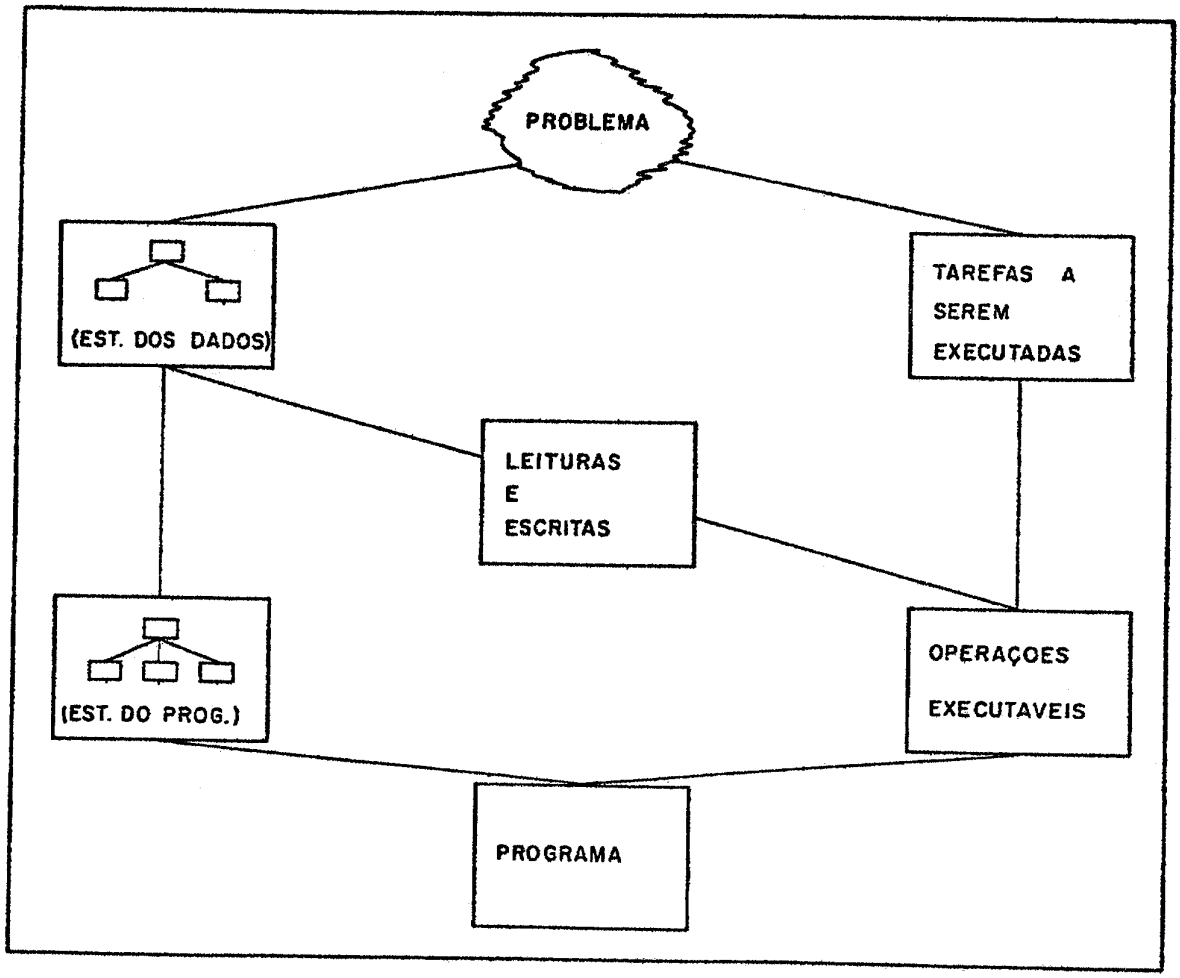


Figura 5.8 - Esquema de Desenvolvimento de Programas proposto por Jackson (adaptado de [15])

Conforme já foi mencionado, o esquema de desenvolvimento de programas apresentados pode ser traduzido numa série de passos, a serem executados "cronologicamente", que serão evidenciados durante o desenvolvimento do exemplo de aplicação, nosso próximo item.

Antes porém de examinarmos a solução de um problrma através do método de Jackson, gostaríamos de salientar que o mesmo não se resume ao que está mencionado neste trabalho, sendo a proposta deste autor bastante mais ampla.

O método básico aqui discutido abrange uma parte significativa dos problemas normalmente encontrados no dia a dia de um programador. Contudo, o seu uso não pode ser generalizado, existindo classes de problemas onde são necessárias outras considerações.

Estas outras classes de problemas, caracterizadas pela existência de conflitos entre a estrutura dos dados de entrada e dos de saída ("structure clashes") - ocorrência de um não "casamento" perfeito entre as estruturas dos dados do programa, o que impede a dedução direta de sua estrutura - merecem procedimentos próprios, que também são apresentados por Jackson.

Por outro lado, na referência já apontada, também são analisadas as implicações de arquivos contendo dados errados e inválidos, e a otimização de programas.

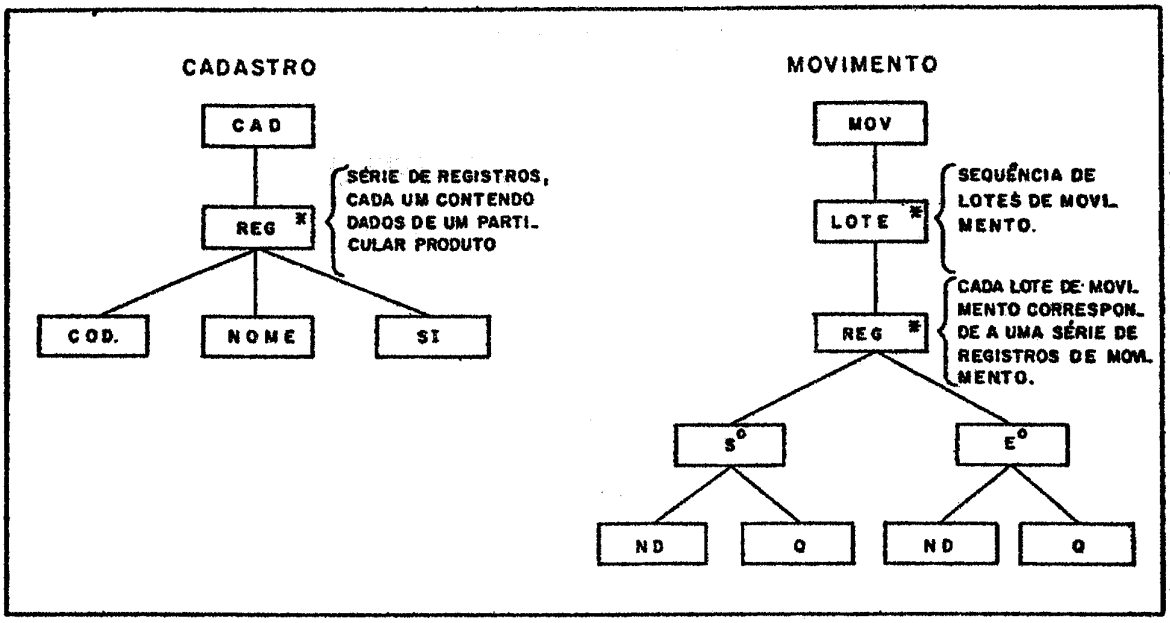
5.2.2 - Exemplo

A seguir desenvolvemos o exemplo definido em 3.1.3., adotando o esquema proposto por Jackson.

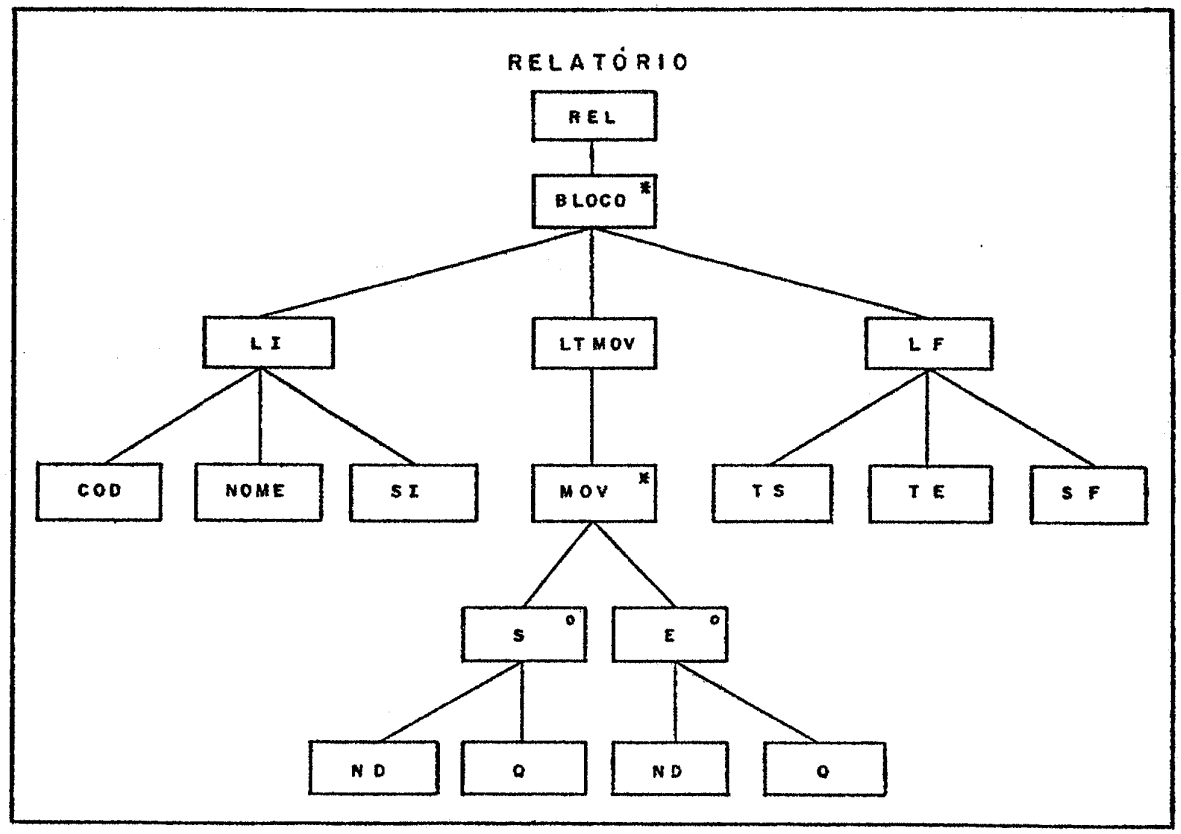
1º passo: Identificar e representar a estrutura dos dados de entrada e saída, adotando a notação definida.

Os dados de entrada e saída no problema em questão são: (1) Arquivo de Cadastro, (2) Arquivo de Movimento e (3) Relatório de Movimentação, cujas representações são as que seguem:

ESTRUTURA DOS DADOS DE ENTRADA



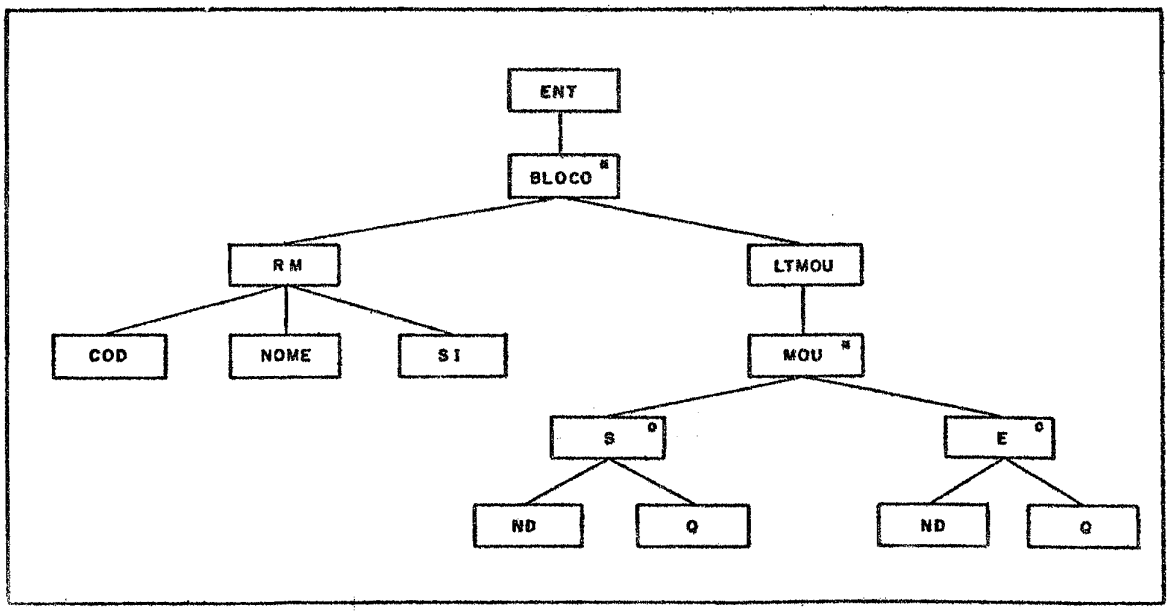
ESTRUTURA DOS DADOS DE SAÍDA



Da análise das estruturas dos dados depreende-se que o arquivo de saída do programa corresponde a uma intercalação dos dois arquivos de entrada, pois cada um dos blocos (Nível 2) é formado a partir de um registro obtido do arquivo cadastro (LI) seguido, quando ocorrerem, dos respectivos dados encontrados no arquivo movimento (MOV). O terceiro componente do bloco (LF) é obtido a partir do processamento dos dados de cadastro e de movimento não constituindo um problema de correspondência de estruturas.

Para situações desta natureza Jackson propõe a definição de um "arquivo lógico de entrada" - para o exemplo equivalente ao arquivo resultante da intercalação dos dois arquivos de entrada, onde ter-se-ia a seguinte estrutura: Uma série de blocos de dados por produto, cada bloco possuindo um registro mestre (dados do produto e saldo inicial) seguido ou não de lotes de movimento. A representação da estrutura deste arquivo lógico é apresentada a seguir.

ESTRUTURA LÓGICA DOS DADOS DE ENTRADA



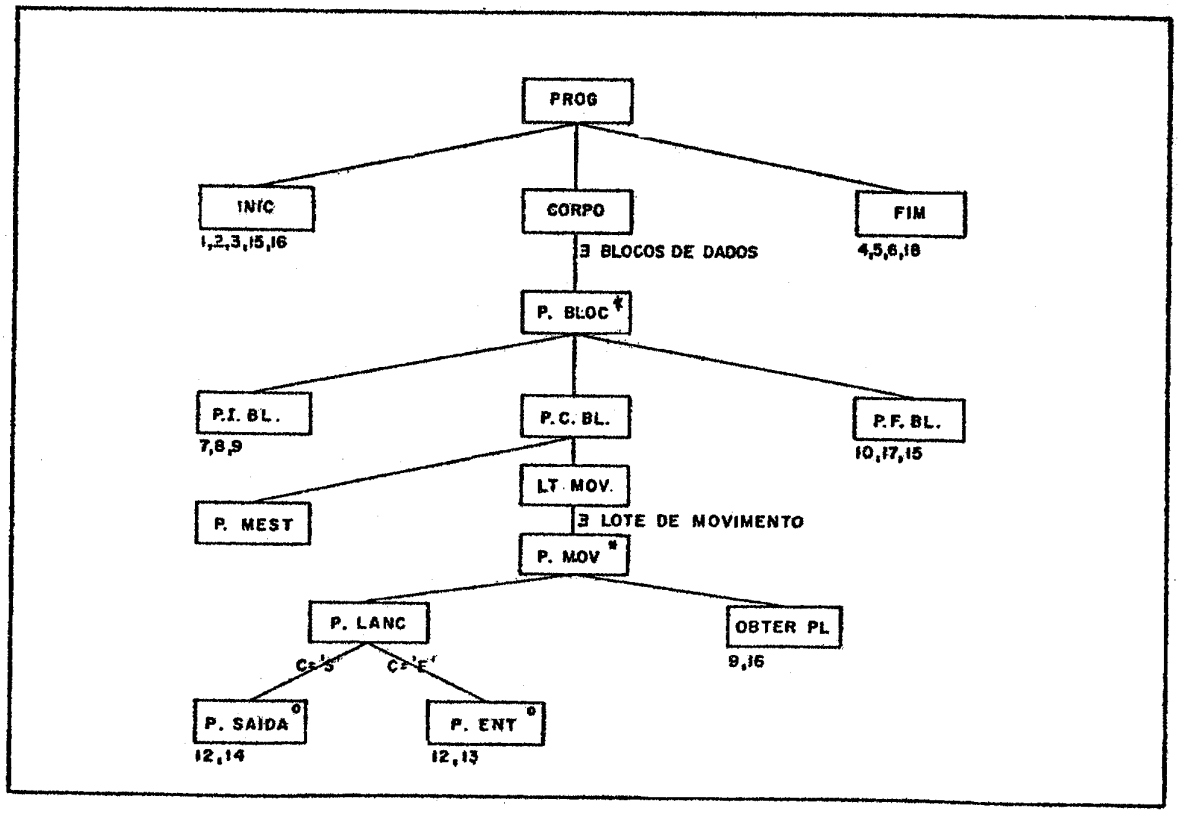
2º passo: Projetar a estrutura do programa a partir das estruturas de dados de entrada e saída.

Com a criação do arquivo lógico obtem-se um "casamento" perfeito entre as estruturas dos dados, o que permite a dedução direta da estrutura do programa a qual é apresentada a seguir.

Ressaltamos que na estrutura do programa são adicionados certos componentes intrínsicos ao processamento, tais como: leitura, chaves, ... bem como previstos os componentes de abertura e fechamento de arquivos, e também os dedicados a inicialização de variáveis.

Por outro lado, na estrutura do programa são eliminados os componentes terminais da estrutura dos dados - componentes elementares - que não condicionam a organização do programa. Estes componentes só voltam a ser considerados quando da elaboração da lista de operações executáveis.

ESTRUTURA DO PROGRAMA



3º passo: Identificação das operações executáveis.

