Programação Estruturada

1. Programação Estruturada

1.1 Definições

Não existe uma definição universalmente aceita para a programação estruturada; ao contrário, existem várias escolas de pensamento que a conceituam.

No sentido mais restrito, o conceito de programação estruturada diz respeito à forma do programa e do processo de codificação. É um conjunto de convenções que o programador pode seguir para produzir o código estruturado. As regras de codificação impõem limitações sobre o uso das estruturas básicas de controle, estruturas de composição modular e documentação. Com este sentido, a programação estruturada enfoca os seguintes tópicos:

- Programação **sem GO TO** (eliminação completa ou parcial do comando GO TO)
- Programação com apenas três estruturas básicas de controle:
  seqüência, seleção e **iteração**
- Forma de um programa estruturado
- Aplicação de convenções de codificação estruturada a uma linguagem de programação específica

Em um sentido mais amplo, a programação estruturada é dirigida tanto aos métodos de programação como à forma do programa. Implica seguir uma metodologia estruturada para organizar o projeto e a implantação do programa. Os tópicos envolvidos incluem:

- Programação modular
- Refinamento gradual (stepwise refinement)
- Níveis de abstração
- Programação top-down e bottom-up

1.2 Objetivos

O principal objetivo da programação estruturada é produzir um programa de alta qualidade a baixo custo, proporcionando uma disciplina de programação para conseguir:

- Melhorar a confiabilidade do programa
- Aumentar a legibilidade do programa
- Minimizar a complexidade do programa
- Simplificar a manutenção do programa
- Aumentar a produtividade do programador
- Estabelecer uma metodologia disciplinada de programação

1.3 Propriedades

As características mais notáveis de um programa estruturado são sua forma hierarquizada e o conjunto reduzido de suas estruturas básicas de controle. Além dessas, também são consideradas propriedades igualmente importantes, a estrutura de controle

padronizada, os requisitos de programação e as convenções de estilo. A seguir, um resumo das propriedades de um programa estruturado.

Propriedade 1 - o programa é dividido em um conjunto de módulos dispostos em uma hierarquia que define suas relações lógicas e de tempo de execução.

Propriedade 2 - o fluxo de execução de módulo para módulo restringe-se a um esquema simples, fácil de entender, no qual o controle deve ser passado para o módulo em seu único ponto de entrada, sair do módulo de seu único ponto de saída e retornar sempre ao módulo que o chamou.

Propriedade 3 - a construção do módulo é padronizada de acordo com as regras tradicionais de modularização, e as estruturas básicas de controle são limitadas a seqüência, seleção, iteração e desvio.

Propriedade 4 - é requerida a documentação relativa ao código fonte, para descrever a função do programa como um todo e descrever, para cada módulo, sua função, sua estrutura de dados e sua relação com outros módulos do programa.

1.4 Estruturas Básicas de Controle

As três estruturas básicas de controle para a construção de programas estruturados :

Seqüência - é usada para controlar a execução do programa, os comandos são executados na mesma ordem em que aparecem no código fonte.

As estruturas de seleção, iteração e desvio são usadas para alterar o fluxo de execução do programa de sua ordem seqüencial normal.

Seleção - é usada para testar uma condição e, então, dependendo de ser o teste verdadeiro ou falso, um dos dois conjuntos alternativos de instruções é executado.

Iteração - é usada para executar um conjunto de instruções em um número inteiro de vezes - isto é, para construir um laço (loop). Deve-se observar que existem duas formas básicas para a estrutura de iteração: DO UNTIL e WHILE.

1.5 Formato Padronizado do Módulo

Um programa estruturado é um programa modular. Porém, em um programa estruturado o módulo, além de seguir as regras básicas de modularização, tem sua forma mais limitada. A imposição de regras de formato rigorosas para a construção de um módulo estruturado é um outro meio de simplificar a estrutura do programa. Elas permitem que o programador concentre sua atenção no conteúdo e não na forma do programa.

As regras para a construção de um módulo estruturado são as seguintes:

- Um módulo tem um único ponto de entrada e um único ponto de saída.
- Um módulo é uma estrutura completa - executa uma única tarefa lógica no programa e é limitado pelo seu ponto de entrada e seu ponto de saída.
- As estruturas básicas de controle legais restringem-se à seqüência, seleção (admitindo-se extensões), iteração e desvio para o ponto de saída do módulo.

1.6 - Estrutura de Controle do Programa

Existe uma *relação de chamada* entre dois módulos, A e B, em um programa estruturado, se o módulo A requisita o módulo B para executar sua função.

Do ponto de vista da estrutura do controle do programa, isto significa que o módulo A chama o módulo B. Quando o módulo A chama o módulo B, o controle do programa é passado do A para o B, o código do B é então executado e quando este completa sua execução, passa o controle de volta ao módulo A.

O módulo A (módulo chamador) é denominado ***pai*** do módulo B (módulo chamado) e o módulo B é denominado ***filho*** do módulo A. Em um programa estruturado, a relação de chamada entre dois módulos só funciona em uma direção; isto é, se o módulo A chama o módulo B, o módulo B não pode chamar o módulo A, direta ou indiretamente, através de um filho ou neto. Além disso, um módulo não pode chamar a si próprio. Estas relações não são permitidas porque não podem ser implantadas facilmente em muitas linguagens de programação e porque tendem a aumentar a complexidade do programa.

1.6.1 - Regras para o Fluxo de Controle do Programa em um Programa Estruturado

- Existe um e somente um módulo raiz, em um programa estruturado.
- A execução do programa deve começar com o módulo raiz.
- Somente um pai pode chamar um filho. Um filho não pode chamar seu pai, nem um módulo pode chamar a si próprio.
- O controle do programa deve ser passado para um módulo em seu ponto de entrada e deve sair do módulo em seu ponto de saída.
- O controle retorna sempre ao módulo chamador, quando o módulo chamado completa a execução.

OBSERVAÇÃO: estas regras não se aplicam a linguagem C. Em C é permitido que o módulo filho chame ao módulo pai ou que um módulo chame a si mesmo.

O menor programa estruturado consiste apenas em um módulo raiz. Outros programas podem conter um módulo raiz e um ou mais filhos. Com exceção do módulo raiz, todos os módulos devem ter pelo menos um pai. Se um módulo tem múltiplos pais é chamado de *módulo comum*. Qualquer módulo, incluindo o módulo raiz, pode ter zero, ou múltiplos filhos. Um módulo sem filhos é chamado de *folha*. Todo ramo em uma hierarquia modular deve terminar com um módulo folha.

