AVALIAÇÃO DE MÉTODOS PARA O DESENVOLVIMENTO DE SOFTWARE

Emilia Maria Holanda Crispim

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

Aprovada por:

Prof. Ana Regina C. da Rocha, D.Sc.
Presidente da Banca

Prof. Jano Moreira de Souza, Ph.D.
COPPE/UFRJ

Prof. Regina Célia de Souza Pereira, D.Sc.
UFF

Rio de Janeiro, RJ - Brasil

Maio de 1991

CRISPIM, Emilia Maria Holanda

Avaliação de Métodos para o Desenvolvimento de Software [Rio de Janeiro] 1991.

xiv, 320 p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1991)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Métodos 2. Avaliação 3. Desenvolvimento de Software I. COPPE/UFRJ II. Título (série)

A José e Idilva,

A Ivete.

AGRADECIMENTOS

A Prof. Ana Regina pela orientação, acompanhamento e sugestões valiosas ao longo do desenvolvimento do trabalho.

Aos meus pais José e Idilva e a minha tia-mãe Ivete pelo carinho, incentivo e pela presença marcante e indispensável em todos os momentos de minha vida.

Ao Sérgio pelo carinho, paciência, compreensão e apoio significativos em todos os momentos que estivemos separados.

A Teresa pela colaboração nos estudos e elaboração da pesquisa.

A minhas irmãs Ana, Teresa e Clícia pela amizade e incentivo constante.

A Cláudia, Riverson e Wamberto pela amizade e companheirismo, indispensáveis, nos momentos mais difíceis.

A Universidade Federal do Ceará e ao Núcleo de Processamento de Dados pela oportunidade de realizar o mestrado.

A Deus por tudo.

RESUMO DA TESE APRESENTADA A COPPE - UFRJ COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS.

Avaliação de Métodos para o Desenvolvimento de Software

Emilia Maria Holanda Crispim

MAIO DE 1991

Orientador: Prof. Ana Regina C. da Rocha

Programa: Engenharia de Sistemas e Computação

Este trabalho tem por objetivo estabelecer critérios para a avaliação de métodos de desenvolvimento de software, fornecendo subsídios para a construção da base de conhecimentos da estação de trabalho do Projeto TABA.

Os critérios para avaliação foram identificados a partir de um estudo sobre características de métodos. Foram avaliados 23 métodos para o desenvolvimento de software usando os critérios definidos.

ABSTRACT OF THE THESIS PRESENTED TO COPPE/UFRJ AS PART OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE.

Evaluation of Software Development Methods

Emilia Maria Holanda Crispim

MAY 1991

Thesis Supervisor: Prof. Ana Regina C. da Rocha

Department: Systems and Computer Engineering

This work aims to establish criteria to evaluate software development methods, providing subsidies to the construction of the knowledge base of TABA Station.

The criteria was identified from a study of characteristics of methodos. 23 methods were evaluated using such criteria.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO I

INTRODUÇÃO

I.1 O USO DE MÉTODOS PARA DESENVOLVER SOFTWARE

O desafio principal nas primeiras décadas da computação era a produção de hardware que reduzisse o custo do processamento e do armazenamento de dados. Hoje esse quadro mudou. A preocupação está voltada principalmente para questões relativas à produção de software (PRESSMAN, 1988).

Muitos fatores influenciaram a mudança desse quadro. A evolução do hardware e a redução do seu custo de manufatura podem ser citados como pontos decisivos para a disseminação dos computadores nos mais diferentes setores da sociedade. Isso foi de grande importância para alterações ocorridas nas atividades relacionadas ao desenvolvimento de software.

A aplicação dos computadores nas mais diversas áreas propiciou uma demanda, em marcha acelerada, de uma grande quantidade de produtos de software, com características cada vez mais complexas. Ficava difícil manter atualizados os inúmeros programas existentes e suprir a necessidade crescente por mais produtos.

Tornou-se imprescindível a aplicação de uma abordagem mais sistemática para a construção de software. Os primeiros métodos eram uma tentativa de mudar o enfoque do

desenvolvimento de software, buscando, assim, solucionar problemas gerados por uma prática de produção desordenada.

Muitos trabalhos encontrados na literatura mostram que vários métodos e ferramentas têm sido propostos com o objetivo de aprimorar e apoiar o processo de desenvolvimento de software. Esse processo tem evoluído consideravelmente nos últimos anos. Entretanto, apesar de todo o esforço empregado e de todo o aprimoramento, algumas questões relativas à produção de software ainda não foram solucionadas.

A primeira questão refere-se ao fato de que a tarefa de construir produtos de software é passível de vários problemas, tais como atrasos nos cronogramas, custos estimados ultrapassados, o não atendimento da demanda requerida, dificuldade de comunicação entre desenvolvedores e usuários, ou ainda produtos que não satisfazem às necessidades que os levaram a ser construídos.

Outra questão relaciona-se com a escolha dos métodos a serem utilizados na construção de software. Como foi dito anteriormente, existem muitos métodos, mas não há um método que possa ser usado de forma universal em qualquer situação.

Na grande maioria, os métodos são adequados para solucionar problemas com características específicas. Assim, surge a necessidade de escolher quais aqueles mais indicados para o desenvolvimento de um determinado tipo de software.

Segundo SOUZA (1989) uma solução para a busca da produtividade e da qualidade do produto encontra-se na melhoria da qualidade do próprio processo de desenvolvimento. A qualidade a nível do produto não se atinge de uma forma espontânea, e sim através de uma série de técnicas e procedimentos que devem ser aplicados durante a sua construção.

Hoje, a tentativa de solucionar a questão da produtividade e da qualidade é vista de forma promissora através do uso de ambientes de desenvolvimento de software.

Podería-se pensar na construção de ambientes utilizáveis no desenvolvimento de qualquer tipo de produto. Isso, contudo, não é viável, pois uma série de fatores influem na adequação do ambiente de desenvolvimento a um determinado projeto (DEMARCO, 1984). As características de uma aplicação específica devem determinar as características de seu ambiente de desenvolvimento (ROCHA, 1989). Os métodos e ferramentas a serem usados na construção também devem ser escolhidos de forma adequada às particularidades do problema a ser resolvido.

I.2 DEFINIÇÃO DE AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

São encontradas na literatura várias definições de ambientes de desenvolvimento de software. CHARETTE (1986), por exemplo, define um ambiente como sendo

"processo, métodos e automação necessários
para produzir um sistema de software";

já em (DART, 1987) ambiente é definido como

"uma coleção de ferramentas de hardware e software usadas por um desenvolvedor para construir sistemas de software";

(LUCENA, 1987), por sua vez, define ambiente de desenvolvimento de software como uma

"coleção coordenada de ferramentas de software organizadas para dar apoio a algum enfoque para o desenvolvimento de software em conformidade com algum modelo de processo de software".

Ambiente de desenvolvimento de software é considerado neste trabalho como (ROCHA, 1988)

"um conjunto integrado de métodos, instrumentos e ferramentas que suportam o desenvolvimento de um software nas diferentes etapas de seu ciclo de vida".

Segundo a definição adotada, um ambiente de desenvolvimento de software é composto de :

- um **ciclo de vida**, que define as fases do processo de desenvolvimento de software, especificando todas as atividades que devem ser realizadas em cada uma das fases desse processo, e,
- **métodos**, que são conjuntos de diretivas para a seleção e aplicação sistemática de técnicas e instrumentos, de forma a organizar o pensamento e o

trabalho do usuário ao longo do processo de desenvolvimento de software (ROCHA, 1987a).

Os métodos fornecem técnicas, instrumentos e ferramentas que viabilizam a realização das atividades definidas no ciclo de vida.

As técnicas são conjuntos de princípios para a execução de uma tarefa específica do processo de desenvolvimento de software. As técnicas podem ser classificadas em (ROCHA, 1987a):

- técnicas construtivas, guiam o processo de construção de software, oferecendo meios para que ele possa ser construído de forma disciplinada;

- técnicas normativas, estabelecem normas e atributos de qualidade que guiem a construção de um software;

- técnicas gerenciais, oferecem meios para planejar e controlar o processo de desenvolvimento de um software.

Os instrumentos tornam possível a utilização de um método. Tomando como exemplo o método "Structured Systems Analysis" (GANE, 1979), observa-se que este método possui os seguintes instrumentos: linguagem para diagrama de fluxo de dados, linguagem para dicionário de dados, linguagens para descrição da lógica dos processos e linguagem para diagrama de acesso imediato aos dados.

Os instrumentos estão divididos em (ROCHA, 1987a):

- **instrumentos para geração**, dão apoio às técnicas construtivas. Os intrumentos para geração podem ser de dois tipos:

  . **instrumentos cognitivos**, visam aumentar a capacidade intelectual dos desenvolvedores, sendo adequados para usuários inexperientes, e,

  . **instrumentos notacionais**, são linguagens que possibilitam a construção de um software;

- **instrumentos para avaliação da qualidade**, dão apoio às técnicas normativas e tornam possível o uso de métodos para avaliação da qualidade, e,

- **instrumentos para apoio à gerência**, viabilizam a utilização de técnicas gerenciais, tornando possível o planejamento e o controle do processo de desenvolvimento de um produto.

Para que esses instrumentos sejam utilizados de uma forma mais produtiva, é necessário que sejam automatizados. Torna-se, portanto, imprescindível a existência de **ferramentas** automatizadas de apoio ao método.

I.3 A ESTAÇÃO DE TRABALHO TABA

O Projeto TABA, em desenvolvimento na COPPE/UFRJ, visa construir uma estação de trabalho configurável para a geração de ambientes adequados a diferentes domínios de aplicação. A Estação TABA poderá ser utilizada com quatro objetivos (ROCHA, 1989):

i. auxiliar o engenheiro de software na especificação e no instanciamento do ambiente de desenvolvimento de software mais adequado à construção de um produto específico;

ii. auxiliar o engenheiro de software a implementar as ferramentas necessárias ao ambiente definido no item i);

iii. permitir aos desenvolvedores do produto usar a estação através do ambiente especificado em i) e produzido em ii);

iv. executar o software.

A Estação TABA é composta de quatro ambientes estruturados de acordo com as funções básicas acima citadas. A figura I.3.1 mostra o Diagrama de Fluxo de Dados da Estação TABA contendo esses ambientes, os quais são descritos a seguir (ROCHA, 1989):

i. **Ambiente Configurador de Ambientes (Meta-Ambiente)** - o meta-ambiente possui duas funções principais: especificar e tornar operacional o ambiente mais adequado ao desenvolvimento de uma determinada aplicação;

ii. **Ambiente Gerador de Ferramentas** - este ambiente auxilia na construção de ferramentas sugeridas pelo sistema especificador de ambientes;

iii. **Ambiente de Desenvolvimento de Software** - a partir dos resultados obtidos em i) e ii), obtém-se o ambiente

específico que deverá ser utilizado no desenvolvimento de uma determinada aplicação;

iv. **Ambiente de Teste e/ou Execução de Aplicação** - este ambiente permite executar na própria estação o software desenvolvido em iii).

O trabalho desenvolvido nesta tese está inserido no contexto do Projeto TABA, em especial na construção do Especificador de Ambientes do Meta-Ambiente da Estação TABA.

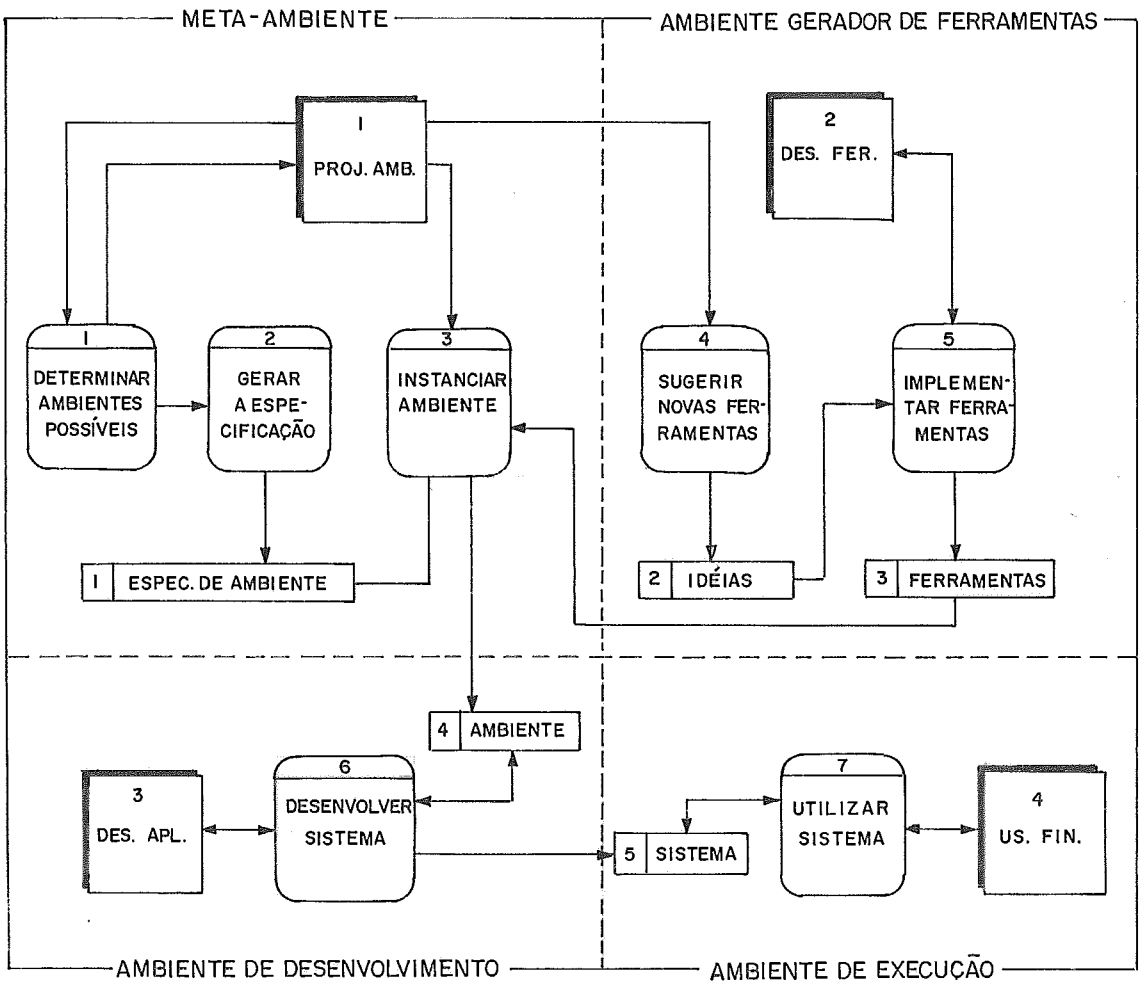


Figura I .3.1 : Diagrama de Fluxo de Dados da Estação Taba
Fonte : (ROCHA , 1989)

I.4 O ESPECIFICADOR DE AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE DA ESTAÇÃO TABA

O Especificador de Ambientes é um sistema que auxilia ao engenheiro de software, fornecendo um acesso conveniente aos dados e conhecimentos obtidos. Esse sistema ajudará na seleção dos componentes de ambientes de desenvolvimento de software que sejam mais adequados à construção de um determinado produto (AGUIAR, 1989).

O Especificador de Ambientes TABA possibilitará, portanto, aos engenheiros de software, um acesso conveniente aos dados e conhecimentos para definir ambientes adequados ao desenvolvimento de aplicações específicas (ROCHA, 1989).

A especificação do ambiente de desenvolvimento de software, na Estação TABA, pode ser realizada diretamente pelo usuário, através de uma linguagem para definição de ambientes, ou através de um sistema especialista de apoio à decisão, que auxilia na tomada de decisão sobre os componentes desejáveis ao ambiente de desenvolvimento de software, tendo em conta as características do produto a ser desenvolvido (ROCHA, 1989).

O Especificador de Ambientes da Estação TABA possui uma base de conhecimento contendo (AGUIAR, 1989):

- conhecimento sobre modelos de ciclo de vida;
- conhecimento sobre domínios de aplicação;
- conhecimento sobre aspectos gerenciais, fatores humanos e produtividade, e,

. conhecimento sobre métodos para o desenvolvimento de software.

É necessário que a base de conhecimento tenha informações sobre características de métodos que auxiliem a selecionar quais os métodos que mais se adaptam ao desenvolvimento de uma aplicação específica.

I.5 OBJETIVO DA TESE

A definição de um ambiente de desenvolvimento de software depende do estado da arte em métodos e do conhecimento de especialistas no uso dos mesmos (ROCHA, 1988). É necessário agrupar informações que auxiliem na tomada de decisões sobre quais métodos melhor se aplicam na configuração de um ambiente para determinado tipo de problema.

Este trabalho tem por objetivo avaliar uma série de métodos propostos na literatura, fornecendo subsídios para a construção da base de conhecimento da Estação TABA, no que se refere a métodos de desenvolvimento de software. Para tanto, foi feito um estudo sobre propostas que auxiliem na escolha dos métodos que melhor se adaptem ao desenvolvimento de uma aplicação específica. A partir desse estudo foram identificadas características a serem utilizadas para avaliar os métodos selecionados.

I.6 ORGANIZAÇÃO DA TESE

O presente trabalho está organizado da seguinte forma:

No Capítulo II é feita a revisão da literatura sobre características de métodos, além de serem apresentadas propostas para avaliação e seleção de métodos.

No Capítulo III é definida a estrutura do Método para Avaliação, onde são descritos os objetivos, fatores e critérios que compõem a abordagem de avaliação de métodos proposta no trabalho.

O Capítulo IV contém a revisão da literatura sobre métodos para desenvolvimento de software. é apresentada uma descrição dos métodos a serem avaliados, contendo as principais características e os procedimentos para utilização desses métodos.

No Capítulo V é feita a avaliação dos métodos para desenvolvimento de software descritos no Capítulo IV, segundo os critérios definidos no Capítulo III. A partir dessa avaliação são elaboradas algumas comparações entre os métodos avaliados.

Finalmente, no Capítulo VI são apresentadas as conclusões do trabalho.

CAPÍTULO II

CARACTERÍSTICAS DE MÉTODOS

II.1 INTRODUÇÃO

Existe hoje, na literatura, uma grande quantidade de propostas para construção de software. Com tantas propostas existentes, fica difícil escolher o método ideal para desenvolver uma aplicação específica. É necessário a utilização de critérios para avaliar a qualidade de métodos e facilitar o processo de escolha.

Este capítulo apresenta uma revisão da literatura sobre propostas para avaliação e seleção de métodos, bem como a descrição das características utilizadas nessas propostas.

II.2 PROPOSTA DE WASSERMAN E FREEMAN

O trabalho desenvolvido em (WASSERMAN, 1983), apresenta uma relação de características genéricas e um conjunto mais detalhado de critérios para a avaliação de métodos.

Em primeiro lugar os autores apresentam características consideradas desejáveis para métodos que podem ser utilizadas na seleção de tecnologias para uma determinada organização. Um método deve atender aos seguintes requisitos:

- cobrir todo o processo de desenvolvimento de um determinado produto de software e facilitar as transições entre as diversas fases do ciclo de vida;

- facilitar a comunicação entre as pessoas envolvidas na construção do produto, durante todos os estágios do desenvolvimento. O método deve determinar a forma de representação dos modelos gerados e a documentação a ser utilizada como instrumento de comunicação entre desenvolvedores e usuários;

- oferecer suporte para a análise e o entendimento do problema. O método deve prover técnicas para auxiliar na elaboração de modelos que representem a estrutura com uma solução do problema;

- suportar as abordagens "top-down" e "bottom-up", bem como uma combinação delas, para a especificação de software;

- prover uma estratégia para alcançar um nível de qualidade desejável para o produto, suportando a validação e a verificação das diversas representações do software, geradas ao longo do processo de desenvolvimento;

- facilitar a inclusão, na especificação de requisitos, de restrições de projeto, implementação e desempenho. O método deve ainda verificar se essas restrições foram obedecidas;

- oferecer suporte para a gerência do processo de desenvolvimento;

- oferecer suporte automatizado. As ferramentas devem ser, de preferência, integradas através de uma base de dados comum e oferecer recursos para a gerência do uso do próprio método;

- suportar a alteração ou a incorporação de novos requisitos, possibilitando que a evolução do produto seja visível e controlável em todos os estágios do desenvolvimento;

- identificar e controlar as partes componentes do produto, bem como as versões geradas;

- poder ser utilizado na construção de vários tipos diferentes de software;

- poder ser ensinado e transferido para outras pessoas ou equipes da organização, e,

- poder sofrer modificações, permitindo a inclusão de novos conceitos técnicos e gerenciais, bem como a criação de novas ferramentas.

Além dos requisitos definidos acima, também é especificado, no trabalho apresentado nesta seção, um conjunto mais detalhado de critérios para a avaliação de métodos. O grau de aceitabilidade para cada critério deve ser determinado de acordo com a estrutura da organização e o tipo de produto a ser desenvolvido.

Esses critérios estão divididos em quatro categorias principais, contendo características técnicas, de uso, de gerência e econômicas. Os critérios pertencentes a cada uma dessas categorias são definidos a seguir:

- **Características técnicas** - estão relacionadas com o suporte oferecido pelo método a vários conceitos técnicos. Essas características são avaliadas através dos seguintes critérios:

  . **hierarquia funcional** - é a capacidade do método representar a decomposição de uma função em um grupo de funções mais detalhadas;

  . **hierarquia de dados**: - é a capacidade do método representar a decomposição de um conjunto de dados em um determinadado nível, em elementos de dados interrelacionados em um nível com um maior detalhamento;

  . **interfaces** - é a capacidade do método identificar as fronteiras entre os processos e conjuntos de dados;

  . **fluxo de controle** - é a capacidade do método representar a seqüência na qual os processos ocorrem;

  . **fluxo de dados** - é a capacidade que o método tem de representar o fluxo de informações entre os processos e os dados armazenados no sistema;

  . **abstração de dados** - é a capacidade do método prover tipos de dados embutidos e funções independentes de implementação para utilizar esses tipos de dados;

. **abstração procedimental** – é a capacidade do método abstrair um conjunto de operações com um único nome e chamar esse procedimento sem se preocupar com os detalhes de implementação;

. **paralelismo** – é a capacidade do método expressar uma situação em que dois ou mais processos seqüenciais executam concorrentemente;

. **segurança** – é a característica do método controlar a segurança dos processos a serem realizados;

. **confiabilidade** – é a capacidade do método verificar a ausência de erros;

. **corretude** – é a capacidade do método garantir a fidelidade do produto às especificações funcionais.

– **Características de uso** – estão relacionadas à aplicação do método em situações de desenvolvimento. Essas características avaliam os seguintes critérios:

. **facilidade de entendimento** – é a capacidade do método permitir o entendimento das especificações geradas por pessoas que não são especialistas no método;

. **facilidade de transferência** – é o critério que avalia o grau de facilidade de se ensinar e aprender a utilizar o método;

. **reusabilidade** – é a capacidade do método de poder reutilizar especificações de projeto, código ou outros produtos em um novo projeto;

. **suporte computacional** — é o critério que avalia o apoio automatizado existente para o método;

. **abrangência no ciclo de vida** — é o critério que avalia as fases do ciclo de vida nas quais o método pode ser utilizado;

. **abrangência de tarefas** — é o critério que avalia os tipos de tarefas para os quais o método pode ser utilizado;

. **extensão de uso** — é o critério que avalia o grau de utilização do método;

. **facilidade de transição entre as fases** — é o critério que avalia a facilidade de transição de fases oferecida pelo método;

. **validação** — é a capacidade do método determinar a correção do sistema;

. **facilidade de modificar o produto gerado** — é o critério que avalia o método quanto ao esforço requerido para modificar o produto gerado caso haja alterações.

— **Características de gerência** — estão relacionadas à capacidade do método aperfeiçoar a gerência das atividades de desenvolvimento de software. Nessas características são avaliados os seguintes critérios:

. **capacidade de gerência** — é o critério que avalia os recursos de gerência oferecidos pelo método;

. **equipe de trabalho** – é o critério que avalia a capacidade do método auxiliar ao trabalho em equipe;

. **produtos gerados** – é o critério que avalia os produtos e documentos gerados através da aplicação do método;

. **gerência de configuração** – é a capacidade do método controlar as versões do produto;

. **finalização de tarefas** – é a capacidade do método definir critérios de finalização para cada etapa de seu funcionamento;

. **escalonamento** – é a capacidade do método suportar o escalonamento de atividades;

. **estimativa de custo** – é a capacidade do método suportar estimativas de custo.

- **Características econômicas** – estão relacionadas à capacidade do método produzir benefícios econômicos referentes à qualidade do software e à produtividade da organização. Essas características avaliam os seguintes critérios:

. **benefícios locais** – é o critério que identifica as vantagens esperadas em se usar um determinado método;

. **benefícios no ciclo de vida** – é o critério que identifica as vantagens de se utilizar um método em uma fase específica do ciclo de vida;

. **custo de aquisição** — é o critério que avalia o custo de obtenção, implantação e treinamento de um método e suas ferramentas;

. **custo de uso** — é o critério que avalia o custo relativo ao uso de um método e suas ferramentas;

. **custo de gerência** — é o critério que avalia o custo de gerência do método e de suas ferramentas.

II.3 PROPOSTA DE ROCHA

Segundo (ROCHA, 1983), a qualidade das especificações de um produto está diretamente relacionada à qualidade dos métodos e das linguagens utilizados para construir essas especificações. A proposta apresentada nesta seção define um conjunto de fatores e critérios para a avaliação da qualidade de métodos e linguagens para especificação de produtos de software.

Os fatores de qualidade relacionados com a avaliação de métodos para o desenvolvimento de software são definidos em seguida:

— **Aplicabilidade** — é a característica do método poder ser utilizado durante as várias fases do ciclo de vida de um software e em diferentes classes de projetos. O fator **aplicabilidade** está relacionado aos seguintes critérios:

. **abrangência** — é a característica do método poder ser utilizado durante todas as fases do ciclo de vida de um software;

. **generalidade** – é a característica do método poder ser utilizado em diferentes classes de projetos;

. **seletividade** – é a característica do método fornecer regras claras e precisas indicando quando as suas técnicas devem ser utilizadas.

– **Possuir suporte** – é a característica do método fornecer suporte, automatizado ou não, para realizar as diversas atividades envolvidas na produção de especificações. O fator **possuir suporte** está relacionado aos seguintes critérios:

. **suporte para evolução** – é o atributo que avalia o auxílio fornecido pelo método para a produção de especificações. A avaliação desse critério é feita levando-se em consideração os seguintes sub-critérios:

– suporte para criação – é a característica do método gerar uma especificação a partir de intuições, idéias, experiências e conhecimento das necessidades e expectativas dos usuários;

– suporte para abstração – é a característica do método separar o essencial de detalhes supérfluos;

– suporte para explosão – é o atributo que avalia a capacidade do método detalhar uma especificação já existente;

– suporte para elaboração – é a característica do método aprimorar uma especificação já existente incluindo mais detalhes;

- suporte para translação - é a característica do método converter uma especificação, escrita em uma determinada linguagem, em uma outra especificação utilizando-se uma outra linguagem.

. **suporte para verificação** - é a característica do método fornecer auxílio para avaliar o grau de imprecisão presente na especificação devido a incorreções na evolução. Para avaliar o suporte para verificação oferecido pelo método, deve-se considerar os seguintes sub-critérios:

  - suporte para comparação - é o atributo do método que examina a consistência entre as descrições dos diversos aspectos existentes na especificação;

  - suporte para identificação de corrupção - é o atributo do método que examina a consistência entre uma especificação e as especificações anteriores;

  - suporte para controle de qualidade - é o atributo do método que examina a especificação em relação aos fatores de qualidade da especificação.

. **suporte para validação** - é o atributo do método que fornece auxílio para avaliar a especificação quanto à satisfação das necessidades e expectativas do usuário. A avaliação desse critério é feita levando-se em consideração os seguintes sub-critérios:

  - suporte para predição - é a característica do método examinar a especificação visando determinar se o

sistema atingirá os objetivos, caso seja construído de acordo com o que foi especificado;

- suporte para extração - é a característica do método identificar as alterações, não explicitadas, nos objetivos atuais.

. **suporte para modificação** - é a característica do método fornecer auxílio para modificar a especificação sem perder a qualidade.

. **suporte para reflexão** - é a característica do método fornecer auxílio para criar e/ou alterar uma determinada especificação, a partir de uma outra especificação de nível mais abstrato. Para avaliar esse critério deve-se considerar os seguintes sub-critérios:

- suporte para pesquisa - é o atributo que permite procurar as especificações de nível mais abstrato que deram origem a uma determinada especificação;

- suporte para reconstrução - é o atributo do método que permite a geração de especificações mais abstratas a partir de suas especificações derivadas;

- suporte para identificação de alterações - é a característica do método gerar propostas de alteração em especificações de nível mais abstrato, para torná-las consistentes com as modificações efetuadas em especificações delas derivadas.

. **suporte para gerência** — é o critério que avalia o auxílio fornecido pelo método para a coleta de dados quanto ao desempenho dos processos e/ou das ferramentas empregadas para produzir especificações. Para avaliar esse critério deve-se levar em consideração os seguintes sub-critérios: suporte para monitoramento do desempenho, suporte para controle de alterações, suporte para controle de qualidade e suporte para controle de execução segundo o plano de desenvolvimento.

No trabalho de ROCHA (1983) também são apresentados fatores e critérios para a avaliação da qualidade de linguagens de especificação. Esses fatores de qualidade são definidos a seguir:

— **Construtibilidade** — é a característica segundo a qual uma linguagem auxilia o especificador na produção de especificações. O fator **construtibilidade** é avaliado através dos seguintes critérios:

. **documentação de utilização** — é a característica da linguagem possuir documentação através da qual se possa obter informações sobre sua utilização para construir especificações;

. **exigência de formação para utilização** — é o atributo da linguagem exigir para seu uso um nível de formação compatível com os especificadores disponíveis no mercado;

. **treinamento para utilização** – é a característica da linguagem exigir para seu uso um nível de treinamento possível de ser realizado;

. **tempo X desempenho** – é o atributo da linguagem exigir para a construção de especificações um tempo compatível com a complexidade da tarefa;

. **naturalidade** – é o atributo que avalia a linguagem quanto à proximidade conceitual e de notação ao problema que está sendo especificado, isto é, verifica se a linguagem de especificação é dirigida ao domínio do problema que está sendo especificado;

. **auxílio para evolução** – é a característica que a linguagem tem de facilitar o detalhamento de itens já parcialmente especificados.

– **Inteligibilidade** – é a característica da linguagem de facilitar a leitura de especificações. O fator **inteligibilidade** é avaliado através dos seguintes critérios:

. **documentação para decodificação** – é a característica da linguagem possuir documentação através da qual se possa obter informações sobre como ler especificações utilizando a linguagem;

. **exigência de formação para decodificação** – é o atributo da linguagem exigir um nível de formação compatível com os diferentes tipos de leitores da especificação;

. **naturalidade** – é o critério que avalia a linguagem quanto à proximidade conceitual e de notação ao problema que está sendo especificado.

– **Capacidade de auxiliar na avaliação** – é a característica da linguagem auxiliar na verificação e validação de especificações. Esse fator é alcançado através dos seguintes critérios:

. **formalidade** – é a característica da linguagem de especificação ter uma notação formalmente definida;

. **auto-verificação** – é a característica da linguagem de, através de seu próprio uso, auxiliar na detecção de erros decorrentes de seu uso não apropriado;

. **auto-validação** – é o atributo da linguagem de, através de seu próprio uso, auxiliar na detecção de erros na especificação que afetem o alcance das necessidades e expectativas, com relação ao produto que está sendo especificado.

II.4 PROPOSTA DE ROMAN

O trabalho desenvolvido em (ROMAN, 1985) apresenta um conjunto de critérios para a classificação de métodos para especificação de requisitos. Os critérios definidos por ROMAN (1985) são os seguintes:

– **Fundamentos formais** – este critério está relacionado com a teoria que estabelece as bases para um determinado método. É essa fundamentação teórica que serve de

estrutura básica para a automação do método. Nesse critério estão incluídas várias alternativas de fundamentos teóricos, tais como:

- **máquinas de estado finito** — oferecem um alto grau de análise dos requisitos;
- **diagrama de fluxo de dados** — representa as funções do sistema e o fluxo de dados existente entre elas;
- **caminhos de estímulo-resposta** — os métodos que usam caminhos de estímulo-resposta decompõem os requisitos de acordo com as funções a serem executadas pelo software, após o recebimento de cada estímulo;
- **processos concorrentes de comunicação** — nesta abordagem um processo é representado por um conjunto de estados e por um mapeamento de transição de estados;
- **composição funcional** — esta abordagem foi promulgada pelo método HOS (HAMILTON, 1976) e é suportada pela ferramenta USE.IT que permite a definição gráfica do funcionamento do sistema como uma composição de funções matemáticas;
- **modelos orientados a dados** — especificam o estado do sistema a partir da representação dos dados a serem mantidos.

— **Escopo** — é a característica definida pelo tipo de requisito que um método é capaz de expressar. Os

requisitos de um sistema podem ser funcionais ou não-funcionais.

. **requisitos funcionais** - são aplicados para modelar os estados internos do sistema e determinam o tipo de interação existente entre os componentes e o ambiente desse sistema;

. **requisitos não-funcionais (ou restrições)** - estes requisitos delimitam os tipos de soluções a serem considerados durante a especificação do produto.

Determinados métodos definem apenas os requisitos funcionais de um sistema, alguns limitam-se a expressar determinados requisitos não-funcionais e outros representam tanto a funcionalidade do sistema como um subconjunto selecionado de restrições.

- **Grau de formalidade** - o grau de formalidade de um método é determinado pelo nível de entendimento que uma máquina tem da linguagem (ou linguagens) que o método utiliza, sem haver a interação humana.

- **Grau de especialização** - a especialização de um método com relação a determinado tipo de problema aumenta sua capacidade de análise e a possibilidade de automação, e facilita a sua utilização. Os métodos, de uma maneira geral, podem ser específicos a um domínio, sensíveis a um domínio ou independentes de um determinado domínio.

II.5 PROPOSTA DE DAVIS

Em (DAVIS, 1986) é apresentado um modelo para seleção de tecnologias de especificação elaborado a partir do projeto Diretrizes para a Tecnologia de Especificação (DTE) ("Specification Technology Guidelines"). O projeto DTE, desenvolvido pela "Boeing Aerospace Company", propunha-se a organizar informações existentes sobre tecnologias de requisitos e projeto em um Livro de Normas. Essas informações deveriam ser organizadas de forma que pudessem ser usadas pelos gerentes técnicos da Força Aérea dos EUA na seleção dos métodos mais apropriados para o desenvolvimento de seus projetos.

A escolha dos métodos deveria ser feita levando-se em consideração as características de cada método e a capacidade dessas características de satisfazerem os requisitos determinados pelas categorias de software e pelas fases do processo de desenvolvimento.

Segundo as normas estabelecidas pelo projeto DTE, a seleção de um método é feita através dos seguintes passos:

- cálculo do Nível de Importância Global ("Overall Significance Level") para o projeto;
- identificação da categoria de software na qual o projeto mais se enquadra;
- seleção dos métodos mais adequados para o desenvolvimento do projeto, e,
- escolha do método mais indicado para o projeto.

Cada passo do processo de seleção de métodos é detalhado a seguir:

a) **Cálculo do Nível de Importância Global (NIG)** - o Nível de Importância (NI) avalia o projeto a partir dos seguintes aspectos:

- considerações de projeto, que abordam o ponto de vista do financiador do projeto;
- considerações de software, que abordam o ponto de vista dos desenvolvedores, e,
- considerações de qualidade, que abordam o ponto de vista dos usuários finais.

A cada consideração é atribuído um valor de NI que pode variar de 0 a 3. O NI de um projeto é a combinação de vários níveis, ou seja, é a média ponderada do valor do NI atribuído a cada consideração. Projetos com NI igual a 3 são mais importantes que projetos com NI igual a 0. Verifica-se, com o NI atribuído a um projeto, se a aplicação de uma determinada tecnologia será prática ou não.

O relacionamento de cada consideração com o NI é definido da seguinte forma:

- **Considerações de projeto** - medem a importância que o orgão financiador atribui ao projeto. Nesse item são avaliados:

. **custo** – o NI de um projeto cresce na proporção direta à redução das restrições relacionadas ao custo do desenvolvimento do projeto;

. **segurança** – o NI de um projeto cresce na proporção direta ao aumento da necessidade de segurança do projeto;

. **planejamento** – o NI de um projeto cresce na proporção direta à redução das restrições de planejamento do projeto.

– **Considerações de software** – medem o esforço conceitual do projeto e sua importância para os desenvolvedores. Nesse item são avaliados:

. **complexidade** – o NI de um projeto aumenta na mesma proporção da complexidade da solução do problema;

. **formalidade de desenvolvimento** – o NI de um projeto aumenta na mesma proporção do nível de controle desejado pelo contratante;

. **utilidade do software** – o NI de um projeto aumenta na mesma proporção do aumento da utilidade da aplicação.

– **Considerações de qualidade** – medem a importância que os usuários finais atribuem ao projeto. Nesse item são avaliados:

. **confiança** – o NI de um projeto aumenta em proporção direta ao nível de confiança requerido;

. **correção** – o NI de um projeto aumenta em proporção direta ao nível de correção desejado;

. **manutenção** – o NI de um projeto aumenta em proporção direta ao nível de manutenção necessário;

. **verificação** – o NI de um projeto aumenta em proporção direta ao nível de verificação requerido.

Como exemplo de projetos para cada NI temos:

. NI = 0 -> geradores de testes e programas de conversão de dados;
. NI = 1 -> editores, compiladores e simuladores de ambientes;
. NI = 2 -> planejamento de missões aéreas e vigilância de área;
. NI = 3 -> controle nuclear e software de vida crítica.

O cálculo do NIG para cada projeto é feito da seguinte forma:

. avaliar e atribuir o NI para cada consideração;
. atribuir um fator de peso para cada consideração;
. calcular o NIG através da seguinte fórmula:

NIG = SPr / SPe

onde:

SPr é o somatório dos produtos entre o NI e o fator de peso atribuído a cada consideração;

SPe é o somatório dos fatores de peso para cada consideração.

b) **Identificação da categoria de software** - foram identificadas dezoito categorias nas quais o software deverá ser enquadrado. As categorias de software foram especificadas em uma tabela contendo o nome da categoria, suas características e a descrição do software que se enquadra nessa categoria.

c) **Seleção dos métodos mais adequados** - os métodos candidatos são selecionados através da equiparação das características dos métodos com as capacidades desejadas a uma fase particular do ciclo de vida e a uma categoria de software.

O ciclo de vida considerado nesse processo é composto das fases de análise de requisitos, projeto preliminar, projeto detalhado, codificação, teste de unidades, teste de integração e teste de sistema.

Neste passo é construída, para cada fase do ciclo de vida, uma tabela denominada Tabela de Combinações (TC), que é composta de duas outras tabelas, a Matriz de Qualidades Desejáveis (MQD) e a Matriz de Suporte (MS). A MQD, representada na parte superior de uma TC, mostra categorias de software versus as capacidades de um método apropriadas a uma determinada fase do ciclo de vida. A MS, por sua vez, mostra os métodos a serem avaliados versus as mesmas capacidades identificadas na tabela anterior.

Na tabela MQD as linhas contém dezoito categorias de software e as colunas, características de métodos que são apropriadas a uma fase específica do ciclo de vida. São

encontradas marcas no ponto de interseção entre uma linha e uma coluna indicando que aquela capacidade do método é desejada naquela categoria de software.

Para cada método foram identificadas trinta capacidades. Algumas dessas capacidades estão relacionadas com a fase de especificação de requisitos, outras com a especificação de projeto e algumas são independentes da fase do ciclo de vida e da categoria do software:

- **Capacidades relacionadas à especificação de requisitos:** modelagem de estado, modelagem de fluxo de dados, modelagem de fluxo de controle, modelagem de objeto, especificação de desempenho de tempo e especificação de desempenho de segurança.

- **Capacidades relacionadas à especificação de projeto:** decomposição funcional, decomposição de dados, decomposição de controle, abstração de dados, abstração de processo, definição de banco de dados, especificação de concorrência/sincronização, definição de interface de módulo, verificação formal, gerência de configuração, análise de completude, análise de consistência e notação para comportamento de código.

- **Capacidades genéricas e independentes da fase do ciclo de vida e da categoria do software:** prototipagem, geração de plano de teste, validação, utilizabilidade, maturidade, rastreabilidade, transição entre fases, disponibilidade de ferramentas, treinamento, nível de experiência e documentação.

As linhas da tabela MS contêm os métodos, enquanto as colunas representam as mesmas características idênticas às das colunas da tabela anterior. Se um método suporta determinada capacidade, estão existe uma marca na interseção da coluna do método com a coluna das capacidades.

Os métodos são selecionados através da inspeção visual das tabelas MQD e MS, ou seja, através da equiparação entre os pontos marcados nas matrizes que compõem a TC. Verificam-se quais as linhas da tabela MS que estão marcadas da mesma forma que as linhas da tabela MQD, ou seja, quais os métodos que suportam as mesmas características requeridas pelas categorias de software. Para tanto podemos observar o exemplo representado na tabela II.5.1, que mostra uma TC composta pelas matrizes MQD e MS.

d) **Escolha do método mais indicado** — nesta etapa é escolhido o método mais indicado para cada fase específica do desenvolvimento do projeto, a partir dos métodos selecionados no passo anterior. A escolha é feita através da comparação dos métodos candidatos com um método ideal. O método ideal provê todas as capacidades desejadas no nível de suporte indicado pelo NIG do projeto.

O processo de escolha pode ser efetuado aplicando-se duas formas de tratamento diferentes para o nível de suporte oferecido pelo método ideal. Os níveis de tratamento são:

MATRIZ DE QUALIDADES DESEJÁVEIS							
		TÉCNICAS DE MODELAGEM				ESPECIFIC. DE DESEMPENHO	
		ESTADO	FLUXO		OBJETO	TEMPO	PRECISÃO
			DADO	CONTROLE			
CATEGORIAS DE SOFTWARE	1				X		X
	2			X		X	
	3		X	X		X	
	4		X	X		X	X
	5		X	X	X	X	
	6	X	X	X	X	X	X
	7	X	X	X	X	X	
	8		X	X	X	X	X
	9		X		X		X
	10	X	X	X	X	X	
	11	X	X	X	X	X	X
	12		X		X		X
	13	X	X	X	X	X	X
	14		X		X		X
	15	X	X	X	X		X
	16		X	X	X		X
	17		X	X	X	X	
	18		X		X		
MATRIZ DE SUPORTE							
MÉTODOS	A		X	X	X		
	B	X		X			X
	C		X		X	X	
	D	X		X	X	X	X
	E	X			X		X
	F	X	X	X	X	X	
	G	X	X	X			

Tabela II.5.1: Exemplo de uma Tabela de Combinações

Fonte: (DAVIS, 1986)

- **nível de suporte uniforme**, onde cada capacidade do método ideal recebe como NI o mesmo valor do NIG calculado para o projeto;

- **nível de suporte não uniforme**, onde para cada capacidade do método ideal é atribuído um valor de nível de suporte separado, ou seja, cada capacidade recebe um NI específico.

A comparação entre métodos candidatos e o método ideal pode ser feita através da comparação dos Números de Pontos do Método (NPM) ("Methodology Scores"), calculados para cada método selecionado. O cálculo do NPM varia de acordo com o tratamento (uniforme ou não uniforme) dado ao nível de suporte do método ideal.

O melhor método deve ter o valor do NPM mais próximo de zero. Um método com valor de NPM maior do que zero provê mais suporte do que o necessário e um método com valor de NPM menor do que zero provê menos suporte do que o desejado. Assim, após o cálculo do NPM para cada método candidato, deve-se observar qual o método que possui o NPM mais próximo de 0.

Para calcular o NPM nos níveis especificados acima, devem ser aplicadas as seguintes fórmulas:

- Cálculo do NPM com nível de suporte uniforme:

$$NPM = SUM - NI * COUNT$$

onde:

SUM é a soma dos valores atribuídos a cada capacidade do método e corresponde ao nível de suporte que o método oferece para a fase onde ele pode ser aplicado;

NI é o nível de importância;

COUNT é a cardinalidade do conjunto de capacidades desejadas para o método.

- Cálculo do NPM com nível de suporte não uniforme:

$$NPM = \sum_{\text{todas as capacidades desejadas}} (R_{método} - R_{ideal})$$

onde:

NPM é o somatório de $(R_{método} - R_{ideal})$ para todas as capacidades desejadas;

$R_{método}$ é o valor do NI atribuído a uma capacidade particular do método;

R_{ideal} é o valor desejado da capacidade atribuída para o método ideal.

O nível de tratamento uniforme é ideal para gerentes de projeto com pouca experiência em selecionar métodos. Já o tratamento não uniforme deve ser utilizado por gerentes experientes que possam aplicar conhecimentos adquiridos sobre a seleção de métodos em projetos anteriores.

II.6 PROPOSTA DE FLOYD

A proposta desenvolvida em (FLOYD, 1986) apresenta um esquema para classificação de características de métodos.

Segundo FLOYD (1986) um método pode ser classificado tendo por base:

- uma área de aplicação;
- sua filosofia, e,
- diretrizes que englobam princípios de organização, técnicas e ferramentas.

Para entender um método é importante perceber a forma como esse método interage com o processo de desenvolvimento, ou seja, como o método "vê" o processo. Os métodos baseiam-se no seu entendimento do que é o desenvolvimento de um software, de como esse software pode ser modelado, de quais são as atividades importantes e de como elas podem ser suportadas.

A área de aplicação de um método pode ser caracterizada através dos seguintes aspectos:

- o tamanho do sistema;
- o tipo de problema a ser resolvido e a área do problema;
- a arquitetura desejada para o sistema;
- a produção técnica e o ambiente de desenvolvimento, incluindo o hardware, as linguagens, os sistemas operacionais, os bancos de dados e outros software de suporte, e,
- a forma de cooperação entre os desenvolvedores e o relacionamento entre esses desenvolvedores e os usuários.

As características de um método podem ser divididas em características orientadas ao produto e características orientadas ao processo.

- **Características orientadas ao produto** - relacionam-se com a elaboração da parte que é considerada o produto do desenvolvimento do sistema. Essas características podem ser estudadas a partir de três dimensões:

  . universos de discursos relevantes para o processo de desenvolvimento do software, que podem ser:

    - os objetivos baseados em uma análise geral das mudanças desejadas;
    - os requisitos do relacionamento entre o sistema a ser implementado e seu contexto de uso, e,
    - as funções do sistema especificadas no projeto da arquitetura;

  . elementos do projeto detalhado do sistema, e,

  . componentes de cada programa.

- **Características orientadas ao processo** - ajudam na estruturação, organização e na melhora do processo de desenvolvimento de um produto de software.

II.7 PROPOSTA DE KELLY

No trabalho de KELLY (1987) é apresentada uma série de características para a avaliação de métodos para o projeto de sistemas de tempo real. A partir dessas características

foi elaborada uma estrutura para comparação de métodos para o desenvolvimento de software.

Os critérios para avaliação de métodos apresentados na proposta de KELLY (1987) são definidos a seguir:

- **Base formal** - este critério descreve a forma como o método vê o software (por exemplo, como um conjunto de objetos que se comunicam entre si ou a partir do fluxo de dados que atravessa o sistema);

- **Integridade semântica** - este critério verifica se a especificação de projeto, gerada pelo método, leva o programador a perceber as intenções expressas pelo projetista. A semântica de um método pode ser forte ou fraca. Uma semântica forte é geralmente carregada de fundamentos matemáticos e requer um esforço considerável para criar e manter especificações. Já um método com semântica fraca permite muita liberdade para interpretar o significado de uma especificação;

- **Técnicas para verificação e validação de projeto** - este critério avalia a capacidade do método verificar e validar a especificação de projeto a partir da especificação de requisitos, antes de passar para a codificação. Também identifica a forma e os recursos utilizados pelo método para verificar e validar o produto;

- **Facilidade de mudanças/manutenibilidade** - este critério avalia se o método oferece ou não facilidades para efetuar mudanças no projeto por ele especificado;

- **Processamento concorrente e comunicação** — este critério avalia a existência no método de recursos para projetar concorrência;

- **Desempenho do projeto** — este critério avalia a capacidade do método permitir que considerações de tempo de resposta e precisão estejam integrados no projeto;

- **Maturidade/história do método** — este critério indica se o método já foi utilizado em vários projetos reais ou se ainda encontra-se em um estágio experimental;

- **Reutilizabilidade** — este critério avalia a capacidade oferecida pelo método para reutilizar partes de projetos anteriores em novos projetos;

- **Grau de suporte automatizado** — este critério avalia a disponibilidade de ferramentas para automatização do método, o equipamento onde podem ser executadas e o custo dessas ferramentas;

- **Treinamento** — este critério verifica a disponibilidade de informações sobre o método e a facilidade de aprender a utilizá-lo;

- **Grau de suporte para outras fases do ciclo de vida** — este critério avalia as fases do ciclo de vida nas quais o método pode ser aplicado;

- **Considerações de linguagem** — alguns métodos são ajustados às construções suportadas por uma determinada linguagem. Esse critério verifica, portanto, se o método está

ligado a construções de alguma linguagem particular e qual é essa linguagem;

- **Passos gerais** – este critério descreve a seqüência de passos a ser executada pelo método;

- **Notação gráfica** – este critério identifica a notação gráfica e textual da linguagem (ou linguagens) do método;

- **Vantagens/desvantagens** – este critério aponta as vantagens e desvantagens do método.