Neste passo são listadas todas as operações a serem executadas no programa - operações primárias e secundárias - tais como: leituras, inicializações, abrir e fechar arquivos acumulações, impressões, etc...

```
 1 - Abrir arquivo de Cadastro
 2 - Abrir arquivo de Movimento
 3 - Abrir arquivo de Relatório
 4 - Fechar arquivo de Cadastro
 5 - Fechar arquivo de Movimento
 6 - Fechar arquivo de Relatório
 7 - Inicializar ZC (ZC ← SALDO INICIAL)
 8 - Zerar AE, AS
 9 - Inicializar LI
10 - Calcular Saldo Final (ZC=ZC+AE-AS)
11 - Imprimir Cod, Nome e Saldo Inicial do Produto.
12 - Imprimir Movimento
13 - Acumular Entradas
14 - Acumular Saídas
15 - Ler arquivo Cadastro
16 - Ler arquivo de Movimento
17 - Imprimir Total de Entradas e Saídas e Saldo Final
18 - Terminar o Programa
```

4º passo: Alocar as operações executáveis na estrutura do programa.

Esta alocação pode ser observada na estrutura definida no passo 2 e corresponde aos números colocados abaixo dos componentes da estrutura.

5º passo: Elaboração do Programa em Pseudo-Código.

Observada a estrutura do programa e as operações alocadas nos seus diversos componentes terminais, elaborar o pseudo-código do programa conforme notação estabelecida.

```
PROG: Seq
      INÍCIO: Seq
              Abrir Cadastro, Movimento e Relatório
              Ler Cadastro
              Ler Movimento
      INÍCIO: End
       CORPO: Iter While (   EOF Cadastro)
              PIB: Seq
                   ZC ← Saldo Inicial
                   AE, AS ← 0
                   LI ← ∅
              PIB: End
              PCB: Seq
                   PMT: Seq
                        LI ← Cod, Nome, Saldo do Produto
                        Imprima LI
                        LI ← ∅
                   PMT: End
                   PMV: Iter While (   EOF mov. AND. C=M)
                        PL:  Select  (CM='S')
                             LI ← nº doc, quanto.
                             Imprima LI
                             AS ← AS + QS
                             Or   (CM='E')
                             LI ← nº doc, quant.
                             Imprima LI
                             AE ← AE + QE
                        PL:  End
                        OL:  Seq
                             LI ← ∅
                             Ler movimento
                        OL:  End
                   PMV: End
                   PFB: Seq
                        ZC ← ZC + AS - AE
                        LI ← AS, AE, ZC
                        Imprima LI
                        LI ← ∅
                        Ler Cadastro
                   PFB: End
              PCB: End
       CORPO: End
         FIM: Seq
              Fechar Cadastro, Movimento e Relatório
              Terminar o Programa
         FIM: End
PROG: End
```

6º passo: Refinamento do pseudo-código e codificação do programa.

Este passo não será por nós desenvolvido já que é condicionado pela linguagem de programação a ser utilizada.

5.2.3 - Considerações

Sem dúvida, a introdução do conceito que os programas devem ser projetados a partir dos dados por ele manipulados torna o MÉTODO DE JACKSON um marco importante dentro do processo de evolução deste tipo de instrumento de trabalho. Até então suas congêneres, tratavam apenas de dar condições para documentar e organizar o fluxo de processamento dos programas, sem no entanto estabelecerem um ponto de partida para a solução do problema.

Deve-se observar que a metodologia apresenta uma notação que naturalmente leva ao desenvolvimento de programas através de refinamentos sucessivos, o que fica refletido na estrutura hierárquica que se obtém para o programa e onde cada nível caracteriza um detalhamento a mais da sua estrutura. Por outro lado, não possuindo representação para desvios arbitrários, sua notação leva a fluxos de processamentos estruturados.

Outro fato a ser ressaltado é que esta metodologia estabelece uma série de passos claros, com resultados bem definidos, o que favorece o planejamento, o acompanhamento e controle da atividade de programação.

A seguir são apresentadas as considerações específicas para a metodologia com relação aos pontos definidos no Capítulo 2.

a) Registro de Soluções e Comunicação

O Método de Jackson é em si um processo para análise do problema de processamento de dados, contendo uma orientação no modo de proceder do programador e definindo uma forma de documentar as soluções intermediárias alcançadas.

Ainda que esta não tenha sido uma preocupação inicial de seu formulador, a metodologia facilita a definição de pontos para o acompanhamento e controle dos trabalhos que, a nosso ver,

poderiam ser: (1) Ao final da execução do segundo passo, - estrutura do programa definido e (2) Ao final do quarto passo - primeira versão do pseudo-código.

A comunicação das soluções é garantida pela notação gráfica e a linguagem em pseudo-código proposta. Contudo vale ressaltar que estas notações não permitem uma comunicação tão ampla quanto a Tabela de Decisão, perdendo parte do seu poder de comunicação junto a pessoas não envolvidas diretamente com o processamento de dados.

b) Produtividade

A produtividade geral dos programadores deve aumentar sensivelmente com a utilização desta metodologia pois favorece a diminuição de erros de programa e facilita sua depuração e manutenção.

Além disso, as estruturas hierárquicas decorrentes de sua aplicação tornam mais evidentes as semelhanças entre as diferentes aplicações, favorecendo desta forma o desenvolvimento de estruturas padrões e agilizando o processo de aprendizagem da equipe de programação.

c) Qualidade

Pelas suas propriedades o Método de Jackson torna o projeto do programa mais claro além de aumentar a compreensibilidade e legibilidade do programa final. Adicionalmente, o programa obtido através de uma aplicação possui a própria estrutura do problema que está sendo modelado e que o torna mais fácil de ser mantido. Por isso tudo, esta metodologia atua de forma positiva quanto ao aspecto qualidade do programa.

d) Praticabilidade

Embora aparentemente simples, esta técnica apresenta algumas dificuldades quanto a sua aprendizagem e uso.

A aprendizagem fica prejudicada pelo fato de que muitos dos conceitos que vão sendo utilizados nas diversas fases do processo de desenvolvimento se encontrarem ainda na forma empírica, sem o estabelecimento de regras que orientem a sua aplicação. Um exemplo disto está na criação dos "arquivos lógicos" de entrada cuja conclusão depende de um "estalo" do programador.

Também não existem regras bem definidas que orientem a passagem das estruturas de dados para a estrutura do programa, o que torna o processo um tanto intuitivo.

A diagramação das estruturas não é padronizada, podendo ser facilitada com o uso de folhas quadriculadas - por ex: com quadrados de 1,5 cm de lado.

e) Generalidade

Este método é voltado para aplicações ditas comerciais, onde a estrutura dos dados é significativa face os processamentos que serão realizados pelo programa. A sua aplicação perde o sentido quando em processamentos científicos, sendo que neste caso poder-se-ia utilizar apenas a notação proposta por Jackson, o que já favorece o desenvolvimento do programa.

f) Sistemas de Suporte

Não existem sistemas automatizados que suportem o uso desta metodologia.

5.3 - LÓGICA DE CONSTRUÇÃO DE PROGRAMAS - (LCP)

5.3.1 - Apresentação

LCP é uma metodologia concebida e aperfeiçoada por Jean Dominique Warnier {29} que, de maneira análoga a de Jackson, utiliza-se da estrutura dos dados para determinar a estrutura do programa.

Ainda que partindo dos mesmos princípios, isto é, a es

trutura dos dados reflete a estrutura do programa, LCP apresenta algumas diferenças ao método que acabamos de analisar. A primeira delas é a de que a estrutura do programa é deduzida apenas considerando-se a estrutura dos dados de entrada, sendo os dados de saída utilizados para fins de controle do programa obtido. Tal procedimento é facilmente compreendido ao se levar em conta a maneira como Warnier vê a organização dos dados de entrada. Segundo Warnier, "a descrição dos dados de entrada se estabelece em função dos tratamentos a efetuar, em vista de obter os resultados demandados" {28}. Desta forma, os dados de saída e os de entrada possuem estruturas semelhantes.

Consequentemente, LCP não contempla os problemas de conflitos de estrutura, analisados no método de Jackson, podendo-se, neste sentido, dizer que é uma metodologia menos abrangente. Vale ressaltar que não é de todo impossível a aplicação desta metodologia nestes casos, apenas ela não apresenta um procedimento sistemático para a abordagem do problema. Uma análise da aplicação da LCP em situações desta natureza pode ser apreciada no artigo de Leon Levy {17}.

Em contrapartida, LCP é um método bem mais sistematizado que o de Jackson. Enquanto este último propõe um esquema de projeto de programa cujos passos são bastante dependentes da intuição do programador, LCP, que tem seus fundamentos na teoria dos conjuntos {29,30}, define uma série de estruturas padrões e estabelece uma série de leis e regras que orientam, e condicionam, o projeto do programa.

O esquema geral de desenvolvimento de programas implícito em LCP está sintetizado na figura 5.14 e também pode ser traduzido numa série de passos conforme pode ser apreciado ao longo do exemplo de sua aplicação.

NOTAÇÃO

De maneira análoga ao método de Jackson, LCP apresenta uma notação única para a descrição da estrutura dos dados e do programa, o que favorece a dedução da estrutura do programa a partir dos dados por ele manipulados.

Ainda que possam ser identificadas as estruturas básicas definidas por Bohm e Jacopine, a notação de LCP não se referencia diretamente àquelas estruturas mas sim, constitui uma representação para classes de estruturas que podem ser observadas em problemas de processamento de dados. Pelas suas características, cada classe de estrutura corresponde a uma atitude de desenvolvimento específica que é viabilizada por um conjunto de definições e regras a serem observadas durante o projeto de programa.

Segundo Warnier, a estrutura de um problema de processamento de dados, que é representada pela estrutura dos dados de saída requeridos, pode ser enquadrada numa das seguintes principais classes:

- Estrutura Repetitiva,
- Estrutura Alternativa e
- Estrutura Complexa,

cujas definições são as que seguem:

Salientamos que todas as definições, regras e leis, relativas a LCP, apresentadas a partir deste ponto, foram retiradas de [28] e, assim sendo, fica convencionado que os trechos entre aspas encontrados ao longo do texto tem sua origem naquela referência.

a) Estrutura Repetitiva

"Um conjunto de dados de estrutura repetitiva é um conjunto dentro do qual se encontra vários subconjuntos de da

dos'de uma mesma natureza".

Como exemplo, suponhamos a emissão de um relatório contendo dados de diversas agências, cada agência compreendendo a vários clientes e estes, por sua vez, compreendendo a vários movimentos. Os conjuntos Agência, Cliente e Movimento correspondem a estruturas repetitivas já que cada um deles é formado de subconjunto de dados de uma mesma natureza.

A notação LCP para esta estrutura é:

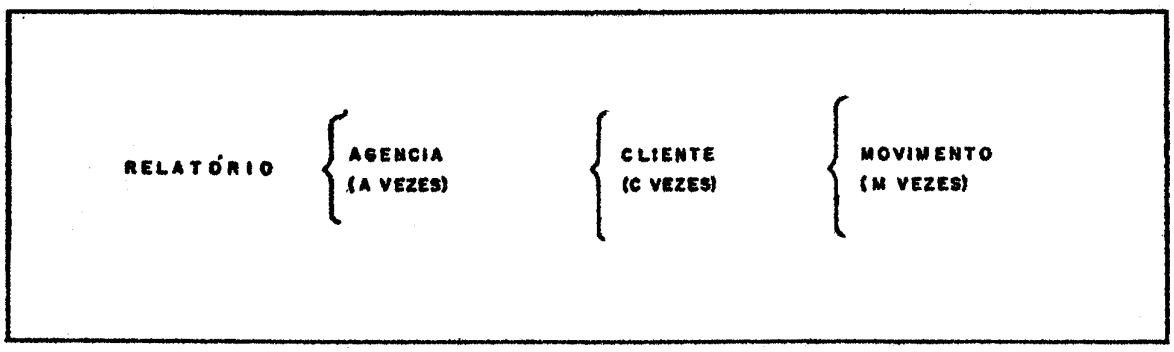


Figura 5.9 - Exemplo de Estrutura Repetitiva

b) Estrutura Alternativa

"Um conjunto de dados de estrutura alternativa é um conjunto dentro do qual se encontra um ou vários subconjuntos cuja presença é aleatória. Se ocorre vários, a presença de um exclui a presença do outro".

Como exemplo, suponhamos que os movimentos definidos anteriormente correspondessem a débitos e crêtidos. Neste caso, no nível 4 do diagrama lógico teríamos caracterizada uma estrutura alternativa, cuja notação é:

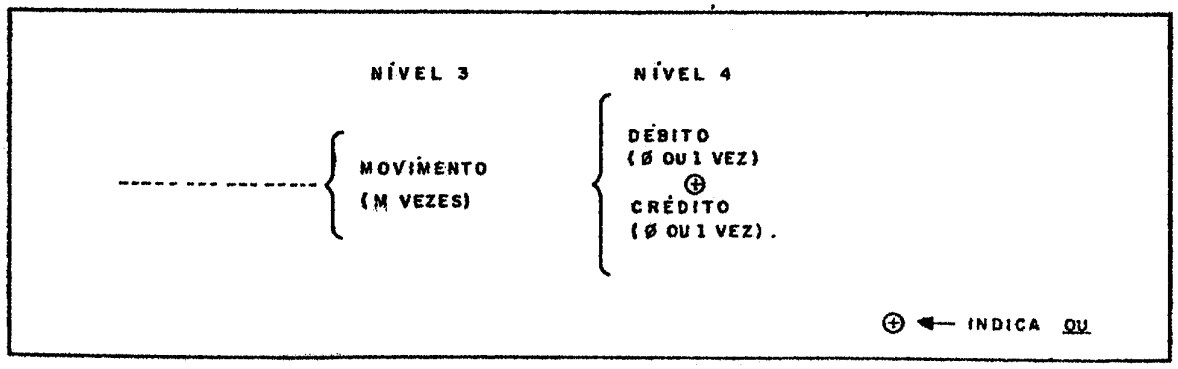


Figura 5.10 - Exemplo de Estrutura Alternativa

c) Estruturas Complexas

"Um conjunto de dados de estrutura complexa é um conjunto dentro do qual encontramos, ao primeiro nível de subdivisão, várias estruturas elementares alternativas ou repetitivas".

Em função do tipo da estrutura elementar encontrada ao primeiro nível de subdivisão do conjunto colocado como referencial, as estruturas complexas são classificadas em:

. Estrutura Complexa Repetitiva

. Estrutura Complexa Alternativa

. Estrutura Complexa Mista

c.1) Estrutura Complexa Repetitiva

"Um conjunto de dados de estrutura complexa repetitiva é um conjunto dentro do qual encontramos, ao primeiro nível de subdivisão, várias estruturas elementares repetitivas".

Como exemplo, suponhamos um relatório por cliente onde esteja especificado os créditos realizados (nº do documento e montante) com uma totalização, e os débitos ocorridos (nº do documento e montante), também com uma totalização. Neste caso, logo no primeiro nível de subdivisão (cliente) tem-se duas estruturas elementares repetitivas. Na notação LCP teríamos:

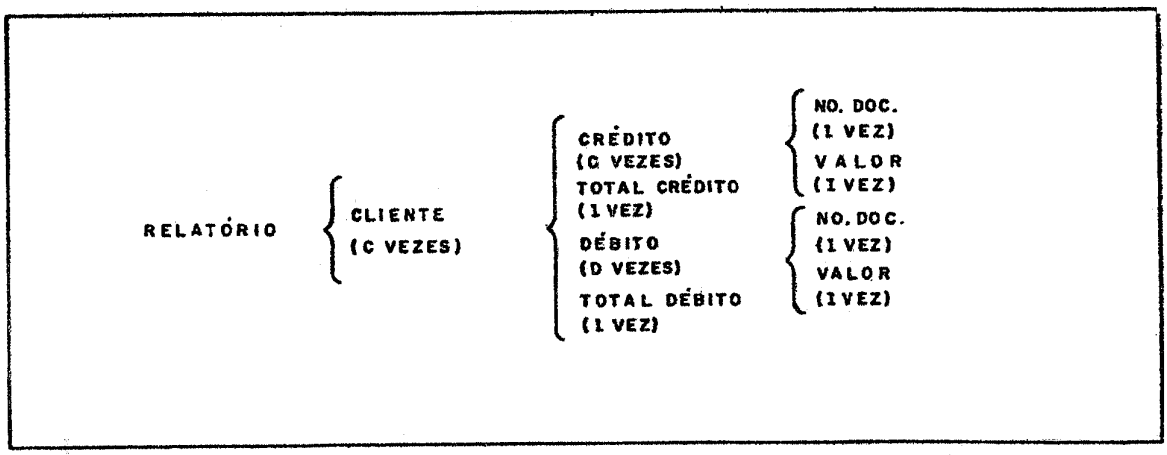


Figura 5.11 - Exemplo de Estrutura Complexa Repetitiva

c.2) Estrutura Complexa Alternativa

"Um conjunto de estrutura complexa alternativa é um conjunto dentro do qual encontramos vários subconjuntos anotados (Ø ou 1 vez) e não exclusivos".

Por exemplo, a obtenção de um arquivo de saldos de clientes a partir de dois outros arquivos, um contendo os saldos atuais e o outro com os movimentos realizados. A estrutura lógica destes dados de entrada é um estrutura complexa alternativa, cuja representação é:

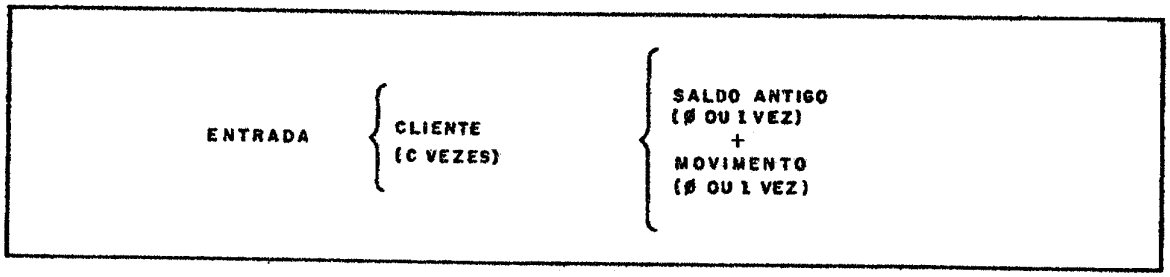


Figura 5.12 - Exemplo de Estrutura Complexa Alternativa

Salientamos o uso do símbolo(+) para indicar que os subconjuntos são não exclusivos, em lugar de (⊕) que indica que são mutuamente exclusivos.

c.3) Estrutura Complexa Mista

"Um conjunto de dados de estrutura complexa mista é um conjunto dentro do qual encontramos várias estruturas elementares, umas sendo repetitivas e outras alternativas, ao primeiro nível do conjunto colocado como referencial".

Como exemplo, imaginemos a emissão de um relatório estatístico periódico dos montantes das futuras relativas a clientes de uma empresa. Após a descrição dos montantes por fatura deverá ocorrer uma linha dando o total das vendas ao cliente e o total dos descontos recebidos. O desconto é função do cliente e corresponde a uma taxa fixa. A notação em LCP é a que

segue.