1.7 - Documentação

A documentação é uma parte essencial de um programa bem-estruturado. Ela é dirigida a três níveis de compreensão do programa:

- Global
- Organização do programa
- Instrução do programa

Cada nível sucessivo fornece uma visão mais detalhada do programa e todos os três níveis são necessários para a sua compreensão e manutenção.

2. Programação Modular

2.1 - Dividir-para-Conquistar

A programação estruturada tem sua origem na programação modular. Grande parte da filosofia estruturada é construída sobre a filosofia de modularização de dividir-para-conquistar. A programação modular utiliza essa filosofia para resolver o problema da complexidade do programa. Quando um programa é dividido em partes independentes, fáceis de serem entendidas, sua complexidade pode ser grandemente reduzida. A filosofia estruturada utiliza dividir-para-conquistar e amplia a filosofia de modularização acrescentando os importantes conceitos de organização hierárquica e níveis de abstração para controlar as relações intermodulares.

A *programação modular* pode ser definida como a organização de um programa em unidades independentes, chamadas módulos, cujo comportamento é controlado por um conjunto de regras. Suas metas são as seguintes:

- Decompor um programa em partes independentes
- Dividir um problema complexo em problemas menores e mais simples
- Verificar a correção de um módulo de programa, independentemente de sua utilização como uma unidade em um sistema maior

A modularização pode ser aplicada em diferentes níveis. Pode ser usada para separar um problema em sistemas, um sistema em programas e um programa em módulos.

2.2 – Vantagens da Programação Modular

- Os programas modulares são mais fáceis de entender, porque a análise do programa pode ser feita através da análise dos módulos, um de cada vez.
- O teste do programa é mais simples.
- Os erros do programa são mais fáceis de serem isolados e corrigidos.
- As mudanças no programa podem ser limitadas a apenas alguns módulos, em vez de percutirem pela maior parte do código do programa.
- Pode-se aumentar mais facilmente a eficiência do programa.
- Os módulos do programa podem ser reutilizados como blocos de construção de outros programas.
- O tempo de desenvolvimento do programa pode ser diminuído, porque diferentes módulos podem ser designados a diferentes programadores, que podem trabalhar com maior ou menor independência.

2.3 - Esquemas de Modularização

Dividir um programa em módulos pode ser um meio muito eficaz de controlar a complexidade. O problema que surge é quanto à melhor forma de dividir o programa. Mesmo para um programa simples, existem inúmeras maneiras de dividi-lo em um conjunto de módulos estruturados. A maneira escolhida pode afetar significativamente a complexidade do programa. Por exemplo, definir módulos grandes demais ou pequenos demais aumenta a complexidade, os custos e a probabilidade de erros no programa.

A forma de dividir um programa em módulos é chamada de seu esquema de modularização. Um grande objetivo da programação modular é definir um bom esquema de modularização (aquele que reflita rigorosamente os componentes do problema a ser programado, minimize a complexidade e possa ser fácil e eficazmente implantado).

2.3.1 – Tamanho do Módulo

Para ajudar o projetista de programa na escolha de um bom esquema de modularização, têm sido desenvolvidas diretrizes relativas ao tamanho do módulo. A IBM, por exemplo, recomenda que se definam módulos com, no máximo, 50 linhas de código de programa ou a uma página de listagem do programa fonte (uma tela de vídeo), pois permite que o leitor veja, e portanto compreenda, todo o código de uma só vez.

Um número semelhante surge de Weinberg, cujos trabalhos demonstram que módulos que possuem mais de 30 instruções de programação são difíceis de entender.

De uma maneira geral, o tamanho do módulo deve estar compreendido entre 10 e 100 instruções. Módulos com mais de 100 instruções são mais dispendiosos em relação a testes e manutenções, enquanto módulos com menos de 10 instruções podem dividir o programa em muitas partes, afetando desfavoravelmente a eficácia do programa.

3. Programação por Refinamento Sucessivo
(STEPWISE REFINEMENT)

3.1 - Definição

É a maneira de desenvolver um programa executando uma seqüência de etapas de refinamento. O processo começa com a definição dos dados e tarefas procedurais básicas, para resolver o problema de programação. Esta definição inicial é feita em nível bem alto e geral. O processo pára quando todas as tarefas do programa são expressas em uma forma que é diretamente traduzível na(s) linguagem(ns) de programação.

A cada etapa de refinamento, uma ou várias tarefas são decompostas em subtarefas mais detalhadas. A definição de uma subtarefa é freqüentemente acompanhada por um refinamento da definição dos dados necessários para as tarefas de interface. Por esta razão o refinamento de tarefa e o refinamento de dados são executados em paralelo.

3.2 - Características do Refinamento

- O processo avança em etapas explícitas.
- A cada etapa, um conjunto de tarefas e estruturas de dados é apresentado. O programa, tal como foi desenvolvido até este ponto, é definido em termos deste conjunto.
- O conjunto de tarefas e estruturas de dados usado em uma etapa define, com maior detalhe, o conjunto que foi apresentado na etapa de refinamento anterior.
- A notação usada para definir as tarefas e estruturas de dados é geral, porém vai tornando-se mais detalhada e mais semelhante à linguagem de programação real, à medida que o processo de refinamento completa-se.
- As etapas de refinamento são o produto das decisões de projeto do programa, envolvendo uma definição adicional das partes funcionais do programa e suas estruturas de dados.

Quão bem as decisões do projeto de programa forem tomadas, tão bem o programa será estruturado em módulos fáceis de serem entendidos e de serem modificados.

4. TOP-DOWN e BOTTOM-UP

4.1 - Metodologias de Programação Estruturada

Os programas podem ser desenvolvidos de cima para baixo ou de baixo para cima. Tanto um como outro são adaptações da programação por refinamento sucessivo.

Quando um sistema é construído de baixo para cima, o projetista primeiro cria os componentes, faz cada um deles funcionar bem e, então agrupa os componentes.

Quando um projetista trabalha de cima para baixo, primeiramente cria a estrutura global, definindo os componentes. À medida que o projeto vai progredindo, ele completa os detalhes, construindo os componentes de níveis inferiores.

Em projetos complexos, é comumente requerida uma combinação de projetos top-down e bottom-up.

4.2 - Programação Top-Down

É um método organizado de projetar, codificar e testar um programa em etapas progressivas. O método é baseado na programação por refinamento sucessivo.

A programação top-down produz um programa modular, estruturado hierarquicamente. O módulo de nível mais alto na hierarquia representa a função global do programa. Os módulos dos níveis inferiores representam subfunções que definem as tarefas do programa com maiores detalhes.