II.8 CONCLUSÃO

Neste capítulo foi apresentada uma série de características para avaliação de métodos conforme estão propostas na literatura. Como pode ser observado, existem vários critérios para selecionar métodos. A escolha dos critérios mais apropriados vai depender do contexto no qual o processo de escolha de métodos está inserido. No próximo capítulo é descrita uma proposta de atributos a serem considerados na avaliação de métodos para o desenvolvimento de software.

CAPÍTULO III

AVALIAÇÃO DE MÉTODOS PARA O DESENVOLVIMENTO DE SOFTWARE

III.1 INTRODUÇÃO

Durante as últimas décadas, vários métodos têm sido propostos visando auxiliar o processo de desenvolvimento de software. Hoje, o número de métodos disponíveis é muito grande, dificultando escolher aqueles a serem utilizados para construir uma determinada aplicação. É necessário a existência de propostas para avaliação que auxiliem e facilitem a escolha de métodos.

No capítulo anterior foi apresentada uma série de trabalhos, presentes na literatura, no que se refere a características de métodos, sendo discutidas algumas propostas para a avaliação e seleção.

Neste capítulo é apresentada a proposta para avaliação de métodos de desenvolvimento de software definida no trabalho. Os conceitos básicos dessa proposta fundamentam-se na estrutura do Método para Avaliação da Qualidade de Software definido em (ROCHA, 1987).

Na próxima seção é apresentada a estrutura geral do Método para Avaliação da Qualidade de Software utilizado. Em seguida são definidos os objetivos, fatores e critérios de qualidade utilizados na avaliação de métodos para o desenvolvimento de software e que compõem o método proposto no trabalho.

III.2 MÉTODO PARA AVALIAÇÃO DA QUALIDADE DE SOFTWARE

Segundo ROCHA (1985), qualidade é um atributo relacionado a alguma coisa, devendo, assim, ser definida de acordo com o objeto que será avaliado. Qualidade é, também, um conceito multidimensional, a ser realizado através de um conjunto de atributos ou características.

O método proposto por ROCHA (1987) organiza um conjunto de características de qualidade que possibilitam a avaliação de software, sob qualquer forma de representação, isto é, a avaliação da especificação de requisitos, especificação de projeto e programas. Esse método está baseado nos seguintes conceitos, representados na figura III.2.1:

- **Objetivos de qualidade** - são propriedades gerais que o produto deve possuir;
- **Fatores de qualidade do produto** - determinam a qualidade do produto sob o ponto de vista dos seus diferentes usuários (por exemplo, usuário final, mantenedores);
- **Critérios** - são atributos primitivos possíveis de serem avaliados;
- **Processos de avaliação** - determinam o processo e os instrumentos a serem utilizados para se medir o grau de presença, no produto, de um determinado atributo;
- **Medidas** - são os resultados da avaliação do produto segundo os critérios;

. **Medidas agregadas** — são o resultado da agregação das medidas obtidas na avaliação segundo os critérios e quantificam os fatores.

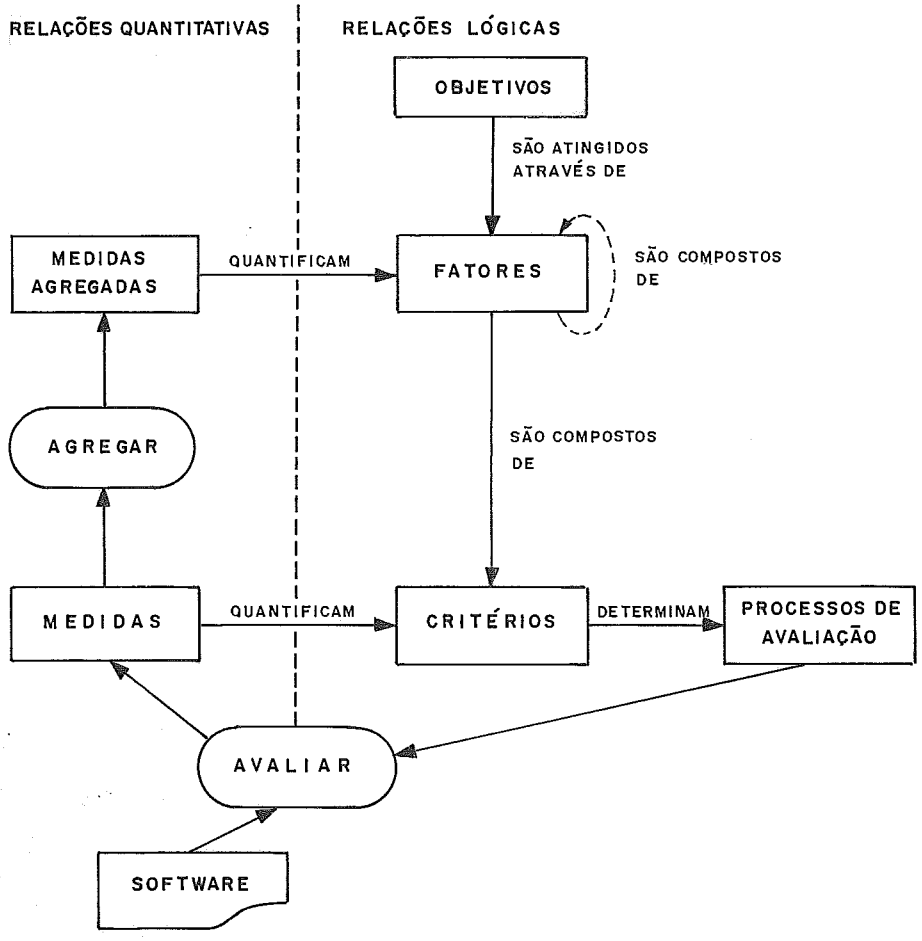


Figura III. 2.1 : Estrutura do Método para Avaliacão da Qualidade de Software
Fonte : (ROCHA , 1987)

A proposta apresentada neste capítulo baseia-se na estrutura do método acima. Para chegar a essa proposta foi identificado, em primeiro lugar, um conjunto de características de métodos de desenvolvimento de software. Em seguida, essas características foram agrupadas e organizadas em termos de objetivos, fatores e critérios de

qualidade. Esses critérios são utilizados para avaliar métodos de desenvolvimento de software. Essa avaliação auxiliará na seleção dos métodos mais adequados à construção de uma determinada aplicação. Os objetivos, fatores e critérios, que compõem a proposta para avaliação de métodos de desenvolvimento de software descrita neste trabalho, são apresentados na próxima seção.

III.3 AVALIAÇÃO DE MÉTODOS PARA DESENVOLVIMENTO DE SOFTWARE

A utilizabilidade é o principal objetivo de qualidade que um método para desenvolvimento de software deve atingir. Métodos existem para serem usados. Eles visam auxiliar no desenvolvimento sistemático de produtos de software, buscando satisfazer as necessidades dos usuários, de acordo com os recursos disponíveis e as restrições estabelecidas.

Um método deve ser utilizado para apoiar as atividades fundamentais a serem realizadas durante o processo de desenvolvimento de um software: construção, avaliação e gerência. Assim sendo, ao avaliar um método para desenvolvimento de software, deve-se considerar os seguintes aspectos:

- utilizabilidade para construção do produto;
- utilizabilidade para avaliação do produto;
- utilizabilidade para gerência do processo de desenvolvimento.

A figura III.3.1 mostra os atributos de qualidade relacionados a esses aspectos, organizados, segundo o método proposto, em objetivos, fatores e critérios para avaliação de métodos para o desenvolvimento de software. Cada um desses atributos são descritos, detalhadamente, nas seções que se seguem.

III.3.1 AVALIAÇÃO SEGUNDO O OBJETIVO UTILIZABILIDADE PARA CONSTRUÇÃO

Um dos principais objetivos de um método é auxiliar na construção de software. **Utilizabilidade para construção** é o objetivo relacionado às características de um método que permitem a sua utilização no processo de construção de um produto de software.

Este objetivo é avaliado através dos seguintes fatores: **aplicabilidade, expressividade, facilidade de aprendizado para construção e facilidade de uso.**

III.3.1.1 O FATOR APLICABILIDADE

Não existe um método que possa ser usado de forma universal no desenvolvimento de qualquer produto de software, independentemente das características do domínio do problema, do próprio produto ou do contexto no qual o produto será desenvolvido. Métodos são propostos com o objetivo de modelar um determinado tipo de problema, possuindo uma série de características que limitam o seu campo de aplicação.

OBJETIVO UTILIZABILIDADE PARA CONSTRUÇÃO

FATOR: APLICABILIDADE

Critério: Adequação ao Ciclo de Vida
Adequação ao Tipo de Processamento
Adequação ao Domínio da Aplicação
Adequação ao Nível de Complexidade

FATOR: EXPRESSIVIDADE

Critério: Abordagem de Desenvolvimento Utilizada
Técnicas Construtivas Utilizadas
Tipo de Instrumentos Notacionais
Forma de Expressão dos Instrumentos Notacionais
Grau de Formalidade dos Instrumentos Notacionais

FATOR: FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO

Critério: Documentação para Construção
Exigência de Formação para Construção
Existência de Instrumentos Cognitivos

FATOR: FACILIDADE DE USO

Critério: Existência de Apoio Automatizado
Automatizabilidade
Extensibilidade

OBJETIVO: UTILIZABILIDADE PARA AVALIAÇÃO

FATOR: VERIFICABILIDADE

Critério: Existência de Normas de Qualidade
Existência de Ferramentas para Verificação

FATOR: VALIDABILIDADE

Critério: Compreensividade
Existência de Ferramentas para Validação

OBJETIVO UTILIZABILIDADE PARA GERÊNCIA

FATOR: SUPORTE PARA PLANEJAMENTO DO PROJETO

Critério: Suporte para Estimativa de Custos
Suporte para Definição do Cronograma
Suporte para Estimativa de Recursos

FATOR: SUPORTE PARA ACOMPANHAMENTO DO PROJETO

Critério: Suporte para Acompanhamento de Custos
Suporte para Acompanhamento de Cronograma
Suporte para Controle de Versões

Figura III.3.1: Atributos para Avaliação de Métodos

O fator **aplicabilidade** avalia o campo de utilização de um método através dos seguintes critérios: **adequação ao ciclo de vida, adequação ao tipo de processamento, adequação ao domínio da aplicação e adequação ao nível de complexidade.**

- Adequação ao ciclo de vida

O termo ciclo de vida é, normalmente, empregado para definir as fases do processo de desenvolvimento de software, especificando todas as atividades que devem ser realizadas em cada uma dessas fases (ROCHA, 1987a). O ciclo de vida de um software envolve, de maneira geral, todas as atividades necessárias para definir, desenvolver, testar, tornar operacional e manter o produto.

Existem, na literatura, várias propostas de modelos de ciclo de vida. Esses modelos abordam diversos aspectos envolvidos na produção do software e definem diferentes atividades e controles a serem exercidos.

É importante observar que não existe um modelo de ciclo de vida apropriado ao desenvolvimento de qualquer produto de software. Para cada tipo de software deve ser selecionado um modelo de ciclo de vida que forneça as bases necessárias para o seu desenvolvimento e manutenção (FAIRLEY, 1986).

Segundo ROCHA (1987a), a escolha do modelo de ciclo de vida apropriado para um determinado software depende do propósito do ciclo de vida, das características do produto

a ser desenvolvido e das características do ambiente onde se dará o desenvolvimento.

Métodos visam auxiliar na execução das várias atividades envolvidas no processo de desenvolvimento de software. De maneira geral, possuem características que os levam a ser adequados a um modelo de ciclo de vida específico.

Neste trabalho são considerados os seguintes modelos de ciclo de vida (DAVIS, 1988):

. **Modelo de ciclo de vida clássico** — este modelo de ciclo de vida é dividido em uma série de fases sucessivas, com pontos de controle após cada uma delas. Cada fase é um detalhamento e um refinamento da fase anterior, e gera resultados que são utilizados na etapa seguinte (FAIRLEY, 1986). É também chamado de modelo de ciclo de vida por fase, modelo tradicional ou modelo cascata.

O ciclo de vida clássico sugere uma abordagem sistemática e seqüencial para o desenvolvimento de software, com um processo de interação entre as suas fases, permitindo o retorno às fases anteriores quando for necessário (PRESSMAN, 1988).

Segundo (WEITZEL, 1989), no modelo cascata assume-se que o usuário conhece seus requisitos e que os pontos de controle entre as fases garantem a satisfação desse usuário, com o produto que está sendo gerado.

Existem algumas variações quanto à nomenclatura e à composição das fases do ciclo de vida clássico. FAIRLEY (1986), por exemplo, divide o modelo clássico nas fases de análise, projeto, implementação, teste do sistema e manutenção. Em ROCHA (1987) ele é composto pelas fases de definição, projeto, construção, avaliação e operação. PRESSMAN (1988), por sua vez, denomina as fases desse ciclo de vida da seguinte forma: análise de requisitos, projeto, codificação, teste e manutenção.

Na realidade, o que caracteriza o modelo de ciclo de vida clássico é a progressão seqüencial existente entre uma fase e a seguinte, levando à passagem para a próxima fase apenas quando a anterior estiver totalmente terminada. O produto só estará em condições de ser demonstrado ao usuário nas etapas finais do ciclo de vida, quando ele já estiver sido codificado e testado.

. **Modelo de ciclo de vida baseado em protótipos** — a prototipagem é uma abordagem de desenvolvimento que possibilita a construção de um modelo do software a ser construído (PRESSMAN, 1988).

Segundo (AARAM, 1984), o protótipo de um software é um modelo operacional desse software, possível de ser executado em computador. As características funcionais do modelo são similares às características do produto. O protótipo possibilita a verificação imediata das características do software antes dele ser completamente desenvolvido. Para isso um protótipo deve ser fácil de ser construído e modificado.

Em (CHRISTENSEN, 1984) são citados os seguintes objetivos para o uso de protótipos:

- facilitar a comunicação entre desenvolvedores e usuários, que em geral falam linguagens diferentes;
- ser utilizado como uma ferramenta para treinar os usuários;
- ser uma forma de verificar se o software faz realmente aquilo que o usuário necessita;
- auxiliar no levantamento de custos do novo sistema.

Existem na literatura várias formas de prototipagem. São consideradas, entretanto, as classes sugeridas em DAVIS (1988):

- **Prototipagem descartável** - neste tipo de prototipagem é construída uma implementação parcial do produto, antes ou durante a fase de definição dos requisitos do sistema, contendo os aspectos menos entendidos.

O protótipo é utilizado pelo usuário como modelo para avaliação, de forma que possíveis mudanças possam ser detectadas. As alterações são efetuadas diretamente na especificação de requisitos do produto final. Assim, o protótipo é usado para identificar as necessidades do usuário e é jogado fora após a identificação dos requisitos.

- **Prototipagem incremental** - neste modelo de prototipagem é construída uma implementação parcial do sistema que possibilite a incorporação de novos

requisitos ao protótipo, de forma a aumentar gradativamente a funcionalidade e o desempenho. Muitos dos requisitos do sistema são entendidos e implementados através de subconjuntos com uma complexidade crescente.

- **Prototipagem evolutiva** — segundo FLOYD (1984), o desenvolvimento de software na prototipagem evolutiva é visto como uma seqüência de ciclos: (re)projeto, (re)implementação, (re)avaliação.

Nesse modelo de prototipagem, a implementação do sistema é iniciada pelos requisitos mais entendidos. Baseia-se no aprendizado através da experimentação do protótipo implementado, procurando-se entender os requisitos que ainda não estão bem claros.

. **Modelo de ciclo de vida baseado em síntese automática de programas** — este modelo de ciclo de vida baseia-se na transformação automática de especificações de requisitos ou de projeto em código executável.

Segundo (DAVIS, 1990), a síntese automática de software a partir de especificações é geralmente vista como um processo de duas etapas: síntese do projeto e otimização do projeto.

O processo de transformação pode ser efetuado através de algoritmos, de uma arquitetura usada em todas as aplicações ou de técnicas baseadas em conhecimento, isto é, usando uma base de conhecimento contendo o conhecimento dos especialistas na construção de projetos de software.

A linguagem utilizada para escrever a especificação é uma linguagem de alto nível de abstração, muito próxima do domínio do problema. Após a construção, a especificação é traduzida automaticamente para código de máquina.

. **Modelo de ciclo de vida baseado em reutilização** - este modelo de ciclo de vida visa incorporar partes de especificação, de projeto ou de código de produtos anteriormente desenvolvidos, em um novo software.

O processo de reutilização de software, segundo (BOLDYREFF, 1989), envolve a identificação de componentes reutilizáveis e de seu desdobramento em novas aplicações. O processo é elaborado através dos seguintes passos:

. identificação dos objetos com potencial de reuso;
. decomposição/abstração desses objetos;
. classificação dos objetos que farão parte da biblioteca de reuso;
. seleção/recuperação dos objetos reusáveis;
. especialização/adaptação dos objetos reusáveis;
. composição/desdobramento da nova aplicação.

O processo inicia-se com o reconhecimento da oportunidade de reuso, que envolve a decomposição de sistemas grandes nos seus conceitos componentes e a extração de conceitos específicos com potencial de reuso. A partir dos conceitos específicos similares é possível abstrair conceitos genéricos.

Os conceitos reutilizáveis precisam ser catalogados e armazenados, a fim de serem selecionados e recuperados para

reuso, quando for necessário. Os conceitos genéricos selecionados para reuso podem requerer a especialização ou adaptação para atender às especificidades da nova aplicação, na qual ele será reutilizado.

O critério **adequação ao ciclo de vida**, portanto, avalia o método com relação a qual desses modelos de ciclo de vida suas características permitem suportar.

- Adequação ao tipo de processamento

Métodos de desenvolvimento de software, também, possuem características que os tornam adequados à utilização no desenvolvimento de produtos de software com um determinado tipo de processamento.

Adequação ao tipo de processamento é o critério que avalia o método quanto à sua capacidade para modelagem de sistemas segundo os diferentes tipos de processamento.

Neste trabalho são considerados os seguintes tipos de processamento (FERREIRA, 1987):

. **Processamento seqüencial** - é caracterizado por:

  . informações acumuladas em lotes, classificados ou não, ao longo de determinados períodos de tempo;

  . freqüência de utilização do processamento fixada pelo costume ou por uma necessidade;

  . grandes quantidades de dados que devem ser consideradas antes de se tomar alguma decisão;

- tempo de formação de lotes dependente das condições das aplicações a serem processadas;
- baixa interação com o usuário;
- instruções de entrada e saída relativamente simples.

. **Processamento "on line"**, que é caracterizado por:

- processamento interativo com o usuário, sendo o tempo de resposta relativo, pois está dentro de um determinado intervalo de tempo;
- chegada de informações ao processador no momento em que estão sendo produzidas, podendo-se dizer que as informações estão "em linha" com o processador.

. **Processamento em tempo real**, que é caracterizado pela:

- modelagem de fenômenos variáveis sobre uma base de tempo, pois nesse tipo de processamento o tempo é a essência da interação do computador com o ambiente;
- dedicação do computador para a execução de uma única e grande aplicação;
- necessidade de um complexo projeto de estruturas lógicas, para controle do processo em andamento, devido à dificuldade de se pré-definir as possíveis ordens de ocorrência das sequências dos eventos de entrada;
- necessidade de eficiência e alta confiabilidade das operações.

O critério **adequação ao tipo de processamento**, portanto, avalia o método quanto à sua adequação para modelagem de sistemas com processamento seqüencial, "on line" ou em tempo real.

– Adequação ao domínio da aplicação

Segundo (WERNECK, 1990), uma aplicação computacional é uma adaptação de funções ou tarefas, que serão executadas em parte ou totalmente pelo computador.

Métodos de desenvolvimento de software não são adequados a qualquer tipo de problema. Cada domínio de problema possui atributos específicos que requerem diferentes recursos para sua solução. As características particulares a cada método fazem com que eles sejam mais adequados a certos domínios de problemas do que a outros.

Adequação ao domínio da aplicação é o critério que avalia as categorias de áreas de aplicação, em geral, suportadas pelo método.

– Adequação ao nível de complexidade

O desenvolvimento de programas grandes é uma atividade intelectual essencialmente distinta da atividade de construir programas pequenos (DEREMER, 1980). Produzir programas grandes apresenta questões que muitas vezes não aparecem no desenvolvimento de programas menores.

A tarefa de produzir programas grandes, ou seja, de produzir sistemas, é denominada segundo DEREMER (1980) de programação grande, programação em larga escala ou

programação de grande porte. Já o desenvolvimento de programas pequenos, ou módulos individuais, é referido pelo termo programação pequena, programação em pequena escala ou programação de pequeno porte.

GHEZZI (1985) afirma que o tamanho de um programa está mais relacionado ao tamanho e à complexidade do problema a ser resolvido do que ao tamanho final do próprio programa.

Um programa pequeno ou um módulo, que implementa um problema de complexidade reduzida, certamente possui um número de instruções que pode ser considerado pequeno e pode ser desenvolvido por uma única pessoa.

Os programas grandes, ou sistemas, conforme DEREMER (1980), são formados por um conjunto de programas pequenos, integrados como coleções de módulos individuais, normalmente escritos por pessoas diferentes e codificados em linguagens distintas. Isso gera a necessidade de coordenar as atividades do grupo de pessoas envolvido no desenvolvimento. É preciso aplicar, na programação em larga escala, técnicas de administração para controlar a equipe e medir o progresso no trabalho (GHEZZI, 1985).

O desenvolvimento de sistemas muitas vezes requer que uma série de restrições sejam atendidas, tais como:

- o alto custo aplicado no desenvolvimento requer que o sistema seja utilizado por um longo tempo;
- o sistema deve funcionar corretamente;

- durante sua vida útil o sistema está sujeito a várias modificações, devendo, portanto, ser fácil de modificar;
- deve ser possível mostrar a correção de um sistema a partir da correção de cada módulo ou programa.

Podemos afirmar que a programação em larga escala possui características, problemas, estrutura de construção e raciocínio diferentes da programação em pequena escala.

Assim sendo, um dos aspectos a serem avaliados para determinar se um método deve ser usado no desenvolvimento de um produto é a identificação da complexidade do problema, ou seja, se o problema encontra-se a nível de especificação de sistemas (programação em larga escala) ou de especificação de programas (programação em pequena escala).

Métodos adequados à construção de programas podem não ser aplicáveis ao desenvolvimento de sistemas. Assim sendo, o critério **adequação ao nível de complexidade** avalia se o método é adequado para o desenvolvimento de software em pequena ou larga escala, isto é, se é adequado para o desenvolvimento de programas ou sistemas.

III.3.1.2 O FATOR EXPRESSIVIDADE

A capacidade de auxiliar na construção de modelos do domínio do problema é uma das principais características de um método. Cada método utiliza uma forma própria para

expressar, ou seja, para representar o software a ser desenvolvido.

O fator **expressividade** pode ser definido como sendo a característica de qualidade que avalia o método quanto aos aspectos relacionados à modelagem do problema. Este fator é avaliado através dos critérios **abordagem de desenvolvimento utilizada, técnicas construtivas utilizadas, tipo de instrumentos notacionais, forma de expressão dos instrumentos notacionais e grau de formalidade dos instrumentos notacionais.**

- Abordagem de desenvolvimento utilizada

O desenvolvimento de um software é uma atividade muito dinâmica. Existem vários enfoques que podem ser utilizados para modelar a solução de um problema. Os diferentes enfoques para modelagem de um software são aqui denominados abordagens de desenvolvimento.

A forma como o problema deve ser visualizado no mundo da implementação e os mecanismos utilizados pelo método para a modelagem de uma solução diferenciam as diversas abordagens existentes. Pode-se dizer, então, que essas abordagens, utilizadas pelos métodos, são uma forma de estruturar o pensamento, visando a modelagem de um produto.

Neste trabalho são considerados os seguintes tipos de abordagens (BOOCH, 1986), (PRESSMAM, 1988), (COAD, 1990) e (YOURDON, 1990):

. **Abordagem de decomposição funcional**, onde o software é modelado em termos de suas funções.

A estratégia básica da decomposição funcional consiste na seleção do processamento necessário para o funcionamento do software e na decomposição desse processamento em uma seqüência de passos e sub-passos.

A solução do problema é mapeada, portanto, a partir do mundo real para um espaço de solução, estruturado de acordo com as funções e sub-funções que compõem o problema. Nessa abordagem também é especificado o relacionamento existente entre cada uma das funções identificadas.

. **Abordagem de fluxo de dados**, onde a modelagem do problema é efetuada a partir das informações que fluem pelo software e de todas as transformações sofridas por essas informações.

Assim, o mundo real é representado em um espaço de solução em termos dos fluxos de dados de entrada e saída e dos processos necessários para gerar os fluxos de dados.

A abordagem de fluxo de dados tem uma ênfase funcional forte, pois os processos são todos decompostos em subprocessos, ficando essa decomposição claramente representada. O refinamento sucessivo dos fluxos de dados não foi percebido com o mesmo grau de facilidade, pois alguns métodos que utilizam essa aboradagem não fazem a decomposição gradual do fluxo de dados, da mesma forma que fazem dos processos.

. **Abordagem orientada a estrutura de dados**, onde a representação do software é feita a partir da representação de suas estruturas de dados, isto é, o problema é mapeado do mundo para o espaço de soluções tendo como ponto de partida o detalhamento das estruturas de dados.

Os métodos que seguem essa abordagem identificam inicialmente os objetos chaves do problema. A partir daí, são identificadas as operações a serem executadas. Esses dados possuem uma estrutura hierárquica, que é representada em termos de construtores elementares de seqüência, seleção e repetição. Além disso esses métodos provêem um conjunto de passos para mapear a estrutura de dados hierárquica em uma estrutura de programa.

. **Abordagem de modelagem da informação**, onde o problema é modelado em termos de entidades e relacionamentos.

Os objetos do mundo real são identificados, seus atributos são descritos e são estabelecidos os relacionamentos entre esses objetos. Assim, a modelagem do problema é feita a partir do seu escopo no mundo real para um espaço de soluções estruturado de acordo com as entidades e seus relacionamentos.

. **Abordagem de orientação a objetos**, onde a modelagem do problema baseia-se no conceito que se tem de objeto. A noção intuitiva das pessoas é que o mundo está cheio de objetos que interagem entre si.

A idéia geral dessa abordagem é ver o software como sendo composto por objetos. O conceito de objetos, adotado na orientação a objeto, vem de duas áreas:

- da modelagem da informação, onde um objeto representa uma coisa do mundo real, com algumas instâncias da mesma;

- da programação orientada a objeto, onde um objeto significa uma instância, em tempo de execução, de algum processamento e valores definidos por uma classe.

Assim, um objeto é um componente do mundo real mapeado no domínio do software. É uma entidade que consiste de uma estrutura de dados (atributos) e todas as operações (métodos) que podem manipular esses dados. O comportamento de um objeto é, então, caracterizado pelas ações que ele pode sofrer ou que pode requerer de um outro objeto.

O acesso aos dados de um objeto dá-se através de suas operações. Essas operações são ativadas por mensagens que especificam os métodos que devem ser executados, ou seja, que fazem um objeto executar uma de suas operações. Os objetos se comunicam entre si através de mensagens.

Existem objetos com características semelhantes e que respondem às mensagens da mesma forma. Esses objetos similares são agrupados em classes. Todos os objetos de uma classe herdam a estrutura de dados e os procedimentos definidos para essa classe, sendo cada objeto individual uma instância da classe.

. **Abordagem baseada em conhecimento**, onde o problema é modelado a partir de técnicas da Inteligência Artificial.

Inteligência Artificial (IA) é o ramo da informática que enfoca o desenvolvimento de produtos de software capazes de desempenhar tarefas normalmente associadas ao comportamento humano inteligente (CHORAFAS, 1988). Possui as seguintes características:

. manipula símbolos em vez de números;
. faz inferências e deduções a partir da informação disponível;
. aplica o conhecimento na resolução de um problema;
. utiliza seu conhecimento e suas regras associadas para limitar o crescimento exponencial que ocorre em situações complexas do mundo real.

Segundo (RICH, 1983), para resolver os problemas complexos encontrados na IA, é preciso utilizar uma grande quantidade de conhecimentos e mecanismos capazes de manipular esses conhecimentos para gerar soluções para esse tipo de problema.

Existem vários mecanismos para armazenar e manipular o conhecimento do domínio de um problema específico. Quando associados a IA, esses mecanismos são denominados de representação do conhecimento. Dentre as formas de representação usadas na IA são destacadas as seguintes (ARAIBÓIA, 1988): representação lógica (ou declarativa),

representação procedimental, sistemas de produção, redes semânticas e "frames".

Um tipo de software desenvolvido com técnicas de IA são os sistemas especialistas que, segundo (CHORAFAS, 1988), são construções de software que os peritos em campos específicos enriquecem com seu conhecimento. Um sistema especialista se caracteriza por conter o conhecimento de um determinado campo, de forma a dar assistência aos especialistas ou prover informações ou auxílio às pessoas que não têm acesso a um especialista em uma área específica (AGUIAR, 1989).

O critério **abordagem de desenvolvimento utilizada** avalia, portanto, os princípios, os mecanismos e a forma de visualização utilizados pelo método para a modelagem do problema, identificando em qual desses tipos de abordagens o método se enquadra.

- Técnicas construtivas utilizadas

Durante o processo de desenvolvimento de um software, várias decisões devem ser tomadas. A forma e a ordem das decisões depende do tipo de técnica inerente ao método usado na construção do produto.

Segundo (ROCHA, 1987a), as técnicas construtivas guiam o processo de desenvolvimento do software, oferecendo meios para uma construção disciplinada. São essas técnicas construtivas que fornecem subsídios para a escolha da ordem na qual as decisões são efetuadas.

Os métodos baseados em técnicas construtivas especificam as decisões, a forma como tomá-las e em que ordem. Neste trabalho são consideradas as seguintes técnicas construtivas definidas em (FREEMAN, 1980):

. **Técnica "top-down"** - nesta técnica o desenvolvimento é efetuado de forma dedutiva, ou seja, do geral para o particular, sendo as decisões de mais alto nível tomadas primeiro.

Inicialmente é obtida uma visão geral do software. A partir desta visão o software é decomposto até se chegar a um nível de detalhe que o torne facilmente compreensível.

. **Técnica "bottom-up"** - nesta técnica o desenvolvimento é realizado de maneira indutiva, partindo-se do particular para se chegar ao geral. Nela as decisões de mais baixo nível são tomadas primeiro.

A estrutura do software é elaborada a partir dos aspectos mais detalhados, ou seja, dos aspectos de mais baixo nível, que comporão gradativamente as abstrações de mais alto nível.

. **Técnica "outside-in"** - é semelhante à técnica "top-down". A diferença encontra-se na direção de detalhamento do produto, que na técnica "outside-in" é definida de fora do software (o que o usuário vê) para dentro desse (a implementação).

. **Técnica "outside-out"** - esta técnica é um corolário da técnica "outside-in". Nela as decisões relacionadas à

implementação do software são tomadas antes das decisões relativas aos seus aspectos externos.

. **Técnica "most-critical-element-first"** - nesta técnica as decisões que afetam as partes mais críticas do sistema são tomadas primeiro. O critério para a tomada de decisões é fazer com que os parâmetros críticos sejam todos satisfeitos.

O critério **técnicas construtivas utilizadas**, portanto, avalia que técnicas construtivas são previstas no método.

- Tipo de instrumentos notacionais

Os instrumentos tornam possível o uso de um método. Instrumentos notacionais podem ser definidos como sendo linguagens que possibilitam a construção de um software. Os métodos utilizam essas linguagens para gerar modelos do produto a ser desenvolvido, ou seja, para construir as especificações do produto.

Existem vários tipos de linguagens que podem ser utilizadas pelos métodos para desenvolvimento de software. Neste trabalho são levadas em consideração as seguintes categorias de instrumentos notacionais:

. **Instrumento para modelagem da estrutura de funções** - auxilia na identificação e representação das funções executadas pelo software através de uma estrutura hierárquica e modular, onde as funções de mais alto

nível são decompostas em funções mais específicas nos níveis de abstração com um maior detalhamento.

- **Instrumento para especificação textual de funções** - auxilia na descrição dos processos a serem executados por um software.

- **Instrumento para especificação detalhada de funções** - auxilia na especificação de elementos críticos ou ambíguos de uma função.

- **Instrumento para modelagem de fluxo de dados** - auxilia na especificação do problema e na definição do processamento a ser realizado a partir dos fluxos de dados que fluem através do sistema.

- **Instrumento para modelagem de fluxo de controle** - possui a capacidade de decompor um problema a partir do fluxo de controle dos procedimentos.

- **Instrumento para modelagem de estrutura de dados** - auxilia na decomposição do problema e na especificação das atividades a serem executadas a partir da estrutura dos dados de entrada e saída.

- **Instrumento para modelagem de relacionamentos entre dados** - modela os dados de um sistema em termos de entidades e define os relacionamentos existentes entre essas entidades.

- **Instrumento para definição de elementos do modelo** - auxilia na definição de todos os elementos a serem utilizados pelo sistema.

. **Instrumento para modelagem de estados** – possibilita a especificação do comportamento, isto é, dos estados que um sistema pode assumir.

. **Instrumento para modelagem de objetos** – permite a decomposição do problema a partir do conceito de objeto, classe, herança e troca de mensagens.

. **Instrumento para modelagem de concorrência e sincronização** – possui a capacidade de modelar problemas que envolvam os conceitos de concorrência e sincronização.

. **Instrumento para definição de banco de dados** – envolve recursos para a definição de bancos de dados.

. **Instrumento para elaboração de código** – auxilia na codificação de procedimentos do sistema.

. **Instrumento para representação do conhecimento** – auxilia a representar o conhecimento.

. **Instrumento para modelagem de diálogo** – auxilia na especificação de diálogos do sistema.

O critério **tipo de instrumentos notacionais** avalia quais desses instrumentos notacionais fazem parte do método sob consideração e podem ser usados para modelar o produto a ser desenvolvido.

– **Forma de expressão dos instrumentos notacionais**

Os instrumentos notacionais são as linguagens usadas na construção de produtos de software ao longo do seu

processo de desenvolvimento. A partir de um estudo feito em (JONES, 1979) e (JONES, 1981), foram observadas diferentes formas nas quais as linguagens podem ser expressas.

As linguagens utilizadas pelos métodos podem ser classificadas, segundo a forma aplicada para modelar as especificações do software, nas seguintes categorias:

. **Linguagens expressas por meio de cadeias de caracteres** - são linguagens que utilizam cadeias de símbolos, que podem ser alfabéticos, numéricos ou especiais, para efetuar a comunicação. Esse tipo de linguagem está subdividido nas seguintes categorias:

- **Linguagem natural sem restrições** - é a linguagem que utiliza o vocabulário e as construções gramaticais usadas para escrita e conversação do dia-a-dia.

- **Linguagem natural com sintaxe e semântica restritas** é a linguagem natural com certas restrições na sintaxe e na semântica, tais como redução de adjetivos, de advérbios e de construções verbais, e utilização de nomes e verbos com significado pré-definido. O objetivo dessas restrições é remover os elementos da linguagem natural que causam problemas de ambigüidades.

- **Linguagem natural acrescida de simbologia** - este tipo de linguagem, além de possuir as mesmas restrições sintáticas e semânticas da categoria descrita acima, utiliza símbolos especiais, tais como símbolos matemáticos, símbolos booleanos e

símbolos metalingüísticos de Backus-Naur para representar o problema.

. **Linguagens com sintaxe/semântica posicionais** - são linguagens formadas por textos expressos normalmente em linguagem natural e sobrepostos em campos geométricos. A sintaxe desse tipo de linguagem é determinada em parte pela disposição geométrica do texto na tabela ou matriz e a semântica muda conforme essa posição geométrica. Essas linguagens buscam mostrar o relacionamento entre elementos. São, normalmente, expressas por tabelas de decisões, árvores de decisões ou matrizes.

. **Linguagens gráficas** - são linguagens que podem ser consideradas como linguagens híbridas pois são expressas por meio de elementos gráficos e elementos da linguagem natural associados a esses elementos gráficos. Esse tipo de linguagem pode ser subdividido em:

- **Linguagem com gráficos em forma livre e texto em linguagem natural** - este tipo de linguagem baseia-se em elementos gráficos e em textos expressos em linguagem natural. Os textos estão associados aos elementos gráficos. A forma gráfica dos símbolos é mais ou menos arbitrária, não derivando de nenhuma teoria. Os elementos gráficos geralmente possuem uma quantidade limitada de conteúdo semântico, sendo utilizados como recipientes para os elementos textuais.

- **Linguagem com gráficos derivados de princípios teóricos e texto em linguagem natural** – neste tipo de linguagem os elementos gráficos são derivados de alguns princípios, como teoria dos conjuntos, conceitos de programação estruturada e teoria de redes. A representação visual desses elementos gráficos é dotada de um alto conteúdo semântico.

- **Linguagem com capacidade de dar um "zoom"** – este tipo de linguagem permite dar um "zoom" de um determinado ponto da descrição para diversos níveis de detalhe, ou seja, tanto para um nível com maior ou menor quantidade de detalhes. A linguagem possibilita uma visão mais específica ou mais geral da descrição do problema.

O critério **forma de expressão dos instrumentos notacionais** avalia, portanto, em qual desses tipos de linguagens se enquadram os instrumentos propostos no método.

- **Grau de formalidade dos instrumentos notacionais**

O grau de formalidade de um produto depende do grau de formalidade da linguagem utilizada em sua construção. Segundo (BOEHM, 1980), as especificações de um software podem ser expressas através de linguagem natural, linguagens semi-formais ou linguagens formais.

As especificações informais escritas em linguagem natural não necessitam de um treinamento específico para sua construção e leitura. A linguagem natural, entretanto,

pode causar problemas de ambigüidade, dificultando o entendimento e a comunicação entre desenvolvedores e usuários.

As especificações semi-formais são expressas por meio de linguagens formatadas ou semi-formais. Essas linguagens possuem uma sintaxe padronizada, o que diminui a ambigüidade das especificações. De forma geral, esse tipo de linguagem requer um nível médio de treinamento para construção e leitura de especificações.

A necessidade de especificar sistemas complexos e com um alto rigor de precisão faz com que seja necessária a utilização de linguagens formais. Essas linguagens oferecem um alto rigor matemático, possibilitando a verificação matemática das especificações produzidas.

Apesar da precisão das linguagens formais, é necessário muito treinamento para seu entendimento e utilização. As linguagens formais possibilitam a eliminação quase que total de ambigüidades e imprecisões. Pode-se dizer que quanto mais estáveis forem os requisitos e maior a necessidade de inexistência de erros, mais formal deverá ser a linguagem aplicada no desenvolvimento da especificação do software.

A diferença básica entre linguagens naturais, semi-formais e formais está na forma de definição dessas linguagens. Os aspectos de estrutura e significado devem ser observados ao se avaliar a forma de definição de uma linguagem. O aspecto de estrutura denomina-se sintaxe e

trata da combinação dos signos, sem considerar o seu significado. Já o aspecto do significado é chamado de semântica e trata do significado dos signos (MENDES, 1989).

Quanto ao grau de formalidade de suas definições, as linguagens podem ser divididas em formais e semi-formais. MENDES (1989) afirma que uma linguagem é formal se ela possui sintaxe e semântica rigorosamente definidas. Uma linguagem semi-formal pode ou não possuir sintaxe bem definida, mas certamente não possui semântica precisa.

O critério **grau de formalidade dos instrumentos notacionais** é definido como sendo a característica que avalia o grau de precisão na definição da sintaxe e semântica das linguagens de um método, isto é, avalia se as linguagens são formais, semi-formais ou informais.

III.3.1.3 O FATOR FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO

O processo de aprendizagem de um método consiste em absorver não apenas a sintaxe das linguagens que ele utiliza para a construção de modelos, mas também as regras de funcionamento do método e inclusive dessas linguagens. Além disso, o desenvolvedor terá de aprender o funcionamento das ferramentas que automatizam o método.

É preciso entender, sobretudo, os mecanismos utilizados pelo método para o desenvolvimento de uma solução para o problema.

O uso de um método na construção de produtos de software requer, portanto, uma etapa inicial de aprendizagem e experiência. As pessoas envolvidas no desenvolvimento do software requerem um determinado grau de conhecimento sobre o método que possibilite a sua correta utilização.

O aprendizado de um método, bem como o seu conseqüente uso podem ser dificultados por vários fatores, tais como a necessidade de conhecimentos específicos para a utilização do método ou a deficiência de documentação sobre o método.

O **fator facilidade de aprendizado para construção** avalia as características inerentes ao método que permitam aos desenvolvedores atingir o grau de conhecimento sobre o método que possibilite o seu uso correto para construir produtos de software.

Este fator é avaliado através dos critérios **documentação para construção, exigência de formação para construção e existência de instrumentos cognitivos.**

– Documentação para construção

O aprendizado de um método está relacionado com a existência e a disponibilidade de documentação sobre esse método. Para que se possa aprender a usar um método, é indispensável a existência de textos, de manuais ou de outro tipo de material didático que auxilie no aprendizado do método, bem como a disponibilidade desse material.

O critério **documentação para construção** avalia, portanto, o tipo e a disponibilidade do material existente sobre o método, através do qual seja possível obter informações de como utilizá-lo para construir produtos de software.

- Exigência de formação para construção

Os aspectos de abordagem, as técnicas, os instrumentos e as regras para uso de um método estão, de maneira geral, fundamentados em um conjunto de princípios teóricos. Em alguns casos, essa teoria básica é complexa e requer, dos usuários de um método, uma série de conhecimentos prévios (por exemplo: matemática, linguagens formais) que muitas vezes não é compatível com o nível de formação dos profissionais disponíveis no mercado.

O critério **exigência de formação para construção** avalia o nível prévio de formação necessário para se aprender a utilizar o método para construir produtos de software.

- Existência de instrumentos cognitivos

Os instrumentos cognitivos têm por objetivo auxiliar aos usuários do método (desenvolvedores, usuários finais, etc) no aprendizado do funcionamento do método, bem como de suas ferramentas. São adequados para pessoas com pouca experiência no uso do método e de suas ferramentas. Como exemplo de instrumentos cognitivos podem ser citados tutores e assistentes especialistas.

O critério **existência de instrumentos cognitivos** avalia, portanto, a existência no método de recursos que facilitem o seu aprendizado.

III.3.1.4 O FATOR FACILIDADE DE USO

O fator **facilidade de uso** avalia em que medida um desenvolvedor, que conheça o método e entenda o problema a ser resolvido, pode construir o produto sem demasiada dificuldade. A facilidade de uso de um método está diretamente associada à existência e à qualidade de ferramentas e à facilidade de realizar alterações a medida que o entendimento do problema aumentar.

O fator **facilidade de uso** é avaliado através dos seguintes critérios: **existência de apoio automatizado, automatizabilidade e extensibilidade.**

- **Existência de apoio automatizado**

A automação do processo de desenvolvimento através do uso de ferramentas tem sido vista como uma solução para a busca do aumento da produtividade e da construção de software de melhor qualidade (PRICE, 1987). Vários aspectos podem ser citados como contribuições importantes para considerarmos as ferramentas como uma solução para essa busca. Dentre eles podem ser citados os seguintes (FISHER, 1990):

. ferramentas podem reduzir vários dos problemas relacionados à construção de software através da

geração automática de partes do produto a partir de suas especificações;

- grande parte dos erros são encontrados nas etapas de codificação e testes. A maioria dessas falhas, contudo, são originadas nas fases de análise e projeto. A utilização de ferramentas que auxiliam na construção de especificações ajuda a reduzir os erros e a detectá-los nos estágios iniciais do desenvolvimento;

- o uso de ferramentas de análise e projeto incentiva a construção de especificações, evitando a tendência de se passar para a fase de codificação dos programas antes do problema ter sido completamente entendido e especificado;

- ferramentas para análise e projeto podem reduzir o esforço e os custos das fases finais do desenvolvimento;

- a utilização de ferramentas tende a reduzir o tempo empregado para a construção de produtos de software;

- ferramentas facilitam a documentação dos produtos e a atualização das especificações geradas;

- o uso de ferramentas torna mais agradável a execução de tarefas para construção de software. É mais interessante desenvolver software usando recursos automatizados do que manualmente.

Em (MCCLURE, 1989) são encontrados exemplos de empresas cuja experiência comprovou que o uso de ferramentas ajuda de forma significativa a:

- aumentar a produtividade;
- aumentar a qualidade dos produtos;
- construir especificações que realmente satisfaçam os usuários;
- cumprir os cronogramas estabelecidos;
- reduzir, consideravelmente, os esforços e as despesas com as tarefas de manutenção do software;
- reutilizar informações anteriormente produzidas;
- melhorar o trabalho de especificar requisitos;
- possibilitar um aumento da criatividade dos desenvolvedores, tornando-os livres de tarefas repetitivas, tais como documentação e geração de código;
- incentivar o uso de métodos para o desenvolvimento de software.

O critério **existência de apoio automatizado** avalia se o método, sob consideração, possui ferramentas.

- **Automatizabilidade**

Uma das razões para que os métodos, que vêm sendo propostos, não atendam a seus objetivos é a ausência de um apoio automatizado. As ferramentas utilizadas em um ambiente de desenvolvimento de software não podem ser mais vistas como um artigo de luxo ou algo supérfluo. O suporte automatizado oferecido pelas ferramentas é algo muito

importante e um dos elementos indispensáveis a um método cujo objetivo é desenvolver software (AGUIAR, 1987).

Alguns métodos são completamente automatizados, outros são totalmente manuais ou possuem alguns instrumentos que podem ser implementados em ferramentas. É, portanto, importante avaliar se os instrumentos presentes em um método podem ser automatizados.

O critério **automatizabilidade** avalia os instrumentos do método que podem ser automatizados através de ferramentas.

- **Extensibilidade**

A construção de um produto de software perfeito e que nunca sofrerá uma única alteração é uma utopia. A realidade é bem diferente. Vários fatores, como por exemplo a característica dinâmica do próprio ambiente onde o software está inserido, influenciam na constante necessidade de mudanças.

As atividades para alteração do produto após sua implantação são denominadas de manutenção. Em (PRESSMAN, 1988) são identificados os seguintes tipos de manutenção:

- manutenção corretiva - onde são realizadas atividades para a correção dos erros encontrados no produto;
- manutenção adaptativa - este tipo de manutenção visa adaptar o produto a mudanças ocorridas no ambiente,

com por exemplo um novo tipo de hardware ou uma nova versão do sistema operacional;

. manutenção evolutiva - onde as tarefas de manutenção visam dotar o software de novas capacidades e funções.

As alterações de um software podem, também, ocorer antes de ele ter sido implantado. É difícil especificar todos os detalhes no início do processo de desenvolvimento. As especificações podem evoluir ao longo desse processo. Isso permite a inclusão de novos requisitos, a eliminação ou a modificação de aspectos que não satisfazem ao propósito final do usuário.

Assim sendo, é necessário que os métodos utilizados ao longo do desenvolvimento permitam que as alterações necessárias possam ser feitas de uma forma viável, isto é, que uma pequena alteração em um conceito resulte em uma alteração similarmente pequena em sua descrição (ROCHA, 1981).

Extensibilidade é o critério que avalia em que grau uma alteração no entendimento do problema impacta no registro do mesmo, ou seja, indica o esforço requerido para modificar as representações já existentes do mesmo, quando algum aspecto de seus requisitos é alterado.

III.3.2 AVALIAÇÃO SEGUNDO O OBJETIVO UTILIZABILIDADE PARA AVALIAÇÃO

A tarefa de controlar a qualidade de software não é algo fácil, pois qualidade não é uma característica que possa simplesmente ser incorporada ao produto final. De acordo com (ARTHUR, 1985), a qualidade deve ser introduzida gradativamente no software no decorrer de seu desenvolvimento. A qualidade é projetada em conjunto com o produto, não sendo imposta ao final da construção (PRESSMAN, 1988).

Deve-se definir, a priori, o nível de qualidade que se deseja obter (especificação dos requisitos de qualidade) e tentar atingi-lo por meio de um processo de construção que vise esse propósito. É necessário buscar esse nível desejado desde as primeiras etapas do ciclo de vida do software, estando ciente de que a participação de todos os desenvolvedores é fundamental para atingir essa meta.

A qualidade de um software depende da satisfação obtida pelo usuário com relação ao serviço fornecido pelo software e inclusive com a facilidade do seu uso (CLUNIE, 1987). Assim, pode-se afirmar que qualidade de software é um conjunto de propriedades a serem satisfeitas em determinado grau, de modo a atender às necessidades de seus usuários (ROCHA, 1987).