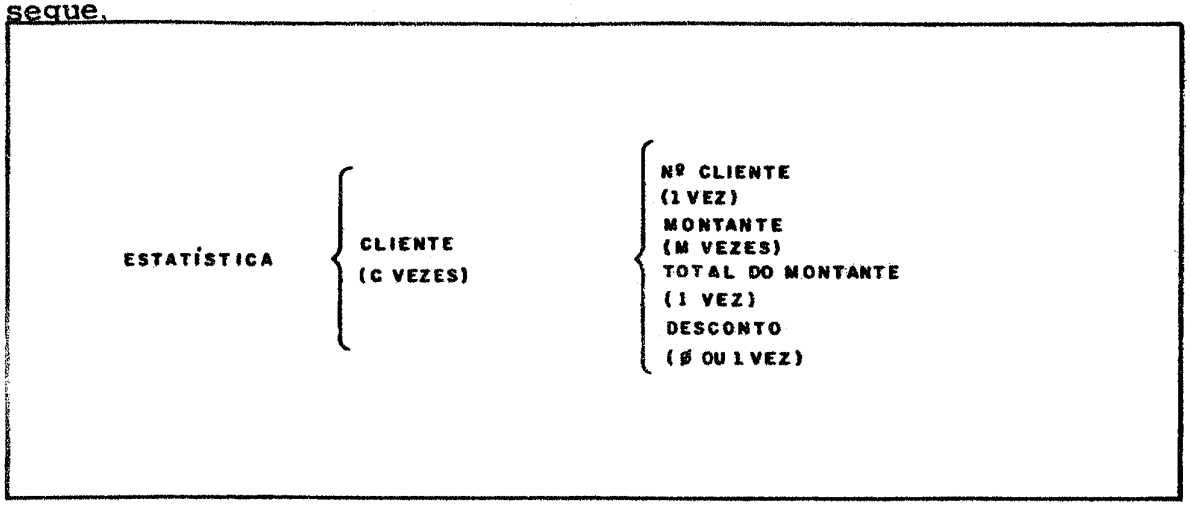


Figura 5.13 - Exemplo de Estrutura Mista

ESQUEMA DE DESENVOLVIMENTO DE PROGRAMAS EM LCP

Na figura 5.14 apresentamos o esquema geral para desenvolvimento de programas proposto em LCP e que pode ser traduzido nos passos definidos no desenvolvimento do exemplo.

Vale ressaltar que LCP não constitui apenas o que aqui estamos mencionando, pois, sendo nosso objetivo apenas caracterizar a metodologia, deixamos de lado muitas de suas particularidades e mesmo, do seu potencial.

Para não sermos totalmente omissos queremos registrar que este método apresenta procedimentos para a otimização de programas que adquirem (no diagrama lógico) à forma de árvores. Também analisa os problemas que envolvem vários arquivos físicos na entrada e/ou saída e a divisão dos programas em diversas fases de processamento, sendo que este último caso se refere a problemas cujo processamento envolvido não são identificados totalmente a partir dos dados de entrada, o que torna necessário a criação de critérios de identificação ao longo do processamento.

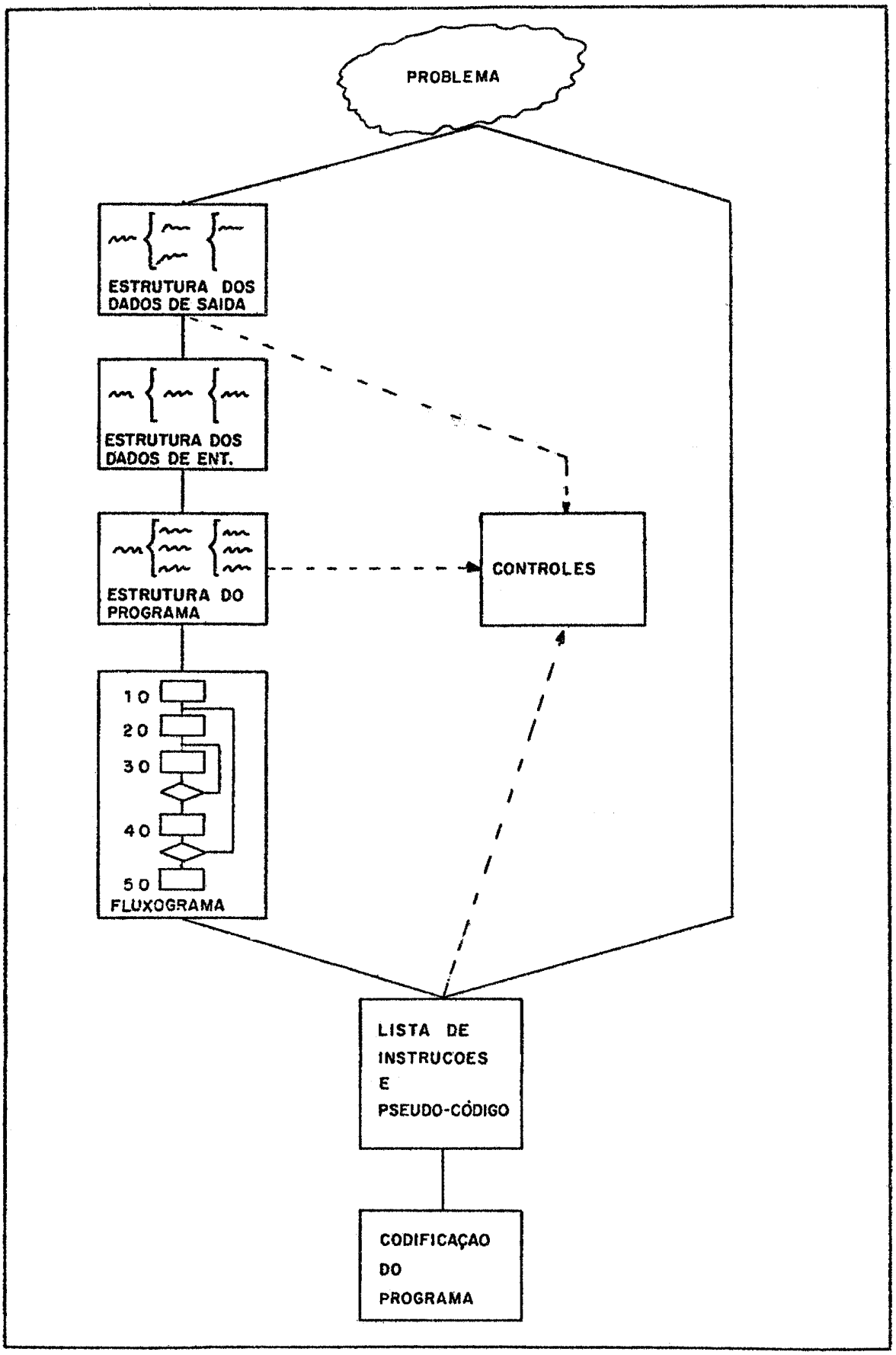


Fig. 5.14 - Esquema de Desenvolv. de Programas proposto em LCP

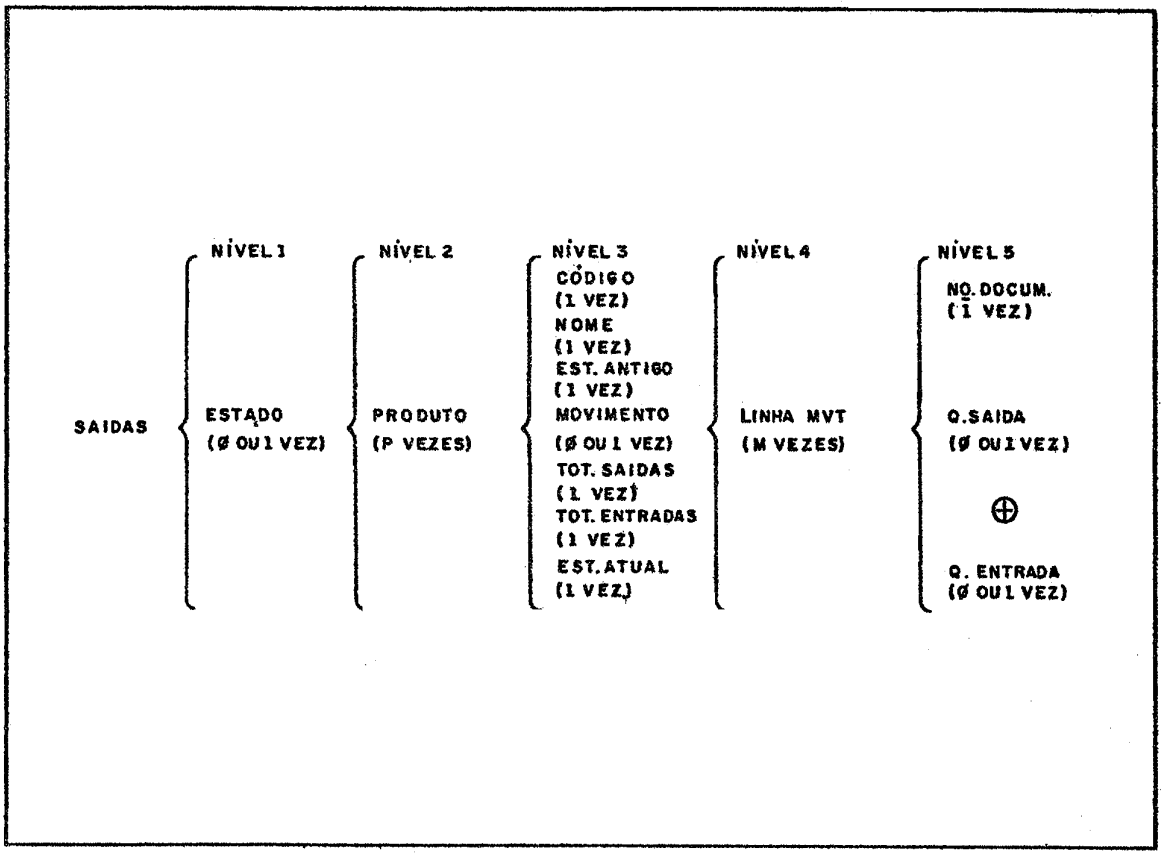
5.3.2 - Exemplo

Para exemplificar a aplicação de LCP, iremos desenvolver o problema definido no início deste trabalho (item 3.1.2).

Antes porém de iniciarmos esta tarefa, gostaríamos de salientar que os trechos entre aspas correspondem a traduções da referência [28]. Também que, de modo a nos ater ao escopo do trabalho, muitas das regras encontradas na referência assinalada serão por nós omitidas.

1º passo: Descrição da Estrutura dos Dados de Saída utilizando a notação LCP - elaboração do Diagrama Lógico de Saída.

A saída do programa consiste de um relatório cuja estrutura é a que segue:



Salientamos a inclusão de um nível adicional representando a existência ou não de saídas no relatório. O conjunto "Estado" é assinalado (Ø ou 1 vez) em função da possibilidade do arquivo de dados de entrada estar vazio ou conter algum dado.

O mesmo ocorre no NÍVEL 3 com o grupo MOVIMENTO que está assinalado (Ø ou 1 vez) devido a possibilidade de não ocorrer qualquer movimento para o produto.

2º passo: Descrição da Estrutura dos Dados de Entrada - elaboração do Diagrama Lógico dos Dados de Entrada, considerando-se a estrutura definida no passo anterior.

Da descrição dos dados de saída, concluimos que o arquivo de entrada deverá ser sequencial e conter os seguintes registros ordenados por código de produto.

- Registro Mestre

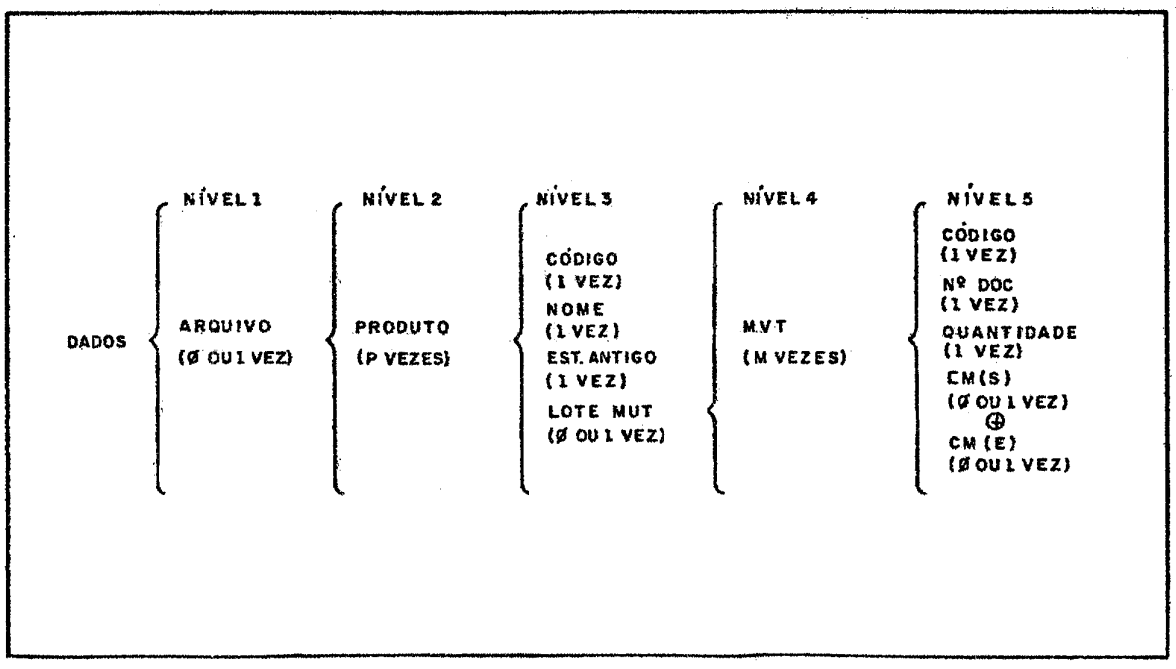
  Código Nome do Produto Estoque Antigo

- Registros de Movimento (Conjunto que poderá ser vazio)

  Código nº Documento Quantidade CM

  onde CM S - Saída
           E - Entrada

Tal arquivo tem o diagrama lógico apresentado a seguir. Observamos que esta definição implica em se executar um "merge" com os dois arquivos de entrada estabelecidos no enunciado.



3º passo: Derivação da Estrutura do Programa - elaboração do Diagrama Lógico do Programa, a partir da estrutura dos dados de entrada, considerando-se as regras de derivação propostas em LCP.

As estruturas existentes nos dados de entrada são do tipo:

NÍVEIS 1, 3, 5: Estruturas Alternativas

NÍVEIS 2 e 4: Estruturas Repetitivas

Regras de derivação propostas em LCP:

"Estrutura repetitiva de dados, então estrutura repetitiva de programa".

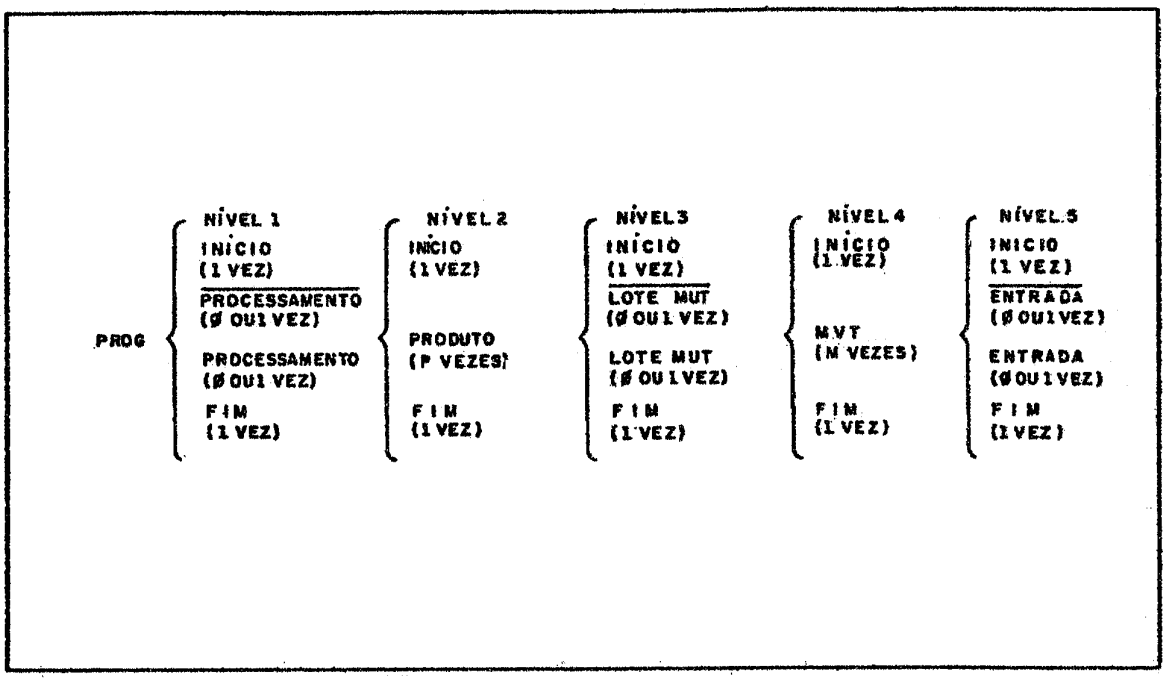
"Um subconjunto de programa de estrutura repetitiva compreende sempre um subconjunto repetitivo precedido de um início a executar 1 vez e seguido de um fim a executar 1 vez dentro do conjunto".

"Um subconjunto de programa de estrutura alternativa compreende sempre dois ou mais subconjuntos executados (Ø ou 1 vez) à exclusão um dos outros. Estes subconjuntos são precedidos obrigatoriamente de um subconjunto inicial a executar 1 vez dentro do conjunto e seguidos de um subconjunto final a executar uma vez no conjunto".

Tal como no método de Jackson, aqui também pode ocorrer a eliminação de algumas informações contidas no diagrama lógico dos dados de entrada. A justificativa disto é a mesma, isto é, são informações que não condicionam a estrutura do programa. Tais informações voltam a ser consideradas apenas quando do estabelecimento das instruções de processamento (sequências lógicas) e da alocação destas instruções na estrutura do programa.

No problema em análise são omitidas do Diagrama Lógico do Programa detalhes como: código de identificação, estoque antigo e estoque atual.

Considerando as regras mencionadas a estrutura do programa é:



4º passo: Controle do Diagrama Lógico obtido para o Programa.