Na programação top-down, a ordem das etapas de projeto, codificação e teste é diferente daquela do desenvolvimento tradicional de programa. No desenvolvimento tradicional, primeiramente o programa inteiro é projetado, depois é codificado e, finalmente, testado. As três etapas são executadas separada e seqüencialmente, chamada de *abordagem por fases* ou *abordagem tudo-em-um*. No desenvolvimento top-down, ao contrário, as três etapas são executadas em paralelo, chamada de *abordagem incremental* ou *abordagem passo a passo*. Primeiramente, um módulo é projetado, codificado e testado. Depois, um outro módulo é projetado, codificado e testado, e assim por diante, até terminarem todos os módulos do programa. Como parte da etapa de teste, cada módulo recentemente codificado é combinado e testado com todos os módulos concluídos até aquele ponto. O programa cresce gradualmente, um módulo por vez. A integração é realizada em etapas e não em grandes blocos no final do projeto, como acontece na abordagem por fases.

Na prática, a abordagem tradicional por fases não tem tido muito êxito. Tende a adiar a descoberta de problemas graves até uma fase bem avançada do projeto. Esta situação revela-se em estouros de cronogramas e orçamentos e em graves problemas de testes de integração. Como conseqüência, a fase de testes é a fase mais mal executada e a que mais foge do esquema planejado. Além disso, os problemas provavelmente continuarão na fase de manutenção, transformando o programa em um custoso pesadelo a ser mantido.

4.2.1 - Vantagens

A abordagem top-down pode evitar muitos destes problemas. Tem a vantagem de iniciar os testes mais cedo, no processo de desenvolvimento. Desta forma os problemas de integração podem ser descobertos e corrigidos antes e a um custo mais baixo.

Uma segunda vantagem é que existe um esqueleto do programa desde as primeiras etapas de desenvolvimento, fornecendo ao usuário uma versão parcial do programa bem antes de sua conclusão.

4.2.2 - Etapas da Programação Top-Down

O desenvolvimento top-down do programa segue a estrutura lógica do programa, iniciando-se com o módulo de nível mais alto e continuando, em etapas bem organizadas, até os módulos de nível mais baixo. As etapas são as seguintes:

Etapa 1 - começa com uma análise do problema e, então, propõe um plano de abordagem, definido como a estrutura geral do programa. Representa a solução como um esboço que identifica o processamento e os componentes de dados gerais do programa. Neste ponto, os conteúdos dos componentes não são definidos em detalhes. O esboço é obtido dividindo-se o problema em partes lógicas que refletem as tarefas funcionais que o programa deve executar para realizar seu objetivo (por exemplo, calcular o pagamento bruto, selecionar o formato da tela).

Etapa 2-n - em etapas, expande o esboço do programa chegando ao programa completo. O processo de expansão segue as regras do processo de refinamento. Os componentes do programa são refinados em subcomponentes (stubs – tocos), os quais são também refinados, e assim por diante. O processo terminará quando o programa puder ser escrito como um conjunto de componentes diretamente representáveis na linguagem de programação.

4.2.3 - Princípios Básicos da Programação Top-Dow

- As decisões de projeto devem ser feitas de tal forma que dividirão o problema em partes cujos componentes estarão relacionados a ele.
- Os detalhes de implantação não devem ser considerados até o processo de desenvolvimento estar avançado, mas em cada etapa devem ser exploradas alternativas.
- As etapas de refinamento devem ser simples e explícitas.

4.3 - Programação Bottom-Up versus Programação Top-Down

Embora a programação top-down seja geralmente considerada como o melhor método de desenvolver um programa estruturado, a programação *bottom-up* é, às vezes, uma alternativa viável. Ambos os métodos compartilham os mesmos objetivos:

- sistematizar o processo de programação
- fornecer uma estrutura para a resolução do problema
- produzir um programa modular

Ambas são compatíveis com a metodologia de programação estruturada. Contudo, abordam o desenvolvimento de programa de maneiras opostas. Na programação *top-down*, a técnica de construção usada no desenvolvimento de programa é a *decomposição* - o sentido do desenvolvimento é de uma solução abstrata e de alto nível para o ambiente de programação real.

Na programação *bottom-up*, o caminho de desenvolvimento seguido é oposto. O programa é construído pela combinação de componentes simples, atômicos, para formar componentes mais abstratos e de nível mais alto. Esta técnica de construção é a *concatenação*.

4.4 - Programação Bottom-Up

A programação bottom-up desenvolve o programa em etapas. Assim como em top-down, o processo pode ser dividido em duas partes. Primeiramente, é proposta uma solução inicial para o programa e depois, são desenvolvidos os componentes que tratam dos detalhes.

4.4.1 - Etapas da Programação Bottom-Up

Etapa 1 - começa com um esboço do programa proposto. Este esboço contém os componentes gerais do programa - componentes funcionais e de dados.

Etapa 2-n - em etapas explícitas, desenvolve o programa combinando os componentes de nível mais baixo para formar componentes de nível mais alto. Este processo é chamado de *concatenação*. Esse processo terminará quando for construído a partir de elementos disponíveis no ambiente de programação real, um conjunto de funções e estruturas de dados do programa, capazes de resolver o problema de programação. Acionadores de testes são usados para representar os módulos de nível mais alto que serão desenvolvidos em etapas posteriores.

4.4.2. - Características do processo de concatenação

- O processo prossegue em etapas explícitas.
- A cada etapa, é construída uma máquina. Os componentes desta máquina são formados a partir dos componentes da máquina desenvolvida durante a etapa de concatenação anterior.
- Cada etapa é o resultado de uma decisão de combinar alguns componentes da máquina anterior.

Na prática, o processo de concatenação, de maneira análoga ao processo de refinamento, não é um procedimento de um passo. Erros e omissões podem forçar a repetição de parte do processo ou de todo o processo.

5. Diagramas Estruturados

5.1 - Introdução

As técnicas de diagramação estruturada oferecem muitas vantagens. Combinam notações gráficas com narrativas, para aumentar a compreensão. Os gráficos são particularmente úteis porque tendem a ser menos ambíguos do que a descrição narrativa. Além disso os gráficos, porque tendem a ser mais resumidos, podem ser desenhados em um tempo bem menor do que poderia ser gasto para escrever um documento narrativo contendo a mesma quantidade de informação.

As técnicas de diagramação estruturada apoiam uma abordagem de desenvolvimento top-down e estruturado. Podem descrever um sistema ou programa com vários graus de detalhes, durante cada etapa do processo de decomposição funcional. Tornam claras as etapas e os resultados do processo de decomposição funcional, ao fornecerem um modo padronizado de descrever a lógica dos procedimentos e as estruturas de dados.