É sabido que existem vários tipos de usuários de um software (mantenedores, operadores, inspetores, etc) e que eles anseiam graus de qualidade que podem ser diferentes. O

produto final deve ter em mira o atendimento das necessidades e expectativas de cada um deles. A qualidade de um software é também algo relativo, dependendo das necessidades do software a ser desenvolvido (ARTHUR, 1985).

Construir um software de boa qualidade não é algo trivial. O uso de métodos ao longo do processo de desenvolvimento de software pode ser visto como uma forma de tentar buscar essa qualidade e avaliar se está sendo atingida.

As diferentes representações de um software (especificação de requisitos, especificação de projeto, etc), geradas pelo método, devem ser avaliadas para garantir o desenvolvimento de um produto de boa qualidade e que atenda às necessidades que o levaram a ser construído. É desejável que os métodos possuam recursos que permitam avaliar a qualidade dos produtos gerados através de sua utilização.

O objetivo **utilizabilidade para avaliação** está relacionado com as características de um método que permitem a sua utilização na avaliação da qualidade dos produtos gerados.

Essas especificações devem ser avaliadas tanto no que se refere à sua forma quanto ao seu conteúdo (ROCHA, 1987). Este objetivo se realiza através dos fatores **verificabilidade** e **validabilidade**.

III.3.2.1 O FATOR VERIFICABILIDADE

O conceito de verificação é definido por BOEHM (1981) através da seguinte pergunta: "Estamos construindo certo o produto?". A verificabilidade, de acordo com (CLUNIE, 1987), preocupa-se com a qualidade do processo de geração das representações de um produto.

É interessante que um método para desenvolvimento de software tenha a capacidade de avaliar a forma das representações do software por ele geradas. Assim sendo, o fator **verificabilidade** avalia as características do método que permitem sua utilização para avaliar a forma do produto. Este fator é avaliado através dos critérios **existência de normas de qualidade** e **existência de ferramentas para verificação**.

- **Existência de normas de qualidade**

A qualidade do produto final está diretamente relacionada à qualidade de suas representações intermediárias (especificação de requisitos, especificação de projeto, etc). A existência de representações com alto nível de qualidade está relacionada, por sua vez, com o método utilizado para a sua construção.

Existência de normas de qualidade é o critério que avalia se o método, sob consideração, define normas e atributos de qualidade, que auxiliem na construção de representações.

- **Existência de ferramentas para verificação**

O critério **existência de ferramentas para verificação** avalia o apoio automatizado oferecido pelo método para verificação das normas de qualidade.

III.3.2.2 O FATOR VALIDABILIDADE

Segundo BOEHM (1981), o conceito de validação pode ser entendido através da pergunta: "Estamos construindo o produto certo?". O processo de validar centra-se na avaliação do conteúdo do produto, ou seja, no grau em que esse atinge os objetivos que motivaram seu desenvolvimento.

O fator **validabilidade** está relacionado à capacidade de um método poder avaliar aspectos relacionados à confiabilidade conceitual do produto. O fator é alcançado através dos critérios **compreensividade** e **existência de ferramentas para validação**.

- **Compreensividade**

A validabilidade de um produto depende, muitas vezes, da possibilidade de participação do usuário em sua avaliação. Essa participação depende do entendimento que o usuário tem do produto final e de suas especificações. É necessário que o método de desenvolvimento possibilite ao usuário compreender as representações do produto, geradas durante o processo de construção do software.

O critério **compreensividade** avalia os aspectos relacionados à facilidade de leitura dos documentos

produzidos, ou seja, qual a facilidade que uma pessoa (não especialista no método) tem de entendê-los (WASSERMAN, 1982).

- Existência de ferramentas para validação

O critério existência de ferramentas para validação avalia o apoio automatizado oferecido pelo método para auxiliar na validação do produto gerado.

III.3.3 AVALIAÇÃO SEGUNDO OBJETIVO UTILIZABILIDADE PARA GERÊNCIA

As atividades técnicas e gerenciais são igualmente importantes para o sucesso de um projeto de software. As atividades gerenciais controlam os recursos e o ambiente onde as atividades técnicas ocorrem. Um dos principais objetivos da gerência é garantir que o produto de software seja entregue dentro do prazo estimado, de acordo com os custos estabelecidos e que esse produto apresente os atributos funcionais e qualitativos requeridos pelo usuário (FAIRLEY, 1986).

As atividades de gerência estão relacionadas com o controle das atividades de construção. Visam estabelecer um plano de desenvolvimento que possa ser executado a fim de construir um produto que atenda às necessidades do usuário, de acordo com as restrições e os requisitos estabelecidos.

Uma das principais características a serem avaliadas em um método para desenvolvimento de software é a sua

capacidade de oferecer suporte à gerência do processo de desenvolvimento.

O objetivo **utilizabilidade para gerência** é avaliado através dos fatores **suporte para planejamento do projeto** e **suporte para acompanhamento do projeto**.

III.3.3.1 O FATOR SUPORTE PARA PLANEJAMENTO DO PROJETO

O planejamento de um projeto está relacionado à definição das tarefas intermediárias que proporcionarão o alcance dos objetivos do projeto. FAIRLEY (1986) afirma que a falta de planejamento é uma das principais causas para o atraso nos cronogramas, os custos ultrapassados, a baixa qualidade dos produtos e os altos custos de manutenção do software. Pode-se dizer, inclusive, que o planejamento do software é uma estrutura que permite fazer estimativas de recursos, custos e cronograma (PRESSMAN, 1988).

Em (PRESSMAN, 1988) são citadas várias atividades associadas ao planejamento, dentre as quais destacam-se as seguintes:

- definição das etapas do desenvolvimento, ou seja, do ciclo de vida, especificando as tarefas a serem realizadas e os produtos a serem gerados;
- estabelecimento dos recursos humanos, de software e de hardware necessários à realização do projeto;
- cálculo da produtividade e das estimativas de custo para o projeto, e,

. definição de cronogramas para o projeto.

Os métodos devem oferecer suporte para planejamento através da gerência do seu próprio funcionamento e do processo de construção de modelos do software a ser desenvolvido.

Suporte para planejamento do projeto é o fator que avalia um método quanto à sua capacidade de auxiliar no planejamento das tarefas a serem realizadas durante o processo de desenvolvimento de software. Este fator é avaliado através dos critérios **suporte para estimativa de custos, suporte para definição do cronograma e suporte para estimativa de recursos.**

- **Suporte para estimativa de custos**

O critério **suporte para estimativa de custos** avalia se o método fornece auxílio para estimar os custos do desenvolvimento do projeto.

- **Suporte para definição do cronograma**

O critério **suporte para definição do cronograma** avalia se o método fornece auxílio para estimativa de tempo de desenvolvimento ajudando na elaboração do cronograma.

- **Suporte para estimativa de recursos**

O critério **suporte para estimativa de recursos** avalia se o método fornece auxílio para estimar os recursos de hardware e software necessários para desenvolver o produto.

III.3.3.2 O FATOR SUPORTE PARA ACOMPANHAMENTO DO PROJETO

O fator **suporte para acompanhamento do projeto** avalia se o método oferece monitoração do processo de desenvolvimento do produto.

O fator é avaliado através dos critérios **suporte para acompanhamento de custos, suporte para acompanhamento de cronograma** e **suporte para controle de versões.**

- Suporte para acompanhamento de custos

O critério **suporte para acompanhamento de custos** avalia se o método fornece auxílio para realizar o acompanhamento dos custos do projeto durante o seu desenvolvimento.

- Suporte para acompanhamento de cronograma

O critério **suporte para acompanhamento de cronograma** avalia se o método fornece auxílio para realizar o acompanhamento do cronograma durante o desenvolvimento do projeto.

- Suporte para controle de versões

Durante a utilização de um método são elaborados vários modelos, de forma manual ou automatizada. As alterações de um software podem ocasionar mudanças nos modelos e para facilitar a gerência desses modelos é necessário haver uma forma de controlar as versões alteradas.

O critério **suporte para controle de versões** avalia se um método oferece recursos para gerenciar as diversas representações do software geradas durante o seu processo de desenvolvimento.

III.4 CONCLUSÃO

Neste capítulo foi apresentada uma proposta para avaliação de métodos para o desenvolvimento de software, com base no Método para Avaliação da Qualidade de Software definido em (ROCHA, 1987).

No próximo capítulo são descritos alguns métodos para desenvolvimento de software, selecionados dentre as propostas existentes hoje na literatura.

CAPÍTULO IV

DESCRIÇÃO DE MÉTODOS PARA O DESENVOLVIMENTO DE SOFTWARE

IV.1 INTRODUÇÃO

A quantidade de informações envolvidas no processo de desenvolvimento de software é muito grande e exige a aplicação de métodos que organizem essas informações e disciplinem o próprio processo de desenvolvimento.

O uso de métodos facilita a comunicação entre desenvolvedores e usuários, auxilia no entendimento e na decomposição do problema, torna mais fácil a definição dos limites desse problema e provê recursos para elaborar uma solução viável e satisfatória. Os métodos auxiliam na elaboração das etapas mais complexas e difíceis do processo de desenvolvimento de software (DAVIS, 1990).

O presente capítulo apresenta a descrição dos métodos que serão avaliados neste trabalho. Foram escolhidos vinte e três métodos dentre a grande quantidade de propostas existentes na literatura. Na descrição de cada método, procurou-se dar uma idéia geral da sua filosofia, mostrando as suas principais características, os seus instrumentos, as ferramentas que os automatizam e os passos necessários para sua aplicação na construção de produtos de software.

IV.2 "BOP PROTOTYPING SYSTEM" (AARAM, 1984)

"BOP Prototyping System" (BOP) tem por objetivo apoiar a especificação de sistemas de informação interativos, através da construção de protótipos, especialmente da interface com o usuário e das características funcionais do sistema.

É o resultado do projeto "Base Operations in Factory Management Systems", desenvolvido pelo Laboratório de Engenharia de Produção do NTH-SINTEF em cooperação com a Universidade Técnica da Dinamarca. O objetivo do projeto era formular e testar operações básicas de sistemas de gerência de produção a serem utilizadas por ferramentas no desenvolvimento de novos sistemas.

O Sistema BOP não lida apenas com funções específicas a sistemas de gerência de produção. A maior parte das funções de BOP são para o manuseio de informações gerais. Além disso, ele não implica o uso de uma determinada configuração de software e hardware para o novo sistema. Assim, BOP é adequado para a especificação de sistemas de informação em geral.

O Sistema BOP é composto de:

- Banco de Dados Relacional - em BOP um banco de dados relacional, na terceira forma normal, é usado para armazenar os dados do sistema e as relações entre eles. As descrições desses objetos são armazenadas em dicionários. Existem dois tipos de dicionários no

Sistema BOP: o dicionário de atributos e o dicionário de relações.

- Conjunto de Operações Básicas - este conjunto é composto por funções macro, pré-programadas, que representam operações lógicas sobre os dados do sistema. Essas funções são encontradas geralmente em sistemas de gerência de produção e estão implementadas em BOP na linguagem APL. Conceitualmente dividem-se em uma hierarquia de três níveis de funções:

  . primeiro nível, consiste em um conjunto de funções matemáticas relacionadas à representação do banco de dados. Normalmente, as funções nesse nível são gerais, sendo uma extensão de um grupo de funções de APL. Elas não são utilizadas de forma direta nas transações dos usuários;

  . segundo nível ou nível de gerência de informações, onde se encontra a maior parte das operações básicas. As funções típicas desse nível são funções de manuseio de telas e relatórios, e funções de recuperação e atualização do banco de dados. Elas manuseiam informações gerais e são freqüentemente utilizadas nas transações dos usuários;

  - terceiro nível ou nível de gerência de produção, onde se encontram as operações básicas que manuseiam problemas de gerência de produção bem definidos.

- Ferramentas para Geração de Protótipos - estas ferramentas dispõem de funções que auxiliam na

construção do protótipo. Todas as funções do sistema BOP estão disponíveis através de menu do sistema e permitem aos usuários descreverem os objetos do novo sistema e gerarem transações simples, sem qualquer conhecimento de programação. No menu do Sistema BOP são fornecidas as seguintes funções:

- manutenção de objetos do sistema;
- geração de transações;
- documentação do sistema;
- análise do sistema;
- acesso relacional ao banco de dados;
- manutenção de menu;
- manutenção de transação, e,
- módulo de aplicação.

A figura IV.2.1 mostra como exemplo uma tela para geração de transações.

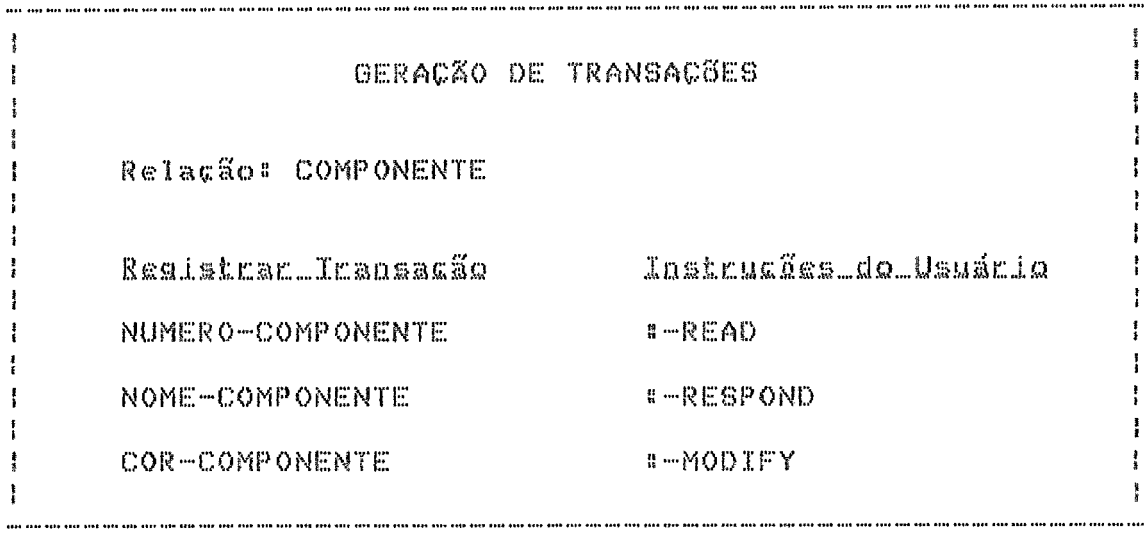


Figura IV.2.1: Geração de Transação no Sistema BOP

Fonte: (AARAM, 1984)

As ferramentas de desenvolvimento, as operações básicas pré-programadas, o banco de dados relacional e os dicionários formam, portanto, uma plataforma para o desenvolvimento de protótipos.

Segundo a abordagem de prototipagem utilizada em BOP, o protótipo não é descartável, sendo visto inicialmente como uma especificação detalhada do novo sistema. O desenvolvimento do protótipo dá-se de forma incremental, começando pelas funções centrais e sendo expandido, gradativamente, até que abranja todo o sistema. Isso é possível graças à facilidade de modificação e expansão do protótipo oferecida por BOP.

A prototipagem é considerada em BOP como um processo interativo, onde o protótipo é construído a partir das descrições do usuário e em cada passo desse processo é possível retornar, quando necessário, aos passos anteriores a fim de modificar as especificações definidas.

Uma das primeiras tarefas a ser realizada para construção de protótipos em BOP é descrever os atributos e relações, isto é, os elementos de dados do novo sistema e como eles se relacionam entre si. Essa tarefa é executada através da opção "manutenção de objetos do sistema", fornecida pelo menu principal. As descrições dos atributos e relações definem o banco de dados do protótipo, que serve de base para a programação e teste das transações do produto final.

O Sistema BOP permite a geração automática de transações do usuário, tais como transações para gravação, modificação, exclusão ou acesso ao banco de dados. Essas transações podem ser incluídas em um menu do usuário e serem alteradas quando for necessário.

O Sistema BOP foi implementado na linguagem APL. O protótipo gerado, entretanto, pode ser executado por pessoas que não tenham nenhum conhecimento prévio dessa linguagem.

As experiências com o uso do Sistema BOP na construção de software mostram que o protótipo desenvolvido pode ser utilizado como um recurso para os usuários decidirem e testarem as características funcionais do futuro sistema, além de se familiarizarem com o uso de sistemas interativos. BOP também foi utilizado para demonstração de produtos e ainda com propósitos educacionais por estudantes da Universidade Técnica da Noruega.

IV.3 "DESIGN APPROACH FOR REAL-TIME SYSTEMS" (GOMAA, 1984)

"Design Approach for Real Time Systems" (DARTS) é um método para análise e projeto de sistemas de tempo real. Foi desenvolvido por Hassen Gomaa no Laboratório de Desenvolvimento de Eletrônica Industrial da General Eletric.

DARTS pode ser considerado como uma extensão dos métodos "Structured Systems Analysis" (DEMARCO, 1978) e

"Structured Design" (PAGE-JONES, 1980), provendo uma abordagem para estruturar o sistema em tarefas e mecanismos de definição da interface entre essas tarefas. O método utiliza os seguintes instrumentos:

- Linguagem para Diagrama de Transição de Estados (DTE) - o DTE representa os possíveis estados que um sistema pode assumir e as condições que ocasionam as mudanças de estado. A notação gráfica da linguagem para DTE é composta dos seguintes elementos:

  . círculos - representam os estados, que mostram o comportamento do sistema, e,

  . setas - representam as transições de estados, que ocorrem em um tempo específico e são geradas por um evento derivado de uma determinada condição.

- Linguagem para Diagrama de Fluxo de Dados (DFD) - o DFD representa as funções executadas pelo sistema, o fluxo de dados existente entre essas funções e os depósitos onde os dados são armazenados.

- Dicionário de Dados - define os itens de dados contidos nos fluxos e nos depósitos de dados.

- Linguagem para Gráfico de Estruturas de Tarefas (GET) - o GET representa as tarefas concorrentes do sistema, com suas interfaces de comunicação e sincronização. O GET é gerado a partir da análise das funções do DFD, identificando-se aquelas funções que podem ser executadas de forma concorrente e as que são

seqüenciais. A linguagem para representar GET é composta pelos seguintes elementos, cuja notação gráfica pode ser vista na figura IV.3.1:

. tarefa - representa uma ou mais funções do DFD;

. interface de comunicação - controla a troca de mensagens entre as tarefas;

. interface de sincronização - indica a ocorrência de um evento que ocasiona a ativação de uma tarefa. No mecanismo de sincronização, a tarefa origem sinaliza um evento que ativa a tarefa destino;

. depósito de dados - onde os dados do sistema são armazenados, por uma ou mais tarefas, e,

. fluxo de dados - representa a passagem de dados entre as tarefas e os depósitos de dados.

- Linguagem para Gráfico de Estruturas (GE) - o GE representa a organização das tarefas em módulos e a interface entre cada um deles.

O processo de desenvolvimento de sistemas, segundo o método DARTS, é efetuado através de uma série de etapas, que abrangem as fases de análise e projeto. As etapas de DARTS são descritas a seguir:

- Definição dos estados do sistema - nesta etapa é construído o DTE, cujo exemplo pode ser visto na figura IV.3.2.

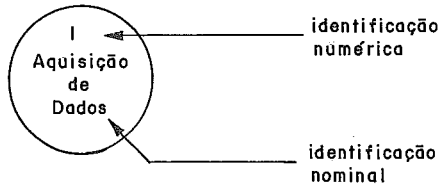
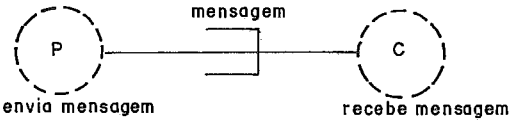
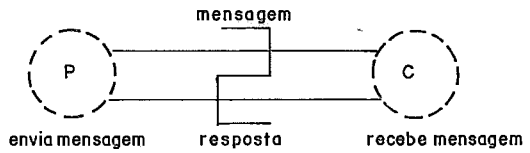
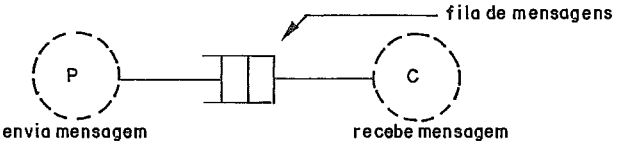
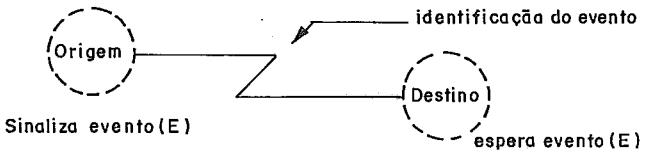
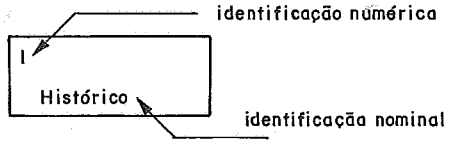
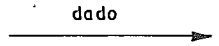
Nome do Elemento	Convenção Simbólica
a) Tarefa	
b) Comunicação por Mensagem	
c) Comunicação por Mensagem fortemente Acoplada	
d) Comunicação por Mensagem Frouxamente Acoplada	
e) Sincronização	
f) Depósito de Dados	
g) Fluxo de Dados	

Figura IV.3.1 : Notação da linguagem para Gráfico de Estruturas de Tarefas do Método DARTS
Fonte : (GOMAA , 1984)

- Análise do fluxo de dados do sistema - nesta etapa são identificadas as principais funções do sistema, sendo construídos o DFD e o Dicionário de Dados.

- Decomposição do sistema em tarefas - nesta etapa as tarefas são definidas a partir da análise dos processos do DFD, identificando-se, de acordo com um conjunto de critérios, quais os processos que podem ser executados de forma concorrente e quais são seqüenciais. O DFD é redesenhado, incluindo-se uma caixa ao redor de cada processo ou conjunto de processos que formem uma tarefa.

O método utiliza os seguintes critérios para auxiliar na identificação das tarefas: dependência de entrada/saída, função de tempo crítico, requisitos computacionais, coesão funcional, coesão temporal e periodicidade de execução. A figura IV.3.3 mostra um exemplo do DFD com a definição das tarefas.

- Definição das interfaces entre as tarefas - nesta etapa são identificados os tipos de módulos de interface entre as tarefas e é construído o GET. Em DARTS as interfaces entre as tarefas são manipuladas através da definição de duas classes de módulos:

  . Módulo de Comunicação de Tarefa (MCT) - responsável pelo controle de todos os tipos de comunicações entre as tarefas. O método DARTS suporta dois tipos distintos de MCT, o Módulo de Comunicação de Mensagem e o Módulo de Informação Escondida:

- Módulo de Comunicação de Mensagem, que suporta os seguintes tipos de comunicação:

  . Comunicação por mensagem fortemente acoplada, onde a tarefa produtora envia uma mensagem e fica esperando por uma resposta da tarefa consumidora;

  . Comunicação por mensagem fracamente acoplada, onde as tarefas produtora e consumidora constroem sua própria fila de mensagens e prosseguem a execução independentes do envio ou recebimento de respostas.

- Módulo de Informação Escondida, que define os depósitos de dados e o acesso a eles.

. Módulo de Sincronização de Tarefas (MST) - indica a ocorrência de um evento que ocasiona a ativação de uma tarefa. No mecanismo de sincronização, a tarefa origem sinaliza um evento que ativa a tarefa destino.

A figura IV.3.4 mostra um GET com a representação das interfaces entre as tarefas.

- Projeto das tarefas - nesta fase as tarefas são decompostas em uma estrutura modular. É elaborado o projeto da arquitetura, através da construção de um GE para cada tarefa identificada no GET, sendo que cada uma das tarefas representa um programa seqüencial.

Antes de desenhar o GE, deve-se decompor o DFD detalhando as funções de cada tarefa, até se alcançar um nível de detalhes suficiente para formular o GE.

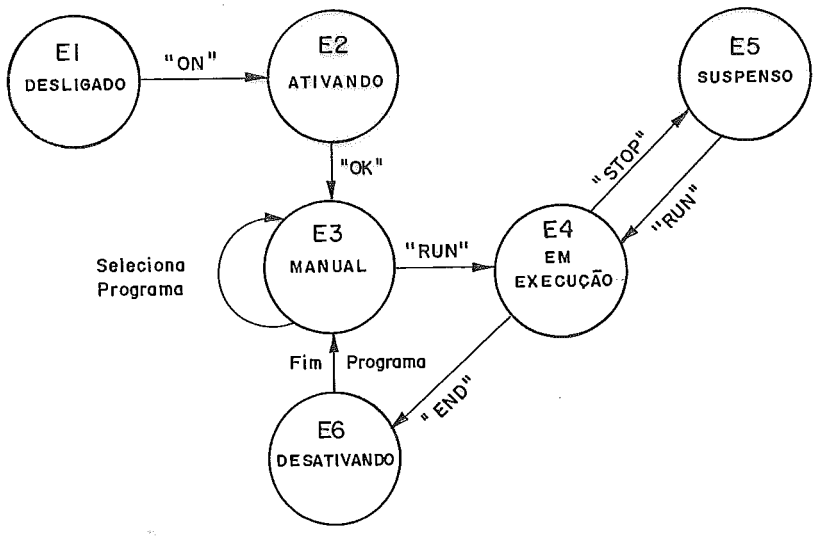


Figura IV.3.2: Diagrama de Transição de Estados
Fonte: (GOMAA , 1984)

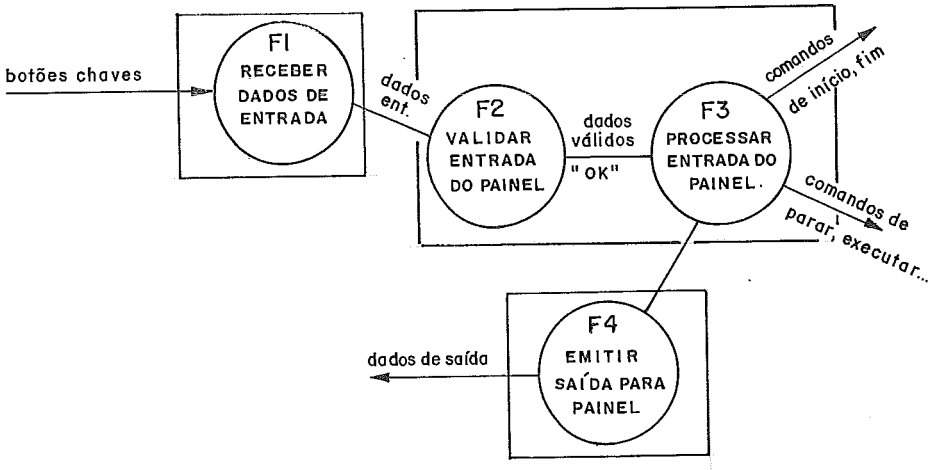


Figura IV.3.3: Identificação de Tarefas no Diagrama de Fluxo de Dados
Fonte : (GOMAA , 1984)

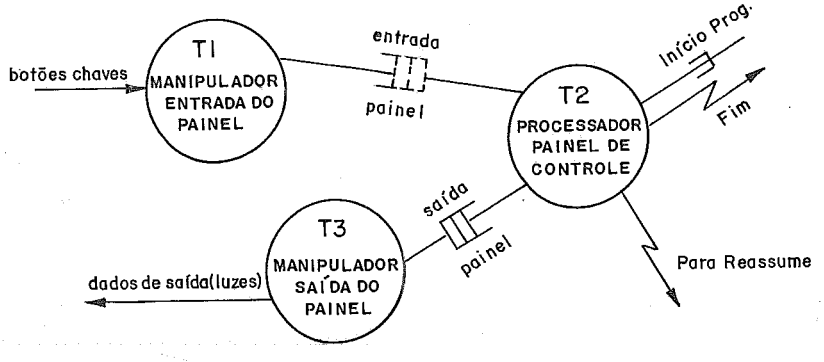


Figura IV.3.4: Gráfico de Estrutura de Tarefas
Fonte: (GOMAA , 1984)

Existem propostas para automatização de instrumentos do método DARTS. Como exemplo, podem ser citadas ferramentas desenvolvidas na COPPE/UFRJ: a ferramenta DARTS-DT (CAVALCANTE, 1988) automatiza o GET e a ferramenta DDDARTS (ANGULO, 1988) é a automação de um Dicionário de Dados para o método DARTS.

IV.4 "ENTITY-RELATIONSHIP MODEL" (FERNANDES, 1989), (ORR, 1989), (CHEN, 1990), (GANE, 1990), (YOURDON, 1990)

"Entity-Relationship Model" (ERM) é um método para modelagem lógica de dados, que permite construir um modelo da estrutura dos dados do sistema em termos de entidades e dos relacionamentos existentes entre elas. É utilizado para a construção do modelo conceitual das informações a serem armazenadas em um banco de dados, isto é, para o projeto lógico de banco de dados, a partir dos requisitos do usuário. Foi desenvolvido por Peter Chen.

O método ERM utiliza os seguintes conceitos básicos:

- Entidade - é uma classe de coisas do mundo real, um conceito, uma abordagem ou um evento que podem ser identificados de forma distinta em função de características comuns. Cada elemento pertencente a uma entidade é dito ser uma ocorrência da entidade.

- Relacionamento - é uma associação entre entidades, ou seja, uma interação entre entidades representando um fato ou situação da realidade.

- Atributo - é uma propriedade inerente a uma entidade ou relacionamento, não possuindo uma existência própria ou independente.

Existem vários tipos de relacionamentos entre as entidades. Dependendo do número de instâncias do relacionamento, ou seja, do grau ou multiplicidade, um relacionamento pode ser classificado em:

. Relacionamento um-para-um - neste relacionamento uma instância de uma determinada entidade A é associada a uma, e somente uma, instância de uma entidade B. Além disso, cada instância da entidade B deve ser associada da mesma forma a uma instância da entidade A. Esse tipo de relacionamento é identificado através do símbolo (1:1). No relacionamento um-para-um deve-se observar se as duas entidades são realmente distintas ou se elas podem ser unidas em uma única entidade;

. Relacionamento um-para-muitos - neste tipo de relacionamento cada instância de uma determinada entidade A é associada a uma e somente uma instância de uma entidade B e essa instância da entidade B é associada a uma ou várias instâncias da entidade A. O relacionamento um-para-muitos é indicado pelo símbolo (1:M).

. Relacionamento muitos-para-muitos - neste tipo de relacionamento cada instância de uma determinada entidade A é associada a uma ou mais instâncias de uma entidade B, e cada instância de B está relacionda a uma ou mais

instâncias da entidade A. O relacionamento muitos-para-muitos é indicado pelo símbolo (N:M).

Os relacionamentos entre entidades ainda podem ser enquadrados em um dos seguintes tipos especiais:

- relacionamento condicional ou opcional - ocorre quando existem instâncias das entidades que não participam do relacionamento;
- auto-relacionamento - ocorre quando uma entidade está relacionada consigo mesma;
- relacionamento múltiplo ou associativo - ocorre quando mais de duas entidades participam do relacionamento;
- relacionamento existência-dependente - ocorre quando uma entidade depende da existência de uma outra entidade.

Um tipo especial de entidades, representadas pelo ERM, são as entidades subtipos/supertipos. Elas são compostas por uma entidade (supertipo) interligada a uma ou mais de suas subcategorias (subtipos) por um mesmo relacionamento.

O método ERM utiliza como instrumento notacional a linguagem para Diagrama de Entidade-Relacionamento (DER). Existem, na literatura, várias propostas de notações para o DER. De maneira geral, as entidades são representadas através de retângulos e identificadas por um substantivo.

Os relacionamentos são identificados por verbos e a sua representação gráfica conta com algumas variações. Em determinadas notações para DER, os relacionamento são

representados por losangos com o nome do relacionamento. Em outras, ele é representado apenas por uma seta com sua denominação. A figura IV.4.1 mostra o exemplo de um DER escrito na notação de Chen.

A modelagem dos dados e o projeto de banco de dados, segundo o método ERM, podem ser feitos através dos seguintes passos:

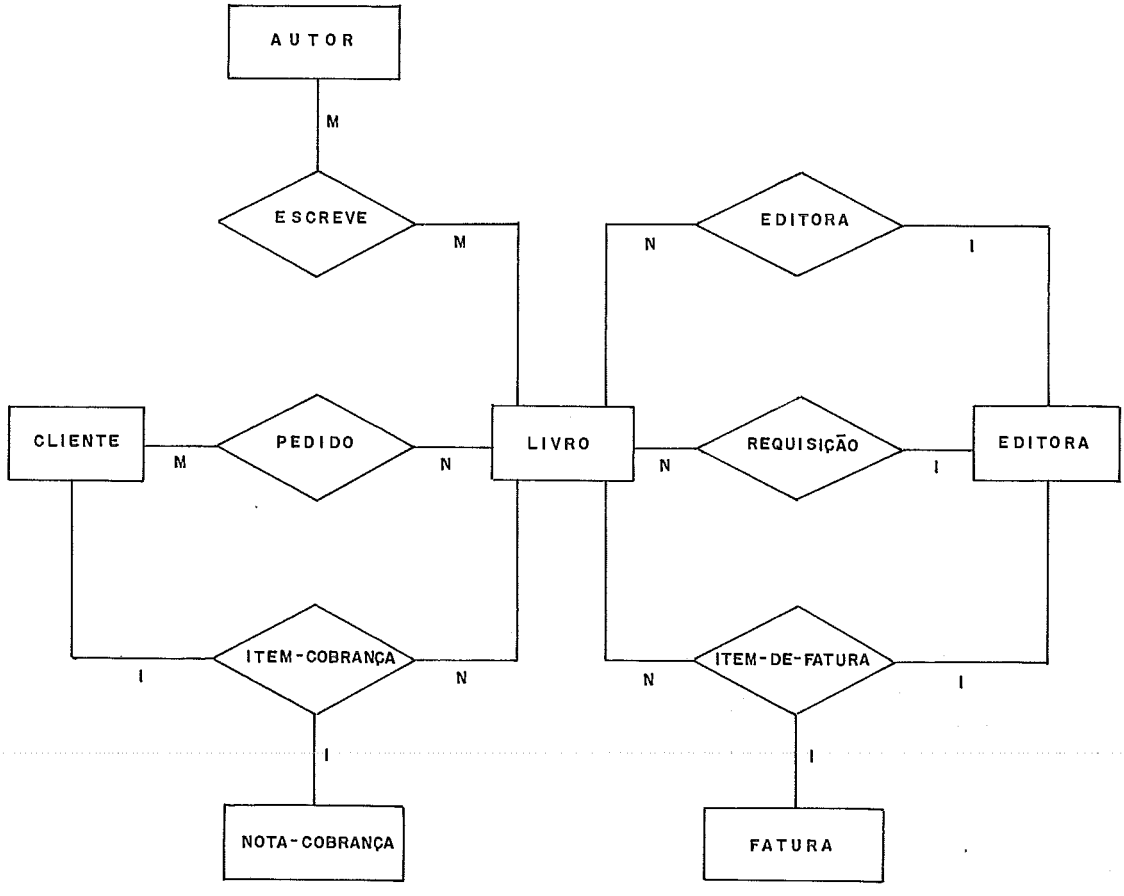


Figura IV.4.1: Diagrama de Entidade - Relacionamento
Fonte: (GANE, 1990)

- Construir o DER - neste passo são identificados os componentes do DER e a chave primária para cada entidade. A construção de um DER pode ser feita através das seguintes etapas:

  . elaborar uma lista contendo as principais entidades do problema;
  . identificar os possíveis relacionamentos existentes entre essas entidades;
  . determinar o grau de cada um dos relacionamentos;
  . identificar os atributos de cada elemento do DER, que podem ser expressos graficamente através de elipses com o nome do atributo;

- Converter o DER em estruturas de banco de dados;

- Desenvolver a aplicação baseando-se nas estruturas do banco de dados.

Embora o método ERM seja manual, existem várias ferramentas que oferecem um apoio automatizado para a construção de DERs. Essas ferramentas são produtos CASE que, em geral, automatizam instrumentos de outros métodos, mas que também apoiam o método ERM.

Como exemplo de ferramentas desenvolvidas no Brasil podem ser citadas PC-CASE do Instituto Brasileiro de Pesquisa em Informática (GANE, 1990) e FEGRES (Ferramentas Gráficas para Engenharia de Software) (TRAVASSOS, 1990) da COPPE/UFRJ.

Das ferramentas desenvolvidas no exterior podemos destacar as seguintes (DAVIS, 1990), (FISHER, 1990) e (GANE, 1990): Analyst/Designer Toolkit da Yourdon Inc., CorVision da Cortex Corp, Deft da DEFT INC., DESIGN/1 da Arthur Andersen & Co., ER-DESIGNER da Chen & Associates, ENVISION da Enriched Data Systems, EXCELERATOR da Index Technology, Information Engineering Workbench da Knowledge Ware Inc., MAESTRO da Softlab Inc., MULTI/CAM da AGS Management Systems, ProKit*WORKBENCH da McDonnell Douglas, Software Through Pictures da Interactive Development Environments, SYSTEM ENGINEER da LBMS, TEAMWORK da CADRE Technologies Inc. e THE DEVELOPER da ASYST Technologies.

O método ERM pode ser aplicado durante a fase de análise, em conjunto com outros métodos, com o objetivo de representar modelos para a especificação de requisitos. Um dos métodos mais utilizados com o ERM é o "Structured Systems Analysis" (DEMARCO, 1978) e (GANE, 1979). Os depósitos de dados do Diagrama de Fluxo de Dados são bons candidatos a serem entidades no ERM, fazendo com que o DER realce os relacionamentos existentes entre os depósitos de dados de um DFD.

IV.5 "ESSENTIAL SYSTEMS ANALISYS" (MCMENAMIN, 1991)

"Essential Systems Analysis" (ESA) é um método para especificação de requisitos de sistemas interativos, onde a modelagem do sistema é feita em termos de respostas dadas a eventos recebidos pelo sistema. O método tem por objetivo apoiar a elaboração de um modelo dos requisitos essenciais,

oferecendo estratégias para identificação e definição desses requisitos. Foi desenvolvido por Stephen McMenamin e John Palmer.

Os requisitos especificados pelo método ESA formam a essência do sistema e retratam todas as características que o sistema deve ter para alcançar seus objetivos, se ele for implementado com uma tecnologia perfeita. O uso de uma tecnologia perfeita, isenta de falhas, leva à definição de um sistema que não execute atividades que compensem as falhas humanas.

A essência de um sistema está dividida nos seguintes componentes:

. Atividades essenciais - compreendem todas tarefas a serem realizadas pelo sistema. Cada atividade essencial é iniciada por um estímulo e gera uma resposta planejada, através de um conjunto de ações efetuadas pelo sistema para executar a atividade. São divididas nos seguintes tipos:

- atividades fundamentais - executam tarefas que visam alcançar os objetivos do sistema. Essas atividades geram saídas para o mundo exterior;

- atividades custodiais - geram e mantém a memória essencial através da obtenção e armazenamento das informações necessárias a atividades fundamentais. Essas atividades geram atualizações na memória essencial.

. Memória essencial – é o conjunto de todos os dados a serem lembrados pelo sistema, isto é, dados produzidos pelo sistema ou recebidos do mundo exterior, que são utilizados pelas atividades fundamentais. A memória essencial também inclui declarações de como as atividades fundamentais devem fazer para acessar os dados armazenados e serve para manter as atividades essenciais ligadas entre si.

Os sistemas interativos percebem as mudanças do ambiente através de eventos. Os eventos podem ser considerados requisitos dos usuários e são de dois tipos:

. Eventos externos – são gerados por entidades no ambiente, que são entidades externas ao sistema;

. Eventos temporais – são gerados pela passagem do tempo, em intervalos determinados, que podem ser fixos ou relativos ao valor de algum dado.

A ocorrência de um determinado evento é informada ao sistema através de estímulos. O sistema responde a esses estímulos por meio de um conjunto de ações. A resposta a um determinado evento é o que caracteriza a execução de uma atividade essencial. Cada evento recebe sempre a mesma resposta, que é definida durante a análise do sistema. Assim, o método ESA é voltado para a especificação de sistemas de repostas planejadas, que é um tipo de sistema interativo.

O método ESA utiliza os instrumentos do método "Structured Systems Analysis", segundo a abordagem de (DEMARCO, 1978) e do método "Entity-Relationship Model"

(CHEN, 1976) para modelar a essência de um sistema. Assim, o método utiliza os seguintes instrumentos:

- Linguagem para Diagrama de Fluxo de Dados;
- Dicionário de Dados;
- Português Estruturado;
- Linguagem para Árvores de Decisão;
- Linguagem para Tabelas de Decisão;
- Linguagem para Diagrama de Entidade-Relacionamento;
- Linguagem para Diagrama de Estruturas de Dados.

O método ESA utiliza duas estratégias básicas para modelar a essência de um sistema:

i) Criar o modelo da essência do novo sistema a partir dos requisitos definidos pelo usuário - nesta estratégia a criação do novo modelo é realizada através das seguintes etapas:

   . identificar a finalidade do sistema;
   . identificar as atividades fundamentais - nesta etapa deve-se determinar a quais eventos o sistema deve responder e definir os estímulos e as respostas correspondentes a esses eventos;
   . identificar as informações a serem armazenadas na memória essencial;
   . identificar as atividades custodais, ou seja, os acessos à memória principal.

Após a realização dessas etapas o modelo é construído utilizando-se os instrumentos identificados acima.

ii) Derivar a essência do novo sistema a partir de um sistema existente - nesta estratégia deve-se, em primeiro lugar, modelar a essência de um sistema existente, para em seguida acrescentar as novas características essenciais que necessitem fazer parte do novo sistema.

Para modelar a essência de um sistema existente, deve-se ter, em primeiro lugar, um modelo físico composto por DFDs, definições de Dicionário de Dados e miniespecificações. Em seguida deve-se realizar as seguintes atividades:

- encontrar e classificar os fragmentos de atividades essenciais;
- reconstruir e modelar cada atividade essencial;
- integrar todos os modelos das atividades essenciais individuais em um modelo essencial único.

O modelo do sistema atual é expandido de forma que as características tecnológicas sejam identificadas e retiradas.

Após a construção do modelo essencial do sistema, deve-se partir para a especificação do projeto do sistema. O método ESA pode ser utilizado em conjunto com o método "Structured Design" (SD) (YOURDON, 1978). É utilizada a estratégia de análise de transformação do método SD para converter o DFD no Gráfico de Estruturas. O tipo de DFD requerido pela análise de transformação, que retrata a transformação de uma entrada em saída, é o tipo de DFD

elaborado pelo método ESA. Assim, o modelo físico elaborado para o novo sistema pode ser convertido facilmente para uma estrutura hierárquica.

IV.6 "JACKSON SYSTEM DEVELOPMENT" (CAMERON, 1983), (PRESSMAN, 1988)

"Jackson System Development" (JSD) é um método para especificação e projeto de sistemas, onde a estrutura funcional do produto é determinada pela modelagem da estrutura dos dados que ele processa. Foi desenvolvido por Michael Jackson e Jonh Cameron.

JSD constrói um modelo do domínio da informação no mundo real, onde a decomposição do problema é derivada em termos das estruturas dos dados de entrada e de saída. O método pode ser utilizado no desenvolvimento de sistemas de tempo real, pois oferece mecanismos para modelar restrições de tempo e controle da sincronização de processos.

O método JSD sofreu influência do método "Jackson Structured Programming" (JSP) (JACKSON, 1975), podendo ser considerado como uma extensão do mesmo. JSP tem por objetivo auxiliar no projeto de programas, enquanto JSD pode ser utilizado tanto na fase de análise quanto na de projeto.

JSP assume a existência de uma especificação de requisitos, a partir de onde as estruturas de dados de entrada e saída serão abstraídas e transformadas em estruturas de programas. O método JSD, por sua vez, oferece

recursos para elaborar a especificação do problema em termos de suas entidades e ações. Essa especificação é, mais adiante, transformada em uma estrutura de programa, utilizando a filosofia básica do próprio JSP.

O método utiliza os seguintes instrumentos:

- Linguagem para Diagrama de Estrutura de Entidades (DEE) - o DEE representa a estrutura de cada uma das entidades do sistema, especificando a ordem, em função do tempo, das ações relacionadas com a entidade. A linguagem para DEE permite modelar estruturas que representem seqüência, seleção e iteração. A figura IV.6.1 mostra a notação para representar as estruturas de entidades e a figura IV.6.2 apresenta um exemplo do DEE.

- Linguagem para Diagrama de Especificação de Sistema (DES) a notação da linguagem para DES é composta de símbolos para representar a conexão entre os processo de um sistema. A figura IV.6.3 mostra a notação para representar as conexões entre os processos e a figura IV.6.4 apresenta um exemplo do DES. A comunicação entre processos pode ser de um dos seguintes tipos:

  . comunicação por fluxo de dados - ocorre quando um processo transmite um fluxo de informações e outro processo o recebe. É representada pelos seguintes símbolos:

  - retângulo - representa um processo;
  - círculo - representa o fluxo de dados;
  - seta - indica a direção do fluxo de informação.

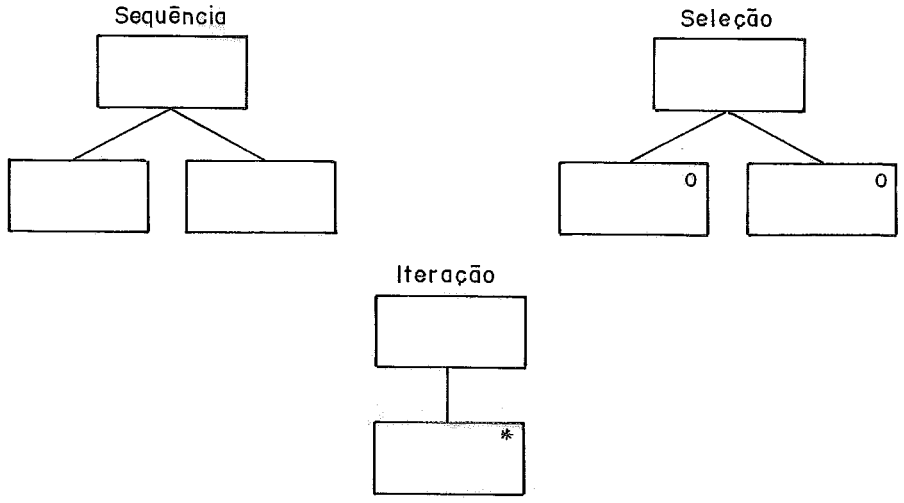


Figura IV.6.1 : Notação para Representar Estruturas em JSD
Fonte: (PRESSMAN, 1988)

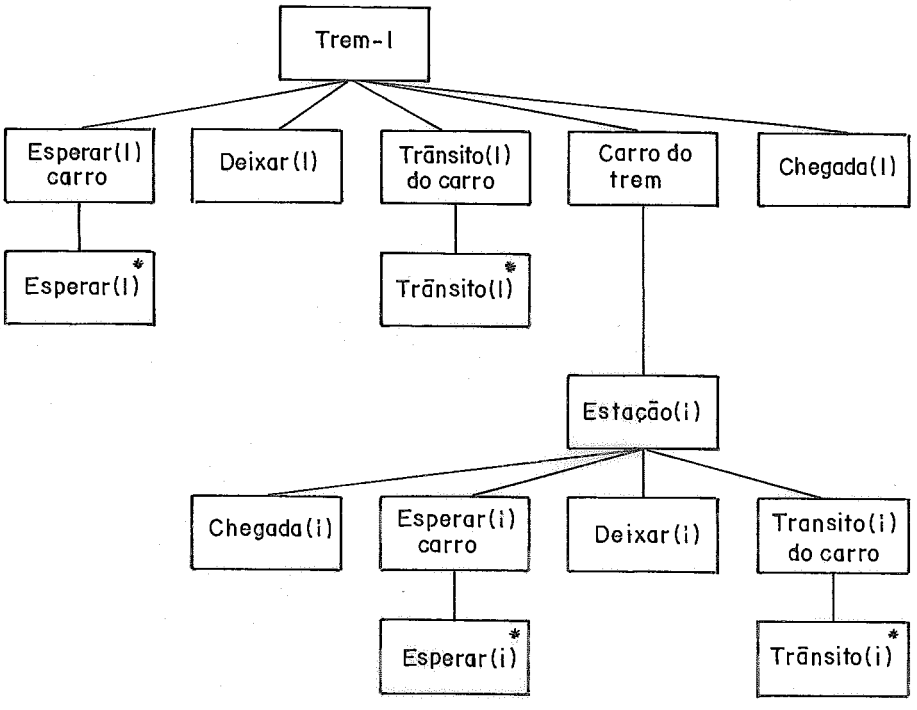
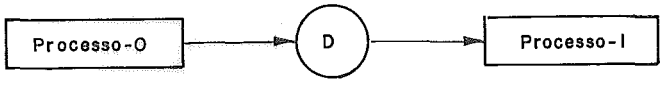
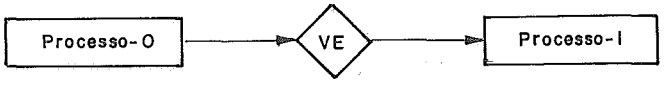


Figura IV.6.2 : Diagrama de Estrutura de Entidades
Fonte : (PRESSMAN, 1988)



Conexão por Fluxo de Dados



Conexão por vetor de Estado

Figura IV.6.3: Notação para Representar Conexão em JSD
Fonte: (PRESSMAN, 1988)

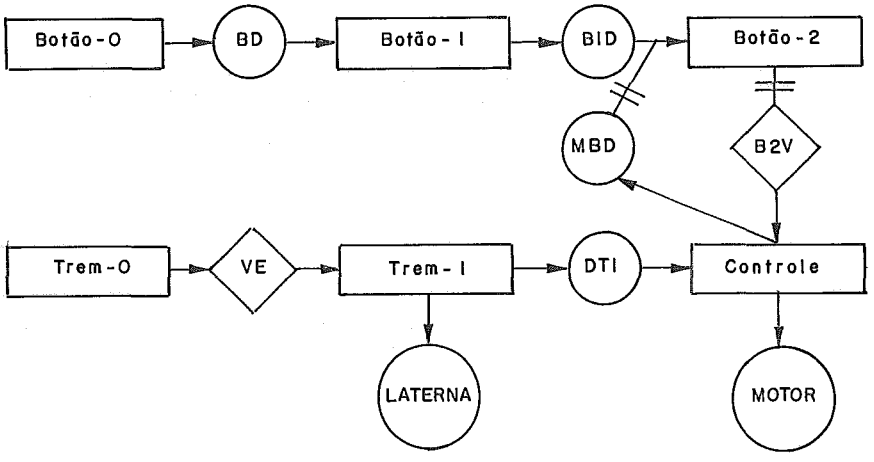


Figura IV.6.4: Diagrama de Especificação de Sistema
Fonte: (PRESSMAN, 1988)

. conexão por vetor de estados - ocorre quando um processo inspeciona diretamente o vetor de estados de outro processo. Nesse tipo de comunicação um losango representa o vetor de estados.