Consiste em comparar a descrição da estrutura do programa com a dos dados de saída, verificando se para cada saída ocorre um subconjunto a ser efetuado o número de vezes requerido no endereço adequado do programa.

5º passo: Projeto do Fluxograma de Sequências Lógicas.

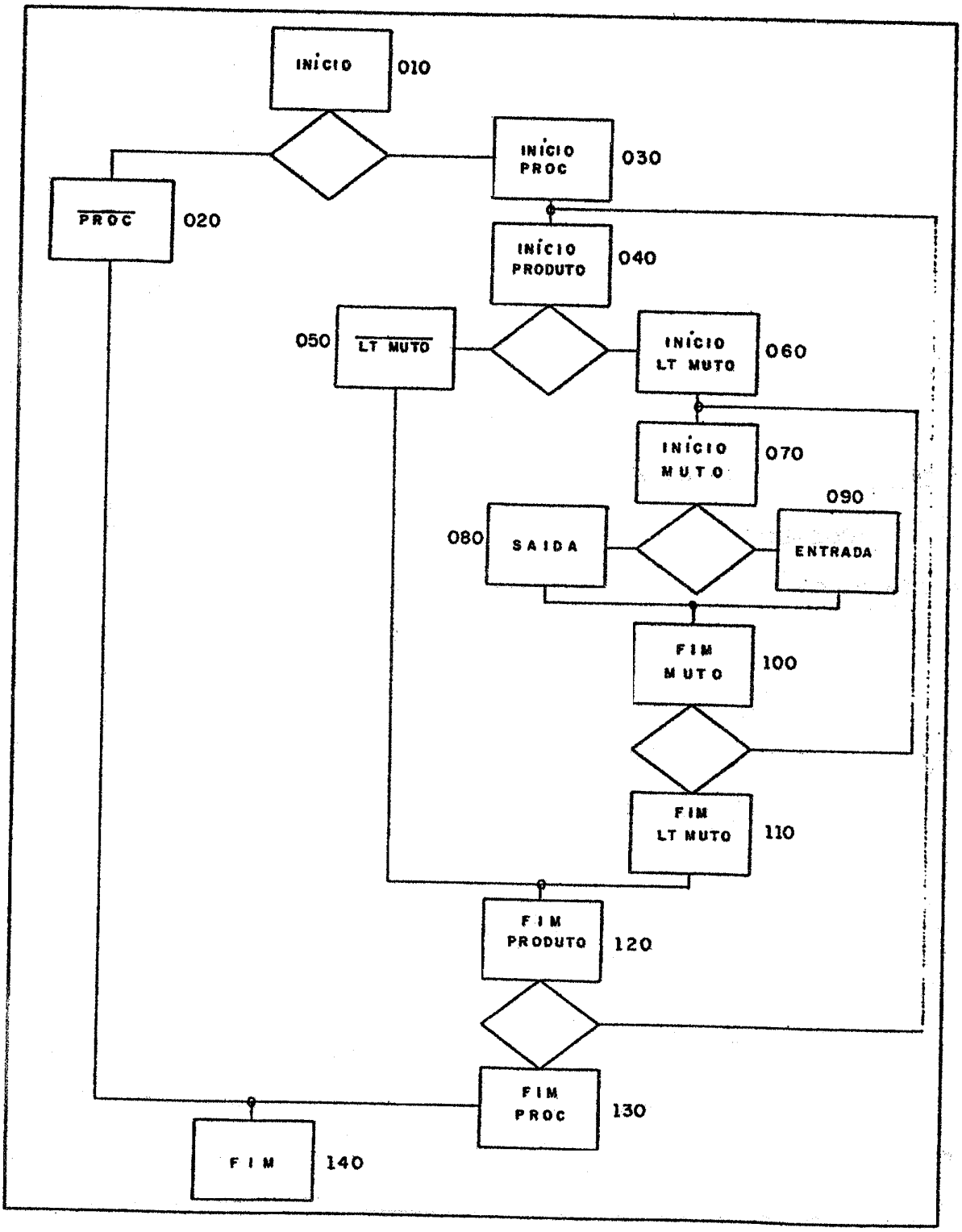
Esta etapa do projeto do programa corresponde a transposição da descrição obtida no passo anterior, em uma figura (fluxograma) que representa o desenrolar espaço - temporal do programa. Cada subconjunto que no fluxograma é representado por um retângulo é denominada "sequência lógica".

Definições

"Uma sequência lógica é um conjunto de instruções de programa executadas todas num mesmo número de vezes no mesmo endereço do programa, isto é, sob as mesmas condições".

"Consequentemente, duas sequências lógicas são obrigatoriamente separadas por uma função de desvio. Inversamente, duas funções ou dois desvios são sempre separados por uma sequência lógica".

Considerando-se estas definições, o fluxograma para o exemplo sendo analisado é:



6º passo: Estabelecimento da lista de instruções e elaboração do pseudo-código.

A primeira tarefa a ser executada neste passo consiste da enumeração das sequências lógicas do fluxograma, que pode ser observada no lado esquerdo dos retângulos no desenho obtido no passo anterior. Esta numeração poderá ser qualquer porém é praxe iniciar com a dezena e ir aumentando de 10 em 10.

Objetivando organizar o estabelecimento das instruções a serem executadas no programa, Warnier propõe que estas sejam inicialmente agrupadas nas seguintes categorias:

- Leituras,
- Desvios e Preparação de Desvios,
- Preparação de Cálculos e Cálculos,
- Preparação de Saídas e Saídas, e
- Chamadas de Sub-Rotinas.

Em paralelo, deverá ser realizada a associação entre o conjunto de instruções e o conjunto de sequências lógicas do programa, obedecendo-se a seguinte lei de correspondência:

"Uma instrução corresponde a uma sequência (elemento do conjunto de sequências) se ela é executada o mesmo número de vezes no mesmo endereço do programa".

Como consequência desta lei, se uma instrução corresponde à várias sequências lógicas, esta instrução deverá ser programada várias vezes: uma vez em cada sequência. Desta forma é verificada a aplicação de cada conjunto de instruções, tomado como conjunto de partida, dentro do conjunto de sequências lógicas do programa, tomado como conjunto de chegada.

Considerando-se o exposto e mais algumas regras e orientações colocadas por Warnier, teríamos:

Leituras

Seq. de referência	
010	leitura
040	leitura
100	leitura

Desvios

Seq. de referência		Seq. de destino
010	Não fim de arquivo	030
020		140
040	Se(Cod.prod.lido=cod.prod.em proc.)	060
050		120

Preparação de Desvios

Seq. de referência	
040	Código do Produto em zona de referência

Preparação de Cálculos e Cálculos

Seq. de referência	
040	Saldo antigo em zona de cálculo
060	Inicializar acumulador de saídas
060	Inicializar acumulador de entradas
080	Adição de entradas

Preparação de Saídas e Saídas

030	Inicializar linha de impressão
040	Mover cod.prod.,nome prod. e estoque antigo para linha de impressão
040	Imprimir linha
040	Inicializar linha de impressão

Terminada esta fase de estabelecimento das instruções a serem executadas então seria elaborado o pseudo-código do programa, sabendo-se que dentro de uma sequência a ordem das instruções deve ser a seguinte:

- preparação de desvios,
- preparação de cálculos e cálculos,
- preparação de saídas e saídas,
- leituras,
- desvios.

Assim, para o problema em questão, ter-se-ia o sequinte formato de pseudo-código.

```
010 Leitura
    Se não fim de arquivo                                   030
020                                                         140
030 Inicializar zona de impressão
040 Colocar cod. prod. em zona de referência
    Colocar estoques antigo em zona de cálculo
    Mover cod., nome e estoque antigo para linha imp.
    Imprimir linha
    Inicializar linha de impressão
    Leitura
    Se (cod.produto lido = Cod. de referência)              060
050                                                         120
060 Inicializar acumulador de saída (AS)
    Inicializar acumulador de entrada (AE)
070 Mover nº do doc. para linha de impressão
    Se código de movimento = E                              090
080 Somar quantidade a AS
    Mover quantidade para linha de impressão                100
090 Somar quantidade a AE
    Mover quantidade para linha de impressão
100 Imprimir e inicializar linha
    Leitura
    Se (cod. produto lido = código de referência)           070
110 Somar AE e zona de cálculo
    Subtrair AS da zona de cálculo
    Mover AS para linha de impressão
    Mover AE para linha de impressão
120 Mover zona de cálculo para linha de impressão
    Imprimir e inicializar linha
    Se não fim de arquivo                                   040
140 Finalizar programa
```

7º passo: Controle do Pseudo-Código obtido.

Neste ponto são verificadas se todas as saídas requeridas estão programadas dentro da sequência indicada.

8º passo: Codificação do Programa

Esta etapa é condicionada pela linguagem a ser utilizada e portanto não iremos desenvolvê-la.

5.3.3 - CONSIDERAÇÕES

Dado o paralelismo entre o método que acabamos de analisar e o proposto por Jackson, aqui iremos deixar de realizar uma crítica tão extensa quanto as feitas anteriormente e nos atermos apenas em salientar algumas das características mais importantes de LCP.

A primeira delas é, sem a menor dúvida, o alto grau de sistematização que impõe ao trabalho de programação, traduzido numa série de passos com resultados bem definidos, e viabilizados por um conjunto de leis, definições e regras que, além de orientar o programador, garantem um padrão de trabalho.

Por outro lado, propicia a documentação de todo o processo de entendimento do problema de uma maneira clara e precisa. A forma de descrição das estruturas de dados e programas em LCP é mais "independente" do processamento que a defendida por Jackson, o que permite uma comunicação mais fácil entre os elementos envolvidos com o desenvolvimento da aplicação e também, uma visualização mais rápida de estruturas semelhantes. O desenho das estruturas através de sua notação são bem mais fáceis de serem realizados, não exigindo qualquer material auxiliar, a exceção de uma régua.

Assim sendo, pode-se dizer que LCP atende aos requisitos de Registro de Soluções e Comunicação, favorece o planejamento e controle dos trabalhos, promove o aumento da produtividade dos programadores e leva a programas de boa qualidade.

Como o método de Jackson, LCP é mais próprio para os problemas em que a estrutura dos dados é condicionante da estrutu

ra do programa, nada impedindo contudo que venha a ser utiliza do em outra classe de problemas - só que neste caso não apresen tará a mesma eficiência.

Não se tem notícia de sistemas automatizados que apoiem sua utilização.

5.4 - HIPO (Hierarchy plus Input Process Output)

5.4.1 - Apresentação

O Hipo é um método desenvolvido pela IBM [13] que embo ra tenha como função a documentação de um sistema ou programa, possui também a propriedade de orientar o processo de detalha mento do problema.

Ao contrário do método de Jackson e LCP, que localizam a estrutura dos dados a processar, o HIPO procura visualizar o sistema ou programa através das funções que o mesmo deverá exe cutar. Assim, o objetivo principal do método é o levantamento do que o sistema ou programa irá executar antes de como o irá fazê-lo.

Considerando a dificuldade de se concluir uma documenta ção após a finalização do projeto, o HIPO a considera uma exten são do próprio esforço de detalhamento do projeto.

PACOTE HIPO

A aplicação do HIPO consiste no desenvolvimento de um conjunto de diagramas que descreve as funções de um sistema ou programa para os detalhes, sendo por isto uma extensão lógica do esforço de desenvolvimento "top-down".

Os principais objetivos do HIPO como técnica de projeto e documentação são:

- Prover uma estrutura pela qual as funções do programa possam ser entendidas. Os diagramas são organizados em uma estrutura hierárquica onde cada diagrama em

qualquer nível é um subconjunto do diagrama do nível superior. Desta forma programas complexos são subdivididos em partes administráveis.

- Determinar as funções a serem executadas pelo programa antes de especificar os comandos a serem usadas para realizar tais funções.
- Fornecer uma descrição visual dos dados de entrada e saída de cada função em cada nível de diagrama.

Os diagramas que compõem um pacote HIPO, exemplificados na figura abaixo, são:

1 - Tabela Visual de Conteúdo

2 - Diagrama Geral

3 - Diagrama Detalhado

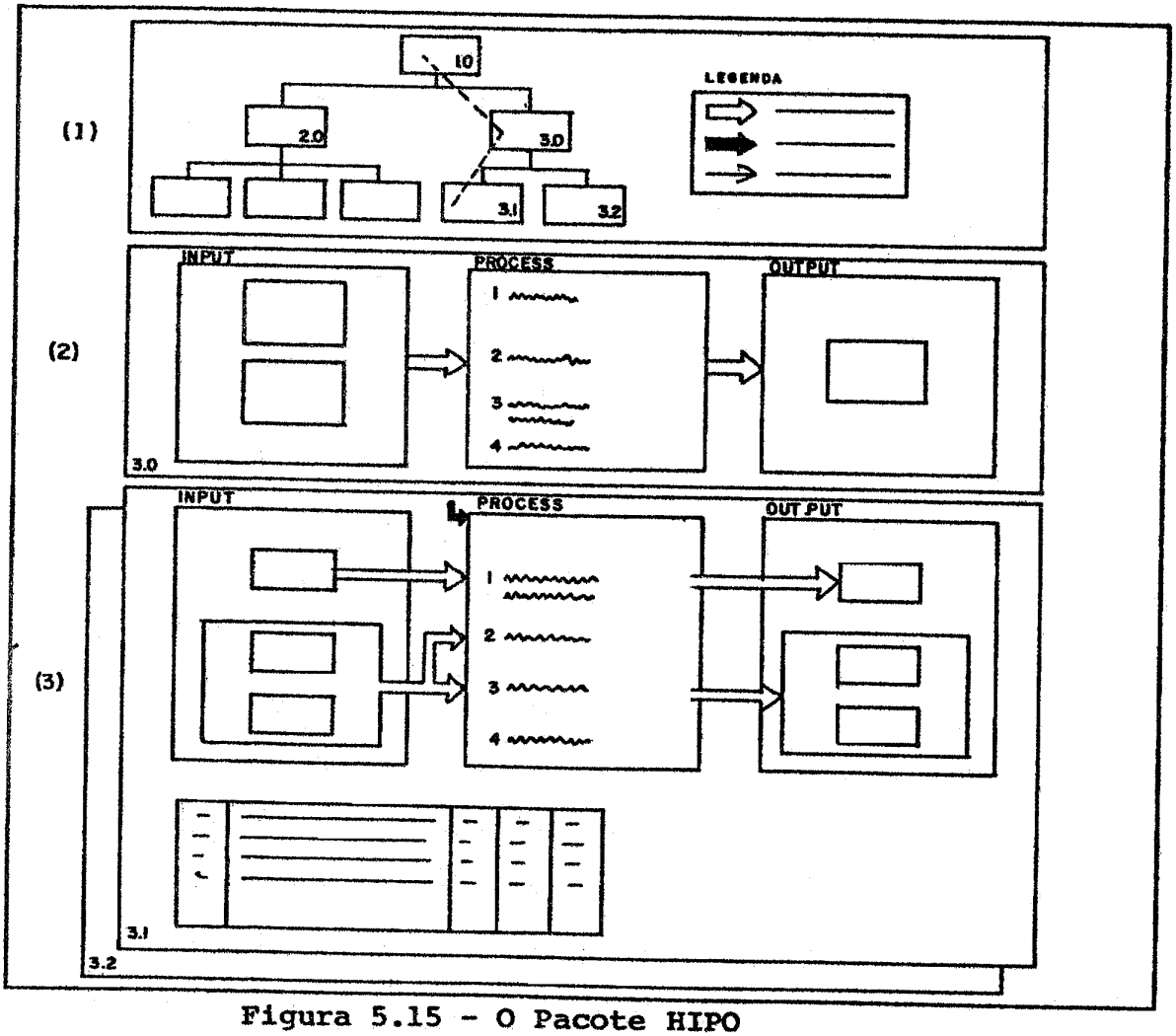


Figura 5.15 - O Pacote HIPO

. Tabela Visual de Conteúdo

A primeira etapa na aplicação do Hipo consiste de determinar de modo detalhado e completo todas as funções do sistema ou programa, bem como o relacionamento entre elas e então criar o diagrama denominado "Tabela Visual de Conteúdo", onde são apresentados em uma estrutura hierárquica, todas estas funções e seus relacionamentos. Também faz parte deste diagrama um quadro com a legenda explicativa dos símbolos utilizados no pacote.

. Diagrama Geral

A intenção deste diagrama é descrever as funções em um nível mais geral e então referenciar todos os "diagramas detalhados" necessários a expansão da referida função.

Em termos gerais, este diagrama contêm 3 seções denominadas: Entradas, Processos, Saídas.

- Seção Entradas

Esta seção contém os principais itens de dados usados pelos diversos passos do processo. Flexas são usadas para conectar os diferentes itens de dados aos passos correspondentes.

- Seção Processos

Contém uma série de passos numerados que descrevem a função sendo executada. Embora não exista limite de número de passos a serem representados nesta seção, devemos evitar que o número ultrapasse a 10, pois assim estaremos garantindo a legibilidade do diagrama.

- Seção Saídas

Contém aqueles itens de dados que são criados ou modificados nos diversos passos de processamento. Flexas são utilizadas para conectarem estes itens de dados aos passos correspondentes.

O diagrama geral poderá conter uma área denominada Des

crição Extendida, onde seriam ampliados os processos ou itens de dados de entrada e saída. Esta área apareceria no diagrama quando se fizesse necessário detalhar alguma informação com o objetivo de aumentar a compreensão do diagrama. Também serve para indicarmos outras documentações (flowchart, tabelas, seções, etc).

. Diagrama Detalhado

Nestes diagramas estão contidos os elementos fundamentais do pacote. São descritos todas as informações necessárias para o entendimento da função em um nível mais alto. Eles descrevem uma função específica e mostram dados de entrada e saídas específicas.

Assim, na seção PROCESSO detalhamos os passos de uma função específica enquanto que nas seções ENTRADA E SAÍDA, ao contrário de nome de arquivos representamos os registros e campos utilizados e gerados no processamento. Na seção de Descrição Extendida ampliamos os passos do processo, bem como referências a outros diagramas HIPO ou não HIPO tal como fluxogramas, tabelas de decisão, lay-outs de registros. O número de níveis dos diagramas de detalhes é função do número de subconjuntos funcionais, da complexidade do programa e da quantidade de informações a serem documentadas.

NOTAÇÃO

Complementando o "pacote" HIPO, e tendo como objetivo facilitar a elaboração dos diagramas e obter padronização na documentação, a IBM propõe o uso de folhas de trabalho padronizadas (HIPO Worksheet) e do gabarito de símbolos (HIPO Template). Como ilustração apresentamos a seguir alguma das notações a serem utilizadas no desenho dos diversos diagramas que, em alguns casos, são semelhantes àquelas propostas para fluxogramas.