5.2 - Funções dos Diagramas Estruturados

- Contribuir para a clareza de pensamento
- Possibilitar uma comunicação precisa entre os membros da equipe de desenvolvimento
- Facilitar a definição de interfaces padrões entre módulos
- Impor uma boa estruturação
- Ajudar a depuração
- Ajudar a efetuar mudanças nos sistemas (manutenção)
- Acelerar o desenvolvimento (com diagramação apoiada por computador)
- Impor precisão nas especificações
- Integrar as ferramentas de administração de dados
- Fornecer condições aos usuários finais para esboçar suas necessidades com clareza
- Integrar a checagem automática com a geração de código

5.3 - Categorias de Diagramas Estruturados

Os diagramas são usados em várias áreas:

Modelo global da empresa - as funções e processos de uma empresa são decompostas hierarquicamente.

Análise de sistema global – o fluxo de dados global entre eventos e processos comerciais é desenhado.
Projeto de sistema - a relações entre os componentes de um sistema são desenhadas.
Arquitetura do programa - a arquitetura global de um programa ou de um conjunto de programas é desenhada, mostrando os módulos independentes.
Detalhe do programa - a lógica detalhada dentro de um módulo de programa é desenhada.
Modelos de dados - um modelo lógico global dos dados é desenhado. É importante aqui que se faça a distinção entre modelos de banco de dados e representação de arquivos.
Estruturas de dados - as estruturas dos bancos de dados ou dos arquivos são desenhadas.
Estruturas de diálogos - as possíveis seqüências das telas e interações em um diálogo homem-computador podem ser desenhadas.

5.4 - Classificação das Técnicas de Diagramação

Técnicas	Sist. Global	Arq. Programa	Lóg. Detalhada	Estrut. Arq.	Estrut. BD
DFD	X				
Decomp. Funcional	X				
Diagr. Estrutura	X	X			
Hipo	X	X	X		
Warnier-Orr	X	X	X	X	
Michael Jackson		X		X	
Fluxograma			X		
Pseudocódigo			X		
Nassi-Shneiderman			X		
Diagr. Ação	X	X	X	X	
Árvores de decisão	X		X		
Tabelas de decisão			X		
D. Estrut. Dados					X
Entidade-Rel. (E-R)					X
D. Naveg. Dados		X			X
Diagr. HOS	X	X	X	X	

6. Diagramas de NASSI-SHNEIDERMAN

6.1 - Introdução

Os fluxogramas, não são técnicas adequadas para descrever programas estruturados. Dão uma visão não-estruturada do mundo e tendem a gerar programas com GO TOs, em vez de programas hierarquicamente estruturados.

I. Nassi e B. Shneiderman resolveram substituir o fluxograma tradicional por um diagrama que apresenta uma visão hierárquica e estruturada da lógica do programa. (N. Chapin usa um tipo de diagrama semelhante, denominado por ele *diagrama Chapin*.)

Os diagramas de Nassi-Shneiderman (N-S) representam estruturas de programas que têm um ponto de entrada e um ponto de saída e são compostos pelas estruturas básicas de controle (seqüência, seleção e repetição). Enquanto é difícil mostrar o embutimento e a recursividade com o fluxograma tradicional, é fácil mostrá-1os com um diagrama N-S e é fácil, também, converter um diagrama N-S em código estruturado.

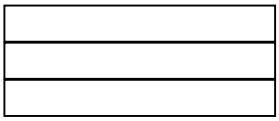
Um diagrama N-S consiste em um retângulo que representa a lógica do módulo do programa. Pretende-se que este retângulo seja desenhado em uma página. Portanto ele deve ter, no máximo, cerca de 15 a 20 subdivisões. Quando um diagrama N-S torna-se muito grande, as subfunções são separadas e desenhadas em um outro diagrama. Foi projetado para ser usado com métodos top-down e de refinamento gradual.

Para representar a lógica do processo, várias estruturas são embutidas dentro do retângulo. ***Não*** é permitida uma instrução de desvio.

6.2 - Estruturas Básicas de Controle

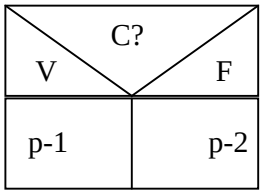
Cada estrutura básica de controle usada em programação estruturada pode ser representada por um símbolo do diagrama de N-S.

Seqüência - é mostrada por uma pilha vertical de retângulos de processo:



Por exemplo, a Figura 6.1 descreve a seqüência de quatro processos : CHECAR NOME E ENDEREÇO, CHECAR CEP NUMÉRICO, CHECAR TERMOS VÁLIDOS e CHECAR PAGAMENTO.

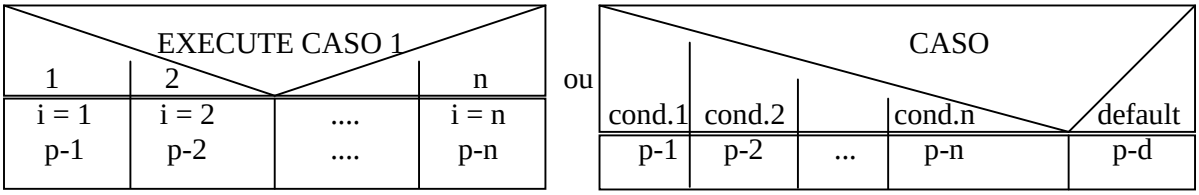
Seleção - apresenta-se a seleção (IF-THEN-ELSE) dividindo-se o retângulo de processo em 5 partes. A metade superior é dividida em três triângulos. O triângulo superior contém a condição a ser testada. Os triângulos inferiores indicam a parte *verdadeira* e a parte *falsa* do IF-THEN-ELSE. A metade inferior é dividida em retângulo de processo *verdadeiro* e retângulo de processo *falso*, p-1 e p-2, respectivamente.



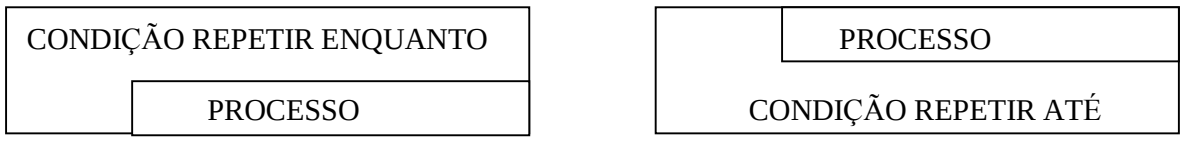
Por exemplo, na Figura 6.1, o último teste de condição no diagrama é a checagem do INDICADOR INVÁLIDO. Se o teste é verdadeiro, o processo ESCREVER MENSAGENS ERROS é executado. Se o teste é falso, nada é feito, pois o processo falso é nulo.