- Linguagem para especificação detalhada de processos - é uma linguagem tipo pseudocódigo, utilizada para representar os detalhes internos dos processos do modelo.

- Linguagem para Diagrama de Estrutura de Dados (DED) - é utilizada na fase de projeto para representar a estrutura dos dados. A notação dessa linguagem é semelhante à notação da linguagem para DEE.

A análise de sistemas é elaborada, segundo o método JSD, através dos seguintes passos:

- Passo de entidade/ação - neste passo as entidades e ações são definidas através de uma estratégia similar àquela adotada pela abordagem de orientação a objetos. As entidades e ações são retiradas de uma breve declaração textual do sistema escrita em linguagem natural.

As entidades são selecionadas examinando-se todos os nomes existentes na descrição do sistema. Para identificar as ações, deve-se examinar os verbos da descrição. Deve-se procurar definir os limites do sistema, identificando somente as entidades e ações que estejam relacionadas com a solução do problema.

- Passo de estrutura da entidade - neste passo as ações que efetam cada entidade são ordenadas pelo tempo e representadas através do Diagrama de Estrutura de Entidades (DEE). É gerado um diagrama para cada uma das entidades identificadas. Além disso, pode-se elaborar um texto narrativo contendo a descrição das restrições que não podem ser representadas no DEE para complementar a definição da entidade.

- Passo de modelo inicial - neste passo inicia-se a construção de uma especificação do sistema como um modelo do mundo real, estabelecendo as conexões entre o modelo gerado e o mundo real. A especificação é elaborada através do Diagrama de Especificação do Sistema e representa as entidades e ações como um modelo de processo. Os detalhes internos do modelo de processo são especificados usando a linguagem para especificação de texto estruturado.

Após a fase de análise, inicia-se a construção de uma estrutura que permita transformar os dados manipulados pelo sistema em uma estrutura de programa. A fase de projeto é descrita a seguir:

- Passo funcional - neste passo é elaborada a expansão do DES através da conexão de novas funções para o modelo por meio de fluxo de dados ou vetores.

- Passo de tempo do sistema - neste passo é feita a especificação das restrições de tempo impostas ao sistema. Os passos anteriores geram um sistema composto

de processos seqüenciais que se comunicam através de fluxos de dados e de vetores de estados.

É utilizado um mecanismo denominado "Time Grain Marker" (TGM) para incorporar as características de sincronização de processos no modelo gerado. O TGM é um registro de dados que indica a ocorrência de um intervalo particular de tempo e possibilita que a passagem do tempo afete as ações de um processo.

- Passo de implementação - neste passo são aplicados conceitos do método JSP para derivar a estrutura de um processo ou programa, a partir da estrutura dos dados do problema.

Os dados são representados em estruturas hierárquicas formadas por construtores que definem seqüências, repetições e seleções. A linguagem para representar as estruturas de dados possui uma notação semelhante à notação da linguagem para DEE.

Dessa forma, a representação procedimental de cada programa é derivada a partir da organização de sua estrutura de dados hierárquica. A partir da estrutura do programa é elaborada a especificação do programa escrita em linguagem para especificação de texto estruturado.

IV.7 "KNOWLEDGE-BASED SYSTEM DEVELOPMENT LIFE CYCLE" (WEITZEL, 1989)

"Knowledge-Based System Development Life Cycle" (KBSDLC) é um método para construção de protótipos de

Sistemas Baseados em Conhecimento (SBCs), através do uso de "shells" de Sistemas Especialistas e ambientes de programação. Foi desenvolvido por John Weitzel e Larry Kerschberg.

A construção de um software, segundo o método KBSDLC, é feita através de uma série de processos que podem ser ativados, desativados e reativados, sempre que for necessário. O método incentiva a execução dinâmica de processos como uma forma de permitir ao sistema evoluir ao longo de seu próprio ciclo de desenvolvimento.

Os processos podem ser executados concorrentemente, mas para que um determinado processo seja executado, é preciso que o anterior tenha sido ativado pelo menos uma vez. A figura IV.7.1 mostra o fluxo dos processos de KBSDLC e os possíveis caminhos de reativação. Esses processos são definidos a seguir, exceto os de identificação do problema e de conversão e manutenção/evolução, que são considerados, segundo os autores, semelhantes aos de outros modelos de ciclo de vida:

- Definição do problema e avaliação da viabilidade - neste processo são realizadas as tarefas básicas para a definição do problema e da solução.

O problema é definido sob dois pontos de vista: comercial e do conhecimento. O problema sob o ponto de vista do conhecimento é mais complexo de ser definido do que sob o ponto de vista comercial. A facilidade de reativação oferecida pelo método, entretanto, contribui

para que o problema, sob o ponto de vista do conhecimento, possa ser redefinido várias vezes. A definição da solução também requer várias ativações da definição do problema e da avaliação da viabilidade.

As questões de viabilidade consideradas por esse processo incluem: viabilidade técnica, econômica, de sucesso do projeto, de integração do sistema ao ambiente operacional e de escalonamento.

- Identificação dos subproblemas - neste processo são realizadas as tarefas básicas para análise do sistema. O problema deve ser dividido em subproblemas de forma a facilitar sua compreensão e o seu tratamento. Para cada subproblema, deve ser construído um pequeno protótipo. Os protótipos gerados serão integrados mais adiante.

- Identificação e definição da estrutura conceitual - neste processo também são realizadas as tarefas básicas para análise do sistema. Um sistema baseado em conhecimento automatiza um processo humano de solução de problema. Assim, os engenheiros do conhecimento devem procurar conceitos que caracterizem o pensamento do especialista sobre o problema. Cada conceito (entidade, atributo ou relacionamento) deve ser visto com o mesmo grau de importância.

- Projeto conceitual - neste processo são realizadas as tarefas básicas para o projeto lógico do sistema. Em sistemas baseados em conhecimento, a definição completa do problema não é obtida antes de iniciar esse processo.

é necessário um ciclo de reativação dos processos para se construir um protótipo, que servirá como ponto de partida podendo sofrer grandes mudanças ou mesmo ser construído novamente até se obter o produto desejado.

No projeto conceitual é selecionada uma representação para o conhecimento, que pode ser lógica de primeira ordem, redes semânticas, sistemas de produção, "frames", "scripts", estruturas de banco de dados, etc. Também é criada a estrutura na qual a representação do conhecimento será utilizada, como por exemplo árvores e/ou.

Durante o projeto conceitual, também podem ser utilizados múltiplos pontos de vista: orientação a objetos, para definir os conceitos; decomposição funcional, para estabelecer o processo de decisão; e decomposição do conhecimento, para dividir a base de conhecimento em componentes baseados em regras e baseados em dados.

- Projeto detalhado - durante este processo são realizadas as atividades básicas para elaboração do projeto dos programas. O engenheiro do conhecimento elabora o pseudocódigo para procedimentos, escreve regras de produção, desenha diagramas de redes semânticas, constrói modelos para representações diretas, identifica e nomeia "slots" para "frames e "scripts" e entradas de tabelas para tabelas de dados.

- Codificação - durante esta fase o projeto detalhado é traduzido para a linguagem da ferramenta de engenharia do conhecimento. Isso faz com que a representação do

conhecimento seja incluída na base de conhecimento da ferramenta.

Nessa fase podem ser identificados os problemas existentes no projeto detalhado, levando à execução do projeto detalhado e da codificação de forma concorrente ou em um "loop". Problemas encontrados durante a codificação podem até ocasionar a reativação do processo de definição do problema e a avaliação da viabilidade.

- Teste do raciocínio - os testes realizados neste processo procuram por raciocínios inválidos, que ocorrem quando o programador traduz incorretamente o conhecimento e revelam quando o sistema viola as expectativas do programador. Os erros encontrados são corrigidos com a reativação do processo de codificação.

- Teste do conhecimento - neste processo procura-se detectar o conhecimento inválido ou ambígüo, pois código correto não significa conhecimento correto.

O conhecimento inválido ocorre quando fatos são declarados incorretos e eles são detectados quando o sistema viola as expectativas dos especialistas. O conhecimento ambígüo, por sua vez, ocorre quando todas as dependências implícitas não são declaradas e é detectado quando o sistema escolhe diferentes soluções, em situações que parecem requerer a mesma solução. Nesse processo há uma busca por mais conhecimento.

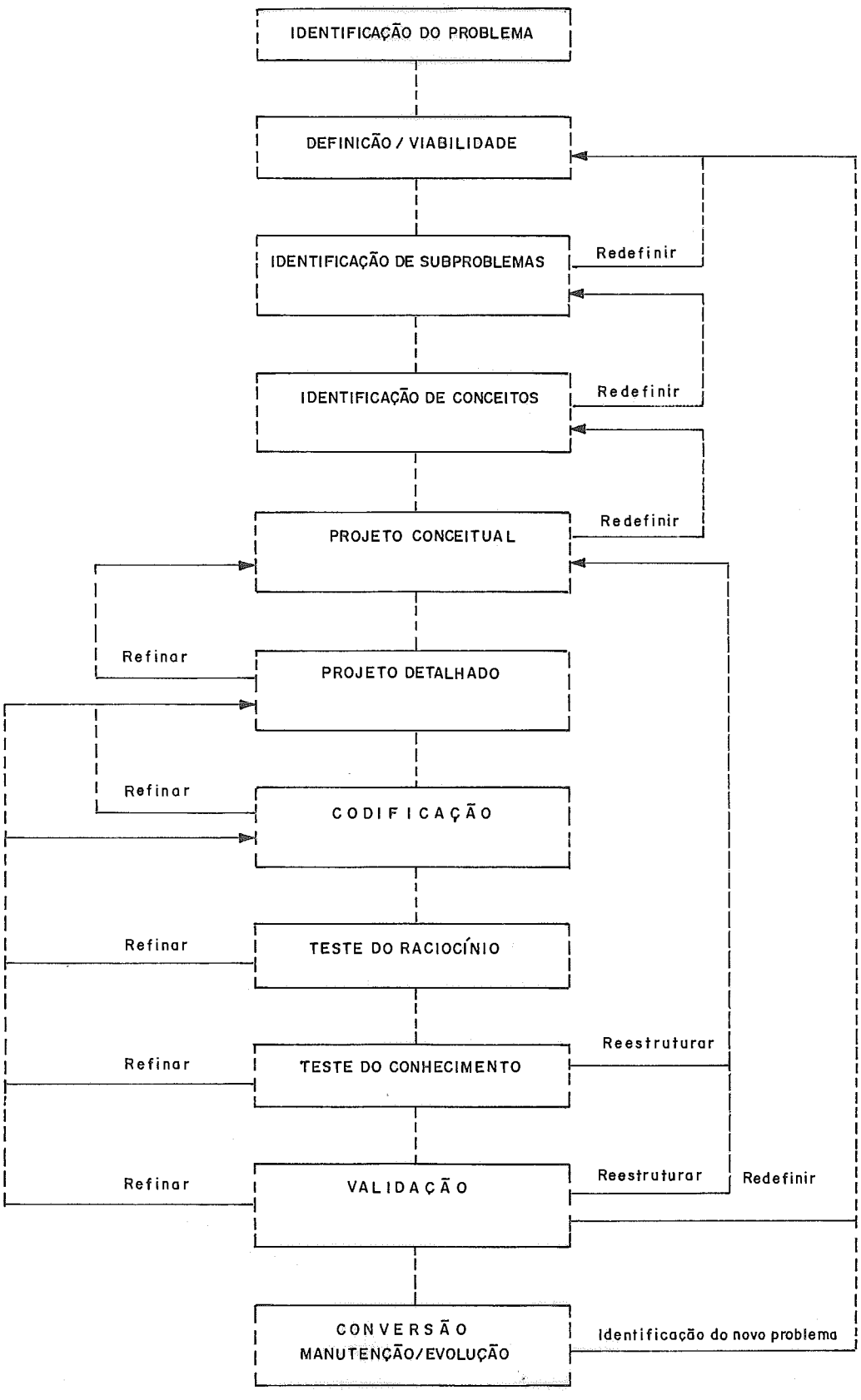


Figura IV.7.1: Fluxo dos processos do método KBSDLC
Fonte: (WEITZEL, 1989)

- Validação - este processo inclui as tarefas básicas de testes de programas e de integração. A tarefa principal é detectar as condições não alcançadas anteriormente e isso requer o uso de muitos exemplos de casos reais.

Após o seu desenvolvimento, o protótipo do sistema pode ser utilizado como o produto final ou ser descartado, servindo, nesse caso, apenas como uma especificação para o sistema que se deseja produzir. O protótipo pode ainda ser usado como uma experiência de aprendizagem.

Como pode ser observado, o método KBSDLC reorganiza as tarefas para o desenvolvimento de software, incluindo considerações específicas para SBC e utilizando ferramentas que permitem a construção de protótipos do sistema.

IV.8 "KNOWLEDGE-BASED SYSTEM DEVELOPMENT METHODOLOGY" (KELLER, 1987)

"Knowledge-Based System Development Methodology" (KBSDM) é um método para o desenvolvimento de Sistemas Baseados em Conhecimento (SBCs), em especial de Sistemas Especialistas (SEs). É aplicado nas fases iniciais do processo de desenvolvimento, tendo especial preocupação com a definição da Base de Conhecimento (BC) e com o processo de aquisição de conhecimento. Foi proposto por Robert Keller.

O método KBSDM utiliza os seguintes instrumentos:

- Linguagem para Diagrama de Fluxo de Dados (DFD), cuja notação gráfica é composta por:

. círculo - que representa uma decisão;

. seta - que representa um fluxo de entrada ou de saída;

. quadrado - que representa a fonte ou o destino dos dados.

- Dicionário de Dados e Regras (DDR), onde são detalhadas as informações presentes no DFD. A figura IV.8.1 mostra um exemplo da definição de uma decisão no DDR.

```
+-----------------------------------------------------------------------+
|                                                                       |
| DECISÃO:                          VALORES:                            |
|                                                                       |
| Avaliar Tipo de Linguagem         Avaliação descrita na Regra 1       |
|                                                                       |
| FATORES:                          VALORES:                            |
|                                                                       |
| Vida útil ..................... pequena e longa                      |
| Tipo de linguagem ............. assembler, L30 e L40                 |
| Restrição quanto ao uso                                               |
|     do computador ............. existente e inexistente              |
| Volume de dados ............... pequeno e grande                     |
|                                                                       |
| REGRA 1:                                                              |
|                                                                       |
| SE    Vida útil                   é pequena                           |
|                                                                       |
| E     Tipo de Linguagem           é assembler                         |
|                                                                       |
| ENTÃO                                                                 |
|          Já que se trata de um sistema com pequena vida               |
|útil, se possível, evite programá-lo em assembler pois os              |
|custos com codificação e testes serão muito altos. Use,               |
|de preferência, uma linguagem de quarta geração. Caso não              |
|seja possível, procure utilizar uma linguagem de terceira              |
|geração.                                                               |
|                                                                       |
+-----------------------------------------------------------------------+
```

Figura IV.8.1: Exemplo de Definição de uma Decisão no Dicionário de Dados e Regras do Método KBSDM

Fonte: (RIBEIRO, 1989)

Para cada decisão representada no DFD, são definidos:

- os fatores que influenciam a decisão (fluxos de entrada);
- os possíveis resultados da decisão (fluxos de saída);
- as regras pelas quais, a partir dos fatores, chega-se aos resultados.

O método KBSDM auxilia nas atividades das fases de levantamento, análise e aquisição do conhecimento.

- Fase de Levantamento, ou estudo da viabilidade de desenvolvimento do sistema, tem como objetivo principal decidir se determinado projeto é importante e definir os objetivos gerais e a estimativa de custo do novo sistema.

Nessa fase é feito um estudo do ambiente onde o produto será operacional a fim de propor estratégias para integração da Inteligência Artificial (IA) com esses ambientes. É feita uma seleção dos domínios de aplicações que são suscetíveis ao uso de técnicas da IA. É importante ressaltar que nem todas as aplicações necessitam ser implementadas como um SBC.

Deve ser observado se a aplicação é apropriada para um tratamento heurístico antes de se partir para a especificação dos requisitos. Para tanto, o método sugere um conjunto de critérios para avaliar aplicações e decidir se o problema deve ou não ser implementado através de técnicas da IA.

Como resultado da fase de levantamento, é elaborado um relatório propondo os passos a serem seguidos pela empresa para integrar IA em suas operações, e as áreas específicas que podem ser beneficiadas com o uso da IA. Para cada domínio da aplicação, o relatório deve conter:

- descrição da importância do sistema e, em alguns casos, um DFD de alto nível mostrando o SBC com suas interfaces com o ambiente da empresa;
- análise de custos e benefícios da aplicação de IA para o domínio;
- análise de risco do desenvolvimento proposto.

A definição da base de conhecimento começa nessa fase. O engenheiro do conhecimento observa como o especialista do domínio toma suas decisões e a forma como elas são realizadas. Nesse ponto pode ser elaborado um protótipo descartável da base de conhecimento, visando tornar mais claras as decisões ou questões de planejamento envolvidas no domínio.

- Fase de Análise, onde são definidos os requisitos do usuário, em termos das funções a serem executadas e do relacionamento de dados existente entre elas. Durante essa fase é produzida a especificação estruturada do sistema e é aperfeiçoado o protótipo da BC. O sistema é decomposto em mini-sistemas, de forma a facilitar a identificação das funções e do fluxo de dados entre elas. A fase de análise é realizada através da execução da:

a) Análise do sistema corrente, onde é elaborada a especificação das tarefas executadas atualmente pelo especialista. Esse passo é realizado através das seguintes atividades:

. especificação do sistema físico corrente, cujo objetivo é documentar o trabalho do especialista da forma como ele é atualmente executado;

. especificação do sistema funcional corrente, cujo objetivo é identificar, a partir do sistema físico corrente, as decisões ou funções de planejamento executadas pelo especialista e as informações específicas usadas para executar essas funções.

Quando desenvolvemos uma base de conhecimento estamos interessados na descrição funcional do trabalho do especialista, ou seja, a especificação das funções do especialista leva à especificação da base de conhecimento. A documentação das operações do sistema físico corrente pode ajudar na identificação da estrutura do conhecimento.

b) Análise do novo sistema, onde é elaborada a especificação do novo sistema, incluindo a tecnologia a ser usada para executá-lo. O objetivo dessa etapa é descrever a essência funcional de tudo aquilo que o especialista faz e embuti-la em algum ambiente tecnológico. Esse passo está dividido em:

. especificação do sistema funcional novo, onde são incluídas novas funções ou dados necessários ao futuro sistema. Nesse ponto ainda se trabalha em

um nível funcional, sem considerar como o sistema será implementado tecnologicamente;

- especificação do sistema físico novo, onde são especificados os detalhes físicos do novo sistema introduzindo-se as considerações de tecnologia.

Após a conclusão da fase de análise, têm-se a especificação estruturada do SBC composta por um conjunto de DFDs, mini-especificações dos processos e o Dicionário de Dados e Regras, com a descrição de todos os dados requeridos pelo sistema.

- Fase de Aquisição do Conhecimento - cujo objetivo é coletar dados para compor a BC. É uma atividade independente que se inicia na fase de levantamento, segue pela fase de análise, estendendo-se além do processo de desenvolvimento. Essa atividade torna-se independente após a fase de análise e a independência garante a expansão e atualização da BC sem interferir na construção do sistema, bem como a disponibilidade da BC durante todas as etapas do processo de desenvolvimento.

O processo de desenvolvimento da BC é realizado através dos seguintes passos:

- especificar as decisões a serem tomadas;
- encontrar a estrutura correta para os fatores de decisão, e,
- treinar o sistema, fornecendo regras gerais ou exemplos de decisões nos quais ele pode basear suas próprias decisões.

Os autores incentivam a construção de protótipos da base de conhecimento através do uso de "shell" de sistemas especialistas e de protótipos do sistema, como ferramentas para auxilar na especificação do produto.

IV.9 "METHODOLOGY FOR BUSINESS SYSTEM DEVELOPMENT" (MATHUR, 1987)

"Methodology for Business System Development" (MBSD) é um método para a especificação de projeto de sistemas comerciais orientados a transação. Foi desenvolvido por Raghubir Mathur.

O método assume a existência de uma especificação de requisitos composta por vários Elementos de Decomposição (ED). Um ED é uma estrutura para especificar os estímulos, o processamento e as respostas associados a cada função do sistema.

Um grupo de EDs, logicamente organizados, formam um Diagrama de Verificação do Sistema (DVS), que representa os requisitos gerais do sistema. O método MBSD tem por objetivo decompor o DVS no projeto detalhado de programa, a partir de onde o código pode ser gerado.

As informações manipuladas durante as várias etapas de funcionamento do método são armazenadas em bibliotecas e em um banco de dados integrado, que oferece mecanismos para segurança dos dados. São utilizados formulários específicos para armazenar as informações produzidas em cada etapa do projeto.

O método utiliza os seguintes instrumentos:

- Linguagem para Elementos de Decomposição (ED) - um ED especifica estímulos, processamento e respostas associados a cada função do sistema. Na figura IV.9.1 pode-se ver a notação gráfica da linguagem utilizada para a construção de um ED;

- Linguagem para Diagrama de Verificação do Sistema (DVS) - um DVS é formado por um conjunto logicamente organizado de EDs e representa os requisitos gerais do sistema;

- Linguagem para Gráficos de Bolhas (GB) - o GB representa o relacionamento entre elementos de dados de telas e relatórios e a forma como esses dados podem ser acessados. A notação gráfica da linguagem para GB é composta de bolhas que representam os elementos de dados e arcos que representam ligações entre os dados.

O método MBSD propõe a realização da seguinte seqüência de passos para que os DVSs, elaborados a priori, sejam decompostos em um projeto detalhado de programas, a partir de onde o código para implementação do produto será gerado:

- Elaboração do Caderno de Projeto do Sistema (CPS) - o CPS oferece a estrutura para o desenvolvimento sistemático e para a documentação do projeto do sistema. Contém as informações geradas durante a especificação de projeto.

O CPS é formado por um conjunto de EDs, ligados de forma a compor a estrutura do sistema. Após o projeto de

todos os EDs, o CNS contém todas as telas de entrada, os nomes dos elementos de saída requeridos para o processamento, as telas de saída, os relatórios, os arquivos e o processamento necessário para produzir cada tela de saída.

- Coleta dos dados - este passo envolve a coleta dos dados relacionados a cada ED do sistema.

- Definição das telas e relatórios - neste passo é feito o projeto das telas de entrada para os elementos de dados associados a cada estímulo, e são definidos as telas de saída e os relatórios de resposta para o estímulo.

- Preparação da visão inicial do usuário - nesta etapa é elaborada a visão do usuário para todas as telas e relatórios definidos. A visão consiste na representação lógica dos dados necessários para responder perguntas, tomar decisões, executar tarefas e dar instruções de uso. O GB é o instumento utilizado para representar os dados associados a todas as telas e relatórios.

- Definição dos caminhos lógicos de processamento - neste passo devem ser identificados todos os caminhos lógicos entre as telas de entrada, telas de saída e relatórios.

As decisões para transferência de controle entre os caminhos podem ser representadas através de um diagrama que pode ser visto na figura IV.9.2. O processamento associado a cada caminho lógico é definido usando uma Linguagem de Projeto de Programa (LPP).

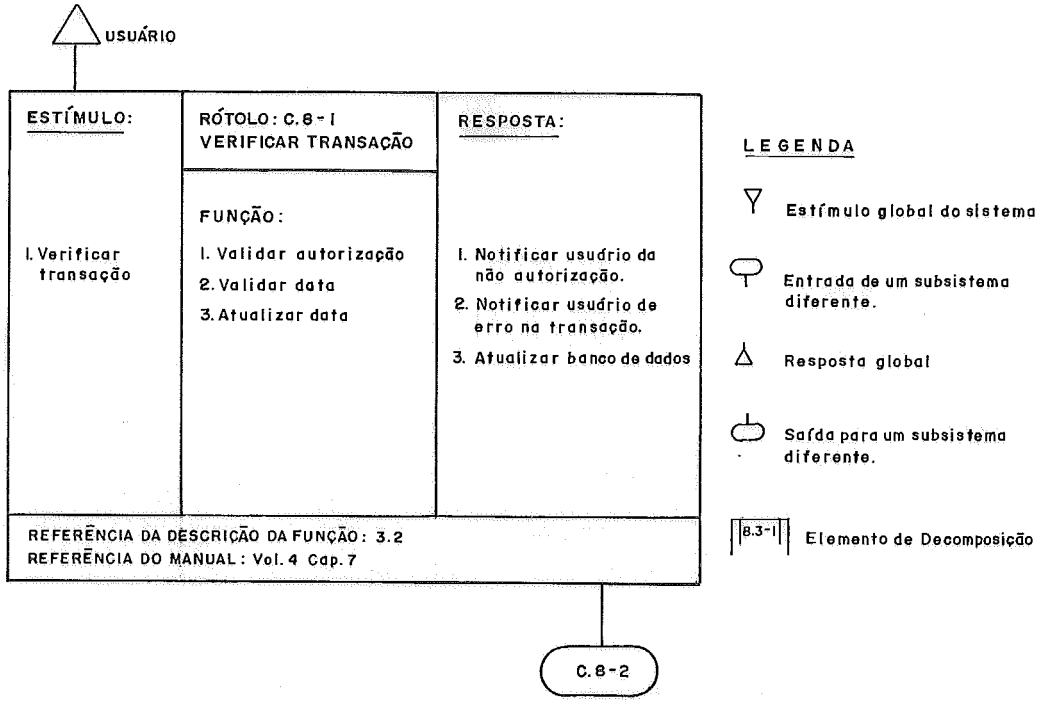


Figura IV.9.1: Exemplo de um Elemento de Decomposição
Fonte: (MATHUR, 1987)

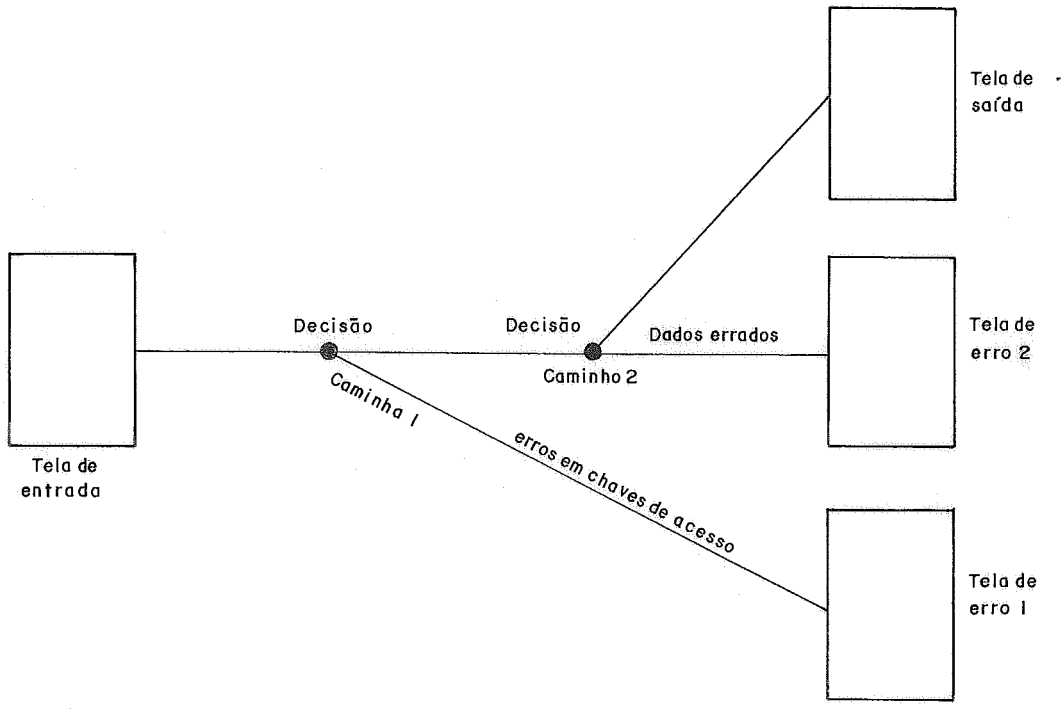


Figura IV. 9.2: Exemplo de Seleção de Caminhos Lógicos
Fonte: (MATHUR, 1987)

- Definição de programas - nesta etapa os caminhos lógicos identificados são decompostos e os elementos gerados são sintetizados em programas. Deve-se elaborar uma definição para cada programa contendo as interfaces do programa, os dados de entrada, os dados de saída e o processamento, especificado na linguagem LPP.

Deve ser feita uma avaliação do projeto do sistema juntamente com o usuário após a execução dessa etapa.

- Elaboração das especificações dos subsistemas - essas especificações são feitas a partir de informações obtidas dos documentos gerados nos passos anteriores, podendo ser escritas em português e em LPP.

Após esse passo, é realizada a Revisão de Projeto Preliminar (RPP), cujo objetivo é verificar se todos os requisitos especificados nos DVSs, para o sistema em questão, foram incluídos nas especificações dos subsistemas.

- Elaboração das especificações de projeto de programas - neste ponto já se sabe quais são as entradas, o processamento e as saídas para cada programa. Deve-se então elaborar uma especificação de projeto detalhado para cada programa. Nessa etapa deve-se, também, definir as interfaces do programa com outros programas do sistema, com utilitários e com o banco de dados.

- Preparação da visão final do usuário - a visão final do usuário é elaborada a partir das informações contidas nas especificações dos programas. Para isso é utilizado

o GB, que identifica o relacionamento entre os dados para cada especificação de programa. Os GBs são codificados a fim de que possam ser utilizados no projeto dos dados.

- Elaboração do projeto dos dados - a codificação dos GBs é utilizada como entrada para a construção de modelos dos dados. Como resultado, é gerado um modelo conceitual do esquema do banco de dados.

- Definição do esquema lógico - o modelo dos dados elaborado deve ser otimizado a fim de se obter um esquema lógico ideal para banco de dados. Esse esquema deve ser armazenado em um Dicionário de Dados Comum (DDC), de forma a ser acessado por todos os programas do sistema.

- Definição dos subesquemas - subesquemas são subconjuntos do esquema lógico utilizados como mecanismos de segurança, restringindo o acesso e uso do banco de dados somente a usuário autorizado. São compilados dentro dos programas e armazenados no DDC.

Após a execução dos passos especificados acima, os documentos gerados são utilizados como entrada para a fase de codificação. Além disso, o método possibilita a realização de teste a nível de programas, de subsistemas e de sistema. Quando todos os caminhos lógicos tiverem sido testados e o resultado verificado, os testes de integração e do sistema estarão completos.

MBSD tem sido utilizado tanto em projetos pequenos quanto em projetos grandes e complexos. A característica do método de especificar as entradas e saídas nas etapas iniciais do projeto do sistema facilita o desenvolvimento de sistemas interativos e provê um melhor entendimento e uma maior visibilidade da interface homem-máquina por parte do usuário.

II.10 "OBJECT-ORIENTED ANALYSIS" (COAD, 1990)

"Object-Oriented Analysis" (OOA) é um método para especificação de requisitos de sistemas segundo a abordagem de orientação a objetos. Foi desenvolvido por Peter Coad e Edward Yourdon.

O método utiliza os seguintes instrumentos:

- Linguagem para Diagramas OOA (DOOA) - o DOOA é utilizado para representar graficamente o modelo do sistema. A linguagem é composta pelos seguintes elementos, cuja notação pode ser vista na figura IV.10.1:

  . objeto - um objeto é uma abstração de dados e do processamento exclusivo que pode ser realizado nesses dados, ou seja, é um encapsulamento de atributos e serviços exclusivos nesses atributos. Reflete a capacidade do sistema manter informações ou interagir com alguma coisa no mundo real;

  . estrutura - uma estrutura é a representação da complexidade do espaço do problema. As estruturas podem ser de dois tipos:

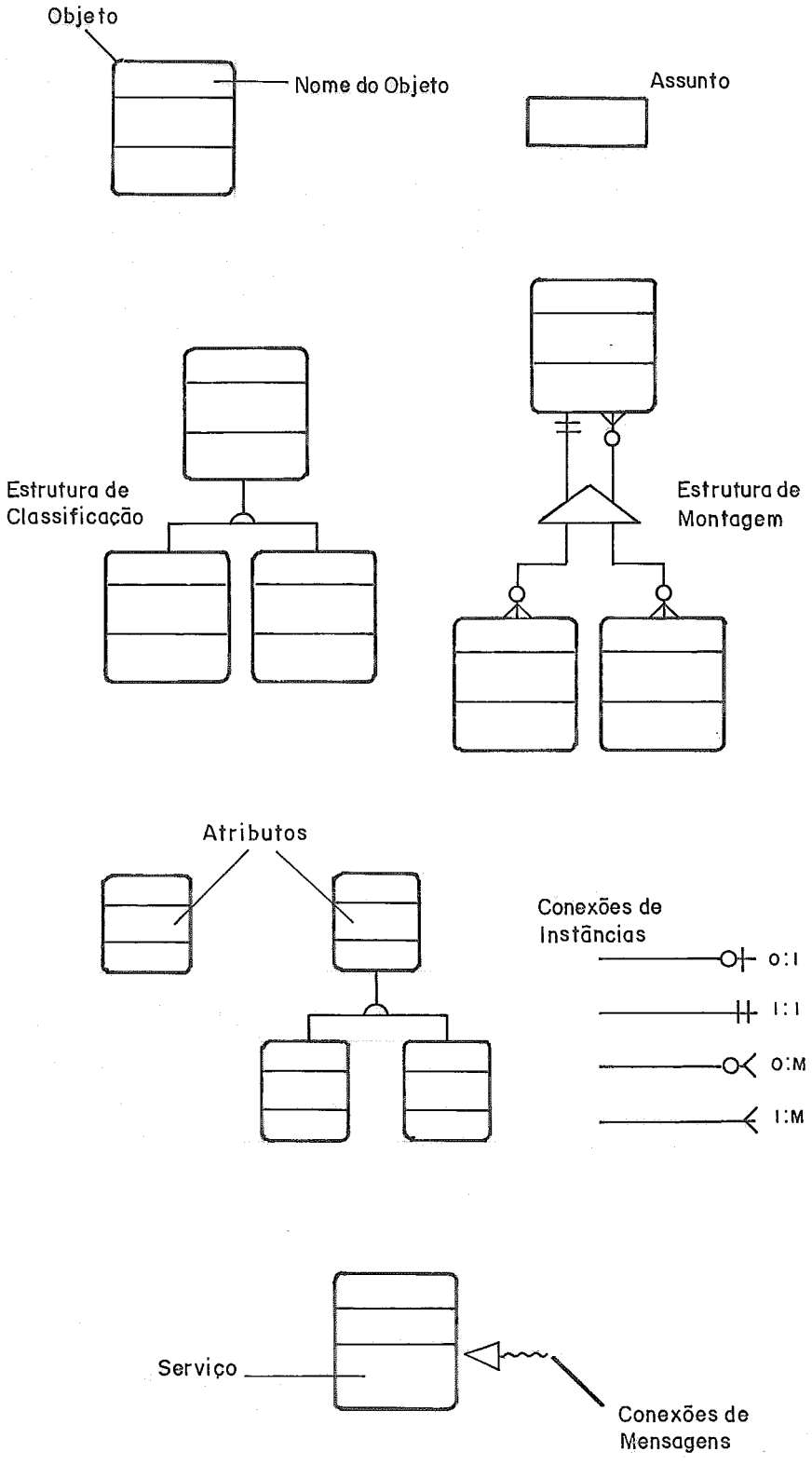


Figura IV.10.1: Notação Gráfica para Representar Modelos em OOA
Fonte: (COAD, 1990)

- estrutura de classificação - decompõe o problema em uma hierarquia de classe e membros, mostrando a generalização e a especialização de coisas do mundo real. O método utiliza o conceito de herança para identificar e representar as características comuns dos objetos, isto é, os atributos e serviços comuns aos objetos;

- estrutura de montagem - decompõe o problema como uma estrutura de objetos com suas partes componentes representando a agregação entre os objetos;

. assunto - é um mecanismo para controlar quanto de um modelo um leitor (analista, gerente ou usuário) é capaz de considerar e compreender de uma só vez;

. atributos - são elementos de dados usados para descrever uma instância de um objeto ou uma estrutura de classificação;

. serviço - é o processamento a ser executado quando chega uma mensagem. Serviços são equivalentes a métodos.

- Linguagem para especificação de requisitos - esta linguagem está baseada em gabaritos e na sintaxe da linguagem Ina Jo (WING, 1989). é utilizada para especificar atributos e serviços. A figura IV.10.2 mostra a estrutura da linguagem para especificação de requisitos no método OOA.

```
specification <nome do objeto>

    descriptiveAttribute <...>
    definitionData <...>
    alwaysDerivableAttribute <...>
    externalSystemInput <...>
    externalSystemOutput <...>

    instanceConnectionConstraint <...>

    stateEventresponse <...>
    objectLifeHistory <...>

    notes <...>

    intent/purpose <...>

    service <...>

end specification
```

Figura IV.10.2: Estrutura para Especificação de Atributos e Serviços no Método OOA

Fonte: (COAD, 1990)

O método sugere como documentação de apoio para especificar os serviços, o uso de Diagrama de Fluxo de Dados, Diagrama de Blocos ou Máquina de Estado Finito. Também podem ser utilizadas as seguintes tabelas: Tabela de Serviços e Estados, Tabela de Execução de Caminhos Críticos e Tabela de Tempo e Tamanho. Além desses diagramas e tabelas, pode ser elaborada, caso seja necessário, uma descrição dos serviços de forma narrativa e informal.

A construção de um modelo do sistema em OOA é realizada através de cinco passos principais, que são definidos a seguir:

- Identificar objetos - os objetos devem ser identificados a partir de estruturas, outros sistemas, dispositivos, eventos, papéis representados, localizações ou unidades

organizacionais. Quando se define um objeto, deve-se considerar os atributos, serviços e requisitos essenciais para cada objeto.

- Identificar estruturas - para definir uma estrutura de classificação, deve-se analisar cada objeto observando se ele representa uma classe genérica ou se pode ser agrupado como sendo uma classe específica de uma determinada superclasse. A definição de uma estrutura de montagem, por sua vez, deve ser feita observando se um objeto é formado por outros objetos, ou se ele é um componente de uma estrutura. Tanto a estrutura de classificação, como a de montagem são definidas de forma "top-down".

- Definir assuntos - os assuntos devem ser selecionados a partir dos objetos e estruturas definidos, pois para cada objeto ou estrutura é incluído um assunto correspondente.

- Definir atributos - os atributos são definidos através dos seguintes passos:

  . identificar os atributos - neste passo o analista retorna à definição do problema e interage com o usuário novamente;

  . posicionar os atributos - os atributos comuns a todos os objetos da estrutura devem ser representados nos objetos mais genéricos, enquanto que as características específicas devem ser associadas às especializações daquela classe;

. identificar e definir conexões de instâncias - uma conexão de instância é o mapeamento entre as instâncias de objetos de um modelo. A figura IV.10.1 mostra a notação para representar conexões;

. especificar as restrições de atributos e de conexões de instâncias utilizando o gabarito definido pela linguagem de especificação de requisitos.

- Definir serviços - neste passo deve-se definir o comportamento requerido para cada objeto e estrutura de classificação e a comunicação necessária entre as instâncias do objeto.

Os serviços são definidos através das seguintes etapas: identificar os serviços, identificar as conexões de mensagens e especificar os serviços. Quando houver troca de mensagens, deve-se incluir apenas um símbolo de conexão, mesmo que os objetos troquem mais de uma mensagem. Esse símbolo deve ser representado na superclasse.

Uma vez construído, o modelo é apresentado e revisado através de cinco camadas, onde cada camada corresponde a uma etapa de funcionamento do método. Essas camadas representam graficamente o modelo e são construídas utilizando-se a linguagem para DOOA:

- Camada de assunto - esta camada representa os assuntos definidos para o modelo. A figura IV.10.3 apresenta um exemplo de uma camada de assunto;

- Camada de objeto - os objetos são identificados a partir da camada de assuntos. São incluídas as linhas de assunto, que delimitam os assuntos e facilitam a navegação pelas diversas camadas. Um exemplo dessa camada pode ser visto na figura IV.10.4;

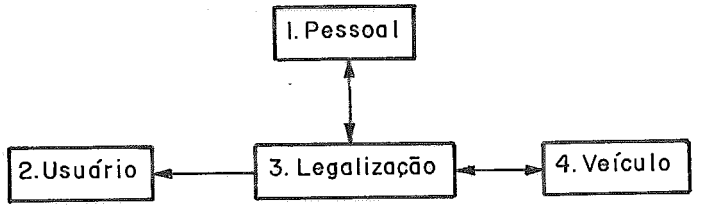


Figura IV.10.3: Camada de Assunto
Fonte: (COAD, 1990)

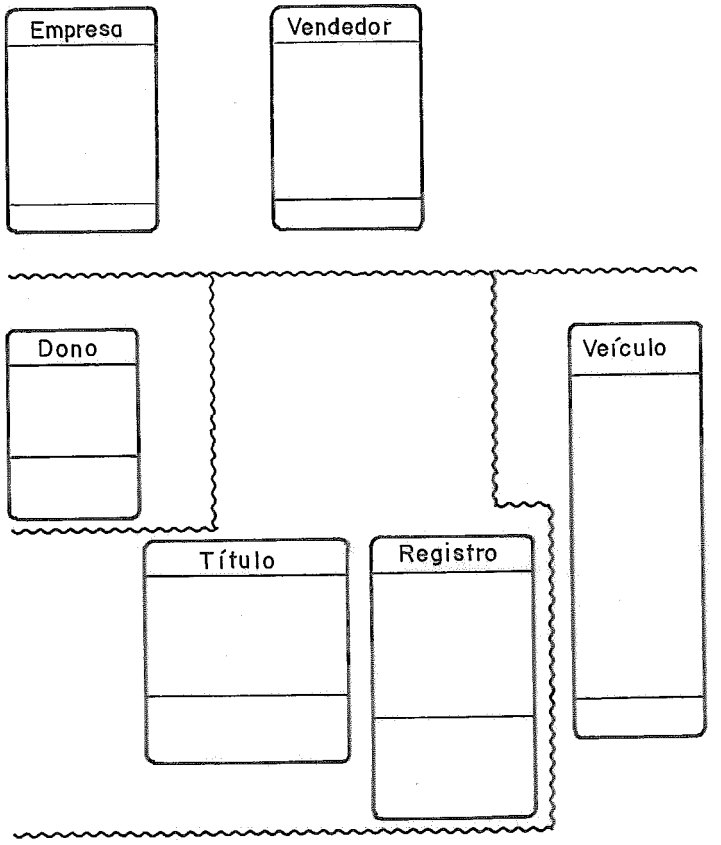


Figura IV. 10.4: Camada de Objeto
Fonte: (COAD, 1990)

- **Camada de estrutura** - nesta camada, cujo exemplo pode ser visto na figura IV.10.5, os objetos são organizados em estruturas de classificação e montagem;

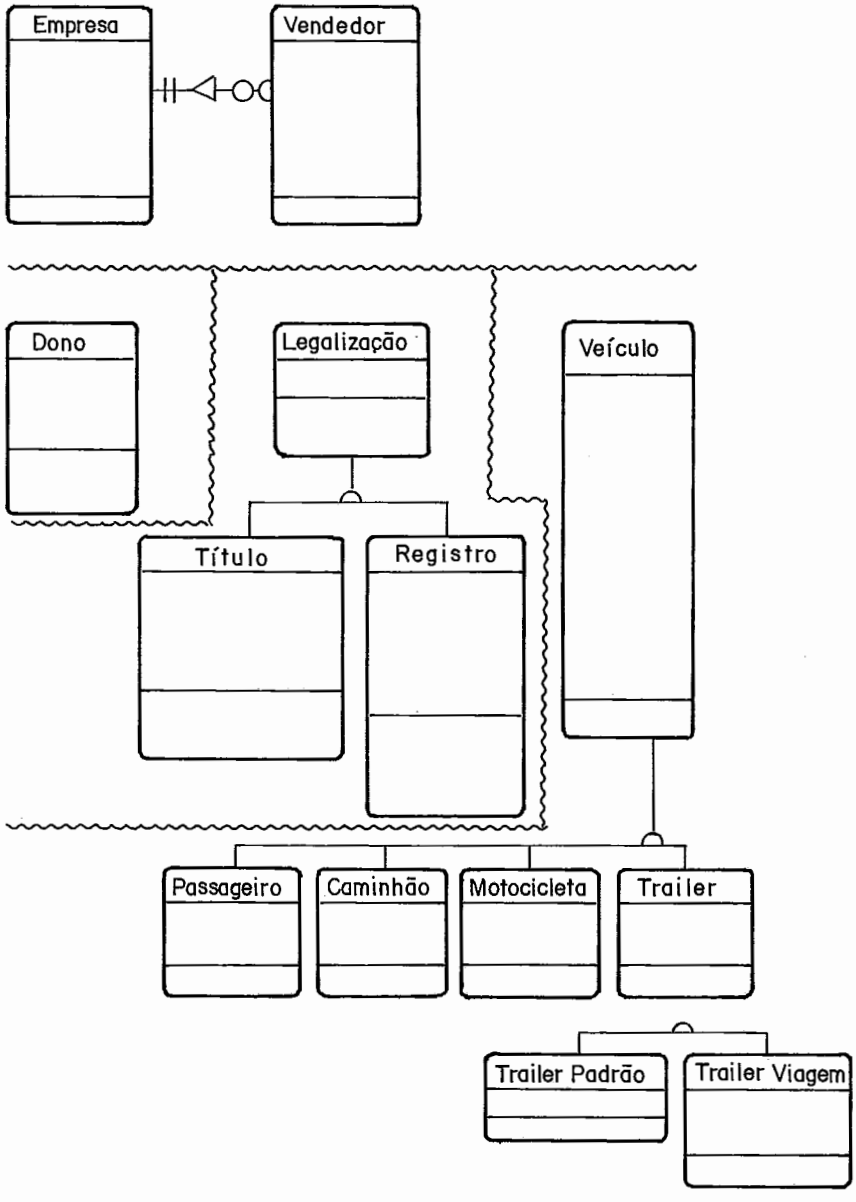


Figura IV.10.5: Camada de Estrutura
Fonte: (COAD , 1990)

- Camada de atributos - nesta camada os atributos são especificados, bem como as conexões entre as instâncias dos objetos. A figura IV.10.6 mostra um exemplo dessa camada;

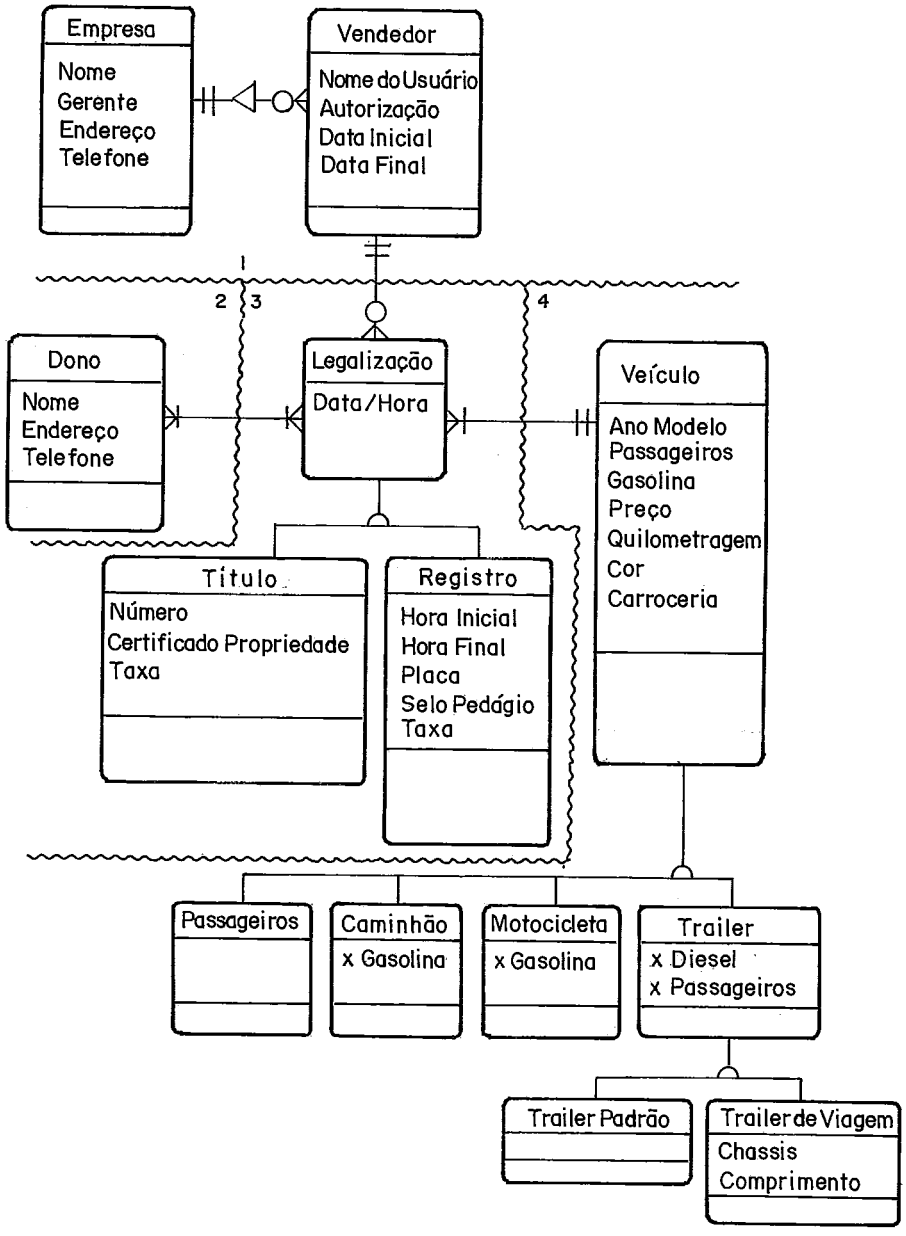


Figura IV.10.6 : Camada de Atributo
Fonte : (COAD, 1990)

- Camada de serviços - esta camada contém os serviços e as conexões de mensagens, cujo exemplo pode ser visto na figura IV.10.7.