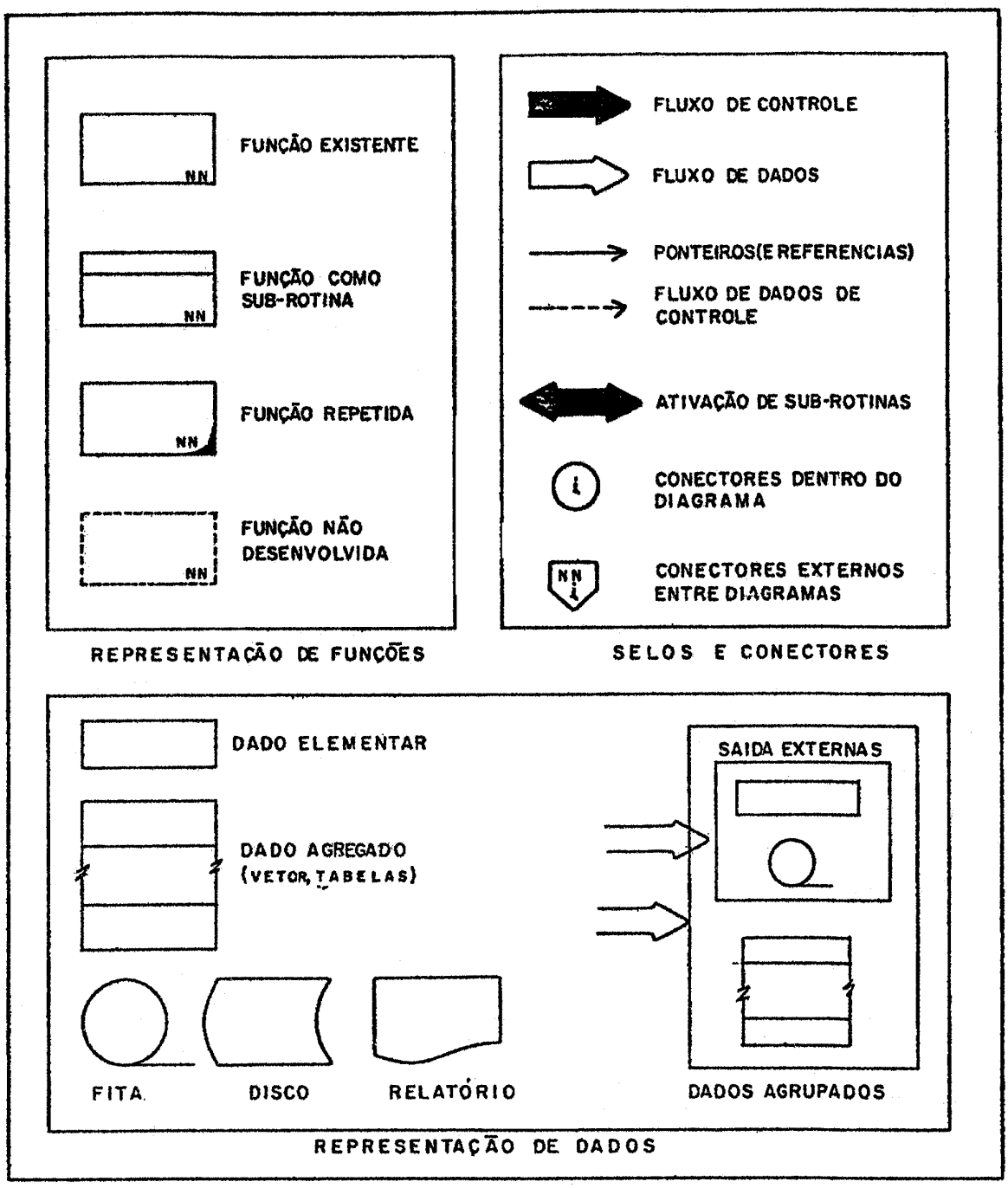


Figura 5.16 - Exemplos da Notação de HIPO

Além da notação exemplificada, são propostas também algumas convenções relacionadas com o seu uso. Estas convenções, omitidas neste trabalho, podem ser encontradas na referência {13}.

UTILIZAÇÃO DO PACOTE HIPO

Na figura 5.17, adaptada de {25}, tem-se esquematizado a proposta de uso do pacote HIPO durante as diversas etapas do ciclo de vida do sistema ou programa.

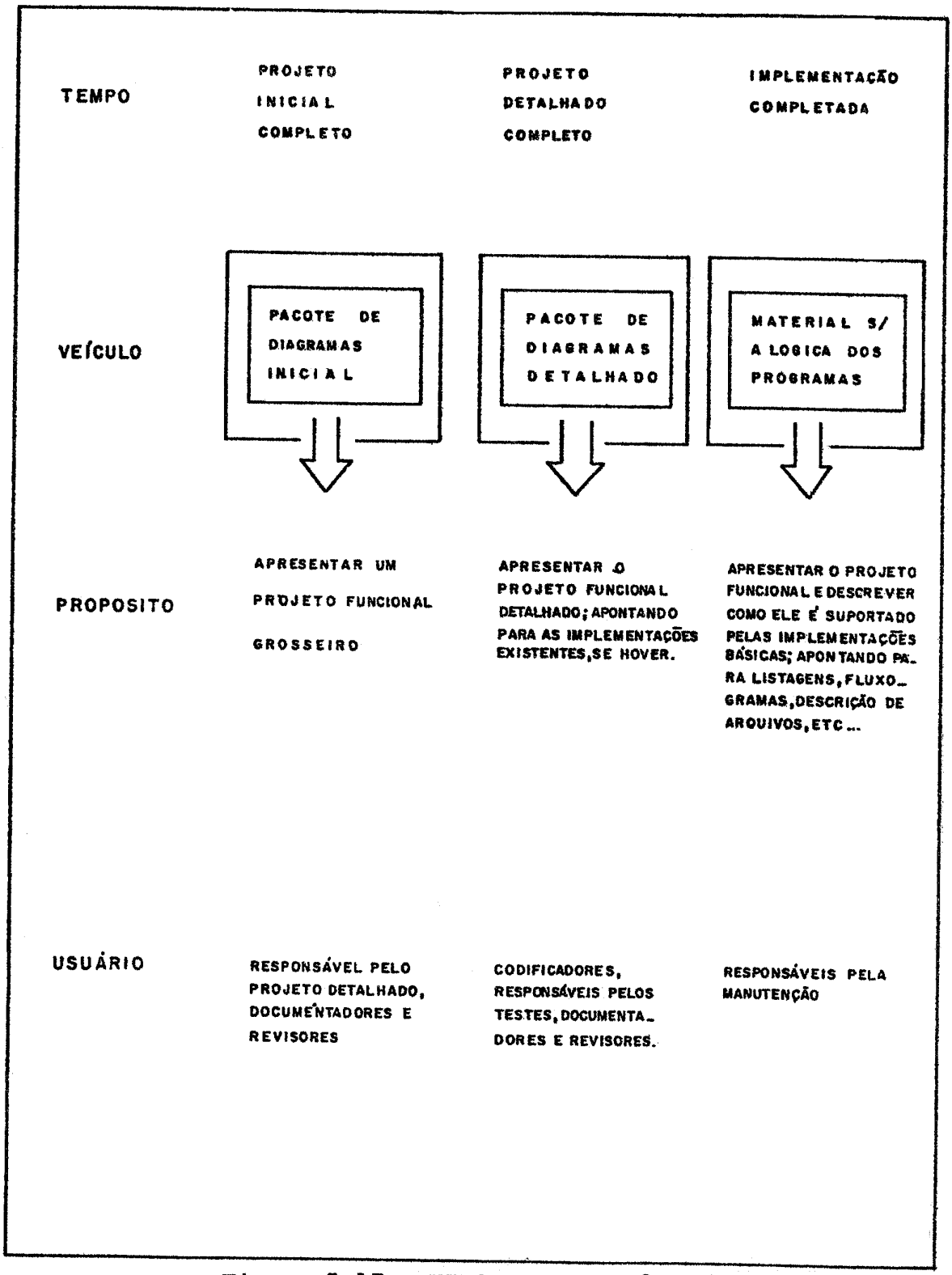


Figura 5.17 - HIPO: Esquema de Uso

Projeto Inicial

Inicialmente várias alternativas são analisadas sendo que uma é selecionada para um maior detalhamento nesta primeira fase do projeto A partir desta decisão, então os responsáveis pelo desenvolvimento do projeto preparam o primeiro plano "top-down" para o novo sistema ou programa, numa tentativa de "quebrar" a aplicação proposta em termos de funções a serem executadas. O resultado deste passo seria uma estrutura semelhante a Tabela Visual de Conteúdo.

Considerando esta estrutura básica obtida, os analistas/programadores passariam a desenhar os diagramas gerais e os detalhados.

Nesta fase, os diagramas gerais contêm apenas um grosseiro projeto funcional e fluxo de dados gerais. Também, os diagramas detalhados contêm apenas idéias preliminares sobre os processos que irão suportar as várias funções. As descrições extendidas deverão se resumir a requisitos preliminares de processamento.

Observamos que os dados levantados nesta fase contribuem para uma estimativa dos recursos humanos requeridos, da programação dos trabalhos e do custo do projeto.

Projeto Final

Nesta fase os diagramas inicialmente obtidos são revisados e expandidos até se alcançar o projeto funcional detalhado do sistema ou programa, tendo-se ao seu término o projeto final completo.

Os diagramas detalhados são mais específicos e contêm detalhes das entradas e saídas, bem como dos processos que suportam as diversas funções ntificadas.

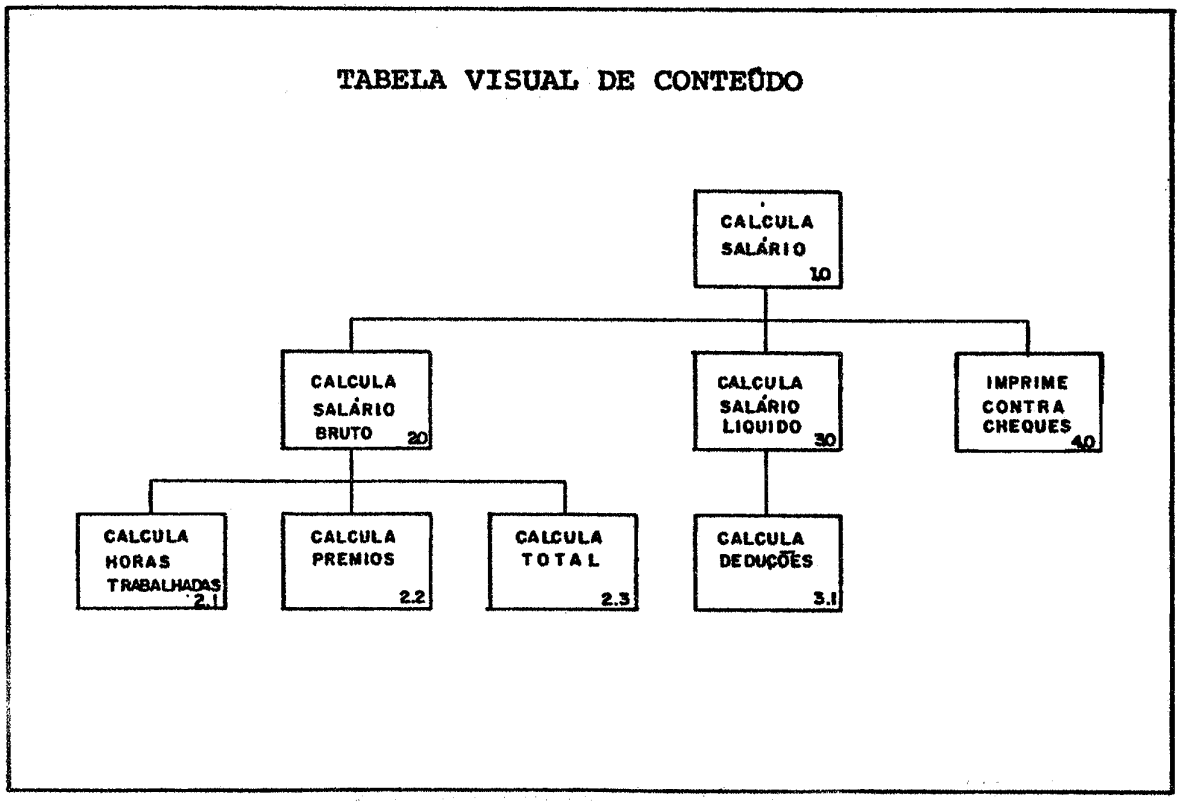
Codificação e Implementação

Aqui os diagramas detalhados são revisados tanto quanto necessário enquanto os programas são codificados e testados. No final deve-se ter desenhado os "lay-outs" de registros e tabelas, e revisadas as descrições detalhadas em referências-cruzadas com os fluxogramas e listagens de programas. Esta revisão seria realizada pelo programador responsável pelo desenvolvimento e implementação do programa e ao seu final ter-se-ia o pacote HIPO completo.

5.4.2 - Exemplo:

Para melhor evidenciar o uso do HIPO adotaremos como sendo nosso problema o cálculo de salários, em substituição ao que vinha sendo desenvolvido até o momento.

De modo a permanecermos dentro do escopo do trabalho iremos apresentar apenas parte dos diagramas.