Observe que a estrutura de seleção pode ser embutida. Na Figura 2.1, o teste de condição para ERROS é embutido no teste de condição para RENOVAÇÃO.

A estrutura de condição pode ser ampliada para a estrutura de casos, na qual uma seleção é feita entre várias opções mutuamente exclusivas, como se segue:



Repetição - é indicada por uma estrutura REPETIR ENQUANTO ou uma estrutura REPETIR ATÉ.



do – while **do - until**

Observe que na estrutura *DO WHILE* a condição é testada em primeiro lugar; então, se a condição for verdadeira, o processo é executado. Na estrutura *DO UNTIL*, o processo é executado em primeiro lugar; depois a condição é testada.

Na Figura 6.1, a estrutura de repetição é usada para indicar que o processo de validação e processamento de uma assinatura é executado uma vez para cada assinatura no arquivo de transações.

6.3 - Exemplo de um diagrama N-S

VALIDAR ITEM ASS:

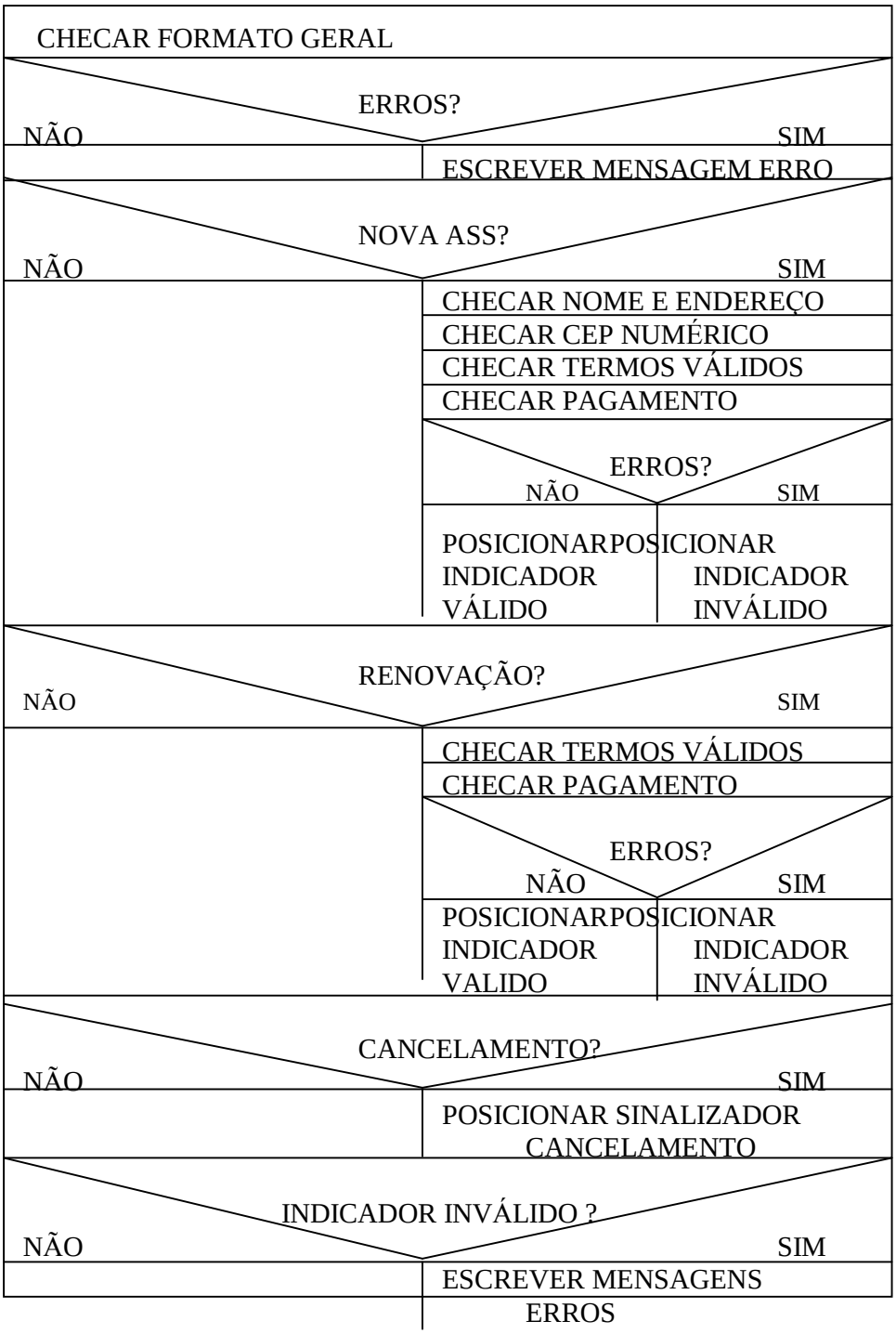


Figura 6.1 – Exemplo de um diagrama N-S para validar assinaturas

6.4 - Crítica ao Diagrama de Nassi-Shneiderman

O diagrama de Nassi-Shneiderman é uma técnica de diagramação usada fundamentalmente para o projeto detalhado de programa. É uma ferramenta fraca para mostrar a estrutura hierárquica de controle, de alto nível, para um programa. Embora ela possa mostrar os componentes procedurais básicos, não apresenta as interfaces ligando estes componentes.

É uma alternativa para os fluxogramas tradicionais de programas, diagramas HIPO e pseudocódigos. Entre estas técnicas, o diagrama N-S é o mais fácil de ser lido e o mais fácil de ser convertido em código de programa. Contudo, é apenas uma ferramenta para o projeto de procedimento e não pode ser usado para projetar estruturas de dados. Embora sua leitura seja fácil, o seu desenho nem sempre o é. O desenho de um diagrama N-S pode levar três ou quatro vezes mais tempo para ser executado do que seu pseudocódigo correspondente.

Uma outra grande falha do diagrama N-S é não ser orientado para banco de dados. Ele não se relaciona com um modelo de dados ou um dicionário de dados.

Os diagramas de ação apresentam todas as informações que estão no diagrama de Nassi-Shneiderman e são bem mais fáceis de serem usados com editores estruturados. Várias ferramentas CASE usam diagramas de ação; nenhuma usa os diagramas de Nassi-Shneiderman.