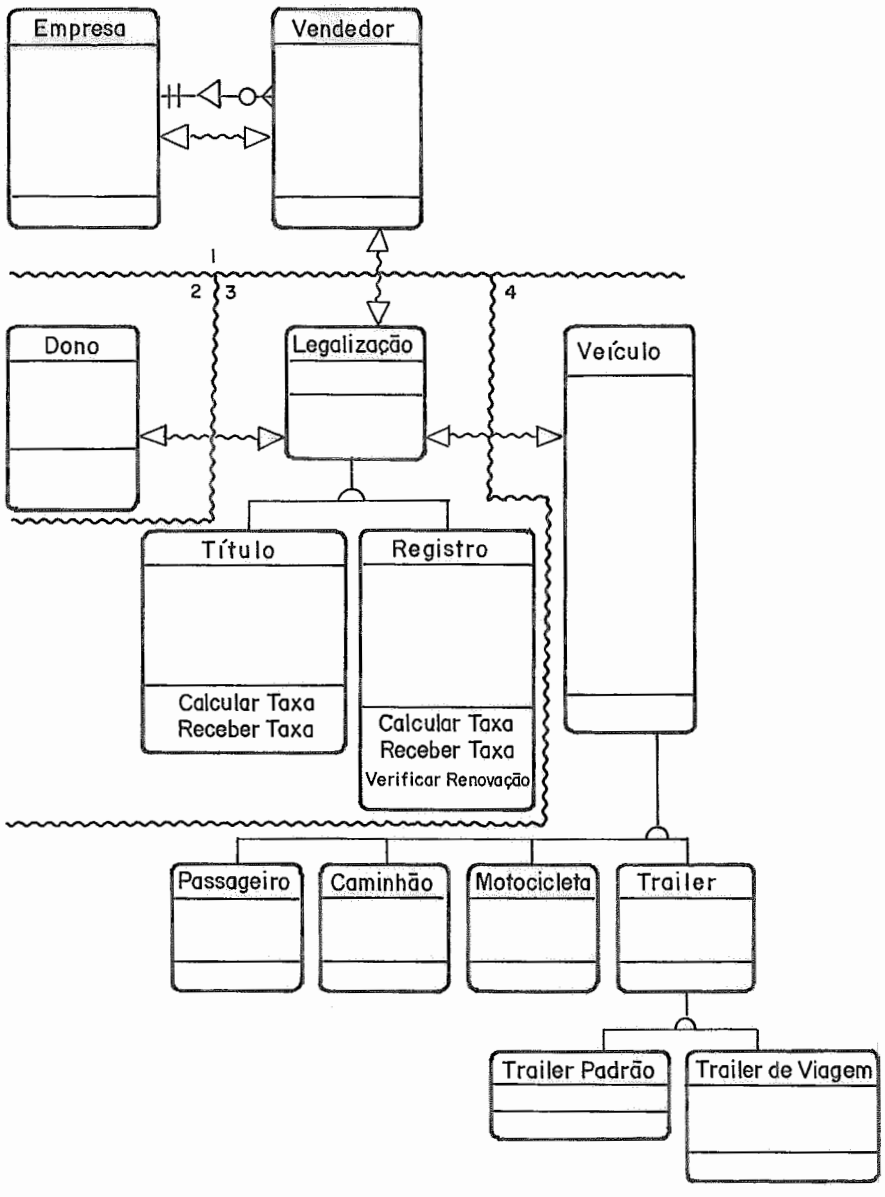


Figura IV.10.7 : Camada de Serviço
Fonte: (COAD, 1990)

A definição do modelo deve ser iniciada com a identificação dos objetos e estruturas, para a partir deles definir os assuntos relativos ao problema. Entretanto, quando é feita a representação gráfica do modelo em camadas, o método sugere que em primeiro lugar seja construída a camada de assuntos. Essa camada servirá de base para a representação dos objetos e estruturas. É a partir da camada de assuntos que se determinam as linhas de assuntos que irão delimitar as estruturas das demais camadas que compõem o modelo.

IV.11 "OBJECT-ORIENTED SPECIFICATION" (BAILIN, 1989)

O método "Object-Oriented Specification" (OOS) tem por objetivo auxiliar na análise e projeto de sistemas, segundo a abordagem de orientação a objetos. Utiliza instrumentos dos métodos "Structured Systems Analysis" (SSA) (DEMARCO, 1978) e "Entity-Relationship Model" (ERM) (CHEN, 1977) para ajudar na especificação dos requisitos. Foi desenvolvido por Sidney Bailin.

O método OOS utiliza os seguintes instrumentos:

- Linguagem para Diagrama de Entidade-Relacionamento (DER) do método "Entity-Relationship Model";
- Linguagem para Diagrama de Fluxo de Dados (DFD) do método "Structured Systems Analysis";
- Linguagem para Diagrama de Objetos (DO);
- Dicionário de Entidades.

- Linguagem para Diagrama de Fluxo de Dados e Entidades (DFDE) - o DFDE representa o fluxo de dados entre as entidades do sistema e está baseado na proposta de (STARK, 1987). A idéia básica do DFDE não é transformar as entradas em saídas, mas sim determinar o conteúdo de cada entidade.

A notação da linguagem desse diagrama é semelhante à da linguagem para DFD, com exceção do conceito de entidades ativas e passivas que é incluído no DFDE. Para diferenciar as entidades das funções em um DFDE, deve-se colocar o nome de cada entidade entre colchetes.

A linguagem para DFDE é composta pelos seguintes elementos, cujo exemplo pode ser visto na figura IV.11.1:

- entidade ativa - representada por um círculo e pode ser decomposta em funções ou subentidades; é semelhante a um processo do DFD;
- função - representada por um círculo e pode ser executada por ou agir sobre alguma entidade; pode ser decomposta em subfunções;
- entidade passiva - representa um fluxo de dados ou um depósito de dados;
- fluxos de dados entre as funções e as entidades;
- entidade externa - fornece ou recebe os dados que são manipulados pelo sistema.

O método assume a existência de uma declaração de requisitos textuais, que pode inclusive estar armazenada em uma base de dados de requisitos. Além disso, OOS aproveita

o conhecimento e a experiência que os analistas de sistemas têm do método SSA, a fim de utilizar o DFD como instrumento para identificar os requisitos do produto com mais facilidade.

A partir das principais entidades do problema, é construído o DFDE, com o auxílio de um DER. A construção do DFDE é um processo paralelo de decomposição de entidades e alocação de funções. A partir do DFDE, com o auxílio de uma ferramenta para geração de projeto, é feita a passagem automática da fase de análise para a fase de projeto, sendo construído um DO, que pode ser implementado na linguagem Ada.

A especificação de requisitos de OOS é formada por um conjunto de DERs e por uma hierarquia de DFDEs representada através de três níveis:

- primeiro nível, composto por entidades de nível mais alto, divididas em subentidades;
- nível intermediário, onde estão entidades de nível mais baixo;
- último nível, formado por funções identificadas a partir das entidades do nível intermediário e pela decomposição dessas funções.

OOS utiliza as seguintes ferramentas:

- Ferramenta para verificação da consistência entre os elementos dos DERs e dos DFDEs;

- Ferramenta para transformação automática de DFDEs em DOs, cujos elementos correspondem a construtores da linguagem Ada.

A especificação de requisitos do software é feita, utilizando-se o método OOS, através dos seguintes passos:

- Identificar as entidades chaves do domínio do problema - nesta etapa podem ser utilizadas uma das seguintes estratégias para a identificação das entidades relevantes do domínio do problema:

  . a partir de nomes e frases nominais extraídos de uma especificação de requisitos textuais, conforme proposto por BOOCH (1983), ou de uma base de dados de requisitos automatizada, ou

  . a partir de um DFD, de onde é retirado o nome do objeto que executa cada processo.

O DFD é utilizado como um instrumento para identificação das entidades do problema. Nesse passo é também construído o DER contendo as principais entidades do problema, identificadas a priori, e os seus relacionamentos. Além disso, é criado um Dicionário de Entidades análogo ao Dicionário de Dados do método SSA, conforme proposto por (STARK, 1987).

- Determinar quais entidades do DER são ativas e quais são passivas - essa distinção é feita a partir das seguintes definições:

. entidades ativas – aparecem como processos no DFDE e podem ser decompostas em subentidades ou em funções e são importantes para a fase de análise;

. entidades passivas – aparecem como fluxos ou depósitos de dados no DFDE.

– Estabelecer o fluxo de dados entre as entidades ativas – neste passo inicia-se a construção do DFDE, sendo construído o DFDE de mais alto nível, composto apenas de entidades. Isso garante que cada função seja fornecida por alguma entidade.

Nesse ponto deve-se procurar garantir a consistência entre o DER e o DFDE. Para tanto o método sugere as seguintes regras de consistência:

. cada entidade do DER deve corresponder a uma entidade no DFDE;

. todos os relacionamentos do DER devem ser expressos de alguma forma no DFDE.

– Decompor as entidades ou funções em subentidades e/ou funções – quando este passo é executado pela primeira vez, é feita a decomposição do DFDE elaborado na etapa anterior. As entidades ativas são então decompostas em subentidades e/ou funções. As funções são decompostas em subfunções nos diagramas de maior detalhamento. A figura IV.11.2 mostra o exemplo da decomposição de uma entidade ativa.

A identificação de uma função é feita através de:

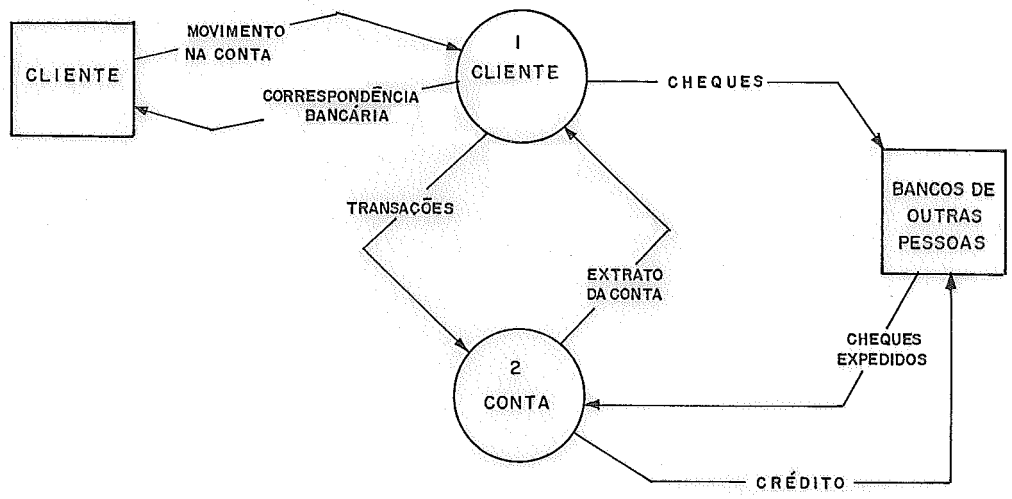


Figura IV. II.1 : Diagrama de Fluxo de Dados e Entidades do Método OOS
Fonte: (BAILIN, 1989)

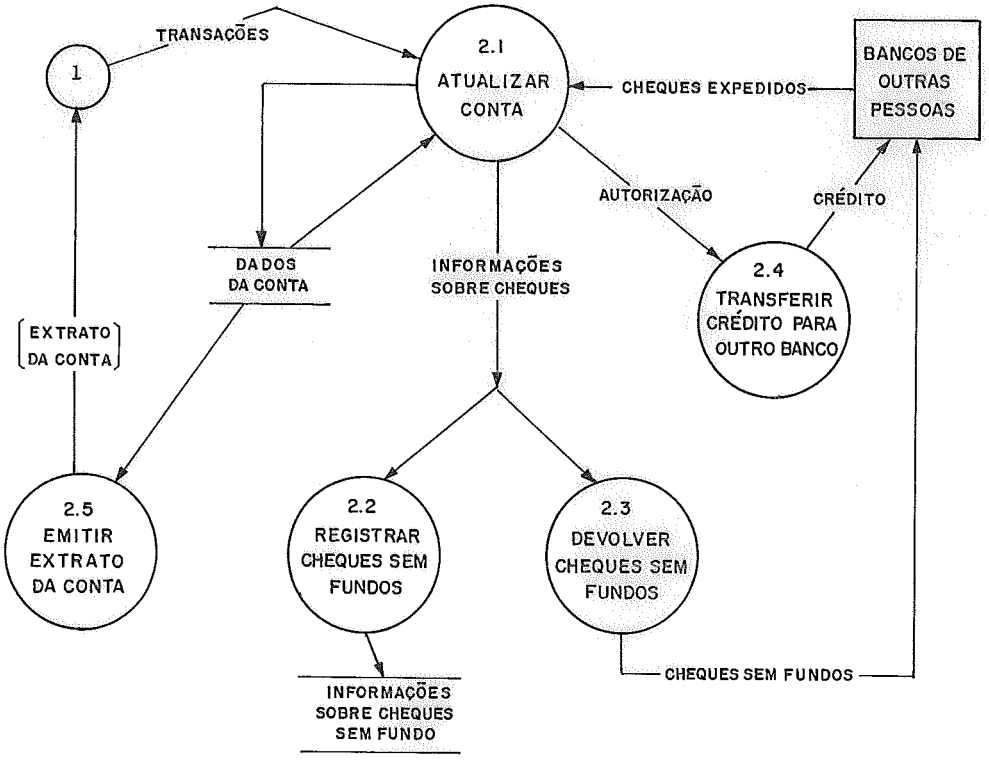


Figura IV. II.2 : Decomposição da Entidade Conta da Figura IV. II.1
Fonte: (BAILIN, 1989)

. verbos e frases verbais retirados de uma especificação de requisitos textuais ou de uma base de dados de requisitos automatizada, ou
. os nomes dos processo do DFD do sistema.

- Verificar a existência de novas entidades - a cada estágio de decomposição do DFDE deve-se verificar se as funções produzidas implicam a existência de novas entidades. Para cada função incluída no passo anterior, deve ser considerada a entidade que a executa. Os fluxos e depósitos de dados incluídos no passo anterior também devem ser examinados a fim de ver se são significantes o suficiente para serem incluídos como entidades passivas no DER.

- Agrupar funções em novas entidades - esta etapa inicia-se com a identificação das funções executadas pelas novas entidades incluídas no passo anterior. Essas funções devem ser reorganizadas e agrupadas, quando for possível, em novas entidades, contribuindo, assim, para o desenvolvimento de componentes reutilizáveis.

- Atribuir as novas entidades a domínios adequados - este passo visa gerenciar a complexidade da proliferação de entidades. Para tanto, deve-se fazer a correspondência de cada nova entidade a um determinado domínio. Se o DER se tornar complexo, é possível reduzir essa complexidade através da representação das entidades relacionadas a cada domínio em um diagrama separado.

Pode-se observar que o processo de especificação de requisitos do sistema é um processo de decomposição iterativo e que existe uma interação entre as entidades e as funções decompostas, fazendo com que entidades gerem novas funções e funções sugiram novas entidades. Com isso tem-se a inclusão de decomposição funcional dentro da abordagem de orientação a objeto.

Após a fase de especificação de requisitos, os DFDEs são transformados automaticamente em DOs. O gerador de projeto mapeia entidades do DFDE em objetos ordinários no DO, que correspondem a "packages" da linguagem Ada.

As funções de mais alto nível, por sua vez, são transformadas em objetos degenerados no DO, que correspondem a tarefas ou procedimentos de Ada. Já as funções de mais baixo nível não são mapeadas para DO, pois elas correspondem a suprogramas das tarefas e procedimentos da linguagem Ada.

IV.12 "OBJECT-ORIENTED SYSTEMS ANALYSIS" (SHLAER, 1988)

"Object-Oriented Systems Analysis" (OOSA) é um método para especificação de requisitos que tem por objetivo principal elaborar o modelo da informação do sistema. O método também gera o modelo de estados e o modelo de processos, que complementam a definição do sistema. Foi desenvolvido por Sally Shlaer e Stephen Mellor.

O método utiliza os seguintes instrumentos:

- Linguagem para Diagrama da Estrutura da Informação (DEI) o DEI é utilizado para identificar os objetos do domínio do problema, seus atributos e relacionamentos. A notação gráfica dessa linguagem está baseada nos Diagramas de Entidade-Relacionamento de CHEN (1977), TSICHRITZIS (1982) e MARTIN (1985) e nos Diagramas de Bachman (MARTIN, 1977), sendo composta pelos seguintes símbolos:

  . retângulo - representa um objeto e especifica o nome do objeto e os seus atributos;
  . flecha - representa um relacionamento binário entre objetos;
  . flecha com um pequeno losango - representa um relacionamento que gera um objeto associativo;
  . barra pequena cortando uma flecha - representa um objeto supertipo e seu relacionamento com objetos subtipos.

- Linguagem para Diagrama de Contexto da Estrutura da Informação (DCEI) - esta linguagem possui a mesma estrutura e simbologia utilizadas pela linguagem para construção de DEI, com exceção dos nomes dos atributos que no DCEI são omitidos do símbolo de objeto. A figura IV.12.1 mostra um exemplo do DCEI.

- Linguagem para Diagrama de Transição de Estados (DTE) - o DTE é utilizado para representar as mudanças de estados dos objetos especificados no modelo da informação. A notação gráfica dessa linguagem é composta pelos seguintes símbolos:

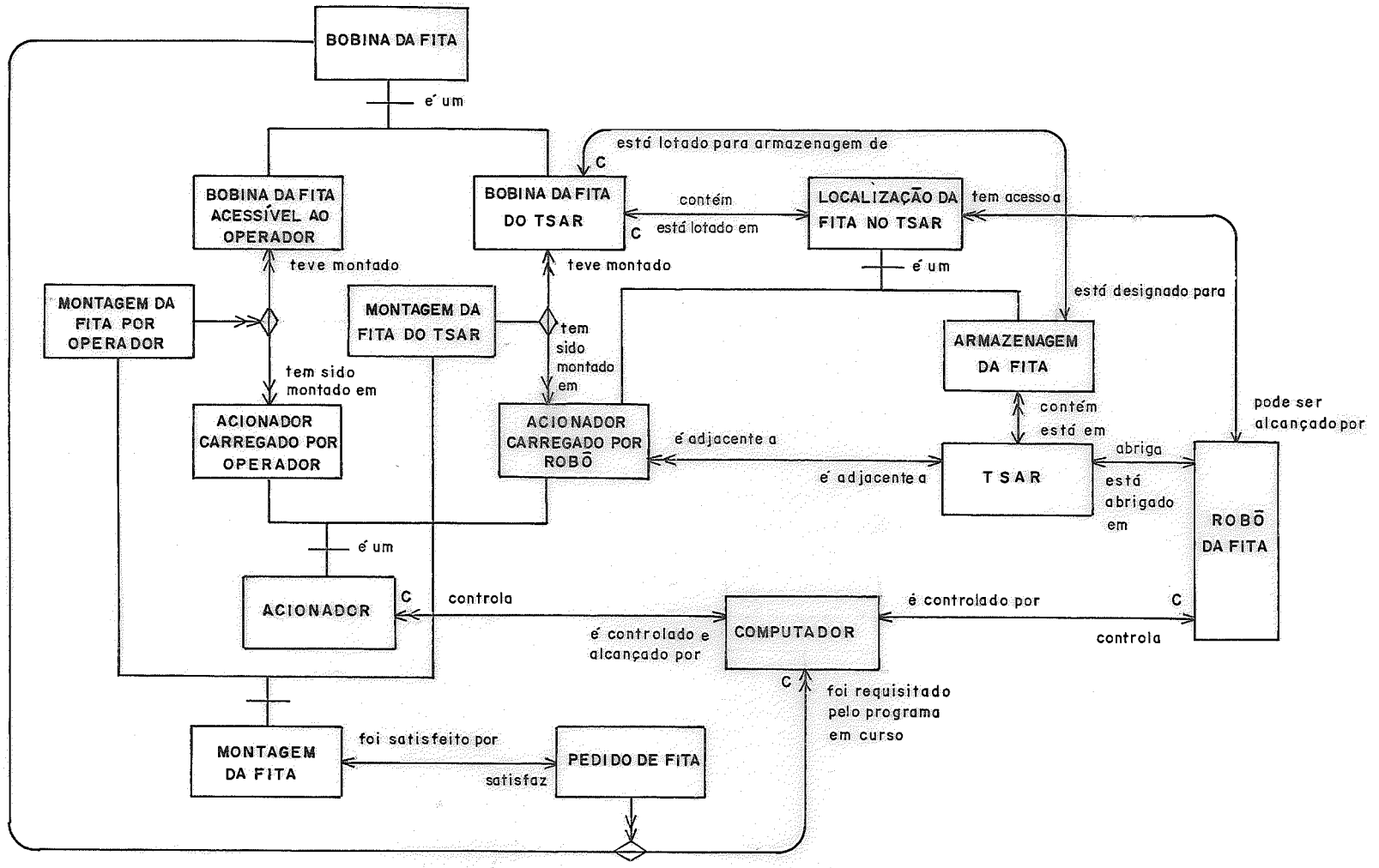


Figura IV.12.1: Diagrama de Contexto da Estrutura da Informação
Fonte: (SHLAER, 1988)

. retângulo - representa um estado;

. flecha - representa as transições de estado e contém o nome do evento que ocasiona a mudança.

No DTE também são especificadas as atividades associadas a cada estado, que são realizadas quando o estado está ativo. A definição das atividades pode ser feita utilizando-se português estruturado.

- **Linguagem para Diagrama de Fluxo de Dados (DFD)** - o DFD é utilizado para modelar as atividades identificadas no modelo de estados, responsáveis pela transformação dos dados. A notação gráfica dessa linguagem segue a abordagem de (DEMARCO, 1978), sendo composta dos seguintes símbolos:

  . bolha - representa um processo ou atividade;

  . flecha - representa um fluxo de dados;

  . linhas paralelas - representa um depósito de dados.

- **Linguagens para Especificação de Processo** - a especificação dos processos é elaborada através de português estruturado, pseudocódigo ou linguagem para árvore de decisão.

O método OOSA utiliza esses instrumentos para gerar os modelos que constituem a especificação de requisitos:

- **Modelo da informação** - é a representação e a especificação do domínio do problema em uma estrutura baseada nos conceitos de objetos, atributos e relacionamentos. É composto por:

. uma parte gráfica, representada pelos diagramas DEI e DCEI, e

. uma parte textual, onde são especificados os objetos, atributos e relacionamentos que formam o modelo. É apresentada pelos seguintes documentos:

. Documento de Especificação de Objeto (DEO) - o DEO é escrito em linguagem natural e mostra a realidade que está sendo modelada e como essa realidade tem sido formalizada através da descrição dos objetos do modelo e de seus atributos. O DEO contém as seguintes informações:

- relação de todos os objetos incluídos no modelo;
- especificação de cada objeto do modelo, contendo nome do objeto, identificadores, descrição e seus atributos. Os atributos do objeto, por sua vez, são especificados através do nome do atributo, descrição, função e domínio.

. Documento de Especificação de Relacionamentos (DER) o DER é escrito em linguagem natural e contém a identificação e a descrição dos relacionamentos de cada objeto, com a sua cardinalidade.

. Documento de Resumo (DR) - o DR é escrito em linguagem natural e apresenta um resumo dos documentos DEO e DER, mostrando os objetos do modelo da informação com seus atributos e relacionamentos.

- *Modelo de estados* - mostra o comportamento dos objetos do modelo da informação, isto é, as transformações

ocorridas nos objetos ao longo do tempo. é representado graficamente através do DTE, que retrata a mudança de estados de cada objeto do sistema.

- Modelo de processo - define o detalhamento das atividades identificadas no modelo de estado. é composto por:

- uma parte gráfica, representada pelo DFD, e,
- uma parte textual, representada pela especificação dos processos.

Os modelos da informação, de estados e de processos formam a especificação de requisitos do sistema. A fase de análise é iniciada com a construção do modelo da informação. Em seguida é elaborado o modelo de estados, e por último o modelo de processos. Os modelos gerados por OOSA são integrados de acordo com uma série de regras que são descritas a seguir:

- todos os objetos do modelo da informação que apresentam mudanças de estado devem ser incluídos no DTE;
- as transições de estado dos objetos implicam na realização de atividades;
- atividades identificadas no DTE são representadas como processos no DFD;
- cada objeto identificado no modelo de informação torna-se um depósito de dados no DFD;

- os elementos de dados incluídos nos depósitos de dados são atributos definidos no modelo da informação;
- os processos só manipulam dados definidos no modelo da informação;
- os processos podem gerar os eventos que aparecem no DTE;
- para cada objeto do modelo de estados deve-se incluir, no modelo da informação, o atributo estado com o seu domínio;
- o domínio do atributo estado, definido no modelo da informação, é formado por uma lista contendo os possíveis estados que um objeto pode assumir, identificados no DTE.

Como pode ser observado, o modelo da informação fornece a base para a construção do modelo de estados e do modelo de processos. Esses modelos podem ser utilizados, durante a fase de projeto, para determinar quais as partes do sistema que deverão ser automatizadas e os dados requeridos.

O método OOSA, portanto, auxilia a elaborar uma especificação de requisitos composta de diferentes modelos que se complementam e podem ser utilizados como ponto de partida para as demais fases do processo de desenvolvimento do software.

IV.13 "PROTOTYPING OF USER INTERFACES" (CHRISTENSEN, 1984)

"Prototyping of User Interfaces" (PUI) é uma proposta para a especificação da interface de Sistemas Interativos (SIs), através da construção de protótipos do diálogo homem-máquina. Foi desenvolvido por Niels Christensen e Klaus-Dieter Kreplin.

A especificação da interface com o usuário de um SI é composta, segundo os autores, de duas partes principais:

- Especificação da dinâmica - consiste na definição das seqüências válidas da interação homem-máquina. A especificação da dinâmica é subdividida em:

  . especificação da estrutura de diálogo, através de estruturas de blocos e construtores de fluxo de controle orientados a diálogo, e,

  . especificação do processamento e das condições dos fluxos de controle associados a cada elemento da estrutura de diálogo.

- Especificação do "layout" de tela - consiste na definição da aparência do diálogo entre o software e o usuário. A especificação do "layout" de cada tela é subdividida, por sua vez, em:

  . especificação da estrutura do "layout" de tela;

  . especificação das máscaras, ou seja, da imagem da máscara, dos atributos dos dados, etc.

Para a especificação da interface com o usuário é utilizado como instrumento uma linguagem para especificação de protótipo, usada para especificar a estrutura do diálogo e o "layout" das telas. Essa linguagem pode ser interpretada por uma ferramenta de simulação.

A linguagem para protótipo de interface possui a seguinte sintaxe:

- cada declaração é escrita em uma linha;
- os ninhos de blocos são representados através de identação;
- M indica máscara;
- G indica grupo de campos;
- F indica um campo único.

As estruturas de controle padrões são escritas como prefixo em cada uma das linhas:

- '-' indica seqüência;
- '.' indica interação;
- '+' indica iteração.

A figura IV.13.1 mostra um exemplo da aplicação da linguagem para definir a estrutura de um diálogo e a figura IV.13.2, a especificação do "layout" de uma tela.

As especificações da dinâmica e do "layout" de tela estão relacionadas entre si. Um diálogo é uma seqüência de interações homem-máquina com o propósito de satisfazer determinadas regras descritas pela estrutura de diálogo.

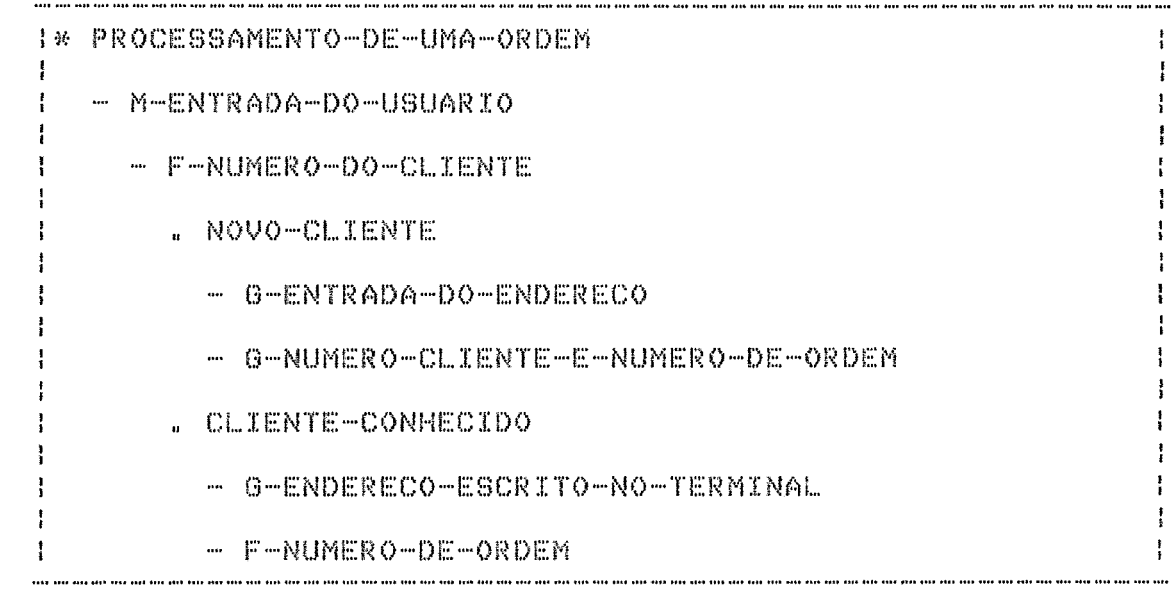


Figura IV.13.1: Especificação da Estrutura de um Diálogo

Fonte: (CHRISTENSEN, 1984)

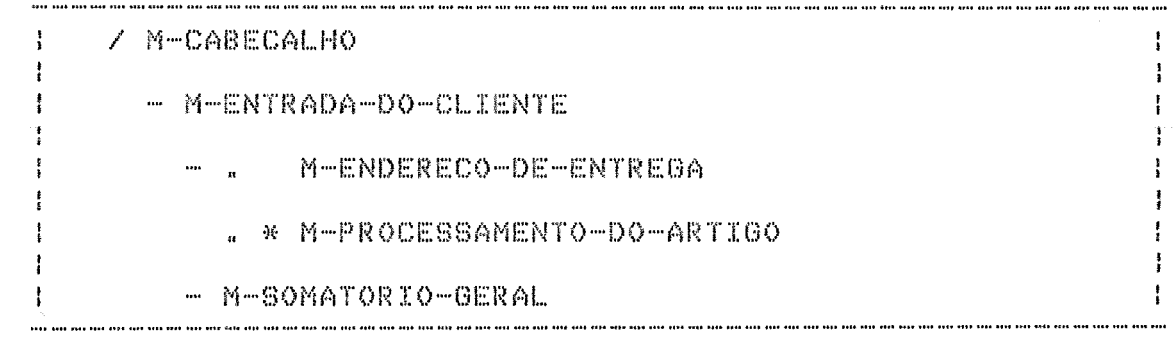


Figura IV.13.2: Especificação do "Lay-out" de uma Tela

Fonte: (CHRISTENSEN, 1984)

A execução de um elemento do diálogo que corresponde, por exemplo, a uma máscara, levará à emissão dessa máscara em uma tela, de acordo com a estrutura do "layout" de tela. A mesma regra vale para os elementos de diálogos correspondentes a itens específicos de uma máscara.

Foi desenvolvido um sistema de gerência de máscara com uma série de características, tais como por exemplo:

- uma máscara ocupa apenas uma parte da tela e pode conter janelas onde outras máscaras são aninhadas;
- uma máscara contém a definição da apresentação visual dos dados (edição, atributos e texto);
- o fluxo de controle de uma máscara é manuseado pela máscara de forma independente do programa.

O protótipo da interface com o usuário é desenvolvido nas fases iniciais da construção do produto e pode ser utilizado como uma especificação. O protótipo gerado pode ser descartado após a fase de análise ou ser uma parte da implementação final do produto.

IV.14 "RAPID INTELLIGENT PROTOTYPING LANGUAGE" (COCHRAN, 1985)

"Rapid Intelligent Prototyping Language" (RIPL) tem por objetivo auxiliar no projeto de interface usuário-sistema (IUS). Oferece um ambiente automatizado para a construção e avaliação da IUS, através do uso de protótipos. Com RIPL os diálogos são especificados e editados interativamente e o protótipo da IUS pode ser utilizado para medir automaticamente o desempenho dos usuários. Foi desenvolvido por Duane Cochran e Frederick Stocker.

RIPL possui dois tipos de usuários: o projetista de IUS e o usuário final. O projetista gera uma simulação do ambiente de interação proposto para o usuário final. RIPL permite ao projetista executar o protótipo, fazendo o papel

do usuário, possibilitando a construção de interfaces que possam inclusive ser utilizadas para treinamento.

O sistema é desenvolvido a partir da descrição da forma como ele deverá operar. Para isso deve-se mostrar, detalhadamente, como o sistema será visto e manipulado pelo usuário para realizar uma tarefa específica. Assim a preocupação com as necessidades básicas do usuário final concentra-se no início do processo de desenvolvimento e os requisitos funcionais do sistema são definidos em função da visão do usuário final.

O projetista de IUS especifica um conjunto de propriedades detalhadas para a interface com o usuário a partir dos objetivos e requisitos estabelecidos para o novo sistema. Durante a especificação da interface, o projetista conta com a ajuda de um Sistema Especialista e com uma base de Conhecimento capaz de recomendar elementos de projetos alternativos e fornecer um acesso "on line" a fatores humanos relacionados com a interface. O Sistema Especialista, por sua vez, é capaz de detectar problemas na interface e sugerir mudanças ao projetista.

Quando a IUS tiver sido especificada pelo projetista, sua simulação pode ser executada pelo usuário final, a fim de avaliar a adequação do projeto da interface. Durante o uso do protótipo, dados sobre o desempenho do usuário final podem ser armazenados e utilizados para otimização do sistema. O protótipo pode ser rapidamente modificado quando for necessário.

O Sistema RIPL utiliza como instrumento a linguagem para Diagrama de Estados, composta dos seguintes símbolos:

- caixas - representam os estados. Cada caixa contém "display" visíveis para o usuário final, que podem ser textuais ou nomes de "ícones";

- arcos com nomes - representam a transição de estados. A hierarquia entre os estados pode ser difinida através de superestados de forma a garantir uma exposição gradual de detalhes.

RIPL oferece apoio automatizado através de quatro subsistemas básicos. A figura IV.14.1 mostra a arquitetura de RIPL:

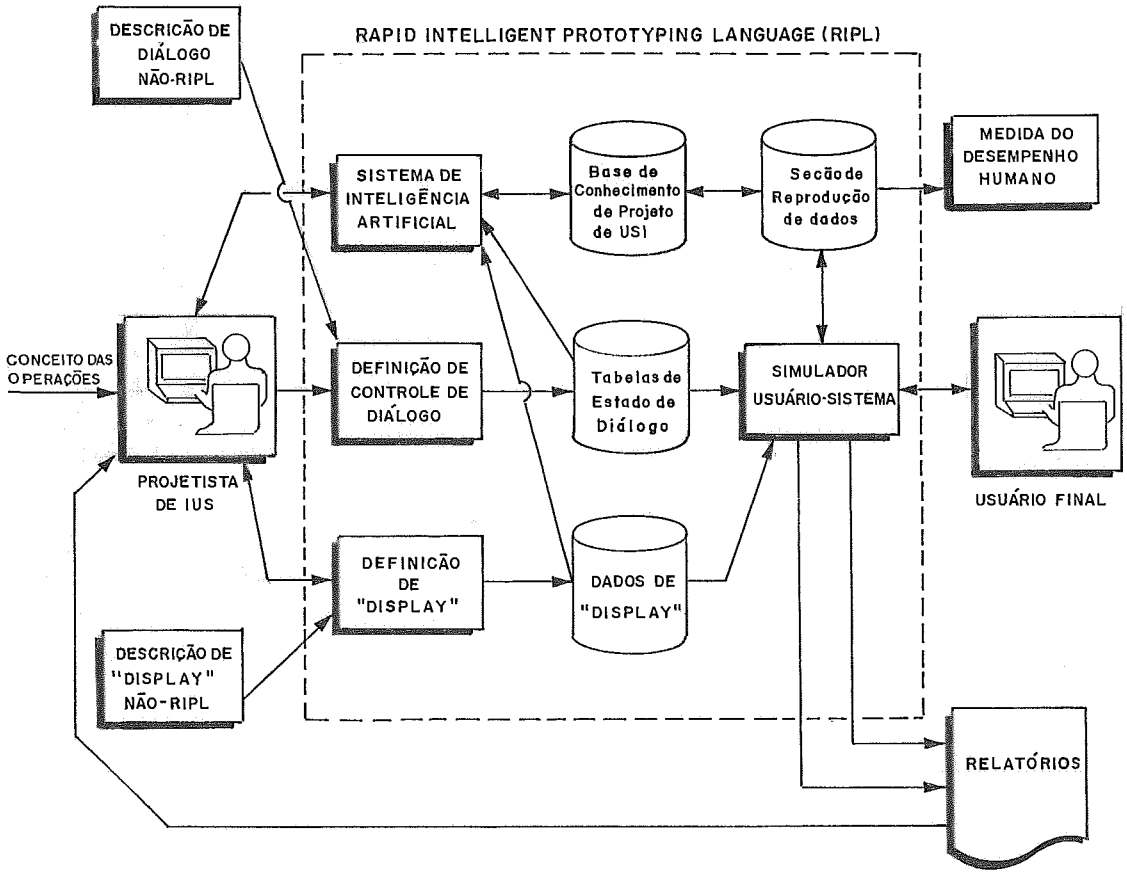


Figura IV.14.1: Arquitetura Básica de RIPL
Fonte: (COCHRAN, 1985)

- Sistema de Inteligência Artificial (SIA) - é formado por três subsistemas:

  . Sistema Especialista (SE) para projeto de IUS - o SE possui uma máquina de inferência independente da aplicação, funções para análise e avaliação de IUS e um banco de dados de fatores humanos. A comunicação com o projetista é feita através de gráficos. O SE pode operar de dois modos:

    - como um monitor, que controla os diálogos e as especificações de "display", de forma implícita e automática;
    - sendo chamado pelo projetista de forma explícita para responder perguntas na base de conhecimento e avalição da interface.

  . Sistema Guia - oferece um apoio informativo sobre as tarefas ao usuário. Também possui uma característica tutorial, provendo uma modelagem especialista em áreas onde o conhecimento baseado em regras é difícil de codificar;

  . Auxílio para Treinamento do Projetista - inclui a modelagem de cores, estrutura, movimentos e interações. Também provê exemplos da interface e tutores "on line".

Esses subsistemas acessam a base de conhecimento de projeto de IUS, as tabelas de estado de diálogo e dados de "display".

- Sistema de Definição de Controle de Diálogo (SDCD) - permite a construção interativa e rápida de diagramas de estado que representam graficamente os diálogos em uma forma orientada para o usuário. Contém um construtor de tabela de transição de estados, um editor de diálogo, um gerente de menu, uma interface de software de aplicação, bem como uma biblioteca de tradutores, usada para importar descrições de diálogo, feitas por outras ferramentas. Isso possibilita a interação de RIPL com outras ferramentas de prototipagem.

- Sistema de Definição de Display (DD) - inclui um pacote gráfico independente de dispositivo, construções de elementos gráficos, especificações de atributos gráficos, biblioteca de símbolos, editor de "display", construtor de menu, gerador de grades e um banco de dados de "display" independente de dispositivo.

- Sistema de Simulação Usuário-Sistema (SSUS) - responsável pela implementação da especificação da interface elaborada pelo projetista. Inclui características para gravar e manusear seções quando for necessário, medir o desempenho humano, tratar exceções e gerar relatórios. Permite a geração de código da interface para as partes de um protótipo validado. Essa característica de geração de código facilita a transição do protótipo para a versão final da interface.

Os componentes SIA, SDCD e DD de RIPL são responsáveis pela geração do ambiente de simulação do usuário final.

IV.15 "SCENARIO-BASED REQUIREMENTS ELICITATION" (HOLBROOK, 1990)

"Scenario-Based Requirements Elicitation" (SBRE) é um método para elicitação de requisitos, desenvolvido no Departamento de Ciência da Computação da Academia de Força Aérea dos Estados Unidos. O método tem por objetivo estruturar as interações entre projetistas e usuários de modo a obter rapidamente os requisitos iniciais. Para isto SBRE utiliza cenários como recurso.

Cenários podem ser vistos como uma descrição de uma determinada instância de uma interação homem-computador. Por causa de seu baixo custo e de sua expressividade limitada, os cenários são mais apropriados para comunicar características específicas do software em situação de alta incerteza.

Além da utilização de cenários, o método enfatiza o desenvolvimento paralelo de requisitos e de um projeto em alto nível, o uso de uma função de avaliação para alcançar um projeto viável e a existência de uma base para manter as questões que surgem durante o processo de elicitação.

SBRE não é um substituto para as atividades das fases de análise e projeto. O que ele procura é coordenar essas atividades de forma a gerar um conjunto exato de requisitos.

A arquitetura conceitual que suporta o método está compreendida por quatro conjuntos de informações que são

obtidas e manipuladas durante o processo de elicitação dos requisitos. Esses conjuntos são descritos a seguir:

- **Conjunto de Objetivos** - neste conjunto são definidos os requisitos do software, as restrições e os recursos disponíveis. Essas informações permitirão determinar a viabilidade do projeto e fornecem critérios para avaliar as alternativas de projeto.

O conjunto de objetivos é representado como uma decomposição em um conjunto de subobjetivos. Isso é uma forma de estruturar os requisitos e permitir o seu detalhamento e evolução. Esses requisitos devem ainda ter sua prioridade determinada, principalmente nas situações em que a ordem de tratamento dos subobjetivos tem um grande impacto na solução final do problema.

- **Conjunto de Projetos** - este conjunto serve de repositório para as decisões de projeto que permitirão construir o software desejado. Vários modelos de projeto podem ser definidos, sendo que cada um deles está associado a uma solução específica e pode representar diferentes configurações de software e/ou hardware necessárias para alcançar os objetivos.

- **Conjunto de Cenários** - esse conjunto é a visualização do projetista de como o projeto do sistema irá executar os objetivos definidos no conjunto de objetivos. Na essência, ele é uma especificação do comportamento, que apreende como o software reage a seu ambiente.

Cenários são desenvolvidos para representar uma situação específica e estão diretamente relacionados aos modelos do projetos que eles representam. Os cenários são agrupados em subconjuntos, de forma que haja a correspondência, de um para um, entre os subconjuntos de cenários e seus modelos de projetos. Um cenário pode estar relacionado a vários objetivos e como resultado, eles ligam as características da solução ou projeto aos objetivos específicos.

- Conjunto de Questões - uma questão pode ser vista como um problema, posição ou indagação relevante para um projeto. Nesse conjunto podem existir quatro tipos de questões:

  - suposições, que são feitas pelo projetista quando existem lacunas no conjunto de requisitos;

  - respostas, que expressam a reação dos usuários a uma solução proposta, representada pelos cenários;

  - implicações, que são restrições impostas na solução e/ou requisitos que estão baseados na adoção de uma determinada solução. As implicações podem ser:

    . externas ao conjunto de projetos, quando afetam os requisitos;
    . internas ao conjunto de projetos, quando agem como restricões, influenciando as considerações futuras de projeto;

- notas, que são informacões gerais ou perguntas que podem vir a ser importantes mais tarde no processo de elicitação de requisitos.

A figura IV.15.1 apresenta uma visão geral da arquitetura do método SBRE. O conjunto de objetivos situa-se no mundo do usuário e o de projeto, no mundo do projetista. A comunicação entre esses dois mundos é realizada através do conjunto de cenários e do conjunto de questões. A figura IV.15.2 mostra a arquitetura detalhada do SBRE, evidenciando as relações entre os conjuntos de informações.

O processo de elicitação dos requisitos é realizado, segundo o método SBRE, através de uma série de etapas onde os requisitos iniciais são refinados iterativamente, enquanto um projeto em alto nível é desenvolvido em paralelo. Os conjuntos de objetivos, projetos e cenários são construídos de forma concorrente e regulados pelo conjunto de questões.

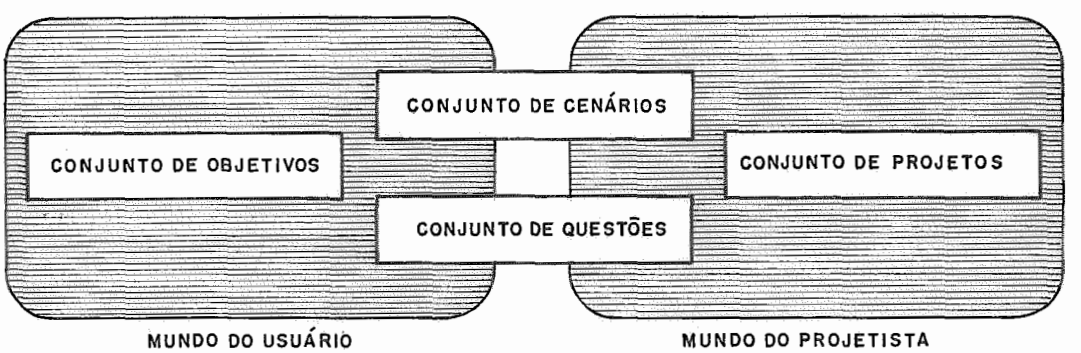


Figura IV.15.1 : Visão geral da arquitetura do método SBRE
Fonte: (HOLBROOK, 1990)

As etapas do processo de desenvolvimento do método SBRE formam um ciclo de geração/apresentação de cenários descrito a seguir:

- Fase de elaboração dos objetivos iniciais - nesta fase ocorrem as primeiras interações com o usuário, de onde são retirados os subsídios para a elaboração do conjunto de objetivos iniciais do sistema.