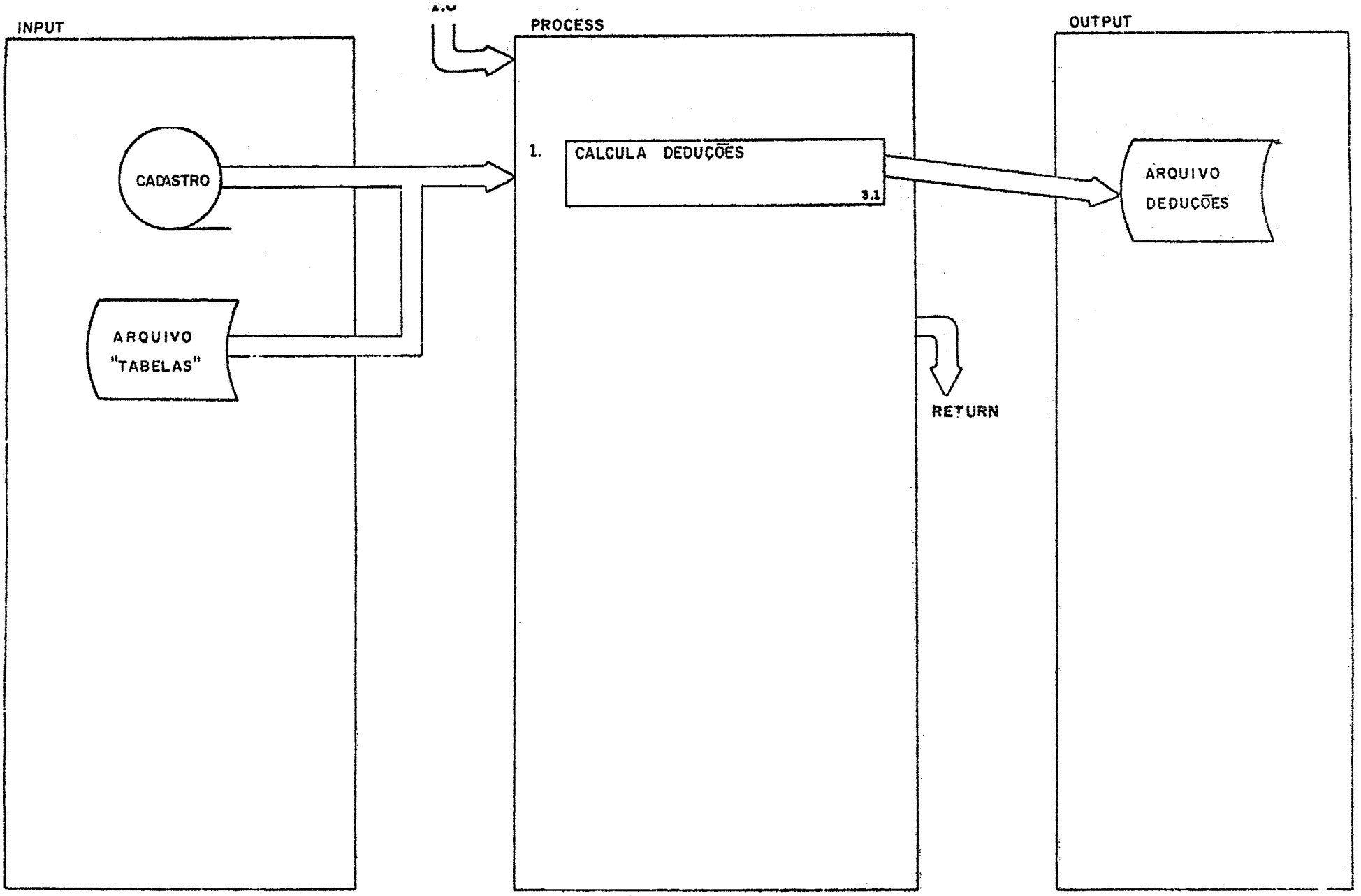


DIAGRAMA Nº 3.0

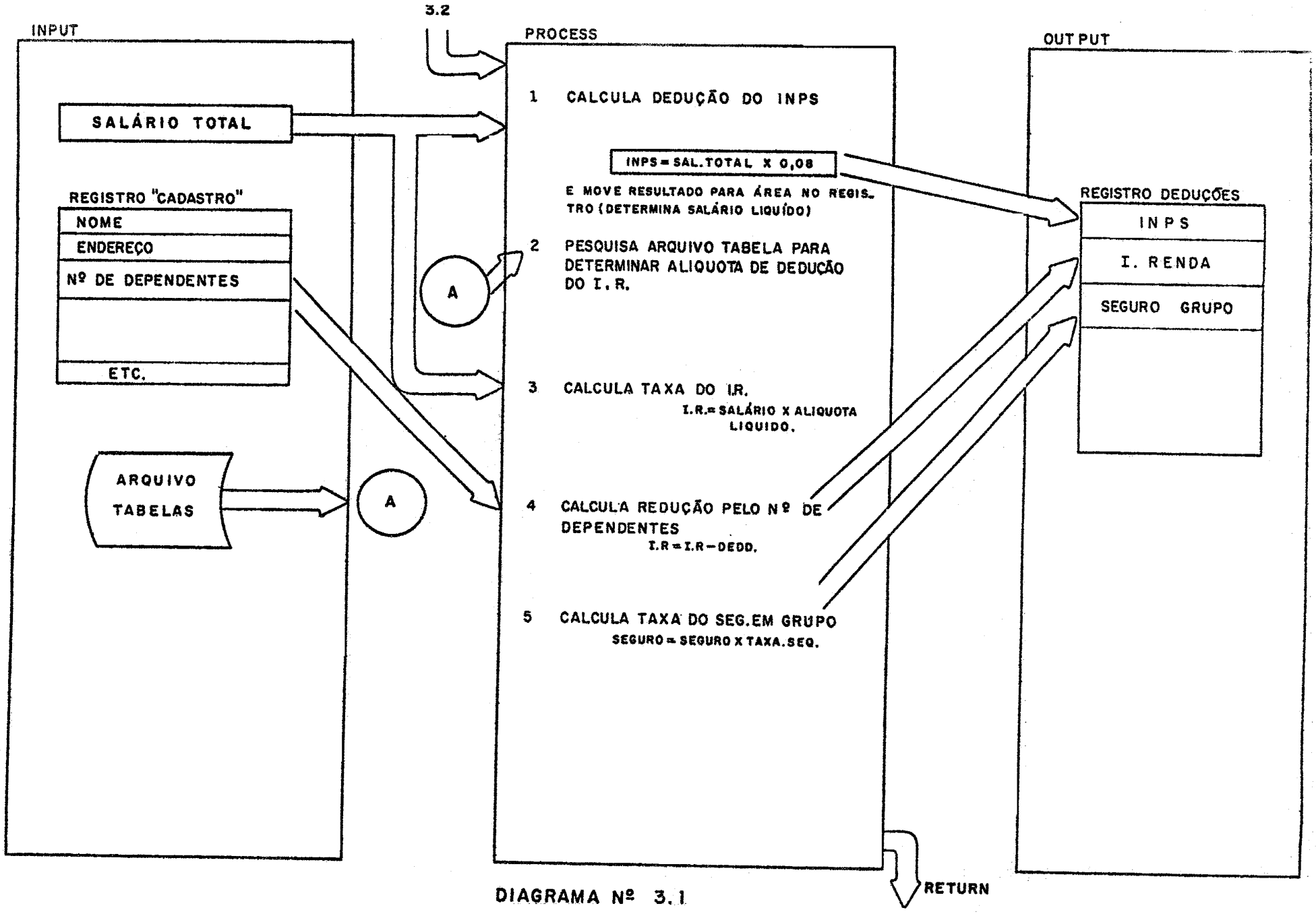


DIAGRAMA Nº 3.1

5.4.3 - Considerações

A descrição do problema através da Tabela Visual de Conteúdo e dos Diagramas Geral e Detalhado, caracteriza uma forma de resolução através de níveis sucessivos de refinamento e torna o HIPO um método de trabalho compatível com o processo humano de soluções de problema. Ao restringir inicialmente o profissional apenas a definição de atividades/funções a serem executadas pelo programa, este método leva primeiro a busca de uma compreensão global sem envolvimento com detalhes do processamento, eliminando assim a situação conhecida como "a perda de visão da floresta devido a contemplação de árvores individualmente".

Talvez por ter sido desenvolvido, a princípio, como uma ferramenta de documentação, o HIPO deixa muito a desejar, quanto a procedimentos para a determinação e divisão das atividades/funções que comporão o programa. Dessa forma, a estrutura do programa será decorrente apenas da experiência do responsável pela sua concepção. Essa deficiência poderá ser eliminada incorporando ao HIPO os conceitos descritos no método Structured Design - o qual é discutido a seguir - conforme foi assinalado por Stay {26}. Em resumo, o que este autor propõe é que os módulos (atividades/funções) sejam determinados utilizando-se as regras especificadas no método Structured Design, sendo que a documentação e detalhamento dos módulos seriam realizados obedecidos os critérios de HIPO.

Também no tocante a estrutura interna dos módulos, o método HIPO não apresenta qualquer orientação no sentido de que se obedeça os preceitos estabelecidos pela programação estruturada. Inclusive, sua simbologia, ao prever uso de conectores, contém facilidades para desvio arbitrário do fluxo de controle (GO TO's). Uma solução para este problema, apontada por ORR {24}

e Neuhold {23}, é a utilização do método Nassi-Schneiderman Chart na descrição interna dos blocos de processo do diagrama geral e detalhado.

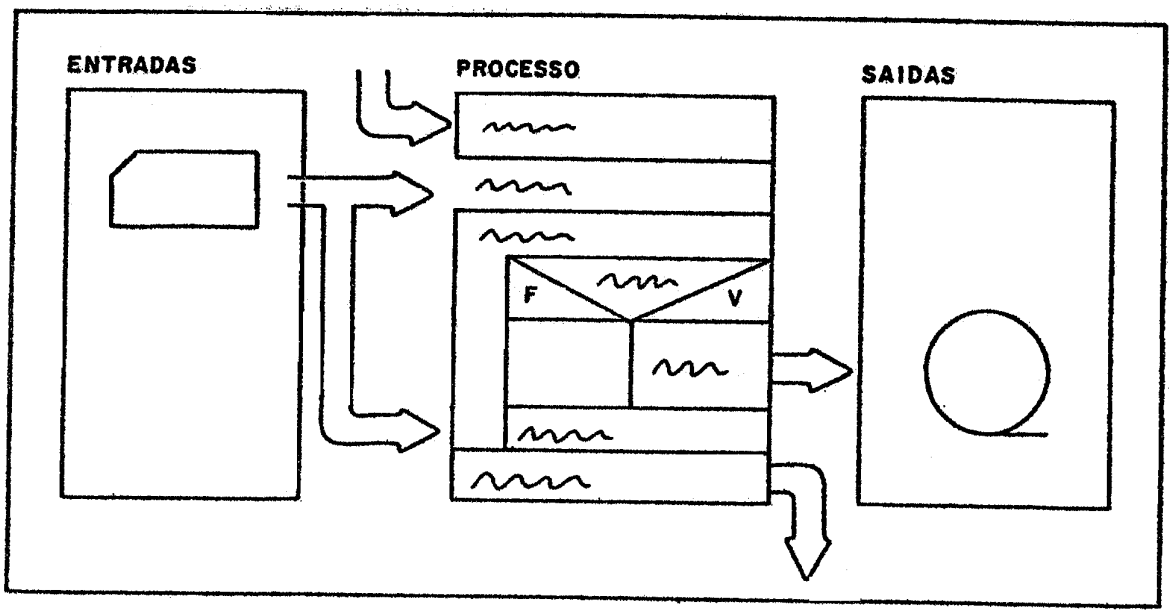


Figura 5.18 - HIPO com Nassi-Schneiderman Charts

Implantados estes enxertos, tem-se no HIPO uma metodologia mais ativa no processo de desenvolvimento de programas, já que também orientaria o processo de análise do problema. No seu estado puro, este método tem uma participação passiva servindo apenas como forma de registro das soluções e como um meio de comunicação.

Considerações específicas para o HIPO em função dos parâmetros por nós estabelecidos no capítulo dois são:

a) Registro de Soluções e Comunicação

Efetivamente, a seu modo, o HIPO promove o registro das soluções alcançadas ao longo do desenvolvimento do programa, garantindo a preservação dos conhecimentos que venham a ser adquiridos. Como já foi dito, este método, na maneira como já foi proposto, não participa ativamente da compreensão do problema, porém esta falha pode ser corrigida com a incorporação de outros

conceitos.

É uma técnica que apresenta etapas de desenvolvimento dos trabalhos com resultados bem definidos, o que facilita o planejamento e controle da atividade de programação. Os diversos diagramas a serem construídos constituem uma maneira clara e precisa para o acompanhamento e avaliação dos trabalhos em andamento.

A notação proposta para uso no HIPO é relativamente simples o que facilita a compreensão dos diagramas envolvidos com sua aplicação mesmo por pessoas estranhas a área de processamento de dados. Contudo, a comunicação pode vir a ser prejudicada com o excesso de informações colocadas em um único diagrama.

b) Produtividade

A produtividade na atividade de programação será aumentada com o uso do HIPO em função da organização do trabalho imposta por este método e que cria uma atitude padrão perante os problemas.

Contudo, este acréscimo de produtividade não será tão significativo quanto o proporcionado pelo método de Jackson, LCP e Composite Design que, além de criar um padrão de comportamento, auxiliam na compreensão do problema.

De um modo geral, já que está implícito o processo de refinamentos sucessivos, os programas obtidos com o uso deste método deverão ser mais isentos de erros e fáceis de serem depurados e mantidos. Aqui também os enxertos propostos poderão ser de grande utilidade.

Quanto ao aspecto de destaque de estruturas semelhantes, achamos que este parâmetro deixa de ter sentido para crítica do HIPO, dado a classe de aplicações a que se destina. Natu

ralmente, alguma experiência pode ser repassada de uma aplicação para outra, porém nunca se terá semelhanças como no caso das aplicações a que se destinam os dois métodos que acabamos de analisar.

c) Qualidade

O HIPO produz um projeto claro e preciso, o que redunda em programas de melhor qualidade. Outrossim, ao levar a programas modulados tem-se que estes necessariamente serão mais compreensíveis e manuteníveis.

Sobre este aspecto de modularização, salientamos o fato de o HIPO não apresentar qualquer orientação quanto a composição dos módulos e sobre o relacionamento entre eles, o que, pode comprometer a qualidade de um programa. Desta forma, o resultado final da aplicação deste método poderá ser programas perfeitamente documentados porém difíceis de serem mantidos e mesmo depurados. Observamos também a facilidade com que os desvios podem ser representados na descrição dos processos.

d) Praticabilidade

Este método é fácil de ser aprendido, contudo o mesmo não se pode dizer quanto ao seu uso.

Como dá para ser deduzido do exemplo apresentado, a elaboração dos diagramas não é uma tarefa fácil de ser realizada, demandando bastante tempo para sua conclusão. A combinação de setas e demais símbolos deverá ser cuidadosamente balanceada para não se chegar a diagramas com excesso de dados e cruzamentos que os tornam difíceis de serem compreendidos. Pelas mesmas razões, a manutenção da documentação é bastante trabalhosa o que significa um desistímulo para a sua realização.

e) Generalidade e sistemas de Suporte

O HIPO é um método que pode ser aplicado em qualquer

situação, porém sua eficiência se torna maior quando utilizado na classe de problemas onde são identificados uma série de funções a serem executadas, sem ligações com a estrutura dos dados - caso em que as duas últimas técnicas analisadas são mais indicadas.

Não existem sistemas automatizados que suportem a aplicação deste método.

5.5 - COMPOSITE DESIGN (ou STRUCTURED DESIGN)

5.5.1 - Apresentação

Concluindo nossa discussão sobre técnicas de desenvolvimento de programas iremos examinar agora uma metodologia bastante divulgada nestes últimos anos e que tem seus fundamentos estabelecidos por G. J. Meyers {21} e E. Yourdon {33}.

É um método semelhante ao HIPO quanto a ter como objetivo central o evidenciamento das funções que compõem a aplicação sendo modelada. Contudo, Composite Design difere sensivelmente da metodologia que acabamos de analisar. Enquanto o HIPO apenas estabelece que devem ser identificadas as funções a serem executadas pelo programa, sem propor qualquer caminho para que isto venha a ser realizado, Composite Design é todo um procedimento para se determinar estas funções e seus relacionamentos.

Como LCP e o método de Jackson, Composite Design também parte dos dados para determinar a estrutura do programa. Todavia, as duas primeiras metodologias defendem o princípio que a estrutura do programa é decorrente da estrutura dos dados, ao passo que Composite Design se baseia nas transformações ocorridas com os dados ao longo do processamento para determinar a estrutura do programa. Estas transformações definiriam as diversas funções, módulos do programa, e seus relacionamentos

Na figura 5.19 tem-se esquematizado o processo de desenvolvimento de programas com a aplicação de Composite Design.

PRINCÍPIOS DE COMPOSITE DESIGN

Genericamente, Composite Design consiste da aplicação de uma série de princípios que podem ser agrupados em:

- um conjunto de regras que especifica a sintaxe da estrutura de um programa, e visa a independência dos módulos;

- um conjunto de técnicas que constrange o programador a decompor ou a dividir os programas dentro da abordagem "TOP-DOWN".

O primeiro conjunto de regras trata da medida de relação dentro do módulo (module strength), da medida de relação entre pares de módulos (module coupling) e do tamanho e extensibilidade do módulo.

Determinar a medida de relação dentro do módulo envolve a análise da função ou funções executadas pelo módulo e então encaixar o módulo entre uma das categorias descritas abaixo, em ordem crescente de coesão:

Ruim
- Por coincidência - Encerra elementos sem relações significativa entre si;
- Lógica - Durante cada chamada, executa uma função selecionada entre uma classe de funções relacionadas entre si;
- "Procedural" - Sequencialmente executa uma classe de funções relacionadas entre si em termos do procedimento do programa;
- Temporal ou clássica - Sequencialmente executa uma classe de funções relacionadas;
- Comunicacional - É um módulo "procedural", onde todas as

suas funções são relacionadas em termos de usos de dados;

- Sequencial - Combinação de funções sequenciais no fluxo dos dados;
- Informacional - Contém várias funções que operam sobre uma mesma estrutura de dados;

BOM
- Funcional - O módulo encerra uma função única e simples.

Já a medida de relação entre módulo se refere aos dados processados por estes e diz respeito tanto a forma usada para passar dados e atributos, como ao próprio dado. Cada par de módulo é analisado e colocado dentro de uma das seguintes categorias de acoplamento:

Ruim
- Por conteúdo - Um módulo se referencia ao conteúdo do outro módulo;
- Por "commom" - Os módulos utilizam uma mesma estrutura global;
- Externo - Os módulos se referem a uma mesma variável global;
- Controle - Um módulo controla a lógica de um outro módulo;
- Por estrutura - Os módulos se referem a uma mesma estrutura de dados não global;

Bom
- Dados - Os módulos se comunicam através de parâmetros de itens de dados (não estrutura).

Estas medidas de relações, dentro e entre módulos, juntamente com as orientações de tamanho de módulo e sua extensibilidade - o tamanho de um módulo deve ser tal que sua compreensibilidade não fique prejudicada - permitem avaliar soluções alternativas mas não determinam o processo de concepção do programa.

Para tanto, Composite Design compreende um conjunto de técnicas que definem um processo de raciocínio "TOP-DOWN". Este processo é iterativo, e inicia com a anälise da estrutura do problema e como o dado é transformado ao fluir através desta estrutura. Esta informação é usada para decompor o problema em uma hierarquia de módulos. Cada módulo é visualizado como um subproblema e a análise é repetida. Este processo de decomposição é continuado até que seja alcançado um ponto onde consideramos ter a estrutura "ideal". Então passamos a definição das interfaces entre os módulos e, em seguida, a codoficação dos módulos e montagem do programa.

ESQUEMA DE DESENVOLVIMENTO COM COMPOSITE DESIGN

O esquema de desenvolvimento está representado na figura 5.19 e corresponde as descrições apresentadas a seguir.

Elaboração do Diagrama de Fluxo de Dados (Data-Flow Charts)

"Data-Flow Chart" é a técnica proposta em Composite Design para representar as transformações sofridas pelos dados ao fluir pelo sistema. O gráfico resultante não contém informações de controle e nem pretende definir sequeências de ações ao longo do tempo. Seu propósito é ajudar na análise das transformações ocorridas com os dados de entrada a fim de alcançar a saída desejada. Em essência, ele apresenta os caminhos do fluxo de informações.

Para construção deste diagrama, Composite Design propõe a notação apresentada na figura 5.20 onde temos a descrição dos dados através de linhas retas e as transformações representadas por círculos ou, mais raramente, por retângulos. A direção normal do fluxo é da esquerda para direita.

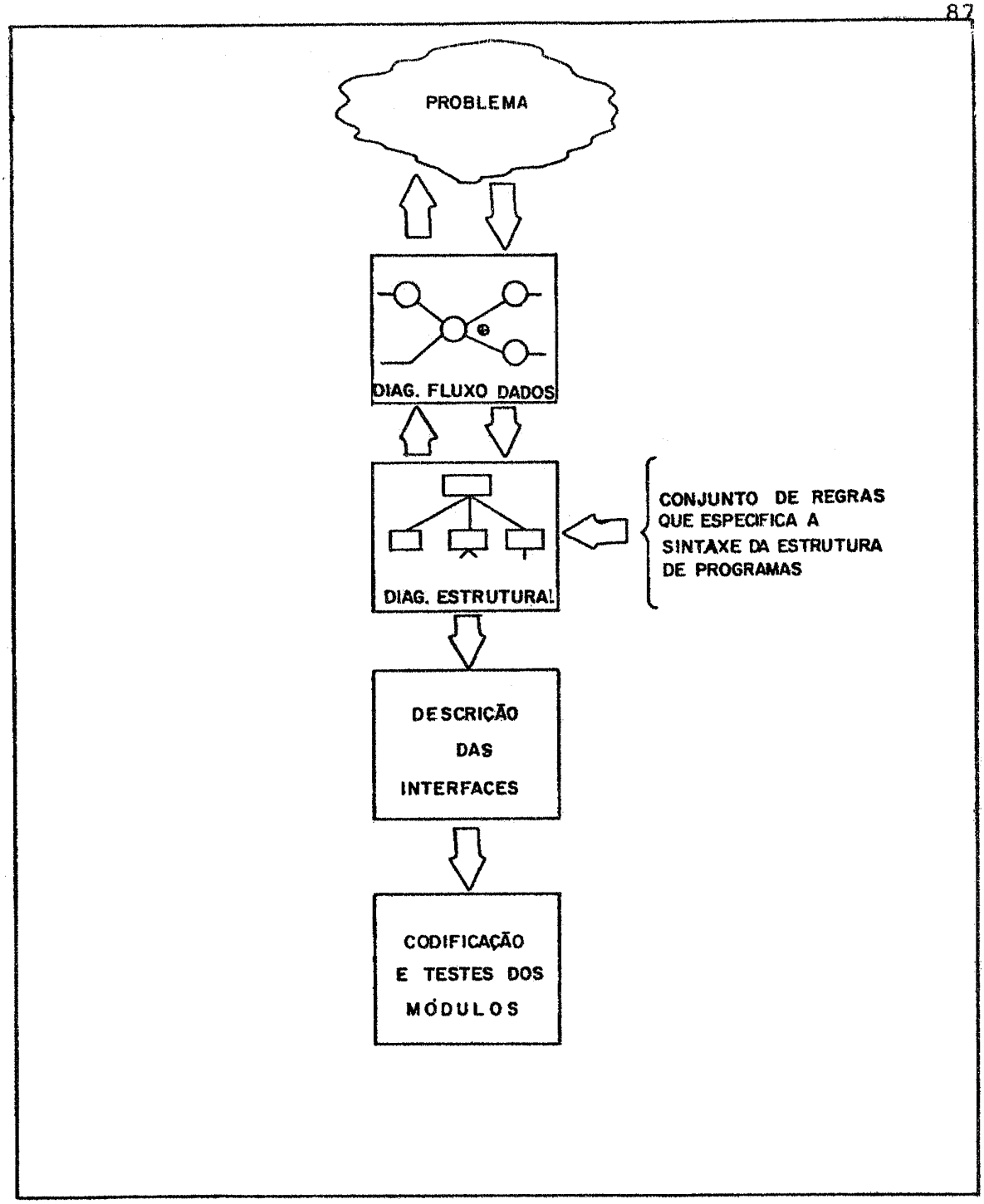


Figura 5.19 - Esquema de Desenvolvimento de Programas com Composite Design.