7. Diagrama de Fluxo de Dados

7.1 - Introdução

Um diagrama de fluxo de dados (DFD) apresenta os processos e o fluxo de dados entre eles. Em alto nível, é usado para mostrar eventos de negócios e as transações resultantes desses eventos. Em nível mais baixo, é usado para mostrar programas ou módulos de programas e o fluxo de dados entre estas rotinas.

Um DFD é usado como o primeiro passo em um projeto estruturado. Apresenta o fluxo de dados global em um sistema ou programa. É principalmente uma ferramenta de análise de sistemas, para desenhar os componentes procedurais básicos e os dados que passam entre eles.

O diagrama de fluxo de dados mostra como o dado flui através de um sistema lógico, mas **não** dá informação sobre controle ou seqüência.

7.2 - Componentes de um DFD

Um DFD é uma representação em rede de um sistema que mostra os processos e as interfaces de dados entre eles. O DFD é construído a partir de 4 componentes básicos:

- o fluxo de dados
- o processo
- o depósito de dados
- os pontos terminais

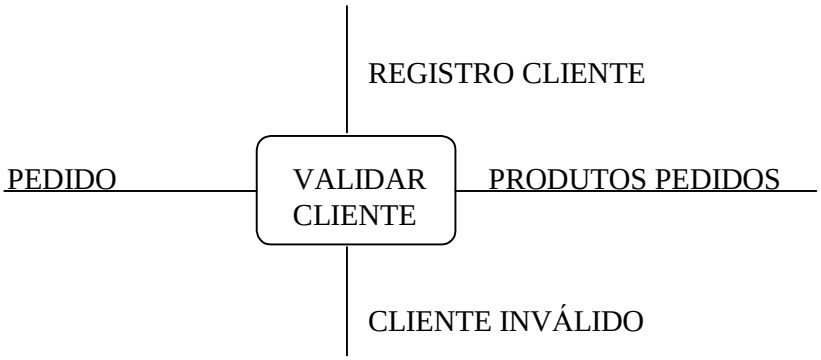
Fluxo de Dados – conduz o fluxo de informações através dos processos de um sistema. O sentido do fluxo de dados é indicado por uma seta. Na verdade, mostra como os processos são interligados. Os dados são identificados por nomes escritos ao lado de sua flecha correspondente; por exemplo:

PRODUTOS PEDIDOS

Processo – é um componente procedural do sistema. Opera sobre (ou transforma) os dados. Por exemplo, pode executar operações aritméticas ou lógicas com os dados, para produzir algum(ns) resultado(s). Cada processo é representado no DFD por um círculo ou um retângulo com os vértices arredondados. O nome do processo é escrito dentro do círculo. Deve ser usado um nome significativo, para definir a operação executada pelo processo, por exemplo:

VALIDAR CLIENTE

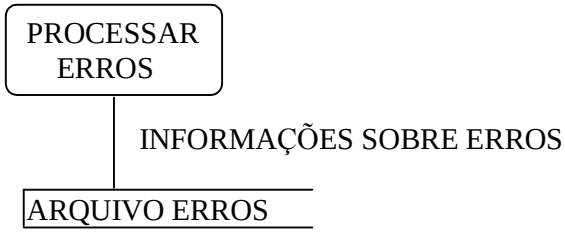
Nenhuma outra informação sobre o que faz o processo é mostrada no DFD. Normalmente, os dados entram e saem de cada processo. Geralmente, existem múltiplos fluxos de dados entrando e saindo de um processo; por exemplo:



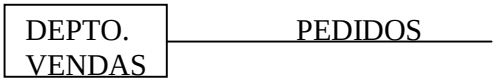
Depósito de Dados – representa um arquivo lógico. É desenhado no DFD como um par de linhas paralelas (às vezes, fechadas em um dos lados). O nome do depósito de dados é escrito entre as linhas; por exemplo:

Cada depósito de dados é ligado a um "retângulo" de processo por meio de um fluxo de dados. O sentido da seta do fluxo de dados mostra se os dados estão sendo lidos do depósito de dados para o processo ou produzidos pelo processo e então enviados para o depósito de dados.

No exemplo mostrado a seguir, as informações sobre erros são produzidas pelo PROCEDIMENTO ERROS e gravadas no ARQUIVO ERROS:



Ponto Terminal - mostra a origem dos dados usados pelo sistema e o último receptor de dados produzidos pelo sistema. A origem dos dados é chamada de *fonte* e o receptor dos dados é chamado de *destino*. Para representar um ponto terminal em um DFD, é usado um retângulo (como mostrado) ou um quadrado duplo. Na verdade, os pontos terminais situam-se fora do DFD.



7.3 – Nivelamento de um DFD

Um DFD é uma ferramenta para análise *top-down*. Pode-se usar DFDs para fornecer tanto uma visão em alto nível como também visões detalhadas de um sistema ou programa. O que acontece dentro de um dos retângulos do DFD pode ser mostrado com detalhes em um outro DFD.
A seguir um exemplo de um sistema de distribuição de vendas.