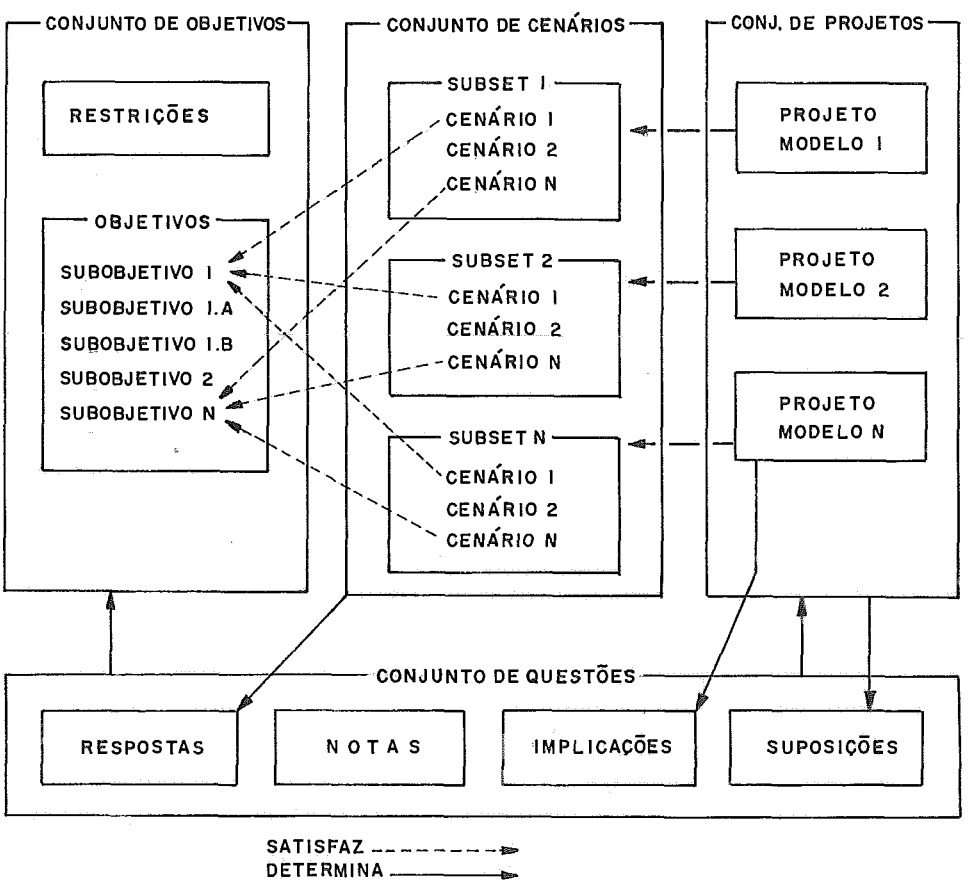


Figura IV.15.2 : Arquitetura detalhada do método SBRE
Fonte: (HOLBROOK, 1990)

Aqui são definidos, por meio de entrevistas ou síntese dos sistemas existentes, os requisitos e as restrições para o sistema, bem como as demais informações que forem consideradas relevantes. Os objetivos são refinados até que se possa construir um modelo inicial do projeto.

- Fase de geração de cenários - esta fase é a etapa de projeto do ciclo onde é elaborado um projeto que procure alcançar os objetivos identificados na etapa anterior e são desenvolvidos os cenários descrevendo o seu comportamento. Além disso, são geradas questões de suposição e implicação externa e interna, que são armazenadas no conjunto de questões.

Como pode ser visto na figura IV.15.3, a medida que o projetista formula modelos de projetos de como os objetivos podem ser alcançados, as considerações são armazenadas no conjunto de projetos.

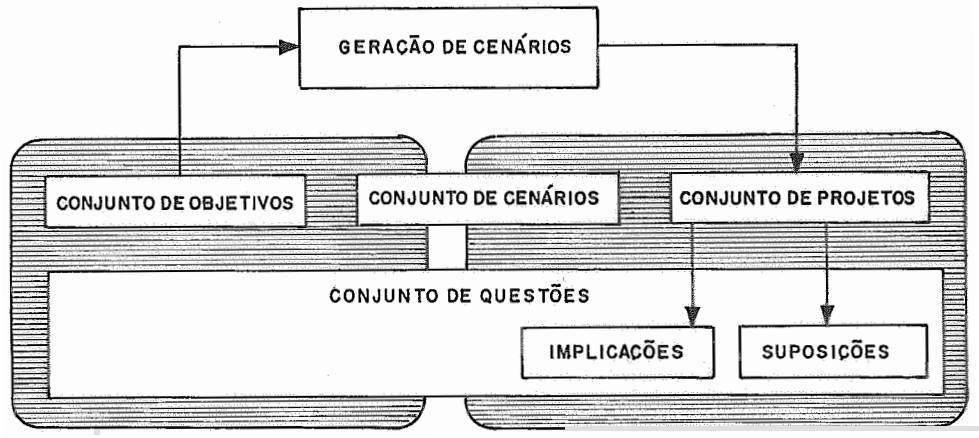


Figura IV.15.3 : Fase de geração de Cenários
Fonte: (HOLBROOK, 1990)

Vários modelos de projetos podem ser desenvolvidos e, baseando-se no comportamento observado desses modelos, são contruídos os cenários que demonstram como o modelo irá satisfazer requisitos específicos. Os cenários completam a ligação entre o modelo de projeto e o requisito específico, que, em conseqüência, mostra como um modelo de projeto irá satisfazer um objetivo particular.

- **Avaliação de cenários** - durante esta fase, os cenários que representam o comportamento de cada modelo de projeto são avaliados pelos usuários. É nesse ponto que verificamos se o projeto atingiu os objetivos definidos.

A figura IV.15.4 mostra a estrutura dessa fase. Nela pode ser observado que o usuário avalia um conjunto de cenários junto com as questões de suposições e implicações externas, geradas pelo projetista durante a fase de projeto e associadas a esses cenários.

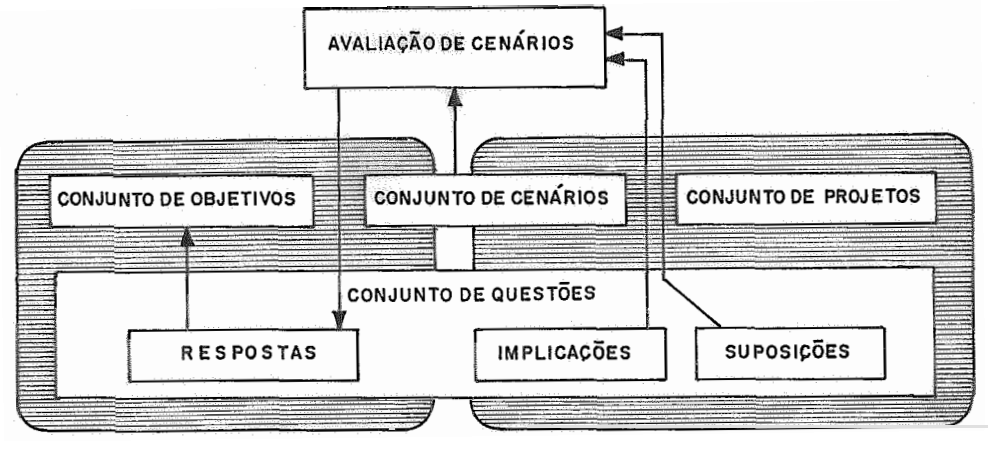


Figura IV.15.4: Fase de Avaliação de Cenários
Fonte: (HOLBROOK, 1990)

As suposições são revisadas para serem incluídas no conjunto de objetivos. As implicações externas, por sua vez, são revisadas a fim de considerar os efeitos no conjunto de objetivos decorrentes da adoção de um componente do conjunto de projeto. O efeito das implicações externas no conjunto de objetivos pode significar novas possibilidades para o projeto ou restrições no conjunto de objetivos.

A fase de geração de cenário é reiniciada utilizando o conjunto de objetivos modificados. O ciclo se repete até que todos os subobjetivos tenham sido atendidos pelos cenários aprovados pelos usuários, que são representações corretas do comportamento do projeto.

SBRE é um método manual cuja automatização pode ser efetuada por ferramentas que permitam estabelecer e manter os conteúdos e os relacionamentos existentes entre os conjuntos de informações. Existe uma ferramenta para o método SBRE em desenvolvimento no Centro de Pesquisas de Engenharia de Software da Universidade da Flórida.

Sistemas de hipertexto, como por exemplo o HyperCard da Apple, podem ser utilizado para a automação do método. Dentre outras vantagens, o HyperCard oferece mecanismos para geração de cenários. A figura IV.15.5 mostra a arquitetura detalhada de SBRE e como as ligações entre os conjuntos são representadas em HyperCard.

SBRE não é um método para construir modelos de software, mas sim um método para auxiliar nas primeiras

interações entre usuários e desenvolvedores, visando estabelecer, de forma clara e precisa, os requisitos do usuário. Como um método para obtenção de requisitos iniciais, pode ser utilizado em quaisquer domínios de aplicações e nos mais variados ambientes. A sua automatização permitirá que ele seja utilizado de forma mais rápida e eficiente.

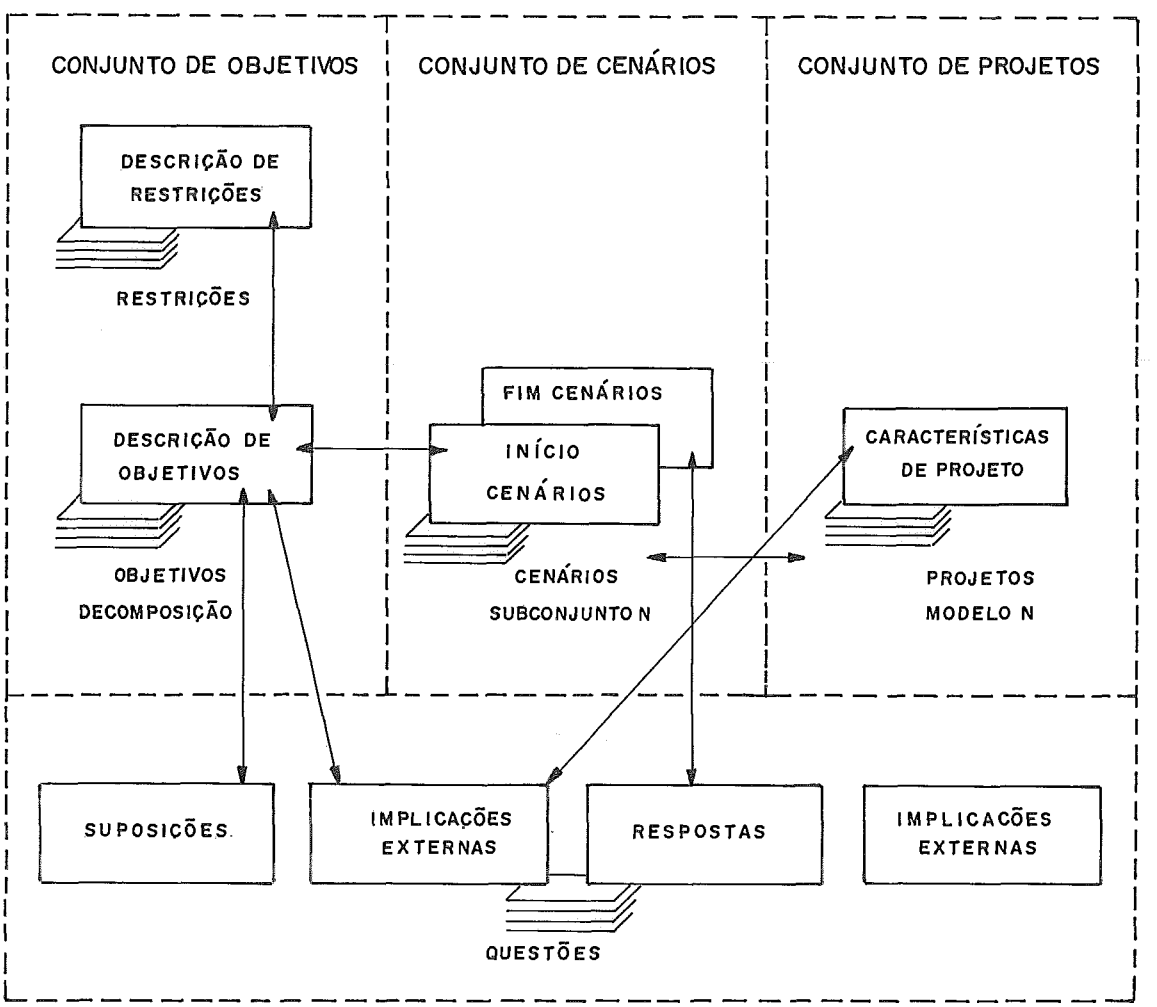


Figura IV.15.5 : Arquitetura de SBRE em Hypercard
Fonte: (HOLBROOK, 1990)

IV.16 "SOFTWARE REQUIREMENTS ENGINEEING METHODOLOGY" (ALFORD, 1977), (DAVIS, 1977), (ALFORD, 1980)

"Software Requirements Engineering Methodology" (SREM) é um método para especificação de requisitos de sistemas de tempo real. Foi desenvolvido pela TWR para o Departamento de Defesa.

Neste método, o problema é organizado como um conjunto de unidades, onde cada uma delas descreve todas as respostas requeridas pelo sistema para um determinado estímulo. Os requisitos são especificados como caminhos compostos por seqüências de processamentos, definidos em termos de estruturas de estímulos/resposta. A estrutura geral do método pode ser vista na figura IV.16.1.

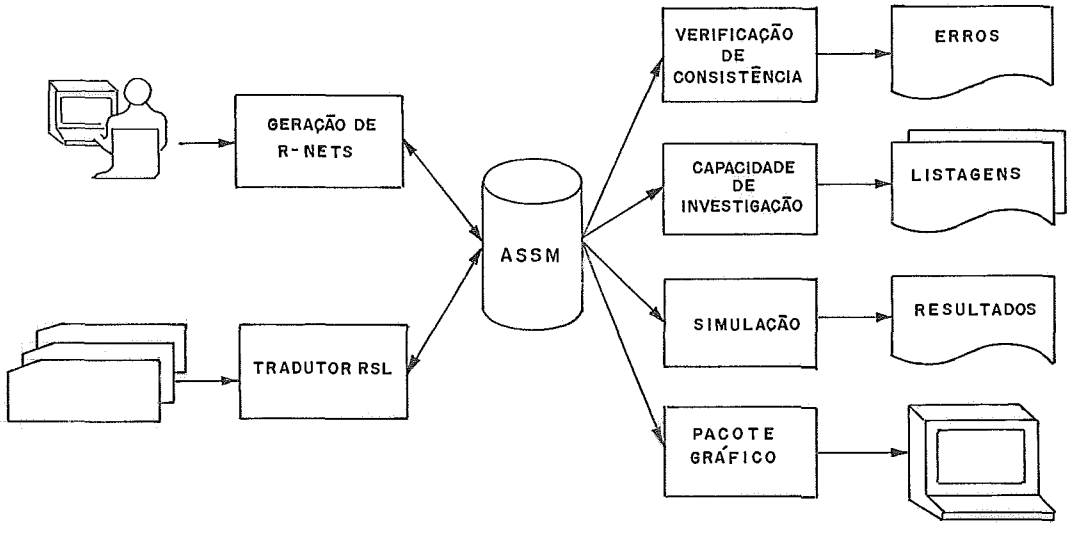


Figura IV.16.1: Estrutura Geral do Método SREM
Fonte: (DAVIS, 1977)

SREM utiliza como instrumento notacional uma linguagem processável em computador para a especificação dos requisitos, denominada de **"Requirements Statement Language" (RSL)**. A linguagem RSL é responsável pela entrada no Sistema REVS, interativa ou em "batch", dos requisitos do produto.

Os conjuntos de requisitos que correspondem a todos os comportamentos do sistema devem ser representados inicialmente como "R-Nets". Uma "R-Net" ("Requirements Network") é um conjunto de redes para representação de requisitos como seqüências de processos executados para prover respostas a estímulos. Uma "R-Net" consiste de:

- um conjunto de nós "ALPHAS", representando passos de um processamento;
- um conjunto de nós "OR", identificando as condições de um processamento e associados a uma variável de seleção. O nó "OR" possui uma entrada e várias saídas, mas apenas uma das saídas é executada, sendo a escolha baseada na condição associada a cada arco;
- um conjunto de nós "AND", que possibilitam que seqüências de "ALPHAS" associadas com ele sejam executados em qualquer ordem ou em paralelo;
- um conjunto de nós "FOR EACH", que determinam a execução de um "ALPHA" uma ou mais vezes;
- um conjunto de nós de interface, onde mensagens externas são recebidas ou transmitidas;

- um conjunto de nós iniciais e terminais;
- um conjunto de arcos ligando os nós;
- um conjunto de eventos nos caminhos da "R-Net", onde cada um dos eventos leva a "R-Net" à execução de determinados processamentos, e,
- um conjunto de pontos de validação usados para identificar caminhos e especificar de onde os dados devem ser retirados para testar o processo.

Em SREM os requisitos são especificados como conjuntos de "R-Nets", onde cada uma delas detalha a resposta do sistema a um determinado estímulo. Esse detalhamento é efetuado de forma "top-down", onde uma resposta é primeiro especificada em termos de um número restrito de "ALPHAS", apresentando uma visão geral do sistema.

Cada "ALPHA" deve ser expandido e a decomposição de um "ALPHA" é representada por um elemento do "R-Net" denominado de "SUBNET". A diferença entre um "ALPHA" e um "SUBNET é que um "ALPHA representa o nível mais elementar na hierarquia de uma "R-Net", enquanto que um "SUBNET" é uma estrutura possível de ser subdividida.

Os requisitos são, portanto, definidos através de "R-Nets" e codificados na linguagem RSL, a fim de que possam ser processados para a geração de protótipos do sistema. A figura IV.16.2 mostra o exemplo de uma "R-Net", enquanto que a figura IV.16.3 mostra a representação dessa "R-Net" na linguagem RSL.

O método SREM é automatizado por um conjunto integrado de ferramentas, denominado **"Requirements Engineering and Validation Systems" (REVS)**, cujo objetivo é auxiliar na construção, verificação e validação de requisitos escritos na linguagem RSL.

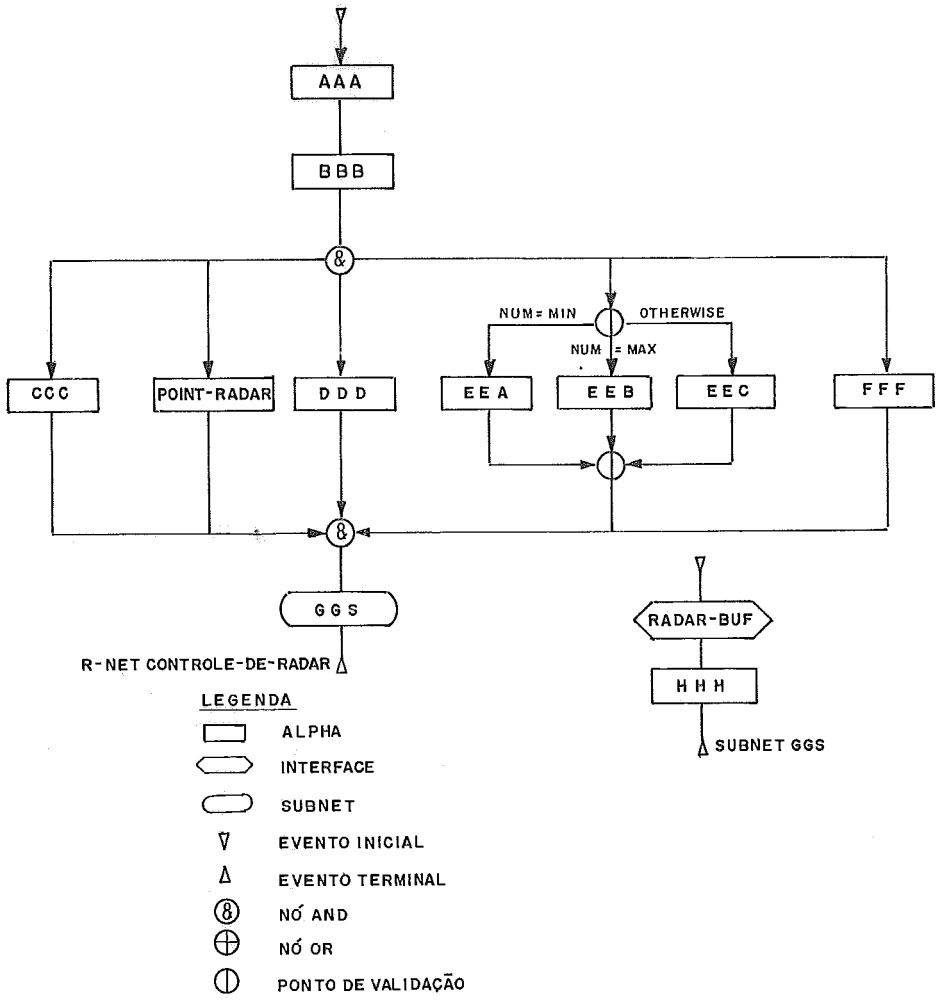


Figura IV.16.2 : Representação de um R-Net
Fonte: (BELL , 1977)

O Sistema REVS recebe os requisitos especificados em RSL, verifica a consistência, ambigüidade e completeza, emite relatórios com os problemas encontrados e armazena os requisitos em um banco de dados para uso posterior. O sistema REVS é composto das seguintes ferramentas:

```
R-NET: RADAR-CONTROL
       STRUCTURE
          ALPHA AAA
          ALPHA BBB
          DO ALPHA CCC
             AND ALPHA POINT-RADAR
             AND ALPHA DDD
             AND DO (NUM-MIN) ALPHA EEA
                      OR (NUM = MAX) ALPHA EEB
                      OTHERWISE ALPHA EEC
                 END
                 ALPHA FFF
          END
          SUBENT GGS
          TERMINATE
          END.
  SUBNET: GGS
          STRUCTURE
          INPUT INTERFACE RADAR-BUF
          ALPHA HHH
          RETURN
          END.
```

Figura IV.16.3: "R-Net" da figura anterior na linguagem RLS
Fonte: (MENDES, 1989)

- **Tradutor da Linguagem RSL** - armazena a definição dos requisitos especificados em RSL no banco de dados ASSM, após a verificação da sintaxe e da semântica das declarações;

- **"Abstract System Semantic Model" (ASSM)** - é um banco de dados centralizado;

- Ferramentas para processar as informações contidas no banco de dados:

  . **Analisador Estático** - responsável por verificar a completeza, a consistência e a não ambigüidade dos requisitos;

  . **Consultor** - extrai informações da base de dados;

  . **Simulador** - gera e processa os algoritmos para simulação dos requisitos definidos;

  . **Pacotes Gráficos Interativos** - possibilitam a representação gráfica da estrutura dos requisitos.

SREM utiliza a seguinte seqüência de passos para o desenvolvimento e validação da especificação de requisitos:

- **Definição de requisitos** - neste passo são identificados todos os estímulos e as repostas associadas a eles, sendo elaborada uma "R-Net" para cada estímulo.

- **Tradução** - nesta etapa são inseridas, de forma interativa ou em "batch", todas "R-Nets" escritas em RSL no Sistema REVS. Os requisitos em RSL são traduzidos pelo tradutor do Sistema REVS, em um forma possível de ser armazenada no banco de dados ASSM.

- **Decomposição** - neste passo são definidos os requisitos de desempenho a serem alcançados e os pontos de validação. É feita uma avaliação dos requisitos com os analisadores do Sistema REVS.

- **Alocação** - nesta etapa gera-se, através do Simulador do Sistema REVS, um protótipo para simular as operações de processamento de mensagens definido pelos requisitos. É elaborado o documento da especificação de requisitos.

- **Demonstração da praticabilidade** - neste passo é construído um simulador analítico para demonstrar se os requisitos de processamentos críticos podem ser alcançados, e se a especificação gerada atende aos requisitos iniciais.

A mais recente implementação de SREM (DAVIS, 1990) foi escrita em Ada e pode ser executada em computadores VAX e em IBM PC/AT.

IV.17 "STRUCTURED ANALYSIS AND DESIGN TECHNIQUE" (ROSS, 1977), (ROSS, 1977A), (LEITE, 1983), (DAVIS, 1990)

"Structured Analysis and Design Technique" (SADT) é um método para a especificação de requisitos de sistemas. O modelo gerado por SADT representa os requisitos através de uma hierarquia de diagramas, que permitem a abstração, a projeção e o particionamento "top-down" do problema. Foi desenvolvido por Douglas Ross da SofTech Inc.

O modelo do problema é composto por uma hierarquia de diagramas representados através de uma linguagem gráfica cuja notação é composta por caixas e flexas. As caixas representam as partes de um todo e as flechas representam a interface e os relacionamentos entre essas partes.

O diagrama de mais alto nível, denominado de diagrama de contexto, apresenta a visão mais abstrata do problema. Esse problema é refinado em subproblemas, e o refinamento é documentado em outro diagrama, onde cada caixa representa uma parte do problema. Cada uma dessas caixas pode ser detalhada em um outro diagrama SADT a fim de documentar a decomposição do problema.

Cada diagrama de nível mais baixo está conectado a uma parte de mais alto nível do modelo, preservando, assim, o relacionamento lógico de cada componente com o sistema total. É esse detalhamento, apresentado de forma gradativa, que compõe a hierarquia de diagramas que formam o modelo do sistema.

A figura IV.17.1 mostra a representação da estrutura hierárquica do método SADT.

O modelo construído através do método SADT considera tanto as funções ou atividades a serem executadas pelo sistema, como os dados a serem manipulados por essas funções. Para representar graficamente os dados e as funções do sistema, o método utiliza os seguintes instrumentos notacionais:

- **Linguagem para Diagrama de Dados (DD)** - o DD representa o sistema em termos dos dados manipulados pelas atividades. A notação gráfica dessa linguagem é composta dos seguintes símbolos, que podem ser vistos na figura IV.17.2:

  . caixas, que representam os dados;

. flexas, que representam as atividades produtoras, atividades consumidoras, de controle e mecanismos.

As atividades produtoras geram os dados representados na caixa e as atividades consumidoras utilizam esses dados. As atividades de controle afetam os dados, modelam a sua estrutura e definem formas de acesso, enquanto que os mecanismos são meios de armazenamento dos dados.

- Linguagem para Diagrama de Atividades (DA) - o DA representa a decomposição do sistema em termos de suas funções. A notação gráfica dessa linguagem é composta pelos seguintes símbolos, que podem ser vistos na figura IV.17.3:

  . caixas, que representam as atividades;
  . flexas, que representam dados de entrada, dados de saída, dados de controle e mecanismos.

Os dados de entrada são utilizados pelas atividades, enquanto os dados de saída são gerados por essas atividades. Os dados de controle, por sua vez, indicam a forma como os dados de entrada são transformados em dados de saída, ou a causa dessa transformação. Já os mecanismos representam os dados que atuam na atividade para efetuar a transformação dos dados. Na figura IV.17.4 é apresentado um exemplo de um DA.

SADT também utiliza textos explicativos, gerados em linguagem natural, para especificar o sistema.

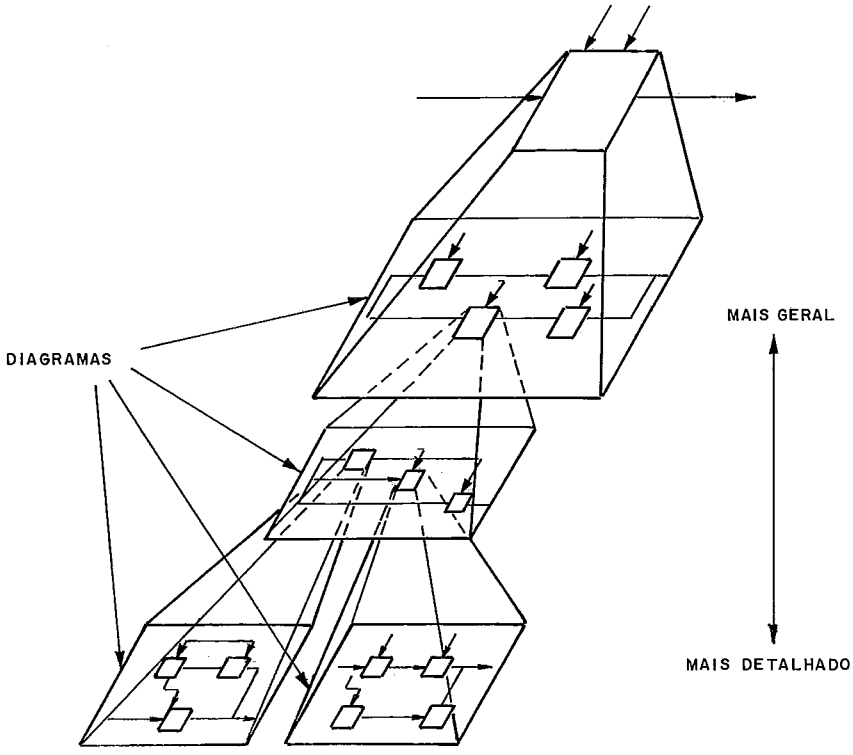


Figura IV.17.1 : Estrutura Hierárquica do Método SADT
Fonte: (ROSS, 1977)

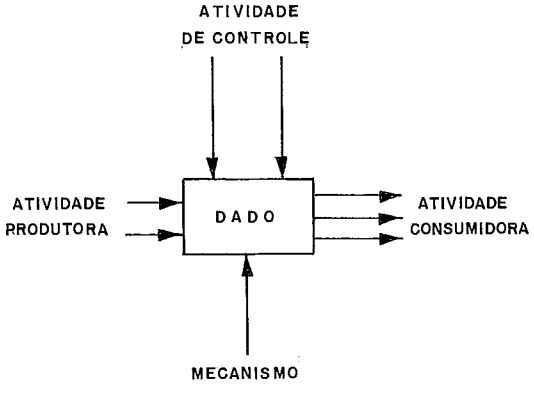


Figura IV.17.2 : Componentes do Diagrama de Dados
Fonte : (ROSS, 1977)

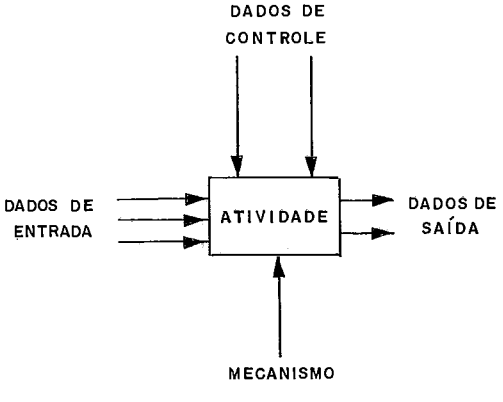


Figura IV.17.3 : Componentes do Diagrama de Atividades
Fonte: (ROSS, 1977)

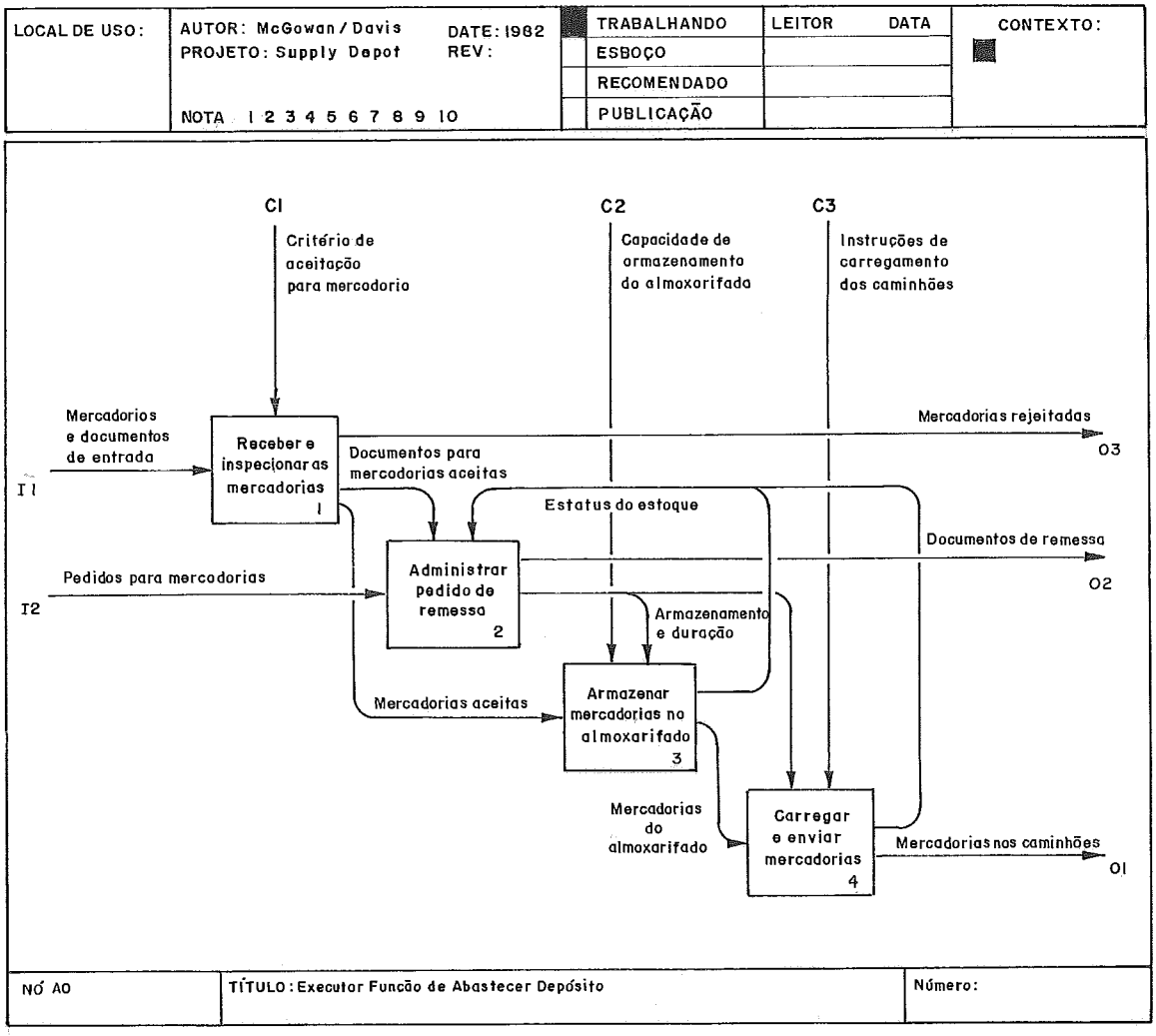


Figura IV.17.4 : Diagrama de Atividades do Método SADT
Fonte: (DAVIS, 1990)

O método oferece uma série de regras para a construção de diagramas, dentre as quais podem ser destacadas:

- cada diagrama, com exceção do primeiro, deve ter mais de duas caixas e menos de sete;
- cada caixa deve ter no mínimo uma entrada e uma saída;
- cada diagrama resultante da decomposição de uma caixa, deve ter a mesma estrutura ICOM ("Input, Control, Output, Mechanism") dessa caixa;
- cada flecha proveniente de um diagrama de nível superior deve ser numerada de acordo com a sua posição no diagrama pai;
- quando uma caixa for decomposta, ela deve ser referenciada no diagrama que compõe a sua decomposição.

Dentre as ferramentas que automatizam o método SADT, podem ser destacadas as seguintes (DAVIS, 1990) A.S.A., Design/IDEF e SPECIF-X. Essas ferramentas automatizam os instrumentos notacionais de SADT, além de manter um banco de dados de projeto, reforçar o papel de cada membro participante do projeto, prover uma documentação de alta qualidade e verificar a consistência, mesmo que de forma rudimentar, dos modelos gerados.

O método de Jackson (JACKSON, 1983) pode ser utilizado em conjunto com o método SADT como uma alternativa para modelagem da fase de projeto. O próprio método SADT pode

ser utilizado nessa fase, mas o seu uso é mais aplicado na fase de análise.

O método tem sido utilizado para o desenvolvimento de aplicações comerciais, manufaturas, treinamento militar, indústria bélica entre outras aplicações, inclusive em produtos que requeriam o processamento em tempo real. O método pode também ser usado para análise e projeto de hardware, bem como para problemas voltados para gerência de projetos.

IV.18 "STRUCTURED DESIGN"

(PAGE-JONES, 1980) (STEVENS, 1988)

"Structured Design" (SD) é um método para o projeto de sistemas, que constrói uma estrutura modular derivada a partir de uma especificação funcional. Permite que os programas de um sistema sejam especificados como um conjunto de módulos com funções únicas e independentes. Foi desenvolvido por Larry Constantine, Wayne Stevens e Glenford Myers e sofreu influência de vários pesquisadores, tais como Meilir Page-Jones e Edward Yourdon.

O método SD utiliza como instrumento notacional a linguagem para Gráfico de Estruturas (GE). O GE representa a decomposição de um sistema em módulos, mostrando a hierarquia, a organização e a comunicação entre os módulos identificados. A notação gráfica da linguagem para GE é composta pelos seguintes elementos, cujos símbolos podem ser vistos na figura IV.18.1:

- **módulo** - representado por um retângulo. Um módulo é definido como uma coleção de instruções de programa, para o qual devem ser identificados os seguintes atributos: informações de entrada e de saída, função que tranforma as entradas em saídas e dados internos para os quais só o módulo faz referências.

Um módulo pré-definido é um módulo que não precisa mais ser escrito porque já existe em algum sistema ou biblioteca de aplicação. Esse módulo é representado graficamente por um retângulo com linhas internas e paralelas às linhas verticais.

- **conexão entre módulos**, representada por uma seta mostrando a chamada de um módulo. A direção da seta mostra qual módulo chama qual.

- **comunicação entre módulos**, representado por uma pequena seta com um pequeno círculo em uma das extremidades. A comunicação entre módulos pode indicar:

  . uma ligação de dados - ocorre quando um módulo envia informação para outro. A direção da seta indica que módulo envia informação para outro;
  . um "flag" - significa que existe uma ligação de controle entre os módulos.

A decomposição de cada módulo de um DE é feita de forma "top-down", onde um módulo pode ser representado por outro diagrama com funções de nível mais baixo, mostrando um maior nível de detalhe. O nível mais alto de um DE

contém as principais funções do sistema com suas interfaces, dando uma visão geral do sistema a nível de projeto. A figura IV.18.2 mostra um exemplo de um GE.

Um dos princípios fundamentais do método SD é particionar um sistema de forma que os módulos sejam o mais independentes possíveis, e que cada módulo execute uma única função. Para alcançar esses objetivos, o método define um conjunto de critérios utilizados para avaliar a qualidade do projeto.

- **Acoplamento** - é o critério que avalia o grau de independência entre os módulos. O objetivo é minimizar o acoplamento, ou seja, tornar os módulos tão independentes quanto for possível.

Um acoplamento baixo entre módulos indica que o sistema está bem particionado e pode ser obtido através da eliminação das relações desnecessárias entre os módulos, reduzindo o número de relações necessárias e enfraquecendo a dependência entre as relações. Um dos pontos principais em se obter uma relação fraca entre os módulos é que nenhum módulo deve se preocupar com os detalhes da construção interna de qualquer outro módulo.

Existem vários tipos de acoplamento em dois módulos e eles podem estar acoplados de mais de uma maneira:

. **acoplamento de dados** - ocorre quando os módulos se comunicam por parâmetros;
. **acoplamento de imagem** - ocorre quando os módulos se referem à mesma estrutura de dados;

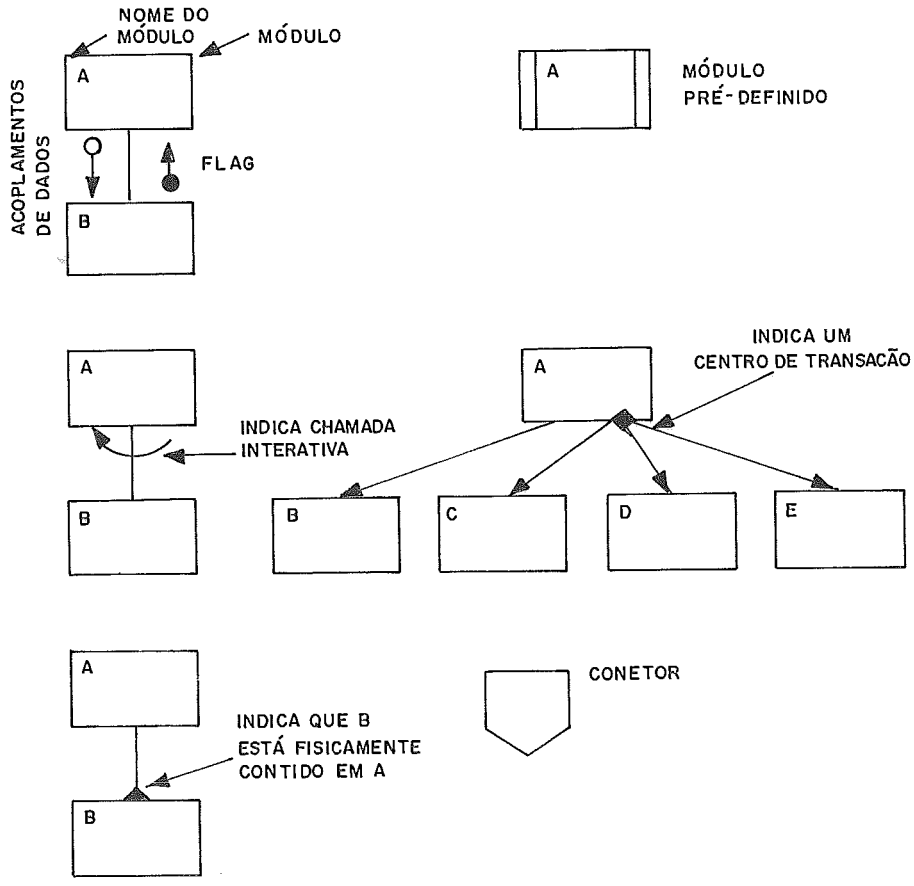


Figura IV.18.1 : Notação gráfica para representar Gráficos de Estrutura
Fonte: (ROCHA, 1987)

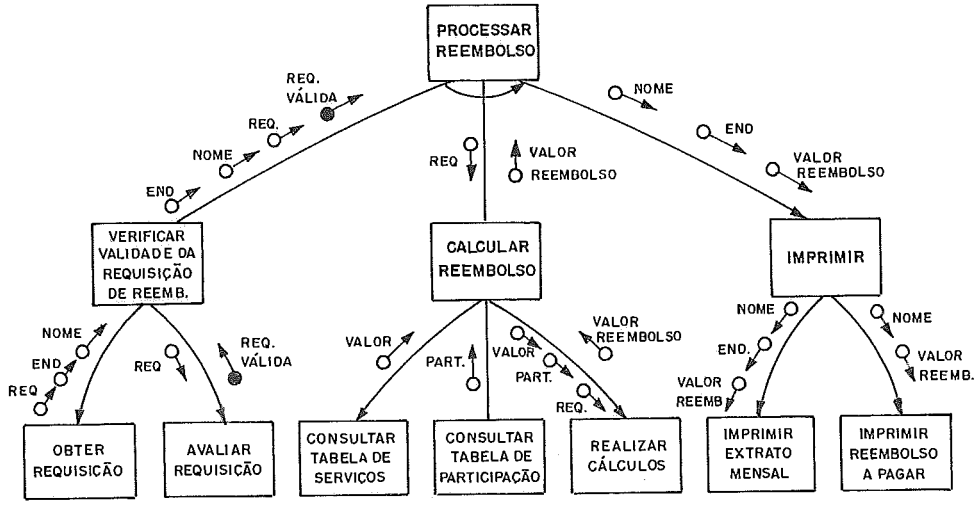


Figura IV.18.2 : Gráfico de Estrutura
Fonte: (ROCHA, 1987)

- **acoplamento de controle** — ocorre quando um módulo passa para o outro módulo um grupo de dados destinados a controlar a lógica interna do outro;
- **acoplamento comum** — ocorre quando os módulos se referem à mesma área de dados;
- **acoplamento de conteúdo** — ocorre quando um módulo faz referência ao interior de outro.

— **Coesão** — é o critério que avalia a medida da intensidade da associação funcional dos elementos de um módulo. O objetivo é maximizar a coesão, isto é, tornar as atividades de um módulo intimamente relacionadas uma com as outras. Existem vários tipos de coesão:

- **coesão funcional** — ocorre quando um módulo contém elementos que contribuem para a execução de uma e apenas uma tarefa relacionada ao problema;
- **coesão seqüencial** — ocorre quando os elementos de um módulo estão envolvidos em atividades de forma que os dados de saída de uma atividade servem como entrada para a próxima atividade;
- **coesão comunicacional** — ocorre quando os elementos de um módulo contribuem para a execução de atividades que utilizem a mesma entrada ou a mesma saída de dados;
- **coesão procedural** — ocorre quando os elementos de um módulo estão envolvidos em atividades diferentes e possivelmente não relacionadas, nas quais o controle flui de uma atividade para outra;

. **coesão temporal** – ocorre quando os elementos de um módulo estão envolvidos em atividades que estão relacionadas no tempo;

. **coesão lógica** – ocorre quando os elementos de um módulo contribuem para a execução de atividades selecionadas fora do módulo;

. **coesão coincidental** – ocorre quando os elementos de um módulo contribuem para a execução de atividades sem relação significativa entre si.

O método SD utiliza dois tipos de estratégias para a elaboração do projeto de um sistema:

– **Análise de transformação** – onde é elaborada uma estrutura composta por uma ramificação com as funções de entrada, uma ramificação de transformação das entradas em resultados e por uma ramificação de saída que trata de todas as saídas do resultado produzido.

– **Análise de transação** – onde é elaborada uma estrutura composta por uma série de transações a serem executadas conforme uma determinada condição, onde um módulo denominado de centro de transação determina o tipo de cada transação e a sua execução.

A construção de um DE a partir de um DFD pode ser elaborada através dos seguintes passos:

– Identificar as atividades manuais e as atividades automatizadas do DFD, desenhando os limites da parte automatizada;

- Retirar as partes manuais e acrescentar, nas atividades automatizadas, os dispositivos físicos de entrada e saída, necessários para passar os dados entre as regiões manual e automatizada. Esse passo resulta em um DFD físico para o sistema;
- Desenhar no novo DFD físico as fronteiras que identificam e separam os diferentes programas;
- Verificar se é estrutura de transformação ou transação;

Se a estrutura for de transformação:

- Começar com a forma mais abstrata de entrada e acompanhar o fluxo de dados até o ponto onde ele não pode mais ser considerado como uma entrada;
- Começar com a saída final e acompanhar o fluxo de dados até o ponto onde ele não pode mais ser considerado como uma saída;
- Desenhar o nível mais alto do GE;
- Acrescentar os módulos de nível mais baixo, detalhando o gráfico;

Se a estrutura for de transação:

- Identificar todos os fluxos que entram ou saem de um processo;
- Dividir o DFD, transação por transação;
- Para cada DFD obtido do partilhamento realizado no passo anterior:
  - Acrescentar os processos necessários para realizar a implementação do sistema;
  - Identificar a transformação central do DFD;
- Desenhar o nível mais alto do GE, e,

- Acrescentar os módulos de nível mais baixo detalhando o gráfico.

O método SD supõe a existência de uma especificação de requisitos contendo o detalhamento das funções a serem executadas pelo sistema. O método "Structured Systems Analysis" (SSA) (DEMARCO, 1978) e (GANE,1979) tem sido bastante utilizado em conjunto com o método SD. A partir do Diagrama de Fluxo de Dados do método SSA é derivado o GE, identificando-se estruturas de transformação e transação.

IV.19 "STRUCTURED SYSTEMS ANALYSIS" (DEMARCO, 1978), (GANE, 1979), (GANE, 1988), (YOURDON, 1990)

"Structured Systems Analysis" (SSA) é um método para especificação de requisitos de sistemas. A modelagem do sistema é feita através da representação dos fluxos de dados existente entre funções. Surgiu na década de 70 e foi divulgado pelos trabalhos de DEMARCO (1978) e GANE (1979).

Na proposta de Chris Gane e Trish Sarson (GANE, 1979), o método SSA propõe os seguintes instrumentos:

- **Linguagem para Diagrama de Fluxo de Dados (DFD)** - o DFD representa as funções do sistema a partir dos fluxos de dados existentes entre elas. O DFD mostra as origens dos dados, para onde vão quando saem do sistema, onde são armazenados, que processos os transformam e quais os dados que entram e saem de cada uma das funções. A

notação gráfica da linguagem para DFD é composta dos seguintes símbolos:

- quadrado duplo - representa uma entidade externa, que são coisas ou pessoas que indicam uma fonte ou destino de dados, para dentro e para fora do sistema, estando, por definição, fora do sistema;
- seta - representa um fluxo de dados e indica a sua direção;
- retângulo arredondado - representa um processo ou uma função que transforma os dados;
- retângulo aberto - representa um depósito de dados, que indica um lugar no sistema onde os dados são armazenados.