Obtenção do Diagrama de Estrutura

O propósito desta etapa é a obtenção da organização hierárquica do programa. São definidos os módulos e suas relações, e identificadas suas interfaces com os métodos de controle.

A estrutura do diagrama é gerada a partir do Diagrama de Fluxo de Dados, sendo que as transformações neste diagrama tornam-se os módulos procurados. A decomposição em módulos é

realizada obedecidos certos critérios, encontrados nas referências {21} e {33}, que resumidamente significam a identificação dos ramos aferentes (entradas) que coletam e formatam os dados de entrada na forma adequada ao processamento, os de transformação que executam as funções básicas da aplicação e os eferentes (saídas) que formatam e distribuem as saídas. A forma como estes ramos forem seccionados no diagrama de fluxo determina os módulos colocados abaixo do segundo nível da estrutura do diagrama, já que o primeiro nível corresponde ao módulo de controle geral e no segundo nível tem-se os módulos responsáveis pelo controle da captação, transformação e saída de dados.

Ressaltamos que os módulos assim obtidos devem ser analisados considerando-se as regras de sintaxe de estrutura de programas e ao final deste estudo, pode-se concluir pela necessidade de redefinição de um novo Diagrama de Estrutura. Este outro Diagrama de Estrutura é definido mudando-se o seccionamento realizado anteriormente no Diagrama de Fluxo de Dados.

Na figura 5.20 apresentamos parte da notação proposta por Myers {21} para construção do Diagrama de Estrutura que, basicamente, se resume em retângulos, para representar módulos, e setas, para indicar a direção do fluxo. A base da seta indica o tipo de fluxo, se dados ou controle. Símbolos adicionais existem para retratar iteração, fator de decisão, etc...

Salientamos que a figura que segue não contém toda a notação proposta por Myers e, que esta diverge substancialmente daquela definida por Yourdon e Constantine {33}, sendo isto uma das principais diferenças entre Composite Design e Structured Design.

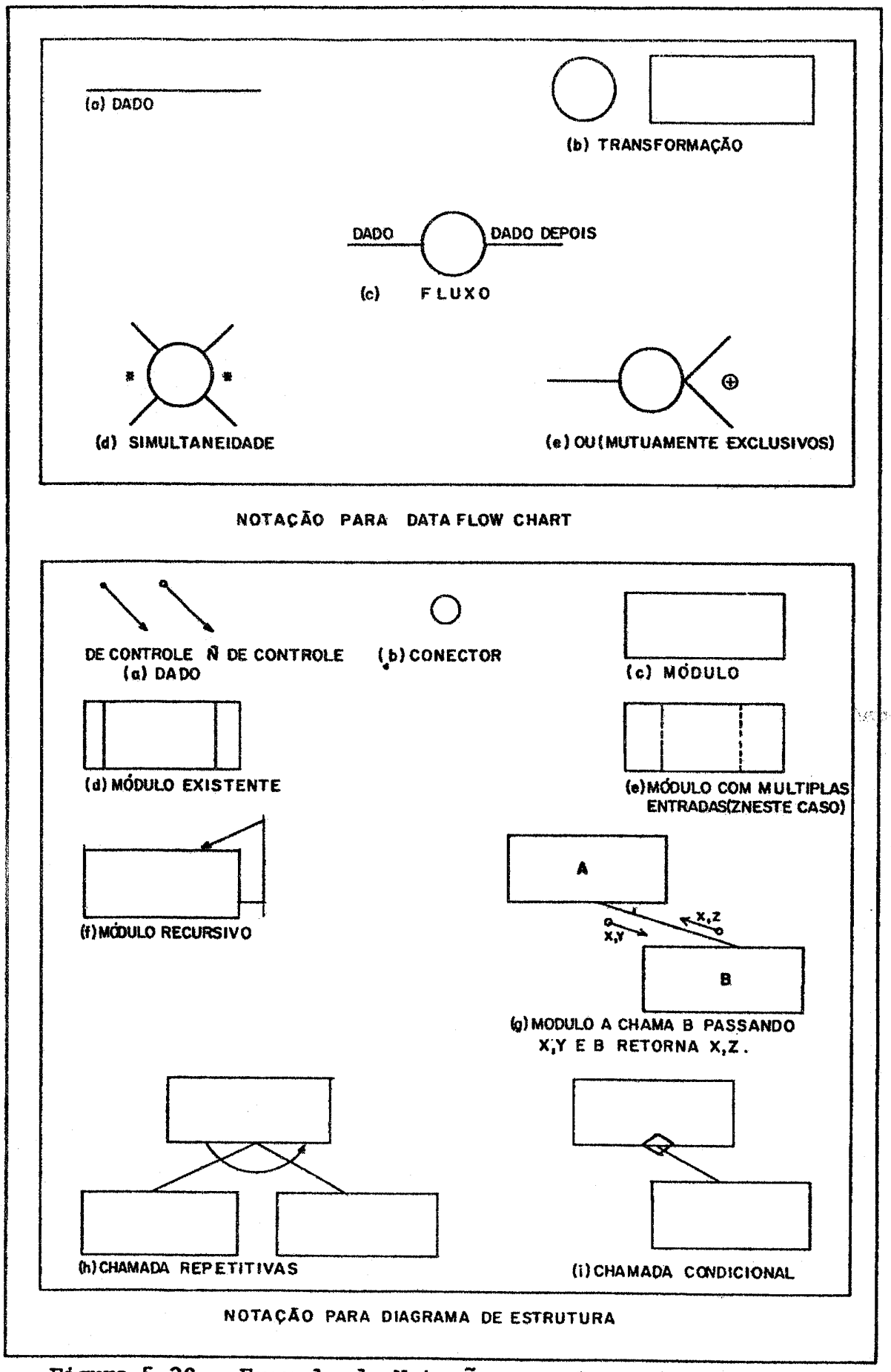


Figura 5.20 - Exemplo de Notacão para Composite Design

Descrição das Interfaces

Como penúltima etapa do processo tem-se a descrição das interfaces entre módulos. Cada dado identificado na estrutura de diagrama - argumentos entre módulos e dados recuperados de arquivos acessados pelo módulo, etc... - são detalhados em termos de tamanho, campos, limites de variação, e demais informações que bem definem o dado. Como resultado, tem-se o dicionário de dados para o sistema.

Codificação do Programa

O método em si não especifica a forma como os módulos devem ser elaborados. Uma solução possível seria a adoção do método do Nassi -Shneiderman para a definição da lógica interna, sendo a documentação final realizada com base na proposta de Meyers {20}, que defende a auto-documentação dos módulos. Nesse caso, cada módulo conteria as seguintes informações:

- Função do módulo;
- Forma de Chamada;
- Descrição dos parâmetros de entrada e saída;
- Causa e efeito: Relacionamento dos parâmetros de entrada e saída;
- Efeitos externos: Ações ativadas fora do sistema ou programa ativado pela chamada de módulo;
- Resumo da lógica do módulo;
- Comandos fonte.

5.5.2 - Exemplo:

Evidenciando as vantagens de Composite Design desenvolveremos a seguir o mesmo problema utilizado para exemplificar o HIPO.

Para nos ater ao escopo do trabalho iremos resumir nos

sa apresentação à apenas parte dos diagramas que se fariam necessários para a completa definição do problema. Também serão omitidas as etapas de análise dos módulos e a de Codificação do Programa.

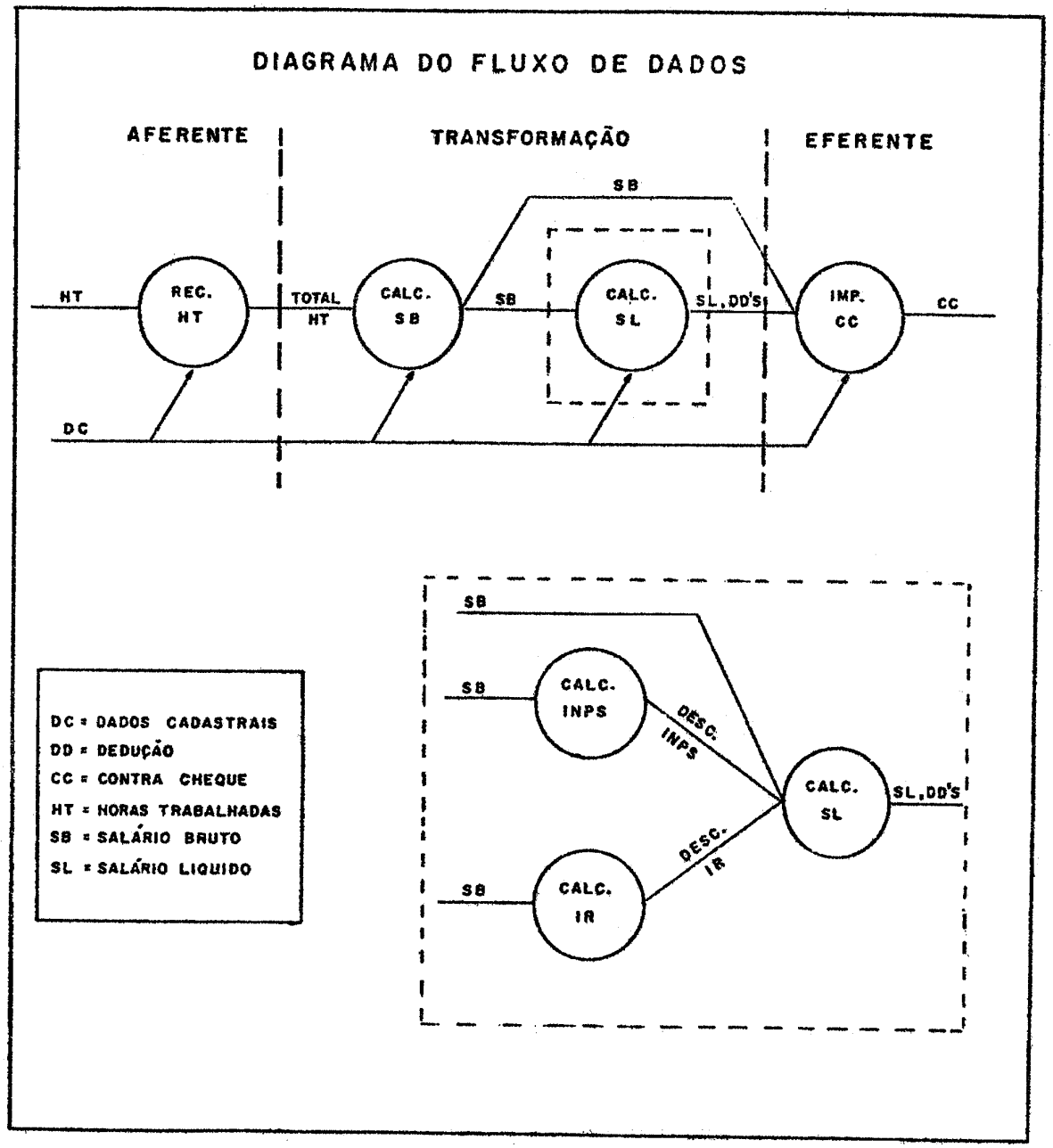
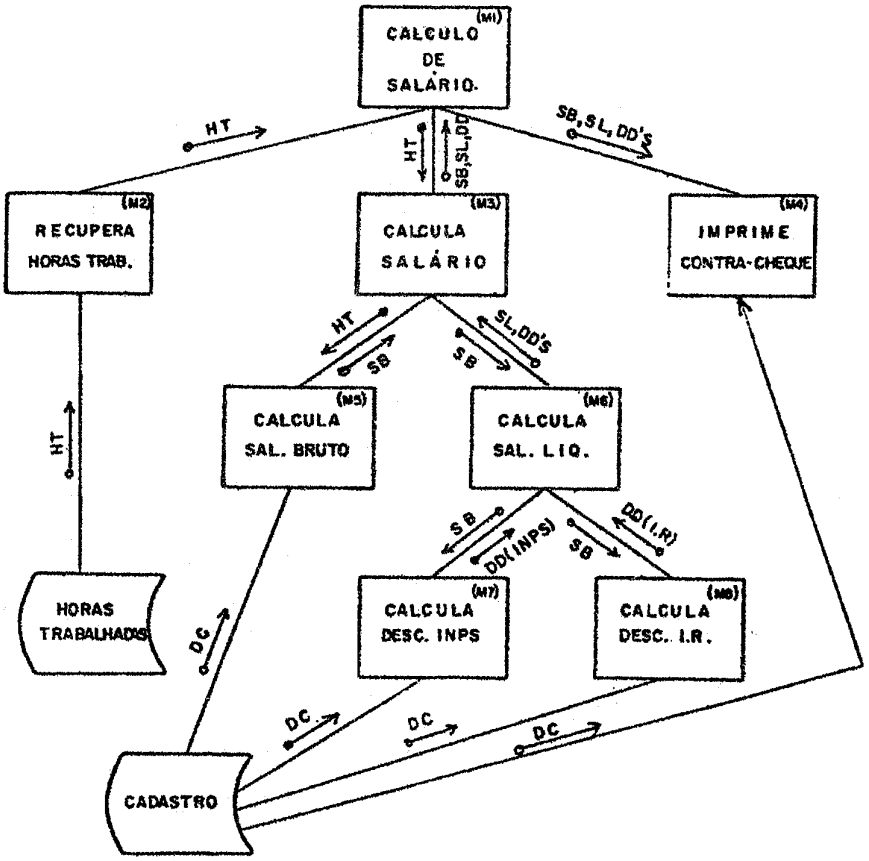


DIAGRAMA DE ESTRUTURAS



DESCRIÇÃO DAS INTERFACES

MODULOS		ARGU-	TIPO	VARIAÇÃO	----------
DE	PARA	MENTO			
M2	M1	HT	NUM.	0 — 300	
M1	M3	HT	NUM.	0 — 300	----------
M3	M5	HT	NUM.	0 — 300	
M5	M3	SB	NUM.	10.000-300.000	

5.5.3 - Considerações

Como o método de Jackson e, consequentemente, LCP, Composite Design constitui um marco dentro da evolução das metodologias para desenvolvimento de programas. Da mesma forma que elas, a técnica que acabamos de apresentar está além de uma mera proposta de documentação ou de estruturação do fluxo de processamento, porquanto estabelece um conjunto de princípios que tem impacto no processo de concepção e na qualidade de um programa.

Por outro lado, juntamente com as metodologias acima mencionadas, Composite Design compõe um grupo de ferramentas que proporciona recursos para se abordar de forma eficaz todas as classes de problemas colocados à área de processamento de dados.

Feitas estas considerações gerais passemos às específicas em função dos pontos estabelecidos no capítulo 2.

a) Registro de Soluções e Comunicação

Composite Design consiste de uma proposta de operacionalização do conceito de análise de problemas através de refinamentos sucessivos de sua estrutura, o que o torna uma ferramenta ativa no processo de concepção do programa. Ainda mais, apresenta uma forma sistemática de documentação dos conhecimentos que vão sendo adquiridos nas diversas etapas do desenvolvimento dos trabalhos, o que garante a preservação das soluções alcançadas.

Por constituir etapas de trabalhos bem diferenciadas, promove o planejamento e controle da atividade de programação.

A comunicação entre os envolvidos com o desenvolvimento do projeto deve ser favorecida com a utilização de Composite Design, pois se utiliza ao máximo de notação gráfica e cria uma linguagem bem determinada.

b) Produtividade

A produtividade deverá ser aumentada como decorrência da sistematização imposta por esta metodologia na abordagem dos problemas. Outro fator que contribuirá para isto é o aspecto da modularização dos programas, que os torna mais fáceis de serem depurados e mantidos..

Vale também aqui a consideração feita durante a análise do HIPO, quanto a visualização de estruturas semelhantes.

c) Qualidade

Sem dúvida alguma o uso desta metodologia leva a projetos claros com uma documentação compreensível e legível, atributos de um programa com boa qualidade.

Em complemento, Composite Design estabelece critérios que permitem avaliar a coesão interna e a independência dos módulos, condições que facilitam a manutenção dos programas.

d) Praticabilidade

Embora sua filosofia seja fácil de ser absorvida e envolva uma simbologia simples, algumas dificuldades podem surgir quando de sua aplicação. Estas dificuldades aparecem na definição do Diagrama de Estruturas a partir do Diagrama de Fluxo de Dados e também na análise da coesão e acoplamento dos módulos, devido as regras existentes serem bastantes vagas e sujeitas a ápreciações subjetivas.

e) Generalidade e Sistemas de Suporte

Valem aqui as mesmas considerações feitas para o HIPO.

6 - CONCLUSÕES E CONSIDERAÇÕES FINAIS

Um fato que fica evidenciado da análise das metodologias aqui apresentadas é que elas são respostas a problemas que foram surgindo ao longo da evolução das aplicações em processamento de dados. Assim sendo, antes de significarem soluções individualizadas, elas podem ser visualizadas como resultados de um processo em contínuo aprimoramento.

Numa primeira instância, a dificuldade estava na maneira de como poderia ser comunicado o fluxo de processamento e as operações executadas por um programa escrito em Linguagem de Máquina ou mesmo em Assembler. A solução apresentada foi o Fluxograma onde símbolos interligados por setas serviam como forma de representação dos intrincados códigos e sequências do programa.

Contudo, problemas mais complexos, que correspondessem a processamentos com vários pontos de decisão, continuavam sendo difíceis de serem analisados e comunicados através desta técnica. A alternativa proposta foi o uso de Tabelas de Decisão.

O aumento quantitativo e da complexidade das aplicações, aliados a necessidade de melhor qualidade dos programas, passaram a requerer uma organização tanto da estrutura do programa quanto do seu processo de desenvolvimento. Como resposta a esta situação foi estabelecido que os programas deveriam ser desenvolvidos através de refinamentos sucessivos de sua estrutura e que esta deveria ser formulada a partir de somente três estruturas básicas. Estes dois conceitos estabeleceram uma filosofia de programação que se denominou Programação Estruturada.