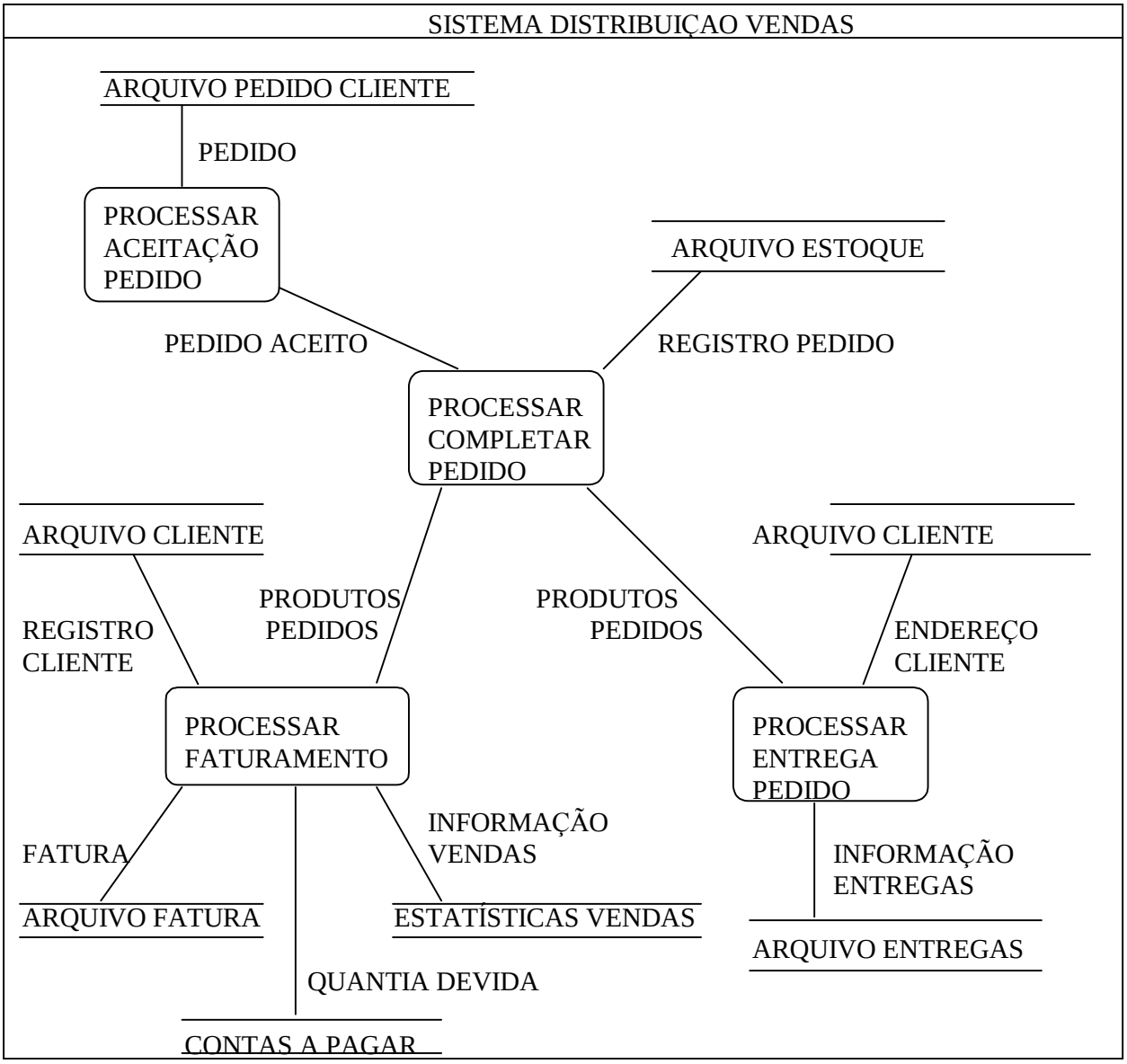


Figura 7.1 – DFD do sistema de distribuição de vendas.

A Figura acima mostra a visão em alto nível do processo aceitação de pedido. Ela nos indica que o processo aceitação de pedido é um dos quatro processos do sistema de distribuição de vendas. Esta visão, porém, não nos fornece nenhuma informação detalhada sobre o processo e o fluxo de dados necessários para registrar um pedido. Se necessitarmos de mais detalhes, devemos olhar dentro do retângulo do processo para ver quais subprocessos estão contidos no PROCESSO ACEITAÇÃO DE PEDIDO.

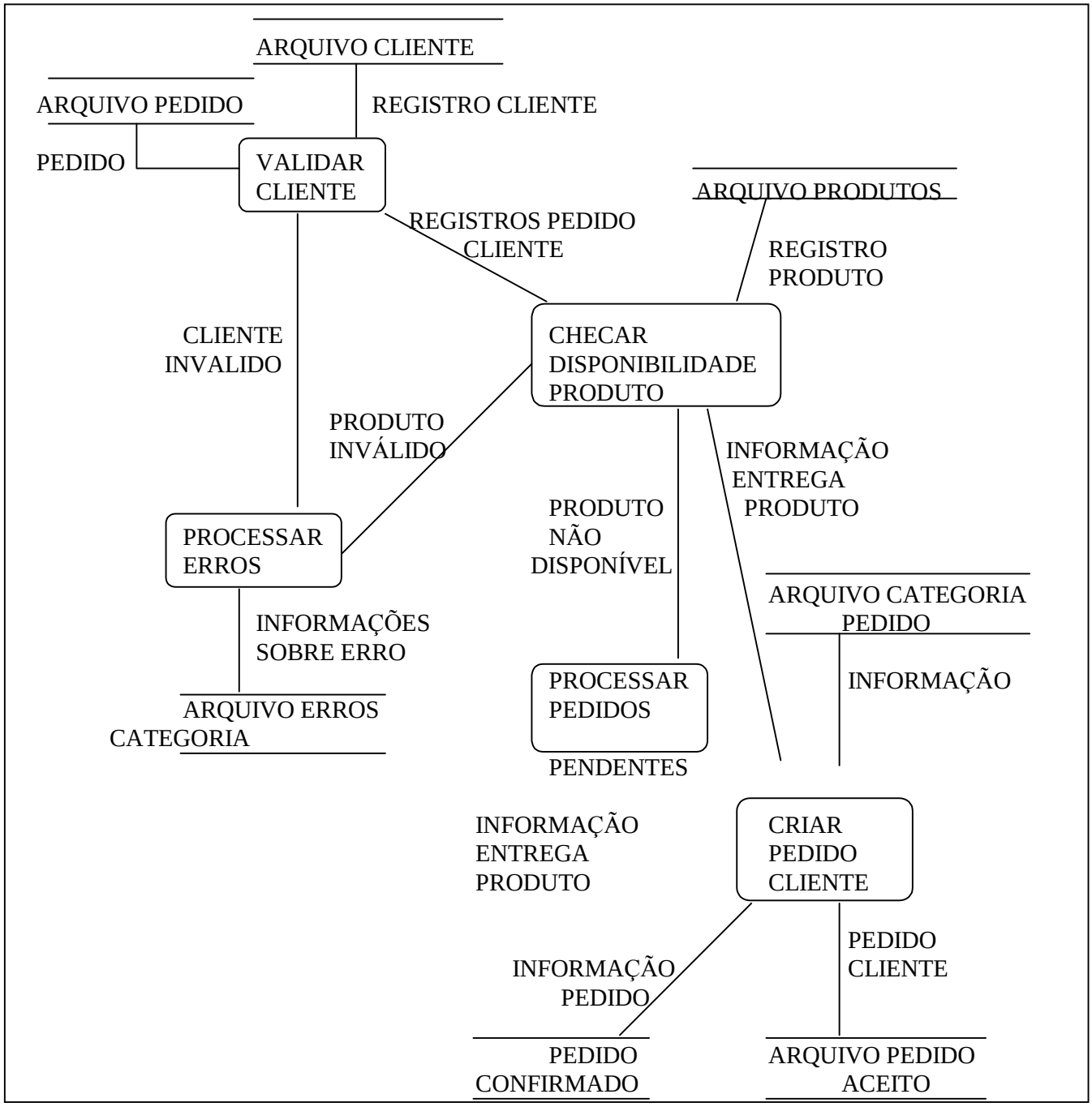


Figura 7.2 – DFD do processo ACEITAÇÃO DE PEDIDO.