A figura IV.19.1 mostra o exemplo de um DFD.

Gane e Sarson recomendam começar o desenvolvimento de um DFD pela elaboração de uma lista contendo as entradas e saídas das entidades externas. Um DFD é construído em níveis e a decomposição do DFD é feita de forma "top-down", por meio do refinamento das funções do sistema em um nível de abstração mais detalhado.

Cada processo de um DFD deve ser expandido de modo a tornar-se um novo DFD, buscando, assim, a especificação de funções mais simples. Todos os processos pertencentes a um nível de abstração inferior devem se relacionar a um processo de nível de abstração superior.

Todos fluxos de dados, entidades externas, depósitos de dados e processos de um determinado nível do DFD devem

ser representados nos DFDs que formam a sua decomposição. A expansão dos processos deve ser efetuada quantas vezes for necessária, até que seja atingido o nível de detalhe desejado.

- **Dicionário de Dados** (DD) - é o local onde são armazenadas informações e definições sobre elementos de dados, estruturas de dados, fluxos de dados, depósitos de dados, processos, entidades externas e entradas no glossário.

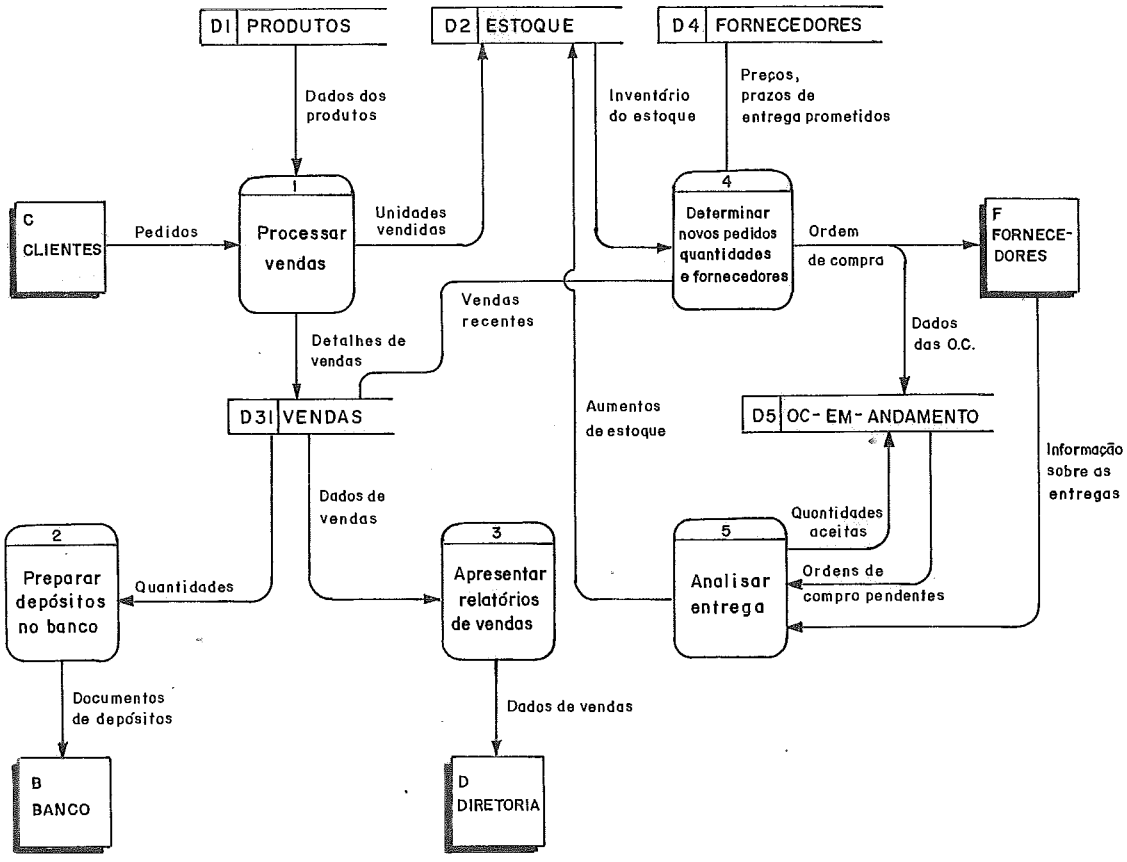


Figura IV.19.1 : Diagrama de Fluxo de Dados segundo Gane e Sarson
Fonte: (GANE, 1990)

- **Linguagens para descrição da lógica dos processos** - os processos podem ser especificados através da linguagem para Árvore de Decisão, linguagem para Tabelas de Decisão, Português Estruturado, Português Compacto e Pseudocódigo.

- **Linguagem para Diagrama de Acesso Imediato (DAID)** - é um instrumento para descrever a organização lógica dos dados, indicando como localizar um determinado dado.

Segundo (GANE, 1979), a especificação de um sistema deve ser elaborada através dos seguintes passos:

- **Realizar um estudo inicial do problema** para verificar as dimensões do problema atual e como ele pode ser solucionado;

- **Realizar um estudo detalhado do problema**, definindo a comunidade de usuários para o novo sistema e construindo o **modelo lógico do sistema existente**, contendo o DFD, a definição dos dados e a especificação da lógica para os processos mais importantes;

- **Derivar os objetivos do novo sistema** a partir das limitações do sistema existente;

- **Desenvolver o modelo lógico do novo sistema**, elaborando o DFD, o DD, a lógica para cada processo primitivo do DFD, o conteúdo e normalização dos depósitos de dados e o DAID, e,

- **Avaliar o modelo lógico** construído para o novo sistema juntamente com o usuário.

Na proposta de Tom DeMarco (DEMARCO, 1978), o método SSA usa praticamente os mesmos instrumentos propostos por (GANE, 1979), com algumas diferenças de nomenclatura e simbologia. São utilizados os instrumentos:

- **Linguagem para Diagramas de Fluxo de Dados (DFD)** - a notação da linguagem para DFD usa os seguintes símbolos, que são ligeiramente diferentes dos utilizados por GANE (1979):

  . seta - representa um fluxo de dados;

  . bolha - representa um processo;

  . linhas paralelas - representam um arquivo ou banco de dados;

  . quadrado simples - representa uma fonte ou destino de dados.

A abordagem de DEMARCO (1978) sugere que a construção do DFD comece com um Diagrama de Contexto, que mostra o sistema como um único processo, as entidades externas que se relacionam com o sistema e todas as entradas e saídas. A figura IV.19.2 mostra um exemplo do DFD na notação de DeMarco.

- **Dicionário de Dados** - contém informações e definições de fluxos de dados, elementos de dados, arquivos ou banco de dados e processos.

- **Linguagem para descrição das transformações ou linguagem para miniespecificações** - estas linguagens podem ser Português Estruturado, linguagem para Tabela de Decisão e linguagem para Árvore de Decisão.

- **Linguagem para Diagrama de Estrutura de Dados** - contém a decomposição da estrutura de dados normalizada.

Nesta abordagem a especificação de um sistema é realizada através das seguintes etapas:

- **Estudar o ambiente físico corrente**, que pode ser manual ou automatizado, gerando o DFD Físico Corrente;

- **Derivar o equivalente lógico do ambiente corrente**, retirando-se os itens físicos e substituindo-os pelos seus equivalentes lógicos. Como resultado é gerado o DFD Lógico Corrente;

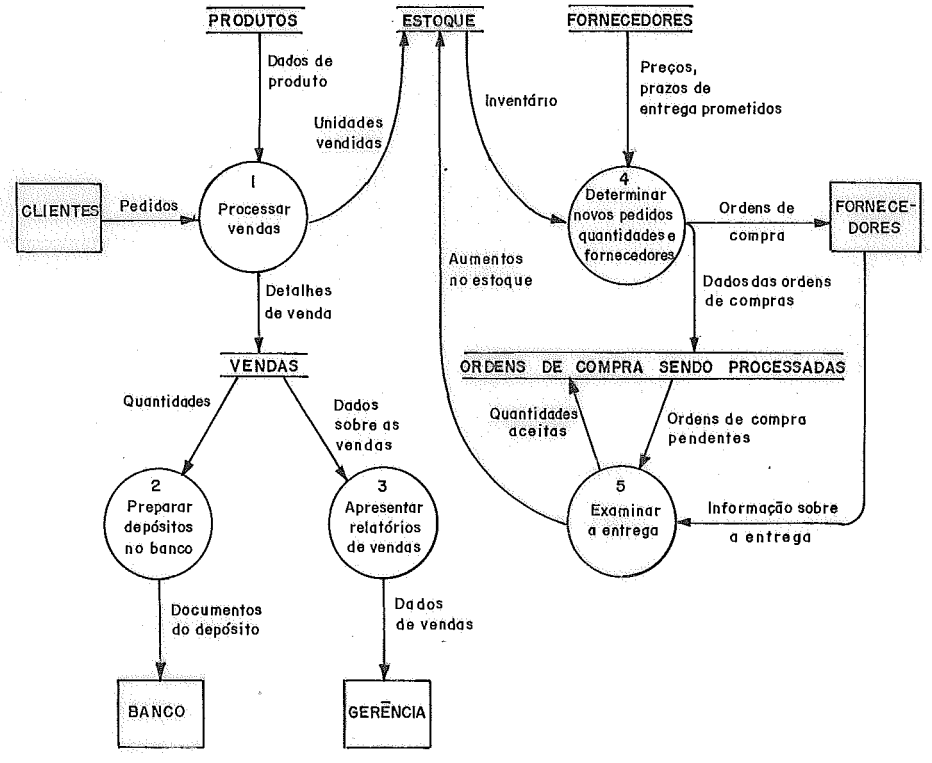


Figura IV.19.2: Diagrama de Fluxo de Dados segundo DeMarco
Fonte: (GANE, 1990)

- **Construir o novo modelo lógico** do sistema;

- **Definir diferentes alternativas de automação** do sistema e documentar cada uma dessas alternativas em um Novo DFD Físico, mostrando claramente as partes que serão automatizadas e as que permanacerão manuais;

- **Analisar**, juntamente com o usuário, **as alternativas de automação** com relação aos custos e benefícios;

- **Selecionar para implementação um dos Novos DFDs Físicos**;

- **Empacotar o Novo DFD Físico** selecionado, juntamente com o Dicionário de Dados e a Descrição das Transformações na Especificação Estruturada.

Extensões do método SSA têm sido propostas, permitindo a sua utilização em conjunto com métodos para a modelagem de dados. GANE (1988) sugere a construção de um Diagrama de Entidade-Relacionamento (DER), após a elaboração do DFD. Assim o DER é aplicado em conjunto com os instrumentos originais do método para modelar os requisitos do sistema.

Na proposta de (YOURDON, 1990) ("Modern Structured Analysis"), além de se utilizar o DER, também foram incluídas várias modificações para tratar de problemas que envolvessem aspectos de modelagem de tempo e controle, através do uso de Diagramas de Transição de Estado (DTS).

Nesta abordagem de YOURDON (1990), a especificação do sistema inicia-se com a construção do modelo essencial do novo sistema, indicando o que o sistema deve fazer para satisfazer os requisitos do usuário. O modelo essencial é

composto de dois modelos: o modelo ambiental e o modelo comportamental.

O modelo ambiental define a fronteira entre o sistema e o ambiente onde reside. Ele é formado por um diagrama de contexto, uma lista de eventos e uma pequena descrição do próprio sistema. O modelo comportamental, por sua vez, descreve o comportamento do sistema necessário para interagir com sucesso com o seu ambiente. Ele é composto de DFDs, DERs, DTSs, itens do dicionário de dados e especificações de processos.

O uso de terminais gráficos e de estações de trabalho incentivou a contrução de ferramentas gráficas para a automatização do método SSA. As ferramentas não automatizam o processo de trabalho, mas dão aos analistas a opção de não mais manipular DFDs e Dicionários de Dados de forma manual.

Dentre as ferramentas disponíveis no mercado internacional podem ser citadas as seguintes (DAVIS, 1990), (FISHER, 1990) e (GANE, 1990): **Analyst/Designer Toolkit** da Yourdon, **Anatool** da Advanced Logical Software, **Deft** da DEFT INC., **DesignAid** da Nastec Corporation, **DESIGN/1** da Arthur Andersen & Co., **Design/2.0** e **Design/OA** da Meta Softwar Co., **ENVISION** da Enriched Data Systems, **EXCELERATOR** e **EXCELERATOR/RST** da Index Technology Corp., **Information Engineering Workbench** da Knowledge Ware Inc., **MacBubbles** da StarSys Inc., **MAESTRO** da Softlab Inc., **MULTI/CAM** da AGS Management Systems, **ProKit*WORKBENCH** da McDonnell Douglas Information Systems Gruop, **ProMod/SA** e **ProMod/RT** da ProMod

Inc., **Software Through Pictures** da Interactive Development Environments Inc., **Structured Architect** da Meta Systems Inc., **TEAMWORK** da CADRE Technologies Inc., **THE DEVELOPER** da ASYST Technologies Inc., **Visible Analyst Workbench** da Visible Systems Corp., **VS Designer** da Visual Software Inc.

Como exemplo de ferramentas disponíveis no mercado nacional são citadas: **MOSAICO** da IESA-TS, **PC-DFD** do IBPI, **TALISMAN** da Staa Informática. Em projetos de pesquisa na COPPE/UFRJ foram desenvolvidas as seguintes ferramentas: **EDIT-DFD** (AGUIAR, 1986), **DDADOS** (BLASCHEK, 1987) e **FEGRES** (TRAVASSOS, 1990).

IV.20 "SYSTEM FOR APPLICATION-ORIENTED REQUIREMENTS SPECIFICATION" (HAGEMANN, 1987)

"System for Application-Oriented Requirements Specification" (SARS) é um método para especificação de requisitos de Sistemas Embutidos, em especial de Sistemas de Controle de Processos (SCP). Foi desenvolvido por M. Hagemann do "Institute for Real-Time Computer Control Systems and Robotics", da Universidade de Karlsruhe, República Federal da Alemanha.

A idéia de SARS é permitir que a descrição dos requisitos seja feita da forma mais natural possível para o usuário, ou seja, de uma forma na qual o engenheiro esteja habituado a descrever um SCP. O método permite a construção automática de um protótipo do sistema, a partir das especificações geradas.

SARS é um sistema automatizado composto por vários elementos, cuja estrutura pode ser vista na figura IV.20.1:

- **Banco de Dados** - é a parte central do Sistema SARS e onde são armazenados todos os dados gerados durante a especificação dos requisitos;

- **Sistema de Recuperação de Informações** (ou Matriz de Acesso) - responsável pelo acesso às informações do Banco de Dados;

- **Editor gráfico e textual** - suporta a criação e manipulação das representações dos requisitos;

- **Analisador da linguagem LARS** - avalia os requisitos definidos em LARS;

- **Gerador de relatórios** - produz relatórios a partir das informações armazenadas no banco de dados;

- **Gerador de código** - permite a geração automática de um protótipo da especificação a partir dos requisitos elaborados na linguagem LARS.

A especificação de requisitos é construída através de instrumentos e ferramentas que permitem a representação do sistema através de gráficos e tabelas, textos formais e informais. A principal linguagem formal de SARS é descrita a seguir:

- **"Language for Application-oriented Requirements Specification" (LARS)** - a estrutura da linguagem LARS

baseia-se no modelo de estímulo-resposta. LARS é composta por três elementos (interface, rede e guarda) que correspondem aos componentes de um SCP. A figura IV.20.2 mostra a estrutura de LARS, cujos elementos são definidos a seguir:

- **Interface** - é responsável pela especificação dos dados e mensagens recebidos (interface de entrada) ou enviados (interface de saída) para o ambiente, cujo exemplo pode ser visto na figura IV.20.3. As informações são descritas em três classes:

  - **evento** - descreve todas as mensagens ou impulsos, que podem ocorrer de forma espontânea ou cíclica;
  - **estado** - contém a especificação dos estados que o ambiente pode assumir;
  - **dado** - contém a definição dos dados.

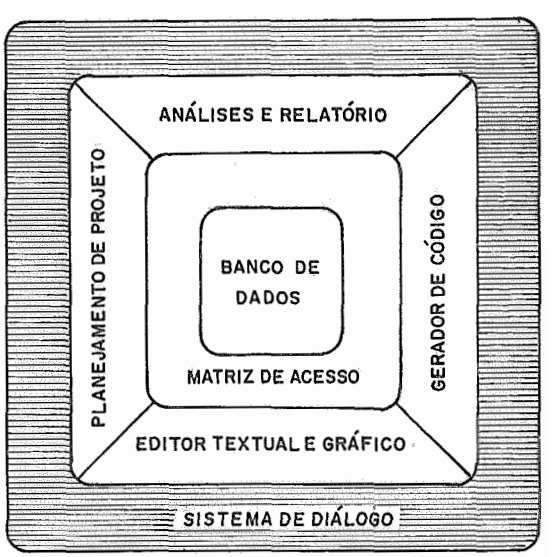


Figura IV.20.1 : Componentes de SARS
Fonte: (HAGEMANN, 1987)

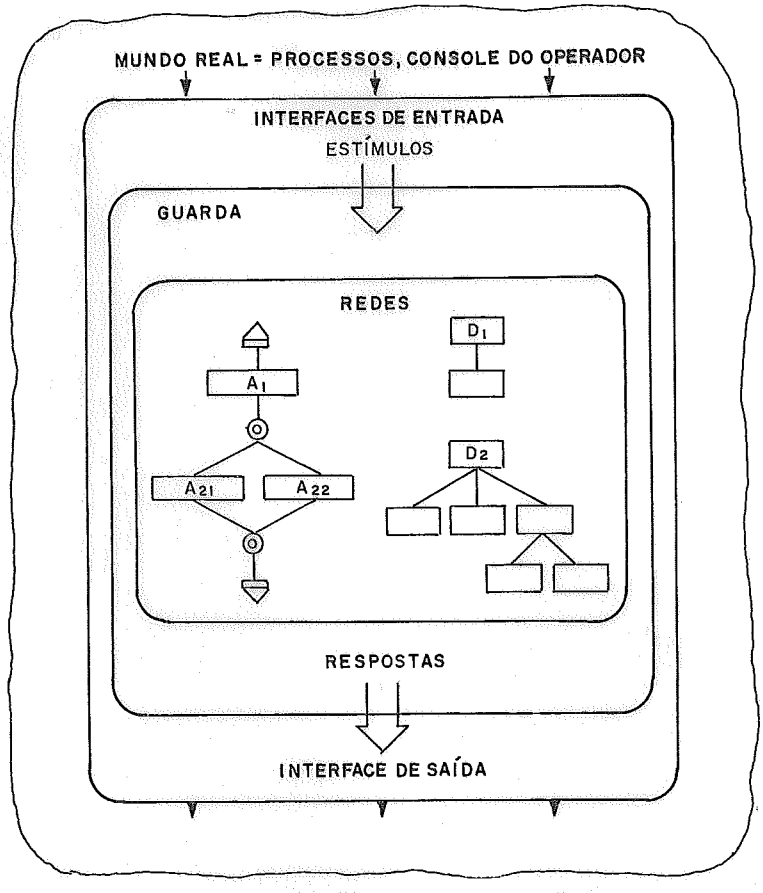


Figura IV.20.2 : Estrutura de LARS
Fonte: (HAGEMANN, 1987)

```
INPUT-INTERFACE pesagem: PROCESS;

    DESCRIPTION
       calcular o peso de um recipiente vazio.
    END DESCRIPTION;

    EVENT chegada: SPONTANEOUS;
    END EVENT;

    DATA peso: NUMERIC (0..999.9);
    END DATA;

    STATUS
       checar-soma: ENUMERATION (válido, não-válido)
    END STATUS;

END INPUT_INTERFACE;
```

Figura IV.20.3: Exemplo de Interface Definida em LARS

Fonte: (HAGEMANN, 1987)

. **Rede** ("net") - é responsável pela especificação das funções e subfunções do sistema. Define a estrutura de controle das funções, que determina a seqüência de subfunções a serem executadas. Uma subfunção elementar é representada por uma ação, que é um módulo do sistema. A figura IV.20.4 mostra um exemplo da especificação de uma rede.

```
NET rede-peso;

   STRUCTURE
      ACTION rede-contagem-peso   INPUT  recipiente
                                  OUTPUT rede;
      PARALLEL
         SIGNAL sinal-peso;
      AND
         ACTION atualiza-estatistica
                     INPUT  rede, dados-estatistica
                     OUTPUT desvios, estatistica,
                            dados-estatistica;
      END PARALLEL;
      EXIT;
   END STRUCTURED;

   COMMOM peso, seletor, dados-estatistica;
   END COMMON;

   PRIVATE
      desvio:      ENUMERATION (ok, medio, ruim);
      estatistica: ENUMERATION (aceita, nao-aceita);
      rede:        NUMERIC (0..9999,9);
   END PRIVATE;

END NET;
```

Figura IV.20.4: Exemplo de Rede Definido em LARS.

Fonte: (HAGEMANN, 1987)

As subredes ("subnets") são utilizadas para elaborar uma estrutura hierárquica das funções. As subredes possuem a mesma estrutura das redes, exceto o fato de que não são controladas pelo componente guarda, sendo ativadas através

de chamadas das redes. As dependências no fluxo de controle também são determinadas. Elas indicam se as funções são executadas de forma paralela ou independente, se são seletivas ou executadas em um "loop".

. **Guarda** ("guard") - é responsável pelo controle do fluxo de dados entre o sistema e o ambiente. O guarda contém uma lista das atividades a serem executadas pelo SCP, a definição dos dados usados por essas atividades e a descrição necessária para a sincronização das redes. Também controla o uso correto dos dados.

Um exemplo do elemento guarda pode ser visto na figura IV.20.5.

```
GUARD encher-recipiente:

   ACTIVATION
      STIMULI-DEFINITION
         peso := chegada OF pesagem
                 AND
                 parar OF operador-sistema = OFF;
      END STIMULI-DEFINITION;
   END ACTIVATION;

   RECEIVE
      peso FROM peso OF pesagem;
   END RECEIVE;

   COMMON
      DATA
         peso: NUMERIC;
      END DATA;
   END COMMON;

END GUARD;
```

Figura IV.20.5: Exemplo de Guarda Definido em LARS.

Fonte: (HAGEMANN, 1987)

Informações explicativas e complementares sobre a definição do sistema, que não podem ser expressas de maneira formal, podem ser armazenadas no Banco de Dados do Sistema SARS, e podem ser acessadas pelo Sistema de Recuperação de Informações.

O método SARS baseia-se no fato de que um SCP tem geralmente que controlar os processos técnicos, supervisionar suas próprias funções e gerenciar o fluxo de dados existente entre essas funções e os processos técnicos.

A especificação de requisitos deve descrever as conexões existentes entre o SCP e o seu ambiente e deve conter, especialmente, as funções necessárias para supervisionar e controlar os processos, bem como as conexões e dependências entre as funções. Segundo o método SARS, a especificação de requisitos é elaborada através dos seguintes passos:

- **Identificação das funções do sistema** - neste passo é feita a análise detalhada do ambiente do sistema. O processo técnico é decomposto em subprocessos e são identificadas e definidas a maior parte das funções, bem como as conexões e dependências entre funções e subprocessos.

- **Definição da interface do sistema** - neste passo é feita a descrição detalhada de todos os sinais e dados do ambiente. Para cada função é elaborada uma descrição dos objetos a serem manipulados, das condições sob as quais

a função deve ser executada e dos objetos a serem supervisionados pela função. A elaboração das redes deve ser realizada paralelamente a esse passo.

- **Definição do fluxo de dados controlado pelo SCP** - neste passo alguns dos dados de processos técnicos devem ser estruturados em uma forma utilizável pelas funções. Além disso, é feita a definição de como os dados globais devem ser construídos a partir dos dados externos, além da descrição do uso dos dados pelas funções.

Esses passos devem ser executados iterativamente, refinando-se a especificação do sistema, até se obter um documento final contendo todos os requisitos do usuário.

O Sistema SARS permite, após a especificação textual e gráfica dos requisitos, a construção de um protótipo do sistema. Esse protótipo, entretanto, não é o produto final. Ele é utilizado para testar os requisitos e detectar a existência de erros na especificação gerada.

IV.21 "TECHNOLOGY FOR THE AUTOMATED GENERATION OF SYSTEMS" (SIEVERT, 1985)

"Technology for the Automated Generation of Systems" (TAGS) é um método para o desenvolvimento de sistemas de tempo real. O método possibilita a validação do produto através da construção de um protótipo, gerado automaticamente na linguagem Ada. Foi desenvolvido pela "Teledyne Brown Engineering".

O método utiliza o seguinte instrumento:

- **"Input Output Requirements Language" (IORL)** - é uma linguagem para especificar requisitos de sistemas que representa o fluxo de dados, o fluxo de controle, a definição dos dados e a decomposição e hierarquia entre os componentes do sistema. A notação gráfica de IORL é expressa por meio de diagramas e tabelas que são definidos a seguir:

  . **Diagrama de Bloco Esquemático (DBE)** - é o diagrama de mais alto nível representado pela linguagem IORL. O DBE mostra uma visão geral do sistema, identificando os seus principais componentes independentes e os fluxos de dados existentes entre eles. A notação gráfica para representação de um DBE, cujo exemplo pode ser visto na figura IV.21.1, é composta de:

  . retângulos, que indicam os componentes do sistema;
  . setas, que indicam o fluxo de dados entre esses componentes.

Os componentes de um DBE são decompostos de forma "top-down" em DBEs de nível mais baixo. Esse processo de decomposição é realizado até que os componentes de cada novo DBE não possam mais ser subdivididos em unidades independentes, ou seja, em componentes cuja execução não depende da execução prévia de outros componentes.

Cada DBE representa um documento diferente cuja identificação deve aparecer nos diagramas e tabelas que compõem a sua decomposição. Deve haver um balanceamento

entre os fluxos de dados externos de um determinado DBE, com os fluxos que entram e saem do diagrama que o originou.

- **Diagrama de Tempo e Relacionamentos de Entrada / Saída (DTRES)** - representa o fluxo de controle dos componentes de um DBE. Para cada componente de um DBE é construído um DTRES específico. A linguagem IORL oferece recursos para suportar o processamento paralelo. Para isso são utilizados construtores especiais representados em um DTRES ou um DPP por um nó "FAN-OUT AND". A figura IV.21.2 mostra um exemplo da notação utilizada para construir um DTRES.

- **Diagrama de Processo Pré-definido (DPP)** - é usado para descrever detalhadamente o fluxo lógico de cada processo referenciado em um DTRES ou em um DPP de nível mais alto. Os DPPs e os DTRESs possuem uma estrutura similar. Entretanto, os DPPs permitem a identificação de componentes dependentes, isto é, de componentes que dependem da execução de outros para serem ativados. A lógica de cada DPP é mantida em bibliotecas armazenadas em discos para possíveis reutilizações. Na figura IV.21.3 podemos ver um exemplo da notação utilizada para construir um DPP.

Os fluxos de dados são representados em um DTRES ou em um DPP através de atribuições feitas a variáveis e por símbolos de entrada/saída que controlam o tempo dos fluxos de dados entre os componentes. Os dados são definidos em IORL por um conjunto de tabelas descritas a seguir:

. **Tabela de Parâmetros de Entrada/Saída (TPES)** - define os dados que fluem entre os componentes de um DBE, isto é, define os dados associados a uma determinada interface no DBE, como pode ser visto na figura IV.21.4.

. **Tabela de Parâmetros Internos (TPI)** - contém a definição dos dados em vários níveis de detalhes. Os dados em sua forma mais geral são definidos em uma TPI especial, a TPI-0. Uma TPI também estrutura os dados definidos para um componente individual e os dados limitados para DPP individual, e define as variáveis internas relacionadas a um DTRES e aos seus DPPs associados.

. **Tabela de Parâmetros de Processo Pré-definido (TPP)** - define os parâmetros locais a um determinado DPP.

O método TAGS é automatizado através do seguinte conjunto de ferramentas:

- **Pacote de Armazenamento e Recuperação** - gera, armazena, recupera e atualiza os diagramas e tabelas da linguagem IORL.

- **Analisador de Diagnósticos** - é responsável pela análise estática dos diagramas gerados em IORL.

- **Compilador de Simulação** - produz código fonte na linguagem Ada e simula o funcionamento do sistema, detectando a existência de erros dinâmicos e gerando os dados de saída.

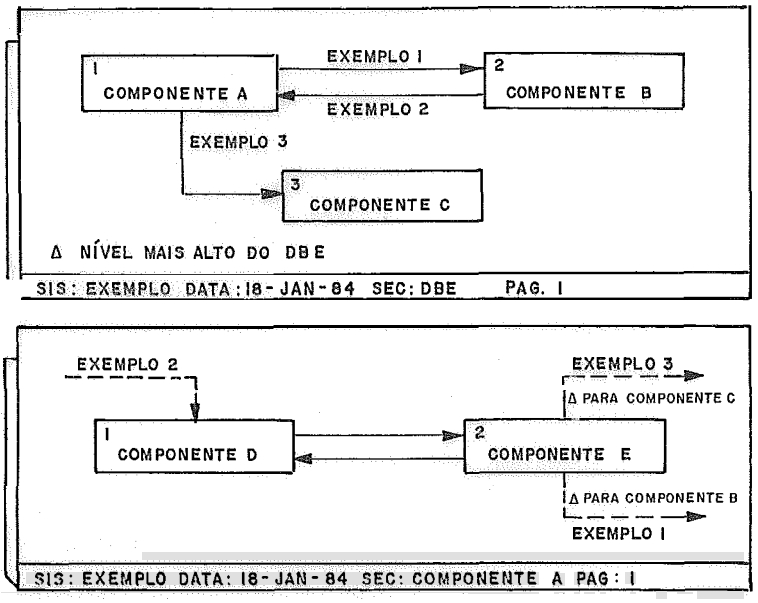


FIGURA IV.21.1 : DIAGRAMA DE BLOCO ESQUEMÁTICO (DBE)
FONTE: (SIEVET, 1985)

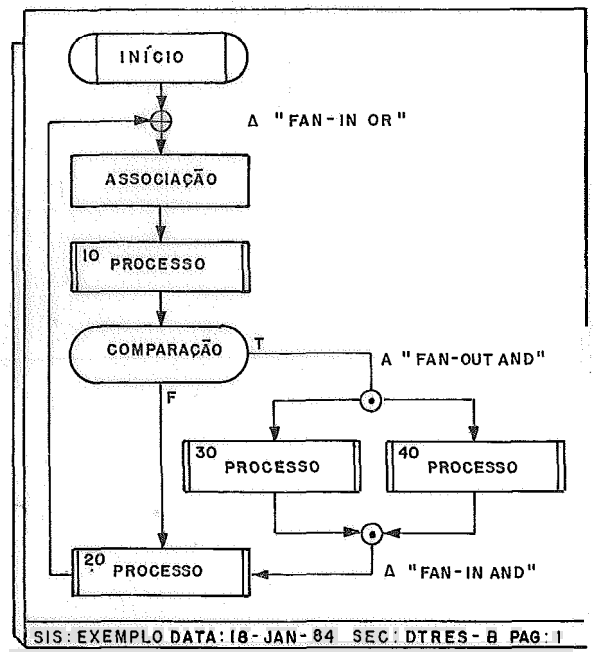


FIGURA IV.21.2 : DTRES ASSOCIADO AO COMPONENTE B DO DBE
FONTE: (SIEVET, 1985)

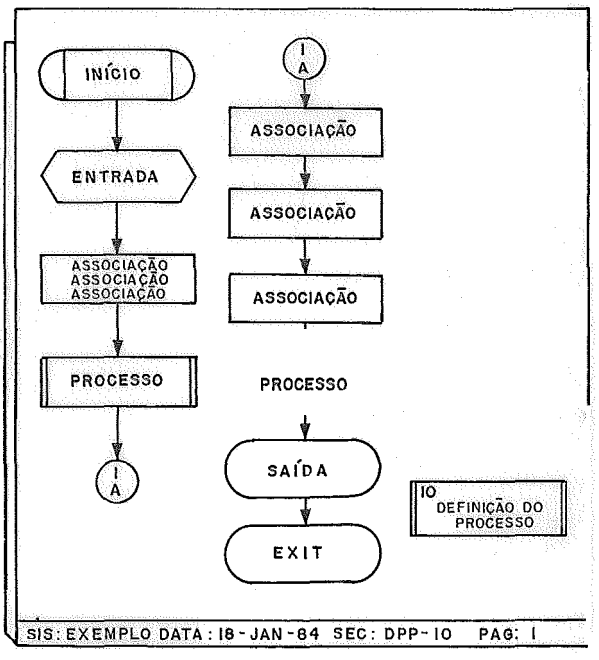


FIGURA IV.21.3 : DPP ASSOCIADO AO PROCESSO 10 DO DTRES
FONTE: (SIEVET, 1985)

GRUPO	DESCRIÇÃO DE PARÂMETROS	NOME	ESCOPO	UNIDADE
6	(GRUPO DE DADOS)	TEMPO	[0-.....60]	SEGUNDOS
		MÊS	[1,2-.....-12]	JAN - DEZ
7	(GRUPO DE DADOS)	MOEDA	[0 ..... .$]	

SIS: EXEMPLO DATA: 18-JAN-84 SEC: TPES PAG: 1

FIGURA IV.21.4 : TABELA DE PARAMÊTROS DE ENTRADA/SAÍDA (TPES)
FONTE: (SIEVET, 1985)

- **Pacote para Gerência de Configuração** - é responsável pelo controle das versões do sistema.

O processo de desenvolvimento, segundo o método TAGS, consiste de três processos interrelacionados: **avaliação, especificação** e **implementação**. O desenvolvimento inicia-se com a identificação dos requisitos do usuário. Esses requisitos são avaliados, juntamente com o usuário, até que o conjunto inicial de requisitos esteja suficientemente entendido para permitir a especificação.

O processo de especificação é então executado e a especificação de requisitos construída é revista e avaliada. A avaliação determina se a especificação será modificada ou implementada. Após implementado, o sistema é avaliado e, como resultado, tem-se a decisão de alterar a especificação ou declarar o sistema operacional.

Os seguintes passos são realizados durante o processo de especificação:

- **Construção do modelo conceitual** - onde é elaborado o DBE de mais alto nível, representando o sistema como um componente único e as principais interfaces com o ambiente. Deve-se definir claramente os limites do sistema para que se possa determinar os requisitos de interface e as características de entrada e saída do sistema.

- **Construção dos DTRES correspondente e dos DPPs de alto nível** - durante este passo são elaborados os diagramas DTRES e DPPs relativos ao SBD construído na etapa

anterior. Os DPPs estão relacionados entre si através do fluxo de controle e podem ser decompostos em DPPs de nível mais baixo.

- **Decomposição dos componentes dos DBE em subcomponentes** - caso seja necessário decompor os componentes do DBE, deve-se executar novamente o passo anterior. Nesse caso, o passo anterior deve ser reiniciado com a alocação dos DPPs, definidos no nível imediatamente superior, aos subcomponentes correspondentes.

- **Definição de todos os DPPs ainda não especificados** - isso é feito através da construção de algoritmos para cada DPP, de forma a permitir a sua validação. Todos os DPPs devem ser armazenados em uma biblioteca para garantir a reutilização dos DPPs existentes.

Uma seqüência de fases de especificações seguidas, cada uma delas, por uma fase de validação de protótipo é realizada em TAGS durante o processo de desenvolvimento. Após a sua especificação, o protótipo é traduzido da linguagem IORL para uma linguagem de implementação, que pode ser a linguagem Ada, e é testado para determinar se a tradução está correta. A linguagem IORL também foi utilizada para construir a própria estação de trabalho do método TAGS.

IV.22 "TWO-DIMENSIONAL PROGRAM DESIGN" (ROTENSTREICH, 1986)

"Two-Dimensional Program Design" (TDPD) é um método para a especificação de projeto de sistemas, através da construção de modelos dos relacionamentos entre os módulos. Permite a decomposição funcional do sistema e o compartilhamento de dados entre ações, característico da decomposição de fluxo de dados. Foi proposto por Shmuel Rotenstreich e William Howden.

O método está baseado no conceito de dimensão, que é definido pela direção dos fluxos de dados entre os módulos do sistema. O projeto do software é modelado, de forma simultânea, a partir das dimensões verticais e horizontais. Os conceitos básicos utilizados por TDPD são definidos a seguir:

- **Dimensão vertical** - é utilizada para definir os relacionamentos entre módulos de diferentes níveis de abstração. Corresponde, em uma implementação procedimental, ao relacionamento entre módulos em uma chamada de subrotinas. Está relacionada à decomposição funcional do sistema.

- **Dimensão horizontal** - é utilizada para definir os relacionamentos entre módulos pertencentes a um mesmo nível de abstração ou entre módulos que não estão relacionados diretamente por fluxos de controle.

Um modelo bidimensional pode ser visto como um conjunto de estruturas compostas com níveis de abstração "top-down" e fluxos de dados verticais e horizontais.

O método utiliza os seguintes tipos de módulos:

- **módulo funcional** - transforma os dados de entrada em dados de saída e é similar a um módulo no método "Structured Design" (YOURDON, 1978);

- **módulo ambiental** - corresponde aos fluxos de dados horizontais e pode ser de vários tipos, tais como:

  . fluxo seqüencial - corresponde a um fluxo de dados horizontal, que passa de um módulo funcional para outro em um mesmo nível de abstração;

  . fluxo paralelo - representa uma estrutura de fluxo de dados horizontal, onde vários módulos preparam os dados a serem utilizados de forma simultânea por um único módulo;

  . tabela - corresponde a um ambiente no qual módulos funcionais podem armazenar ou recuperar dados;

  . canal - representa um conjunto de objetos de dados que podem ser lidos seqüencialmente;

Os fluxos de dados utilizados por TDPD podem ser:

- **fluxo de dados vertical** - são fluxos de dados que podem fluir verticalmente, para cima ou para baixo, entre um módulo, em um determinado nível de

abstração, e seus subordinados, em um nível de abstração mais baixo.

Os fluxos de dados que fluem para baixo tornam-se menos abstratos e mais específicos. Em alguns casos esses fluxos correspondem à decomposição de dados compostos. Em outros, ocorre a discriminação de diferentes tipos de dados.

Os dados que fluem para cima, por sua vez, tornam-se mais gerais, podendo corresponder à recomposição de dados ou à geração de dados abstratos, relacionados a vários tipos alternativos de dados, em um nível de abstração mais baixo.

- **fluxos de dados horizontais** - são fluxos de dados que fluem através do sistema, de um módulo para outro, no mesmo nível de abstração. São representados através do uso de ambientes. Um módulo funcional que prepara dados para serem utilizados por outros módulos no mesmo nível de abstração coloca dados em um módulo ambiental, do qual outro módulo funcional pode recuperá-lo.

A direção vertical e horizontal de um fluxo de dados pode ser ilustrada através das figuras IV.22.1 e IV.22.2. Nesses exemplos o módulo A é difinido em termos dos módulos B e C. Na figura IV.22.1 os dados fluem entre os módulos A e B, e entre A e C. Os dados a serem compartilhados por B e C devem passar através de A. Esse fluxo de dados é dito ser vertical. Na figura IV.22.2 temos um exemplo de fluxo de

dados horizontal, onde os dados fluem diretamente entre B e C. O módulo A se comunica diretamente com os módulos B e C, mas desconhece detalhes de alguns dados compartilhados por B e C.

O método TDPD possui como instrumento a linguagem para Diagrama de Estruturas Bidimensional (DEB). O DEB é similiar ao Gráfico de Estruturas do método "Structured Design" (YOURDON, 1978), com a exceção de que o DEB usa ambientes de fluxos de dados horizontais, bem como pares de fluxos de dados verticais e conexões verticais entre módulos. A figura IV.22.3 apresenta o exemplo de um DEB.

A notação da linguagem para DEB é composta dos seguintes símbolos:

. retângulos - para representar os módulos;

. flechas - para representar os fluxos de dados. Os fluxos de dados verticais são indicados por letras. Se a letra estiver à esquerda da flecha, o fluxo é para baixo; se estiver à direita, o fluxo é para cima. Os fluxos de dados horizontais ou ambientes são identificados por números.

O método utiliza as seguintes regras para evitar falhas no projeto:

- **Redundância vertical** - esta regra determina que os dados que fluem na direção vertical não devem ser passados de um módulo funcional em um determinado nível, para um outro de nível mais alto, e depois, sem ser utilizado,

voltar para outro módulo funcional de nível mais baixo. Nesse caso os dados devem ser passados entre os módulos de nível mais baixo usando um módulo ambiental.

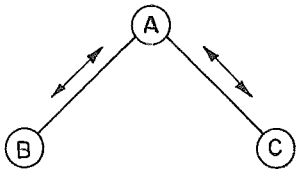


Figura IV.22.1: Fluxo de Dados Vertical
Fonte: (ROTENSTREICH, 1986)

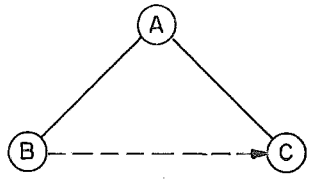


Figura IV.22.2 : Fluxo de Dados Horizontal
Fonte: (ROTENSTREICH, 1986)

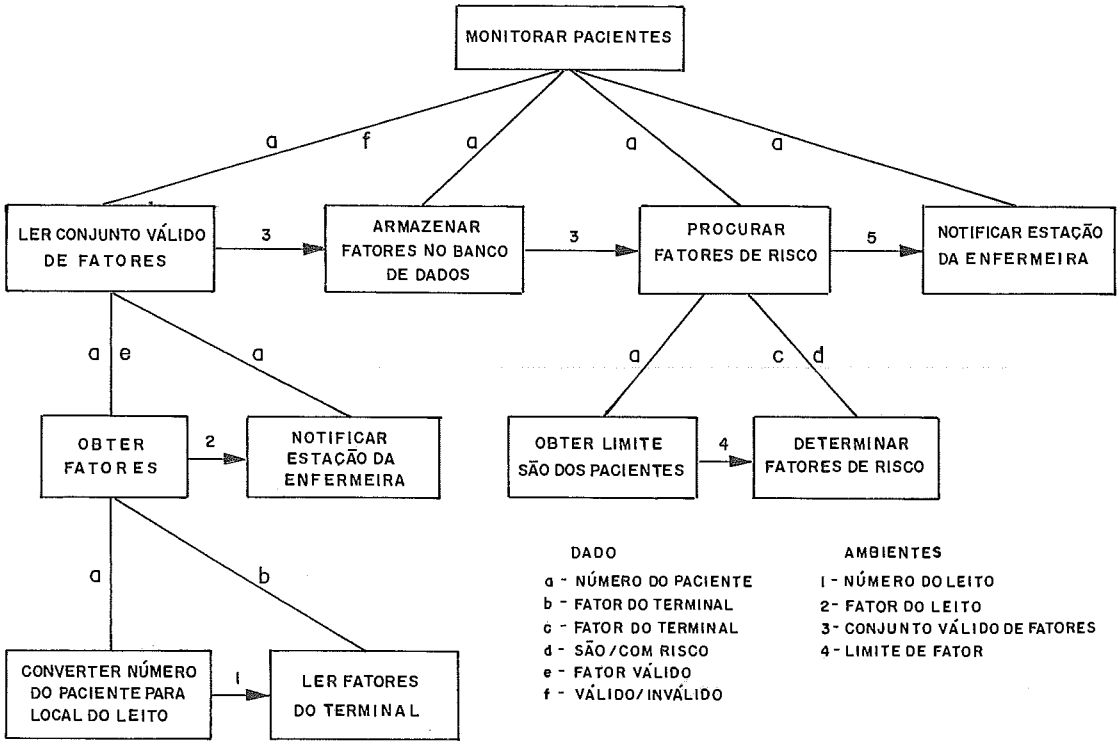


Figura IV.23: Diagrama de Estrutura Bidimensional
Fonte: (ROTENSTREICH, 1986).

– **Redundância horizontal** – esta regra determina que os módulos ambientais não devem ser usados para mover dados de um lugar do projeto para outro, quando a fonte ou o destino dos dados for um módulo de nível mais alto.

Um produto projetado usando o método TDPD pode ser implementado usando diferentes abordagens, incluindo programação orientada a objetos. Podem ser utilizadas linguagens com diferentes paradigmas de programação, tais como, por exemplo, FORTRAN, COBOL, Algol, Pascal, LISP, Módula e Ada.

IV.23 "USER SOFTWARE ENGINEERING" (WASSERMAN, 1982) (WASSERMAN, 1986)

"User Software Engineering" (USE) é um método para o desenvolvimento de Sistemas Interativos (SIs). Abrange todas as fases do ciclo de vida e permite a construção de protótipos do sistema. Foi desenvolvido por Anthony Wasserman.

Esses sistemas são altamente conversacionais, o que leva à participação efetiva do usuário nas fases iniciais de especificação do produto. Assim, o método USE considera a definição dos diálogos como o ponto de partida para a especificação. Permite a construção de um protótipo da interface com o usuário, a partir de onde são definidas e implementadas as funções a serem executadas pelo sistema.

O método utiliza os seguintes instrumentos:

- Linguagem para Diagrama de Entidade-Relacionamento (DER) do método "Entity-Relationship Model" (CHEN, 1976);

- Linguagem para Diagrama de Fluxo de Dados (DFD) do método "Structured Systems Analysis" (DEMARCO, 1978);

- Dicionário de Dados;

- Linguagem para especificação informal de operações;

- Linguagem para especificação de diálogo;

- Linguagem para manipulação de dados do Sistema Troll/USE;

- "Programming LAnguage for INteraction" (PLAIN) - é uma linguagem de programação procedimental derivada do Pascal, que oferece um conjunto de características para a construção de SIs e operações para suportar a definição e a manipulação do banco de dados relacionais;

- Linguagem para Diagrama de Transição USE (DT/USE) - o DT/USE permite a especificação dos dados de entrada, de saída, a ligação desses dados às operações do sistema e a definição do "lay-out" das telas para cada operação. A notação da linguagem para DT/USE é composta de:

  . círculos, que representam os nós ou estados;
  . arcos, que representam as transições;
  . retângulos, que representam as subconversações e as ações a serem executadas.

O DT/USE conta também com variáveis alfanuméricas, símbolos de gerência de tela e cursor e a dependência da seqüência dos diálogos nos resultados das ações.

USE é automatizado por um conjunto integrado de ferramentas, denominado de "Unified Support Environment", que suporta a construção de protótipos, a implementação do sistema e o controle do processo de desenvolvimento. As ferramentas de USE são identificadas a seguir:

- **"Rapid Prototypes of Interactive Dialogues" (RAPID/USE)** - permite a construção rápida de protótipos do sistema, podendo ser utilizadado para verificação e análise do protótipo da interface com o usuário. Possui os seguintes componentes:

  - **Editor de Diagrama de Transição (EDT)** - responsável pela geração de DT/USE e pela sua codificação na Linguagem de Especificação de Diálogos;
  - **Interpretador de Diagrama de Transição (IDT)** - efetua a interpretação do DT/USE codificado, permitindo a construção e modificação de protótipos do diálogo sistema/usário;
  - "Action Linker" - responsável pela integração do IDT com o Sistema Troll/USE e linguagens de programação procedimental.

- **Sistema Troll/USE** - provê uma interface de comunição para sistemas de banco de dados relacional, oferecendo uma linguagem para modelagem de dados;

- **Sistema de Controle USE** - suporta a organização modular do sistema, provê uma estrutura unificada para atividades do desenvolvimento, controla versões e registra automaticamente as atividades efetuadas.

O desenvolvimento de um sistema utilizando o método USE é efetuado através das seguintes fases:

- **Análise dos Requisitos** - durante esta fase são realizados os seguintes passos:

  . construir um modelo conceitual dos dados do sistema usando o Modelo de Hierarquias Semânticas de SMITH (1983) ou o DER (CHEN, 1976);

  . modelar as funções do sistema utilizando o DFD (GANE, 1979).

  . definir textualmente as funções através da linguagem para especificação de operações (figura IV.23.1);

  . construir um dicionário de dados contendo informações sobre todos os dados e operações definidos na fase de análise, de forma que a consistência das informações possa ser mantida ao longo do desenvolvimento.

O método procura obter vários níveis de abstração através da independência dos dados e da independência de diálogo, o que permite a definição de diferentes diálogos no mesmo sistema, para vários tipos de usuários.

- **Projeto da interface com o usuário** - nesta fase é elaborada a especificação da interface com o usuário. Os DTs/USE são construídos e codificados na linguagem de especificação de diálogos pelo EDT. A Figura IV.23.2 mostra o exemplo de um DT/USE com a especificação do controle de tela.