Concretizando esta filosofia de programação, encontra-se atualmente um conjunto de metodologias do qual destacamos Nassi-Shneiderman Charts, Método de Jackson, LCP, HIPO Composite Design.

Destas, as mais simples são Nassi-Shneiderman Charts e HIPO. A primeira trata apenas da estruturação do fluxo de processamento, enquanto a segunda enfatiza o processo de desenvolvimento por refinamentos sucessivos. Ressaltamos o quanto estas duas técnicas são complementares, o que justifica a união aventada em 5.4.3.

Já o método de Jackson, LCP e Composite Design incorporam os conceitos da Programação Estruturada e acrescentam mais um que é o dos dados serem a base da estrutura do programa. As duas primeiras metodologias defendem o princípio de que a estrutura do programa deve ser derivada da estrutura dos seus dados, enquanto a terceira defende que ela deve ser deduzida do fluxo e transformações ocorridas com os dados.

Na figura 6.1 procuramos representar a evolução das metodologias de programação. O núcleo mais interno corresponde a preocupação com a documentação do fluxo/condições de processamento. O segundo nível caracteriza a preocupação com a estruturação do fluxo de processamento e com a aplicação do conceito refinamentos sucessivos, sem perda do aspecto documentação. No último nível tem-se a incorporação do conceito de derivação da estrutura do programa a partir dos dados por ele manipulados.

Uma segunda conclusão que se pode tirar das discussões realizadas ao longo deste trabalho é a não existência, no momento, de uma metodologia completa que possa ser aplicada de forma eficaz em toda e qualquer situação. Ainda que uma metodologia possa ser genérica, como o fluxograma o é, ela apresenta certas deficiências que desaconselham o seu uso.

Nos quadros apresentados ao final deste capítulo procuramos resumir as considerações tecidas para cada metodologia.

Não existindo ainda uma metodologia perfeita, que atenda to

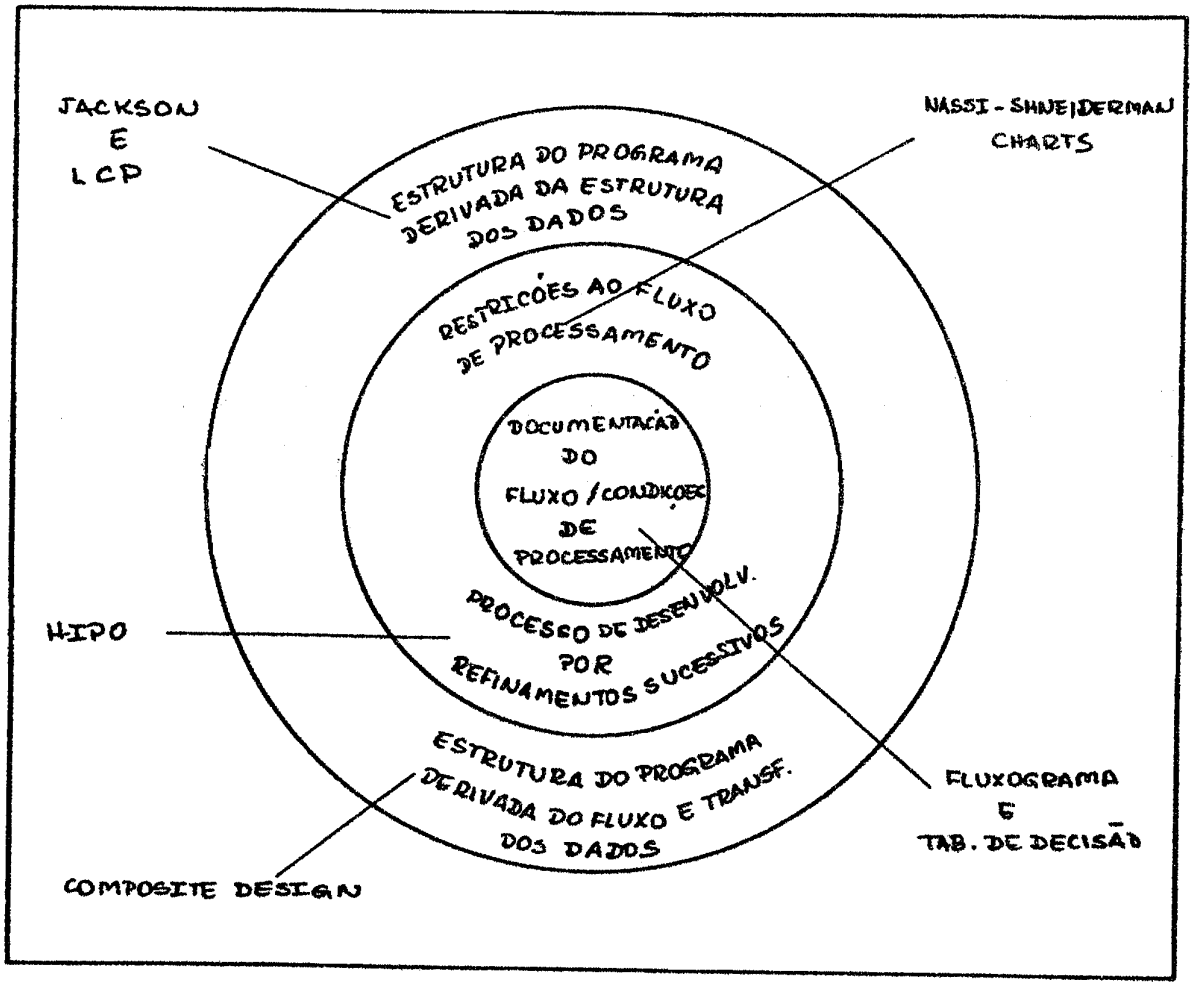


Figura 6.1 - Evolução das Metodologias de Programação

dos os requisitos necessários, se faz necessário a seleção de duas ou mais metodologias, a serem utilizadas em conjuntos ou de forma individualizada, dependendo dos propósitos que se tem em mente.

Observando as características identificadas para cada uma das metodologias concluímos que elas podem ser utilizadas da seguinte maneira:

- Quando o objetivo é o registro do projeto, e o programa final auto-documentado, seria suficiente a utilização de LCP ou Método de Jackson, e "Composite Design". A primeira para suportar o desenvolvimento de aplicações cujos dados têm estruturas hierárquicas e o processamento a ser realizado é condicionado por esta estrutura, tais como as de natureza comercial. Já "Composi

te Design" ficaria reservado para as aplicações onde os algoritmos de processamento não têm nada a ver com a estrutura dos dados, como são as aplicações científicas.

- Se o objetivo é o registro do projeto, e também uma boa documentação externa do programa então seria indicado, além das metodologias anteriores, o uso do HIPO como ferramenta de documentação.

A utilização das Tabelas de Decisão e Nassi-Shneiderman Charts deve se resumir a casos específicos, como auxílio a análise ou documentação de certas situações. O uso dos Fluxogramas não tem sentido nos dias de hoje, a não ser para a esquematização geral de sistemas.

Embora não se tenha alcançado a metodologia ideal, a nosso ver, bastam as existentes para a dimistificação da atividade de programação que, até hoje, ainda é tratada como sendo uma arte. Naturalmente, alguma criatividade se faz ainda necessária na elaboração de um programa, contudo as metodologias apresentadas sistematizam o processo de desenvolvimento de tal maneira que se pode afirmar que a elaboração de programas está no nível de desenvolvimento de um projeto de engenharia.

Não queremos dizer com isto que os problemas na área de desenvolvimento de programas tenham terminados, mas apenas estão bastante minimizados. No momento atual os principais problemas são decorrentes da dificuldade de definição dos reais requisitos do sistema, o que leva a programas perfeitos tecnicamente mas errados quanto as reais necessidades do usuário. Por outro lado, mudanças nos requisitos do problema continuarão a ocorrer e, as alterações a serem processadas, provocarão a degradação dos programas.

A falta de ferramentas adequadas não se restringe à defini

ção das necessidades dos usuários, mas também atinge a parte de testes e prova de correção de programas. Ainda que o uso das metodologias aqui apresentadas proporcionem programas de boa qualidade, sempre existem possibilidade de erro e, atualmente, não se tem recursos para a deteção e eliminação de todos os erros de um programa. Tanto a prova de correção de programas quanto a elaboração de massas de teste constituem uma arte.

Por fim, gostariamos de salientar que a experiência tem mostrado que a simples implementação de uma metodologia de programação não é condição suficiente para que melhores resultados sejam alcançados nesta área. Como qualquer atividade, o desenvolvimento de programas deve ser planejado, acompanhado e controlado a fim de que sejam cumpridos prazos estabelecidos e obtenha-se produtos finais de boa qualidade.

Para este fim, algumas propostas de forma de trabalho têm sido apresentadas e dentre estas destacamos "Chief Programmer Team" e "Structured Walksthru" {20}.

Segue-se os quadros contendo resumo das considerações realizadas ao longo do trabalho.

PARÂMETROS / METODOLOGIA	REGISTRO DE SOLUÇÕES	COMUNICAÇÃO
FLUXOGRAMA	Implicitamente não traz consigo qualquer orientação para à dissecação do problema	Prejudicada pelo excesso de detalhes
TABELAS DE DECISÃO	Não traz implicitamente qualquer orientação para a dissecação do problema	Permite uma boa visualização rápida das condições/ações do processo.
NASSI-SHNEIDERMAN CHARTS	Embora não traga consigo orientação para a dissecção do problema, sua linguagem favorece o proc. refinamentos sucessivos	Torna mais evidente a solução encontrada para o problema.
MÉTODO DE JACKSON	Propõe uma abordagem para o problema e documenta as diversas fases do desenvolvimento. Favorece acompanhamento dos trabalhos	Notação mais própria para pessoal de processamento de dados.
LCP	Propõe processo de desenvolvimento bastante semelhante ao de Jackson, porém mais sistematizado.	As soluções intermediárias são mais independentes do processamento de dados.
HIPO	Propõe desenv./documentação "TOP-DOWN", sem contudo orientar o processo de análise do problema. Também favorece controle dos trabalhos.	Pode ficar prejudicada pelo excesso de detalhes
COMPOSITE DESIGN	Propõe desenv./documentação "TOP-DOWN", suportado por conjunto de regras para derivação e análise de módulos.	Muito boa comunicação das soluções intermediárias e final.

PARÂMETROS / METODOLOGIA	PRODUTIVIDADE	QUALIDADE DO PROGRAMA
FLUXOGRAMA	Não tem qualquer impacto na produtividade.	Totalmente dependente da experiência do programador.
TABELAS DE DESIÇÃO	Restringe mais o processo de desenvolvimento do programa.	Ainda que evidencie a modularização do programa, o produto final é dependente do programador.
NASSI-SHNEIDERMAN CHARTS	Leva a estruturação do programa, o que facilita sua depuração e manutenção.	Induz a projetos mais claros e estruturas finais mais compreensíveis e legíveis.
MÉTODO DE JACKSON	Além de levar a prog. estruturados, sistematiza o trabalho do programador. Também favorece a visualização de Est. semelhantes.	Projeto claro e programa final compreensível e legível. Estrutura do programa igual a estrutura do problema.
LCP	Maior do que a alcançada com a utilização do método de Jackson devido o alto grau de sistematização do trabalho imposto por LCP	Semelhante a alcançada pela aplicação do método de Jackson.
HIPO	Organiza o desenvolv. de programas. Contudo não necessariamente estes serão fáveis de depuração e manutenção.	Projeto claro, e programa final compreensível e legível. Vale observar que os módulos definidos não necessariamente são os adequados
COMPOSITE DESIGN	Estabelece um padrão de trabalho para o programador, e a solução alcançada será mais fácil de ser depuradas e mantida	Projeto claro, e programa final compreensível e legível. Os módulos definidos devem ser os mais adequados para o problema modulado

PARÂMETROS / METODOLOGIA	PRATICABILIDADE	GENERALIDADE	SIST. DE SUPORTE
FLUXOGRAMA	Fácil de aprender, porém sua aplicação torna-se difícil em função do nível de detalhes	Pode ser utilizada para documentar programas de qualquer natureza.	Não existe
TABELAS DE DECISÃO	Fácil de aprender e de usar.	Mais eficaz em problemas com grande número de ponto de decisão	Sim
NASSI-CHNEIDERMAN CHARTS	Fácil de aprender quanto ao uso, vale ressaltar que a elaboração dos diagramas pode ser trabalhosa.	Pode ser utilizada em qualquer classe de problema.	Não
MÉTODO DE JACKSON	As soluções apresentadas para a derivação est. do programa o são feitas de modo muito "empírico".	Notação genérica. Contudo o esquema de desenv. é orientado para aplicações de natureza comercial.	Não
LCP	Aprendizagem e uso facilitado pelo conjunto de regras que orientam cada etapa do processo do desenvolv.	Orientada para problemas de natureza comercial. Notação não tão genérica quanto a proposta por Jackson	Não
HIPO	Fácil de aprender. Quanto ao uso vale ressaltar que a elaboração dos diagramas pode ser trabalhosa.	Voltado para problemas cujas funções a serem executadas não são condicionadas pela estrutura dos dados	Não
COMPOSITE DESIGN	Fácil de aprender, porém muitas das orientações dadas são bastante subjetivas	Mesma classe de problemas para o qual o HIPO é indicado.	Não

REFERÊNCIAS BIBLIOGRÁFICAS

01 - American National Standards Institute Inc.; "Flowchart Symbols and their Use in Information Processing"; Nova York, American Standards Institute, pg. 20, 1970.

02 - Backer, F. T. e Mills, H. D.; "Chief Programmer Teams"; Illinois, Datamation, 19, 12, pg. 58-61, Dez 1973.

03 - Bohm, C. and Jacopine, G.; "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules"; Nova York, Communication of ACM, 9, 5, pg. 366-371, 1966.

04 - Brooks Jr., Frederic P.; The Mythical Man-Month: Essays on Software Engineering; Massachusetts, Addison-Wesley Publishing Company, 195 pg., 1975.

05 - Chapin, Ned; "New Format for Flowcharts"; Nova York, Software Practice and Experience, 4, 4, pg. 341-357, Out-Dez 1974.

06 - Chapin, Ned; "Flowcharting With the ANSI Standard: A Tutorial"; Nova York, Computer Surveys, 2, 2, pg. 119-143, Junho 1970.

07 - Dijkstra, E. W.; "Programming Considered as a Human Activity", em "Proceedings of the IFIP Congress"; pg. 213-217, 1965.

08 - Dijkstra, E. W.; "GO TO Statement Considered Harmful"; Nova York, Communication of ACM, 11, 3, pg. 147-148, Mar 1968.

09 - Dijkstra, E. W.; "A Constructive Approche to the Problem of Program Correctness", BIT, 8 (3), pg. 174-186, 1968.

10 - Enos, J. C. e Van Tilburg, R. L.; "Software Design"; em Jensen, W. Randall e Tonies, Charles C. "Software Engineering", Nova Jersey, Prentice-Hall Inc., pg. 64-220, 1979.

11 - Fergus, R. M.; "Decision Tables-What, Why and How", Michigan, em "Proceedings, College and University Machine Records Conference", University of Michigan, pg. 1-20, 1969.

12 - Hartman, W. "et alii"; Management Information Systems Handbook, Holanda, Mc Graw-Hill Book Company, 2a. edição, 1972.

13 - HIPO: A Design Aid and Documentation Technique, Nova York, IBM Corporation, 130 pg., 1974.

14 - Hopkins, M. E.; "A Case for the GO TO"; em "Proceedings of ACM Conference", Nova York, ACM, pg. 791-797, Agosto 1972.

15 - Jackson, M. A., Principles of Program Design; Londres, Academic Press, 229 pg., 1975.

16 - Knuth, D. E.; "Structured Programming with GO TO Statements"; Nova York, Computer Surveys, 6, 4, pg. 261-301, Dez 1974.

17 - Levy, Leon; "CALL une instruction nuisible"; Paris, Zéro. Un. Informatique, Nº 118, pg. 29-35, Março 1975.

18 - Lucas, H. C.; "The ANSI Flowcharting Standard", em Aurbach Information Management Serie (4-03-02), Pennsauken, Auerbach Publisher Inc.; 18 pg., 1977.

19 - Lucena, Carlos J. P. e Staa, Ardnt Von; Projeto de Programas: Aplicações de Programação Estruturada em Cobol, Rio de Janeiro, SERPRO, 119 pg., 1978.

20 - Meyers, Glenford J.; Software Reliability: Principles & Practices, Nova York, John Wiley & Sons, 360 pg., 1976.

21 - Meyers, Glenford J.; Reliable Software Through Composite Design, Nova York, Petrocelli/Charter, 159 pg., 1975.

22 - Nassi, I. e Shneiderman, B.; "Flowchart Techniques for Structured Programming"; SIGPLAN Notices, pg. 12-26, Agosto 1973.

23 - Neuhold, E. J.; "Software Development and Software Engineering", Notas de Aula, PUC-RJ, 1979.

24 - Orr, Kenneth T.; Structured Systems Development, Nova York, Yourdon Press, pg. 170, 1977.

25 - Rigo, J. T. e Rudikoff, Joel R.; "HIPO: Structured System Design Documentation", em Auerbach Information Management Series (4-01-05), Pennsauken, Auerbach Publisher Inc., 20 pg., 1975.

26 - Stay, J. F. "HIPO and integrated program design", IBM System

Journal, 6, 2, pg. 143-154, 1976.

27 - Swan G. H.; Top-Down Structured Design Techniques; Nova York, Petrocelli Books Inc.; 140 pg., 1978.

28 - Warnier, J. D.; Les Procédures de Traitement e leurs Données, Paris, Les Editions D'Organisation, 153 pg., 1975.

29 - Warnier, J. D.; Entraînement à la Programmation, tome 1: Construction des Programmes, Paris, Les Editions D'Organisation, 303 pg., 1974.

30 - Warnier, J. D.; Entraînement à la Programmation, tome 2: Exploitation des Données, Paris, Les Editions D'Organisation, 274 pg., 1974.

31 - Wirth, N.; "Program Development by Stepwise Refinement.", Nova York, Communication of the ACM, 14, 4, pg. 221-227, 1971.

32 - Wirth, N.; "On the Composition of Well-Structured Programs", Nova York, Computing Surveys, 6, 4, pg. 247-259, Dez 1974.

33 - Yourdon, E. e Constantine, L. L.; Structured Design, Nova Jersey, Prentice Hall Inc., 473 pg., 1979.