Podemos continuar a expandir nossa visão do sistema, olhando dentro de cada processo, até onde quisermos. Quão detalhada deve ser nossa visão? Uma regra prática seria: continuar expandindo os retângulos de processos, para criar DFDs detalhados, até as operações executadas por cada processo poderem ser descritas em uma especificação de uma página.

A cada nível, o DFD deve conter menos que 12 retângulos de processos, preferivelmente, seis ou sete. DFDs maiores são um sinal de tentativa de mostrar muitos detalhes, e são difíceis de serem lidos.

Este processo de definir um sistema em uma maneira top-down é chamado de nivelamento de um DFD.

7.4 – Especificação de Processo e Dicionário de Dados

Quando um DFD é desenvolvido durante a análise estruturada, são desenvolvidos, também,. especificações de processo e dicionário de dados, para darem informações adicionais sobre o sistema.

Uma **especificação de processo** é criada para cada retângulo do DFD de nível mias baixo. Define como o fluxo de dados entra e sai do processo e quais operações são executadas com os dados. Uma especificação de processo é descrita com outras técnicas. A seguir, uma versão em português estruturado do que acontece no retângulo rotulado VALIDAR CLIENTE, da Figura 2.

```
VALIDAR CLIENTE
    PARA CADA PEDIDO REALIZADO;
            ENCONTRAR REGISTRO CLIENTE CORRESPONDENTE;
            SE CLIENTE ENCONTRADO
                  SE CRÉDITO CLIENTE BOA SITUAÇÃO
                      REUNIR TODOS PEDIDOS PRODUTO PARA AQUELE CLIENTE;
                      CRIAR REGISTRO PEDIDO_CLIENTE
                 CASO CONTRÁRIO (CLIENTE MÁ SITUAÇÃO)
                       ESCREVER MENSAGEM ERRO CLIENTE INVÁLIDO
                 FIMSE.
           CASO CONTRÁRIO (CLIENTE NOVO A SER ACRESCENTADO NO
      ARQUIVO)
                  CRIAR REGISTRO CLIENTE NOVO;
                  REUNIR TODOS PEDIDOS PRODUTO PARAAQUELE CLIENTE;
                  CRIAR REGISTRO PEDIDO CLIENTE
           FIMSE.
```

O **dicionário de dados** contém definições de todos os dados do DFD. Pode incluir, também, informação física sobre os dados, tais como dispositivos de armazenamento, e métodos de acesso aos dados. O termo dicionário de dados, usado em conjunção com a definição de um DFD, *não* é o mesmo quando usado em conjunção com sistemas de gestão de banco de dados. A seguir, é apresentado um exemplo

ARQUIVO CLIENTE = [registros de cliente]

Os colchetes [ ] indicam que neste exemplo
ARQUIVO CLIENTE é *repetições* de registros de cliente.

registro-cliente = nome-cliente + endereço-cliente + informação-pagamento + pedidos-pendentes + tipo-cli

O sinal + indica que neste exemplo um registro de cliente é um item de dado composto formado por uma *seqüência* dos itens de dados relacionados acima.

tipo-cli = empr | indivíduo

A barra | indica que tipo-cli é uma empresa ou um indivíduo.

7.5 - Críticas ao DFD

Os diagramas de fluxo de dados são uma ferramenta muito valiosa para traçar fluxos de documentos e fluxos de dados em sistemas complexos. Algumas organizações estendem seu uso para a estruturação interna de programas. Este uso pode ser questionado, pois existem técnicas melhores.

DFDs são usados erroneamente em muitos casos. Especificações grandes e complexas são criadas com a ajuda de tais diagramas, mas a tarefa de fazer a checagem cruzada de todas as entradas e saídas de dados muitas vezes, não é executada adequadamente.

Muitos conjuntos de especificações, a partir das quais os programadores codificam, têm diagramas de fluxo de dados nos quais as entradas e saídas são inconsistentes. Após o código ter sido escrito, é muito dispendioso acertar a confusão resultante.

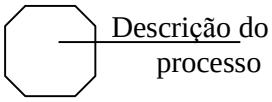
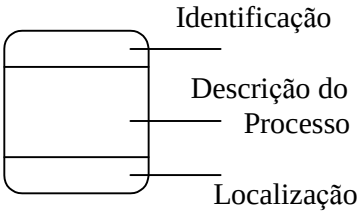
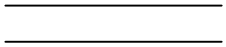
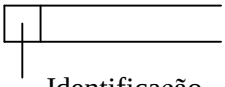
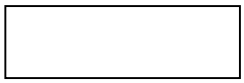
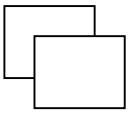
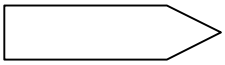
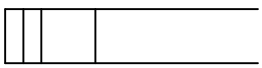
Os autores de diagramas de fluxo de dados declaram que quando desenham os diagramas iniciais, diagramas em alto nível, não podem ainda conhecer os detalhes que surgirão quando estiverem desenhando os diagramas subseqüentes. É verdade que os diagramas iniciais são esboços, ainda sem detalhes e precisão. Contudo, quando o detalhe é definido, ele deve ser refletido nos níveis mais altos, de forma que estes fiquem coerentes. Em outras palavras, o projeto do detalhe deve nos induzir a mudar os demais níveis do diagrama.

Com uma ferramenta computadorizada (CASE), isto pode ser feito automaticamente. As entradas e saídas detalhadas das camadas de níveis mais baixos podem ser refletidas nas camadas de níveis mais altos. Porém, nem todas as ferramentas computadorizadas fazem isto. Elas são simplesmente apoios para o desenho, sem ligação com as desejadas checagens de integridade.

8. Notação de Gane e Sarson

8.1 – Introdução

Gane e Sarson adotaram, para diagramas de fluxo de dados, convenções de diagramação ligeiramente diferentes daquelas tornadas populares por Yourdon e De Marco. Sob alguns aspectos, a notação de Gane e Sarson é melhor. A figura a seguir apresenta um resumo das duas.

	Yourdon, De Marco e outros	**Gane e Sarson**
Fluxo de dados		
Processo que transforma os dados		
Depósito de dados		
Origem ou destino de dados externos		
Fluxo de materiais		
Depósito de dados repetidos em um diagrama		
Origem ou destino de dados externos repetidos em um diagrama