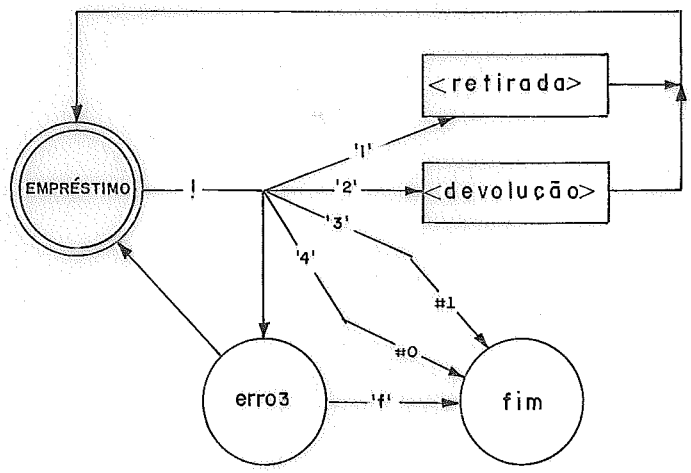
As subconversações de um DT/USE podem ser decompostas em um DT/USE de maior detalhe, como pode ser visto na figura IV.23.3.

```
Operação: Retirada
Entrada:  livro-id, num-cop, cod-leit, data-entr, MAX
Saída:    estado
Função:

 Processa a retirada de uma cópia num-cop de um livro
livro-id, para um leitor cod-leit, com uma data de
entrega data-entr, somente se cod-leit tiver MAX ou menos
livros já retirados.

Operação: Devolução
Entrada:  cod-leit, livro-id, num-cop
Saída:    -
Função:

Processa a devolução de uma cópia num-cop do livro com
o número de identificação livro-id, pelo leitor cod-leit.

Operação: Empréstimos
Entrada:  cod-leit
Saída:    títulos
Função:

 Retorna o conjunto de títulos dos livros atualmente
retirados pelo leitor cod-leit.
```

Figura IV.23.1: Operações descritas através da linguagem para especificação informal de operações.

Fonte: (WASSERMAN, 1986)

- **Construção do protótipo da interface com o usuário** - nesta fase o protótipo da interface é construído pelo IDT, que interpreta os DTs/USE codificados pelo EDT, permitindo ao usuário simular o diálogo com o sistema. Isso possibilita aos usuários apontar as falhas na interface que serão corrigidas de forma dinâmica.




```
diagram  emprestimo  entry emprestimo
                     exit  fim
tab t_0 20
tab t_1 25
node emprestimo
   cs,ra,rv,c_ 'Sistema de Biblioteca', sv,
   r+a, t_0, 'Por favor escolha:',
   r+a, t_1, '1) Retirada de livro',
   r+a, t_1, '2) Devolução de livro',
   r+a, t_1, '3) Sair do Subsistema de Emprestimo',
   r+a, t_1, '4) Sair de Sistema de Biblioteca',
   r+a, t_0, 'Sua opção (1-4):'
node fim
node erro3
   rs-1, rv, bell, 'Por favor digite um número de 1 a 4',
   rs, co, sv,  'aperte a tecla RETURN para continuar ou f para sair desta seção.'
```

Figura IV.23.2: Diagrama de Transição de Estado USE
Fonte: (WASSERMAN, 1986)

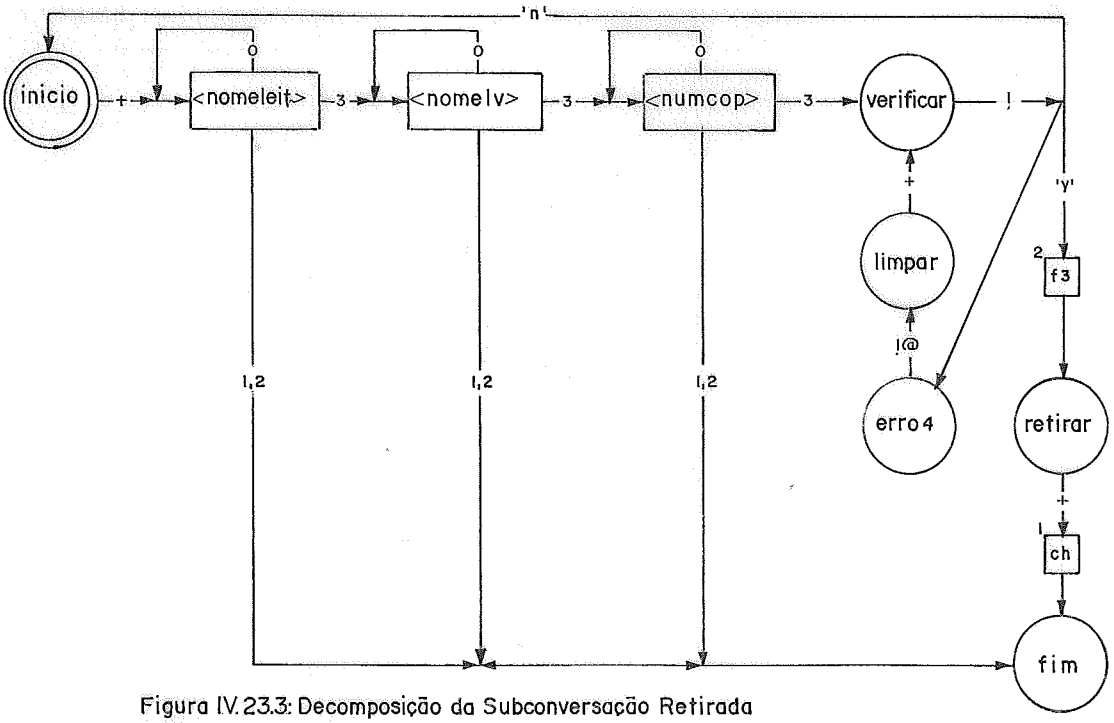


Figura IV.23.3: Decomposição da Subconversação Retirada
Fonte: (WASSERMAN, 1986)

Essas fases iniciais do método USE podem ser executadas de forma paralela, permitindo que a interface com o usuário seja especificada antes de completar a modelagem dos dados e atividades.

- **Avaliação do projeto da interface com o usuário** - nesta fase é feita a avaliação da interface em conjunto com o usuário. A avaliação é possível graças à manutenção de arquivos do tipo "log", contendo as entradas para o sistema digitadas pelo usuário e as transições de estado.

- **Construção de um sistema executável** - nesta fase pode-se optar por uma das seguintes alternativas para a implementação do sistema:

  . expandir o protótipo com funções programadas e operações de manipulação do banco de dados, de forma a abranger todas as operações do sistema. As operações que manipulam o banco de dados podem ser reescritas na linguagem de manipulação de dados relacionais do Sistema Troll/USE, cujo exemplo pode ser visto na figura IV.23.4;

  . abandonar o protótipo e continuar o projeto usando uma linguagem de programação tradicional, ou,

  . abandonar o protótipo e continuar o projeto fazendo uma especificação formal do sistema. Neste caso aconselha-se o uso do método "Behavioral Approach to the Specification of Information System" (BASIS) (LEVERSON, 1983), que é uma abordagem de um modelo

abstrato para especificação, baseado em (HOARE, 1972) e refinado para uso na linguagem de programação ALPHARD (SHAW, 1981).

```
operation livropronome (nome-autor)
  ans1 := livro where autores $ nomeautor;
  ans1 := é o conjunto de livros onde o conjunto de
          autores contém o nome do autor dado

operation retirada (livro-id, num-cop, cod-leit,
                    data-entr, MAX -> estado)
  inset loan [$0, $1, $2, $3, $4];
  copialivro [$1, $2].ultimoleitor := $0;
  copialivro [$1, $2].estado := retirada;
  leitores [$0].retirada := leitores [$].retirada + 1;
  $5 := normal;
```

Figura IV.23.4: Linguagem de manipulação de dados do Sistema Troll/USE

Fonte: (WASSERMAN, 1986)

A construção do protótipo do sistema é feito com o auxílio do RAPID/USE. O "Action Linker" é responsável pela ligação da definição do diálogo, identificada a partir do IDT, com as operações a serem executadas pelo sistema. Essas operações podem ser programadas na linguagem de manipulação de dados do Sistema Troll/USE e em linguagens de programação procedimental, tais como C, FORTRAN 77, Pascal e PLAIN.

O método USE integra o desenvolvimento de interfaces com o usuário no processo de construção de software, utilizando um conjunto de ferramentas para suportar a rápida elaboração e modificação de protótipos dessas interfaces. A participação efetiva do usuário nos estágios iniciais do desenvolvimento leva ao aumento da

confiabilidade do sistema e do grau de satisfação do próprio usuário.

Como pode ser visto, USE permite a combinação de vários métodos formais e informais, e possibilita a alternativa de desenvolver protótipos do sistema, com o objetivo de aumentar a produtividade e aplicar um processo sistemático para o desenvolvimento de software.

IV.24 "VIENNA DEVELOPMENT METHOD" (COHEN, 1986)

"Vienna Development Method" (VDM) é um método formal que segue uma abordagem baseada em modelos, onde as especificações são expressas através de modelos explícitos do sistema. É utilizado principalmente como método para especificação de requisitos, podendo ser aplicado, entretanto, durante todo processo de desenvolvimento de software. Foi desenvolvido nos laboratórios de pesquisa da IBM em Viena ("IBM Vienna Research Laboratories").

VDM está baseado em semântica denotacional, que é uma teoria desenvolvida com o objetivo de especificar formalmente a semântica de linguagens de programação. Na semântica denotacional são fornecidas funções semânticas que mapeiam as construções sintáticas do programa nos valores abstratos que eles denotam.

A partir dessa visão, o método foi usado originalmente para definir linguagens de programação. Contudo, sua aplicação foi ampliada, tornando-se um método de desenvolvimento de software utilizado em vários tipos de

aplicações, destacando seu uso na área industrial. É apropriado para a especificação de sistemas com processamento seqüencial.

VDM é uma abordagem baseada em modelos no sentido de que as descrições dos sistemas, tanto a especificação de requisitos como a de projeto, são dados como modelos. Os modelos são descritos por meio de abstrações matemáticas, tais como conjuntos e mapeamentos finitos, e compostos de:

. dados, que representam entradas, saídas e o estado interno do sistema, e,

. operações e funções, que manipulam os dados.

VDM utiliza como instrumento notacional uma linguagem para especificação denominada Meta-IV. As construções gramaticais da notação de Meta-IV correspondem a objetos no domínio matemático, no qual a notação se baseia. A linguagem possibilita a definição de sistemas com alto rigor matemático. A figura IV.24.1 mostra o exemplo de uma especificação em VDM.

Em VDM um sistema é modelado definindo-se a estrutura de seus estados e as operações que podem ser feitas sobre esses estados. Assim, uma especificação VDM consiste de duas partes principais:

- a definição de variáveis abstratas usadas para representar o estado interno do modelo do sistema. Para cada variável deve ser definido um tipo de dado que denota um conjunto de valores possíveis de serem assumidos por essas variáveis;

- a definição das operações e funções que agem nessas variáveis e que podem estar disponíveis para os usuários do sistema. A especificação das operações é feita através da linguagem da Lógica dos Predicados.

```
Estado:: CANDIDATOS : Pessoa-set
         CANDIDATOS-APROVADOS : Pessoa-set
         Pessoa::   NOME : Ident
                    NOTA-1 : Int
                    NOTA-2 : Int
                    NOTA-3 : Int
                    MEDIA  : Int

Ident = /* um conjunto de nomes distintos */
Int = /* conjunto de inteiros */

CADASTRAR-CANDIDATO (A:Pessoa)
ext CANDIDATOS: wr Pessoa-set
pre a ∉  candidatos
pos candidatos' = candidatos U (a)

ANALISAR-NOTAS (A:Pessoa)
ext CANDIDATOS : wr Pessoa-set
    CANDIDATOS-APROVADOS : wr Pessoa-set
pre NOTA-1 (a) >= 5 ^ NOTA-2 (a) >= 6 ^
    NOTA-3 (a) >= 7 ^ MEDIA  (a) >= 7 ^
    a ∈  CANDIDATOS
pos candidatos' = candidatos - (a) ^
    candidatos-aprovados' = candidatos-aprovados U (a)
```

Figura IV.24.1: Especificação de um Sistema em VDM

Fonte: (MENDES, 1989)

Um modelo em VDM é representado por um estado global do sistema e por operações sobre esse estado. O estado determina o modelo dos dados internos do sistema. A linguagem Meta-IV fornece os tipos de dados que podem ser utilizados para construir o modelo. Em VDM, especifica-se um sistema fornecendo o seu estado inicial. Deve-se também definir as invariantes de estado. As invariantes são condições adicionais que restringem os valores que as variávies de estado podem assumir. É importante verificar

se essas invariantes são satisfeitas no estado inicial do sistema e se são preservadas pelas operações definidas sobre esse estado.

A especificação de um sistema em VDM é feita de forma "top-down", a partir da construção de um modelo abstrato do sistema ao qual são acrescentados gradativamente detalhes de implementação. A construção do sistema é feita através de vários níveis, onde em cada nível são acrescentados mais detalhes de implementação. As especificações são detalhadas e os dados e as funções, definidas inicialmente de forma abstrata, são transformados até chegarem a um nível implementável.

Em cada etapa do desenvolvimento são realizadas verificações, por meio de regras de provas, para avaliar se as estruturas de dados e as operações correspondem àquelas definidas no nível anterior. Na última fase é necessário provar que o código produzido preenche corretamente os requisitos definidos na especificação de menor nível de abstração.

IV.25 CONCLUSÃO

Neste capítulo foram apresentadas descrições de vários métodos para o desenvolvimento de software presentes na literatura. A seleção dos mesmos foi feita buscando-se ter métodos representativos de uma variedade de abordagens e propósitos. No próximo capítulo é apresentada uma avaliação desses métodos segundo os critérios de qualidade definidos no Capítulo III.

CAPÍTULO V

AVALIAÇÃO DOS MÉTODOS

V.1 INTRODUÇÃO

Neste capítulo é feita uma avaliação dos métodos para o desenvolvimento de software descritos no Capítulo IV. Esses métodos são avaliados levando-se em consideração os atributos definidos no Capítulo III.

V.2 AVALIAÇÃO SEGUNDO OS ATRIBUTOS DE QUALIDADE

Durante a avaliação, foram considerados, apenas, os objetivos utilizabilidade para construção e utilizabilidade para avaliação. Isso se deve ao fato de não ter sido encontrado nenhum método que oferecesse apoio à gerência do processo de desenvolvimento quando da avaliação dos métodos. As seguintes abreviaturas foram utilizadas na documentação dos resultados:

- . NA (não se aplica) - indica que o critério não está relacionado ao método em questão;
- . NP (não possui) - indica que o método não possui o critério;
- . SD (sem dados para avaliar) - indica que não foram encontrados dados suficientes para avaliar o critério.

O resultado da avaliação dos métodos descritos no cpítulo IV é apresentado a seguir:

V.2.1 AVALIAÇÃO DO MÉTODO BOP

MÉTODO: "BOP Prototyping System"

DESENVOLVEDOR: Lab. de Engenharia de Produção do NTH-SINTEF

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida baseado em protótipos (prototipagem incremental).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento "on line".

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Sistemas de informação interativos, em particular em sistemas de gerência de produção.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de modelagem da informação, pois em primeiro lugar é elaborada a descrição dos dados e dos relacionamentos existentes entre eles; a partir dessa descrição é feita a especificação das transações do sistema.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

- Técnica "most-critical-element-first": as principais funções do sistema são implementadas em primeiro lugar; a partir daí o protótipo é expandido até abranjer todo o sistema;
- Técnica "outside-in", pois o protótipo é elaborado a partir da visão do usuário.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para definição de elementos do modelo:
  . ling. p/ Dicionário de Atributos (DA)
  . ling. p/ Dicionário de Relações (DR)

- Instrumento para elaboração de código:
  . linguagem APL.

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem natural c/ sintaxe e semântica restritas:
  . linguagem para DA e linguagem para DR

- Ling. natural acrescida de simbologia:
  . linguagem APL

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Formal e semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

- (JORDANGER, 1981b), (JORDANGER, 1981c), (AARAM, 1984);
- Manual do Usuário: (JORDANGER, 1981a).

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

Conhecimento da linguagem APL.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Totalmente automatizado.

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

A utilização de dicionários de dados, para armazenar os atributos e as relações entre os dados, torna as transações do usuário independentes da definição dos dados, facilitando as modificações e extensões do protótipo.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

O usuário pode experimentar o protótipo do sistema, a fim de avaliar se as características funcionais oferecidas atende às suas necessidades.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

O próprio protótipo.

V.2.2 AVALIAÇÃO DO MÉTODO DARTS

MÉTODO: "Design Approach for Real-Time Systems"

DESENVOLVEDOR: Hassen Gomaa da General Eletric

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fases de análise e projeto).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento em tempo real.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Aplicações de tempo real.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

- Abordagem de fluxo de dados, pois a especificação das funções do sistema é feita a partir do fluxo de dados existente entre elas;
- Abordagem de decomposição funcional, pois as funções do sistema são agrupadas em tarefas que são decompostas em uma estrutura modular.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

- Técnica "top-down", para construir o DFD e para decompor as tarefas em uma estrutura de módulos;

- Técnica "bottom-up", para identificar as tarefas no DFD, através da análise dos processos.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de estados:
  - linguagem para Diagrama de Transição de Estados (DTE)

- Instrumento para modelagem de fluxo de dados:
  - linguagem para Diagrama de Fluxo de Dados (DFD)

- Instrumento para definição de elementos do modelo:
  - ling. para Dicionário de Dados (DD)

- Instrumento p/ modelagem de concorrência e sincronização:
  - linguagem para Gráfico de Estrutura de Tarefas (GET)

- Instrumento para modelagem de estrutura de funções:
  - ling. para Gráfico de Estruturas (GE)

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem para DTE
  - linguagem para DFD
  - linguagem para GET
  - linguagem para GE
- Linguagem natural com sintaxe e semântica restritas:
  - linguagem para DD

  As linguagens para DFD e GE têm ainda capacidade de dar um "zoom".

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(GOMAA, 1984), (GOMAA, 1986), (GOMAA, 1987), (GOMAA, 1989)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

Conceitos sobre sistemas de tempo real, como: sincronização, concorrência e comunicação entre tarefas.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Ferramentas desenvolvidas na COPPE/UFRJ:

- DARTS-DT (CAVALCANTE, 1988);
- DDDARTS (ANGULO, 1988).

AUTOMATIZABILIDADE:

Todos os instrumentos do método podem ser automatizados.

EXTENSIBILIDADE:

Os modelos gerados pelo método não oferecem dificuldade de serem alterados, caso tenham sido construídos com o auxílio de ferramentas.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

As normas de qualidade definidas pelo método "Structured Design" (PAGE-JONES, 1980) devem ser aplicadas pois DARTS utiliza instrumento desse método.

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

A compreensão dos modelos gerados é facilitada pelo uso de linguagens gráficas como notação acessível aos usuários.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.3 AVALIAÇÃO DO MÉTODO ERM

MÉTODO: "Entity-Relationship Model"

DESENVOLVEDOR: Peter Chen

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fase de análise).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

NA

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Aplicações onde há necessidade de modelar o relacionamento entre os dados. É aplicado, em particular, em aplicações que utilizem banco de dados.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de modelagem da informação: o método constrói um modelo contendo as entidades do sistema e os relacionamentos existentes entre elas.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "most-critical-element-first" pois a construção do Diagrama de Entidade-Relacionamento é iniciada a partir das principais entidades do sistema.

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza como instrumento a linguagem para Diagrama de Entidade-Relacionamento (DER), que pode ser classificada como:

- Instr. para modelagem de relacionamentos entre dados;

- Instr. para definição de banco de dados.

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

A linguagem para DER pode ser classificada com linguagem com gráficos em forma livre e texto em linguagem natural.

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(FERNANDES, 1989), (ORR, 1989)
(CHEN, 1990), (GANE, 1990)
(YOURDON, 1990)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de um nível de formação específico para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

- Ferramentas desenvolvidas no exterior:

  . Analyst/Designer Toolkit da Yourdon
  . Corvision da Cortex Corp.
  . Deft da DEFT INC.
  . DESIGN/1 da Arthur Andersen & Co.
  . ER-DESIGNER da Chen & Associates
  . ENVISION da Enriched Data Systems
  . EXCELERATOR da Index Technology Corp.
  . Information Engineering Workbench da Knowledge Ware Inc.
  . MAESTRO da Softlab Inc.
  . MULTI/CAM da AGS Management Systems
  . ProKit*WORKBENCH da McDonnell Douglas Information Systems Group
  . Software Through Pictures da Interac. Development Environments Inc.
  . SYSTEM ENGINEER da LBMS
  . TEAMWORK da CADRE Technologies Inc.
  . THE DEVELOPER da ASYST Technologies

- Ferram. disponíveis no mercado nacional:

  . PC-CASE do IBPI
  . TALISMAN da Staa Informática.

- Ferramentas desenvolvidas na COPPE/UFRJ:

  . FEGRES (TRAVASSOS, 1990).

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

Os modelos gerados pelo método não oferecem dificuldades de serem alterados, caso tenham sido construídos com o auxílio de ferramentas.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

As ferramentas que automatizam o método oferecem recursos para verificação da sintaxe dos DERs gerados.

- VALIDABILIDADE -

COMPREENSIVIDADE:

O método utiliza uma linguagem gráfica com notação acessível para os usuários, gerando modelos fáceis de serem entendidos.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.4 AVALIAÇÃO DO MÉTODO ESA

MÉTODO: "Essential Systems Analysis"

DESENVOLVEDOR: Stephen McMenamin

John Palmer

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fase de análise).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

- Processamento seqüencial;
- Processamento "on line".

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Adequado para sistemas interativos, sendo aplicado, em particular, em sistemas de respostas planejadas.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

- Abordagem de fluxo de dados: a modelagem das funções do sistema é feita partir dos dados que entram e saem do sistema;
- Abordagem de modelagem da informação: o método constrói um modelo contendo as entidades e seus relacionamentos.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

- Técnica "top-down" p/ decompor as funções do modelo;
- Técnica "most-critical-element-first" p/ modelar os dados do sistema e os relacionamentos existentes entre eles.

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza instrs. dos métodos "Structured Systems Analysis" (SSA) (DEMARCO, 1978) e "Entity-Relationship Model" (ERM) (CHEN, 1976):

- Instrumento para modelagem de fluxo de dados:
  - linguagem para Diagrama de Fluxo de Dados (DFD)
- Instrumento para especificação textual de funções:
  - português estruturado
- Instrumento para especificação detalhada de funções:
  - linguagem para Árvore de Decisão (AD)
  - linguagem para Tabela de Decisão (TD)
- Instrumento para definição de elementos do modelo:
  - ling. para Dicionário de Dados (DD)
- Instrumento para modelagem da estrutura dos dados:
  - linguagem para Diagrama de Estrutura de Dados (DED)
- Instr. para modelagem de relacionamentos entre dados:
  - linguagem para Diagrama de Entidade-Relacionamento (DER)

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem natural com sintaxe e semântica restritas:
  - português estruturado e linguagem p/ DD
- Ling. com sintaxe/semântica posicionais:
  - linguagem para AD e linguagem para TD
- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem para DFD
  - linguagem para DER
  - linguagem para DED

  A linguagem para DFD tem ainda capacidade de dar um "zoom".

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

— FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO —

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(MCMENAMIN, 1991)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de um nível de formação específico para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNTIVOS:

NP

— FACILIDADE DE USO —

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Não foi encontrada ferramenta específica p/ o método. Ferramentas que automatizam instrumentos dos métodos SSA e ERM podem ser utilizadas como apoio automatizado para o método ESA.

AUTOMATIZABILIDADE:

As linguagens gráficas utilizadas pelo método ESA podem ser todas automatizadas.

EXTENSIBILIDADE:

A ausência de um apoio automatizado pode dificultar as alterações do modelo gerado.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

Ferramentas que automatizam instrumentos utilizados pelo método oferecem recursos para verificação da sintaxe do modelo gerado.

- VALIDABILIDADE -

COMPREENSIVIDADE:

As linguagens utilizadas pelo método possuem notação gráfica simples, gerando modelos fáceis de serem entendidos pelos usuários.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.5 AVALIAÇÃO DO MÉTODO JSD

MÉTODO: "Jackson System Development"

DESENVOLVEDOR: Michael Jackson

Jonh Cameron

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fases de análise e projeto).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

- Processamento seqüencial;
- Processamento em tempo real, pois oferece mecanismos para modelar restrições de tempo e controlar a sincronização entre os processos.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

SD

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem orientada a estruturas de dados, pois as funções do sistema são modeladas a partir dos dados de entrada e de saída.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

- Técnica "top-down" p/ modelar estruturas de dados e para decompor as funções que afetam as entidades do sistema;

- Técnica "most-critical-element-first" p/ identificar as principais entidades do sistema.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de estrutura de dados:
  - linguagem para Diagrama de Estrutura de Entidades (DEE)
  - linguagem para Diagrama de Estrutura de Dados DED)

- Instrumento para modelagem de estrutura de funções:
  - ling. para Diagrama de Especificação de Sistema (DES)

- Instrumento para especificação textual de funções:
  - linguagem p/ especificação detalhada de processos

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Ling. c/ gráficos derivados de princípios teóricos e texto em linguagem natural:
  - linguagem para DEE
  - linguagem para DED

- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem para DES

- Linguagem natural com sintaxe e semântica restritas:
  - linguagem p/ especificação detalhada de processos

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(CAMERON, 1983), (JACKSON, 1983), (PRESSMAN, 1988)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de um nível de formação específico para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

SD

AUTOMATIZABILIDADE:

As linguagens gráficas utilizadas pelo método podem ser automatizadas.

EXTENSIBILIDADE:

A ausência de um apoio automatizado para o método pode dificultar as alterações no modelo gerado.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

As linguagens utilizadas por JSD possuem notação gráfica acessível para os usuários. Os textos que podem ser usados para completar a especificação auxiliam na compreensão do modelo.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.4 AVALIAÇÃO DO MÉTODO KBSDLC

MÉTODO: "Knowledge-Based System Development Life Cycle"

DESENVOLVEDOR: John Weitzel
Larry Kerschberg

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

- Prototipagem descartável, pois o protót. pode ser utilizado como uma especificação;
- Prototipagem incremental, pois o protót. pode ser desenvolvido gradativamente e mantido como o produto final;
- Prototipagem evolutiva, pois o protótipo pode ser utilizado como recurso para aprendizagem;

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

NA

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Sistemas baseados em conhecimento.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

- Abordagem baseada em conhecimento, pois utiliza recursos para representar e manipular o conhecimento de especialistas;
- Abordagem de decomposição funcional, para especificar o processo de decisão.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down".

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza instrumentos para representação de conhecimento:

- lógica de primeira ordem
- representações procedurais
- redes semânticas
- sistemas de produção
- "frames"

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem natural acrescida de simbologia
- Linguagens gráficas

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Formal e semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(WEITZEL, 1989)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não requer uma formação espefíca para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

SD

AUTOMATIZABILIDADE:

SD

EXTENSIBILIDADE:

As alterações do produto são facilitadas pelo uso de protótipos e pela capacidade do método permitir reexecução de fases anteriores do desenvolvimento.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

A compreensão do modelo gerado é facilitada pelo uso do protótipo, que pode ser experimentado pelo usuário para avaliar se é realmente o que deseja.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

O próprio protótipo pode ser utilizado para validar o modelo gerado.

V.2.7 AVALIAÇÃO DO MÉTODO KBSDM

MÉTODO: "Knowledge-Based System Development Methodology"

DESENVOLVEDOR: Robert Keller

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida baseado em protótipos (prototipagem descartável).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

NA

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Sistemas baseados em conhecimento, em particular sistemas especialistas.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

- Abordagem baseada em conhecimento, pois oferece recursos para manipular o conhecimento de especialistas;
- Abordagem de fluxo de dados, pois a especificação das funções do especialista é feita a partir dos fluxos de dados de entrada e saída.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down", para decompor as decisões do especialista.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de fluxo de dados:
  - linguagem para Diagrama de Fluxo de Dados (DFD)

- Instrumento para definição de elementos do modelo:
  - linguagem para Dicionário de Dados e Regras (DDR)

- Instr. para representação de conhecimento

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem para DFD
- Linguagem natural com sintaxe e semântica restritas:
  - linguagem para DDR

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(KELLER, 1987)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não requer formação específica para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

SD

AUTOMATIZABILIDADE:

SD

EXTENSIBILIDADE:

O uso de protótipos facilita a alteração do modelo gerado.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

As linguagens gráficas utilizadas pelo método possuem notação gráfica acessível para o usuário.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

O próprio protótipo.

V.2.8 AVALIAÇÃO DO MÉTODO MBSD

MÉTODO: "Methodology for Business System Development"

DESENVOLVEDOR: Raghubir Mathur

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo ciclo de vida clássico (fase de projeto).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

- Processamento seqüencial;
- Processamento "on line".

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Sistemas comerciais orientados a transação.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem orientada a estrutura de dados: as funções são especificadas a partir dos dados associados a cada estímulo e da identificação dos caminhos lógicos existentes entre as entradas e as saídas.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "outside-in", pois em primeiro lugar é projetada a visão do usuário, para em seguida detalhar as transações a serem executadas.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de estrutura de dados:
  - ling. para Gráficos de Bolhas (GB)
- Instrumento para modelagem de fluxo de controle:
  - linguagem para definição de caminhos lógicos
- Instrumento para modelagem da estrutura de funções:
  - linguagem p/ Elementos de Decomposição (ED)
  - linguagem p/ Diagrama de Verificação do Sistema (DVS)
- Instrumento para especificação textual de funções:
  - linguagem de projeto de programa

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem natural c/ sintaxe e semântica restritas:
  - linguagem de projeto de programa
- Ling. com sintaxe/semântica posicionais:
  - linguagem para ED
  - linguagem para DVS
- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem para ED
  - linguagem para DVS
  - linguagem para GB
  - linguagem para definição de caminhos lógicos

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(MATHUR, 1987)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não requer uma formação específica para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

SD

AUTOMATIZABILIDADE:

Todas as linguagens gráficas do método podem ser automatizadas.

EXTENSIBILIDADE:

O uso de formulários pré-definidos para o projeto de telas e relatórios e para a definição das funções do sistema, facilita as alterações.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

Documentação gerada na forma de gráficos e formulários ajuda no entendimento e na validação do produto.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.9 AVALIAÇÃO DO MÉTODO OOA

MÉTODO: "Object-Oriented Analysis"

DESENVOLVEDOR: Peter Coad

Edward Yourdon

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

- Modelo de ciclo de vida clássico (fase de análise);
- Modelo de ciclo de vida baseado em reutilização, pois permite reuso de classes definidas.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

- Processamento seqüencial;
- Processamento "on line";
- Processamento em tempo real.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Tem sido utilizado para especificação de ferramentas, podendo ser usado em outros tipos de aplicações.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de orientação a objetos.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

- Técnica "bottom-up", para identificar os elementos do modelo;
- Técnica "top-down", para representar as camadas.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de objetos:
  - linguagem para Diagramas OOA (DOOA)
- Instrumento para especificação detalhada de funções:
  - linguagem para Tabela de Serviços e Estados
  - linguagem para Tabela de Execução de Caminhos Críticos
  - ling. para Tabela de Tempo e Tamanho
- Instrumento para especificação textual de funções:
  - ling. para especificação de atributos e serviços

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Ling. natural acrescida de simbologia:
  - linguagem para especificação de atributos e serviços
- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem para DOOA
- Ling. com sintaxe/semântica posicionais:
  - tabelas
- Linguagem natural sem restrições:
  - utilizada para especificar funções

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(COAD, 1990)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de uma formação específica para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Ferramenta desenvolvida na COPPE/UFRJ:

EDC (MATTOSO, 1990)

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

Os modelos gerados pelo método não oferecem dificuldades de serem alterados, caso tenham sido construídos com o auxílio de ferramentas.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

As linguagens utilizadas pelo método não oferecem dificuldades de serem entendidas pelos usuários.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.10 AVALIAÇÃO DO MÉTODO OOS

MÉTODO: "Object-Oriented Specification"

DESENVOLVEDOR: Sidney Bailin

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

- Modelo de ciclo de vida clássico (fases de análise e projeto);
- Modelo de ciclo de vida baseado em reutilização, no caso de existir uma biblioteca para os objetos e as funções possíveis de serem reutilizados.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento seqüencial.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

...

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

- Abordagem de orientação a objetos, para definir objetos e encapsular funções;
- Abordagem de fluxo de dados para decompor as entidades e funções no DFDE.
- Abordagem de modelagem da informação, p/ identificar as entidades do sistema e os relacionamentos existentes entre elas.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

- Técnica "top-down", p/ decompor entidades e funções;

- Técnica "most-critical-element-first" p/ identificar as principais entidades do sistema.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de fluxo de dados:
  - linguagem para Diagrama de Fluxo de Dados (DFD)

- Instrumento para modelagem de objetos:
  - ling. para Diagrama de Objetos (DO)

- Instr. para modelagem de relacionamentos entre dados:
  - linguagem para Diagrama de Entidade-Relacionamento (DER)

- Instrumento para definição de elementos do modelo:
  - ling. para Dicionário de Dados (DD)

- Instrumento para modelagem de objetos e modelagem de fluxo de dados:
  - linguagem para Diagrama de Fluxo de Dados e Entidades (DFDE)

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem p/ DFD, linguagem p/ DFDE
  - linguagem p/ DO, linguagem p/ DER

- Linguagem natural com sintaxe e semântica restritas:
  - linguagem para DD

  As linguagens para DFDE e DFD têm ainda capacidade de dar um "zoom".

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(BAILIN, 1989)

(STARK, 1987) para o DFDE.

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de um nível de formação específico para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

- Ferramenta para verificar a consistência entre elementos do DER e do DFDE;
- Ferramenta para transformação automática de DFDE em DO.

AUTOMATIZABILIDADE:

Todos os instrumentos do método podem ser automatizados.

EXTENSIBILIDADE:

Todas as funções a serem executadas por uma entidade são incluídas no modelo, mesmo que não seja necessário utilizá-las de imediato. Além disso, o método procura manter a portabilidade de componentes reutilizáveis. Esses procedimentos visam aumentar a capacidade de alterar o produto com o mínimo de reespecificação e reprojeto.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

Deve-se verificar o acoplamento, coesão, viabilidade e completeza funcional das entidades durante a especificação de novas entidades e o agrupamento de funções.

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

Ferramenta p/ verificação da consistência entre elementos do DER e do DFDE.

- VALIDABILIDADE -

COMPREENSIVIDADE:

Os modelos são representados através de linguagens com notações acessíveis para o usuário.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.11 AVALIAÇÃO DO MÉTODO OOSA

MÉTODO: "Object-Oriented Systems Analysis"

DESENVOLVEDOR: Sally Shlaer e Stephen Mellor

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fase de análise).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento seqüencial.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

O método tem sido utilizado em aplicações comerciais, podendo ser utilizado em outros tipos de aplicações.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

- Abordagem de modelagem da informação, pois identifica as entidades do sistema e contrói um modelo contendo as entidades e os relacionamentos existentes entre elas;
- Abordagem de fluxo de dados, pois identifica as funções do sistema e o fluxo de dados existente entre elas.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

- Técnica "most-critical-element-first", utilizada para gerar o DEI e o DCEI;
- Técnica "top-down", utilizada p/ decompor as funções do sistema.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instr. para modelagem de relacionamentos entre dados:
  - linguagem para Diagrama de Estrutura da Informação (DEI)
  - linguagem para Diagrama de Contexto da Estrutura da Informação (DCEI)
- Instrumento para modelagem de estados:
  - linguagem para Diagrama de Transição de Estados (DTE)
- Instrumento para especificação textual de funções:
  - português estruturado
  - pseudocódigo
- Instrumento para especificação detalhada de funções:
  - linguagem para Árvore de Decisão (AD)
- Instrumento para modelagem de fluxo de dados:
  - linguagem para Diagrama de Fluxo de Dados (DFD)

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem p/ DEI, linguagem p/ DCEI
  - linguagem p/ DTE, linguagem p/ DFD
- Linguagem natural com sintaxe e semântica restritas:
  - português estruturado e pseudocódigo
- Linguagem posicional:
  - linguagem para AD
- Linguagem natural sem restrições:
  - utilizada para escrever Documento de Especificação de Objeto, Documento de Especificação de Relacionamentos e Documento de Resumo

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal e informal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(SHLAER, 1988)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

Não há necessidade de formação específica para utilizar o método.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

NP

AUTOMATIZABILIDADE:

Todas as linguagens gráficas utilizadas pelo método podem ser automatizadas.

EXTENSIBILIDADE:

A ausência de apoio automatizado pode dificultar a alteração dos modelos gerados pelo método.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

O método utiliza linguagens gráficas com notação acessível para os usuários.

Os documentos textuais, que acompanham os gráficos, auxiliam no entendimento da especificação gerada.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.12 AVALIAÇÃO DO MÉTODO PUI

MÉTODO: "Prototyping of User Interfaces"

DESENVOLVEDOR: Niels Christensen

Klaus-Dieter Kreplin

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida baseado em protótipos. O protótipo gerado pode ser descartado ou utilizado como parte do produto final.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento "on line".

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Aplicações interativas.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em pequena e larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

NA

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "outside-in", pois o protótipo da interface do produto é elaborado a partir da visão do usuário final.

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza uma linguagem para especificar protótipos da interface que pode ser classificada como:

- instrumento para modelagem de diálogo.

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

A linguagem para especificar protótipos pode ser classificada como:

- linguagem natural com sintaxe e semântica restritas.

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(CHRISTENSEN, 1984)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

Não há necessidade um nível de formação específico para utilizar o método.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Totalmente automatizado.

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

O apoio automatizado oferecido pelo método facilita alterações do protótipo da interface.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

O próprio protótipo pode ser utilizado para verificação da interface.

- VALIDABILIDADE -

COMPREENSIVIDADE:

O usuário tem disponível o protótipo da interface, podendo experimentá-lo a fim de avaliar se é realmente o que deseja.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

O próprio protótipo pode ser utilizado para validação da interface.

V.2.13 AVALIAÇÃO DO MÉTODO RIPL

MÉTODO: "Rapid Intelligent Prototyping Language"

DESENVOLVEDOR: Duane Cochran

Frederick Stocker

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida baseado em protótipos.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento "on line".

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Aplicações interativas.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em pequena e larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

NA

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "outside in", pois o protótipo é projetado a partir da visão do usuário final.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de estados:
  . ling. para Diagrama de Estados (DE)

- Instrumento para modelagem de diálogo

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem com gráficos em forma livre e texto em linguagem natural:
  . linguagem para DE

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(COCHRAN, 1985)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de uma formação específica para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

O método oferece tutores "on line" para auxiliar na construção de protótipos da interface com o usuário.

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Totalmente automatizado.

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

Alterações na interface são facilitadas pelo uso de ferramentas para geração de protótipos.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

O próprio protótipo da interface.

- VALIDABILIDADE -

COMPREENSIVIDADE:

O protótipo da interface pode ser experimentado pelo usuário a fim de avaliar se atende às suas necessidades.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

O próprio protótipo da interface.

V.2.14 AVALIAÇÃO DO MÉTODO SBRE

MÉTODO: "Scenario Based Requirements Elicitation"

DESENVOLVEDOR: Depart. de Ciência da Computação da Academia de Força Aérea dos EUA

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

- Ciclo de vida baseado em protótipos, pois após a sua aplicação um protótipo pode ser implementado a partir dos requisitos definidos.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Qualquer tipo de processamento.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Qualquer domínio de aplicação.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de decomposição funcional, pois os conjuntos de objetivos, projetos e cenários são definidos a partir das funções.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down".

TIPO DE INSTRUMENTOS NOTACIONAIS:

SD

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

SD

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

SD

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(HOLBROOK, 1990)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

SD

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Ferramenta em desenvolvimento no Centro de Pesquisas de Eng. de Software da Universidade da Flórida. A proposta é usar sistemas de hipertexto, como HyperCard da Apple, para estabelecer e manter os relacionamentos entre conjuntos de informações geradas pelo método.

AUTOMATIZABILIDADE:

Pode ser automatizado por ferramentas que permitam estabelecer e manter o relacionamento existente entre os conjs. de informações, bem como seus conteúdos.

EXTENSIBILIDADE:

Oferece recursos para alterar conjuntos de informações através do armazenamento das mudanças no conjunto de questões.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

Os modelos gerados devem ser acessíveis aos usuários, pois a participação deles, durante a fase de avaliação do projeto, é considerada importante.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.15 AVALIAÇÃO DO MÉTODO SREM

MÉTODO: "Software Requirements Engineering Methodology"

DESENVOLVEDOR: TWR

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida baseado em protótipos.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento em tempo real.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Aplicações de tempo real.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de decomposição funcional, pois os requisitos são especificados a partir dos procedimentos a serem executados como respostas a estímulos.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down".

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza como instrumento a linguagem "Requirements Specification Language" (RSL), que pode ser classificada como:

- Instr. para especificação de concorrência e sincronização;
- Instrumento para modelagem de fluxo de controle.

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

A linguagem RSL pode ser classificada como linguagem natural acrescida de simbologia.

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(ALFORD, 1977), (ALFROD, 1977)
(DAVIS, 1980)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

Conceitos sobre sistemas de tempo real, tais como: concorrência e sincronização.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Totalmente automatizado.

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

Apesar da complexidade da linguagem RSL, a realização de alterações pode ser facilitada pelo uso de ferramentas.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

- Tradutor da linguagem RSL que verifica a sintaxe e a semântica das declarações escritas em RSL;
- Analisador estático da linguagem RSL que verifica a completeza, consistência e não ambigüidade dos requisitos.

- VALIDABILIDADE -

COMPREENSIVIDADE:

O processo de validação é dificultado pela formalidade da linguagem RSL. Entretanto, a capacidade de simulação dos requisitos oferecida pelo Sistema REVS auxilia ao usuário compreender o produto.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

Simulador do Sistema REVS, que simula os requisitos através do processamento dos algoritmos gerados.

V.2.16 AVALIAÇÃO DO MÉTODO SADT

MÉTODO: "Structured Analysis and Design Technique"

DESENVOLVEDOR: Douglas Ross da SofTech Inc.

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fases de análise e projeto).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento seqüencial.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Tem sido utilizado no desenvolvimento de aplicações comerciais, indústia bélica, treinamento militar, manufaturas e outras.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

- Abordagem de fluxo de dados, pois modela a estrutura funcional a partir do fluxo de dados existente entre as funções.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down", para construir o Diagrama de Atividades e o Diagrama de Dados.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de fluxo de dados:
  - linguagem para Diagrama de Atividades (DA)
- Instrumento para modelagem de estrutura de dados:
  - linguagem para Diagrama de Dados (DD)

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem para DA
  - linguagem para DD

  As linguagens para DA e DD têm ainda capacidade de dar um "zoom".

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(ROSS, 1977), (ROSS, 1977a), (DICKOVER, 1978), (ROSS, 1980), (LEITE, 1983), (ROSS, 1985), (DAVIS, 1990)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de um nível de formação específico para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Design/IDEF e SPECIF-X (DAVIS, 1990).

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

Não oferece dificuldades para alterar modelos, caso tenham sido construídos com o auxílio de ferramentas.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

Gera modelos construídos com notação fácil de ser entendida e acessível aos usuários.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.17 AVALIAÇÃO DO MÉTODO SD

MÉTODO: "Structured Design"

DESENVOLVEDOR: Larry Constantine, Wayne Stevens
Glenford Myers, Meilir Page-Jones
Edward Yourdon

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fase de projeto).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento seqüencial.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

O método tem sido utilizado em aplicações admnistrativas e comerciais, mas pode ser utilizado em outros tipos de aplicações.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em pequena e larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de decomposição funcional, pois modela as funções do sistema em uma estrutura hierárquica.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down", para construir o Gráfico de Estruturas.

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza como instrumento a linguagem para Gráfico de Estruturas (GE) que pode ser classificada como instrumento para modelagem de estrutura de funções.

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

A linguagem para GE pode ser classificada como linguagem em forma livre e texto em linguagem natural.

A linguagem para GE tem ainda capacidade de dar um "zoom".

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(STEVENS, 1974) (MYERS, 1978)
(YOURDON, 1978) (PAGE-JONES, 1980)
(STEVEN, 1988)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de um nível de formação específico para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

- Ferram. disponíveis no mercado nacional:
  - PC-CASE do IBPI
  - MOSAICO da IESA-TS
- Ferram. desenvolvidas na COPPE/UFRJ:
  - EDIT-GE (NOGUEIRA, 1988)
  - FEGRES (TRAVASSOS, 1990)
  - EDIT-GEII (PASSOS, 1991)

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

Os modelos gerados não são difíceis de serem alterados, caso tenham sido construídos com o auxílio de ferramentas.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

Avalia a coesão e o acoplamento entre os módulos.

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

AVALIE-GE (PASSOS, 1991)

- VALIDABILIDADE -

COMPREENSIVIDADE:

A linguagem utilizada pelo método possui notação gráfica simples, gerando modelos acessíveis aos usuários.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.18 AVALIAÇÃO DO MÉTODO SSA

MÉTODO: "Structured Systems Analysis"

DESENVOLVEDOR: Chris Gane/Trish Sarson

Tom DeMarco

Edward Yourdon

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fase de análise).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

- Processamento seqüencial;
- Em (YOURDON, 1990) são incluídos recursos para modelagem de tempo, permitindo o uso para processamento em tempo real.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Tem sido utilizado em sistemas comerciais e administrativos, podendo ser utilizado em outros tipos de aplicações.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de fluxo de dados, pois as funções do sistema são definidas a partir do fluxo de dados existente entre elas.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down".

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de fluxo de dados:
  - linguagem para Diagrama de Fluxo de Dados (DFD)
- Instrumento para especificação textual de funções:
  - português estruturado
  - pseudocódigo
  - português compacto
- Instrumento para especificação detalhada de funções:
  - linguagem para Árvore de Decisão (AD)
  - linguagem para Tabela de Decisão (TD)
- Instrumento para definição de elementos do modelo:
  - linguagem para Dicionário de Dados (DD)
- Instrumento para modelagem da estrutura dos dados:
  - linguagem para Diagrama de Acesso Imediato aos Dados (DAID); na versão de (DEMARCO, 1978) esse instrumento corresponde à linguagem para Diagrama de Estrutura de Dados (DED)
- Instr. para modelagem de relacionamentos entre dados
  - linguagem para Diagrama de Entidade-Relacionamento, usada na versão de YOURDON (1990))
- Instrumento para modelagem de estados:
  - linguagem para Diagrama de Transição de Estados (DTE), usada na versão de YOURDON (1990))

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem natural com sintaxe e semântica restritas:
  - português estruturado
  - pseudocódigo
  - português compacto
  - linguagem para DD
- Ling. com sintaxe/semântica posicionais:
  - linguagem para AD
  - linguagem para TD
- Linguagem com gráficos em forma livre e texto em linguagem natural:
  - linguagem para DFD
  - linguagem para DAID
  - linguagem para DER
  - linguagem para DTE

  A linguagem para DFD tem ainda capacidade de dar um "zoom".

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(DEMARCO, 1978), (GANE, 1979), (GANE, 1988), (YOURDON, 1990)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de um nível de formação específico para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

- Ferramentas desenvolvidas no exterior:

  . Analyst/Designer Toolkit da Yourdon
  . Anatool da Advanced Logical Software
  . Deft da DEFT INC.
  . DesignAid da Nastec Corporation
  . DESIGN/1 da Arthur Andersen & Co.
  . Design/2.0 da Meta Software Co.
  . Design/OA da Meta Software Co.
  . ENVISION da Enriched Data Systems
  . EXCELERATOR da Index Technology
  . EXCELERATOR/RST da Index Technology
  . Information Engineering Workbench da Knowledge Ware Inc.
  . MacBubbles da StarSys Inc.
  . MAESTRO da Softlab Inc.
  . MULTI/CAM da AGS Management Systems
  . ProKit*WORKBENCH da McDonnell Douglas Information Systems Group
  . ProMod/SA e ProMod/RT da ProMod Inc.
  . Software Through Pictures da Interac. Development Environments Inc.
  . Structured Architect da Meta Systems
  . TEAMWORK da CADRE Technologies Inc.
  . THE DEVELOPER da ASYST Technologies
  . Visible Analyst Workbench da Visible Systems Corp.
  . VS Designer da Visual Software Inc.

- Ferram. disponíveis no mercado nacional:

  . PC-DFD do IBPI
  . MOSAICO da IESA-TS
  . TALISMAN da Staa Informática

- Ferramentas desenvolvidas na COPPE-UFRJ:

  . EDIT-DFD (AGUIAR, 1986)
  . DDADOS (BLASCHEK, 1987)
  . FEGRES (TRAVASSOS, 1990)

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

O método não oferece dificuldades para alterar os modelos gerados, caso tenham sido construídos com o auxílio de ferramentas.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

Ferramentas que automatizam instrumentos utilizados pelo método oferecem recursos para verificação da sintaxe dos modelos gerados.

- VALIDABILIDADE -

COMPREENSIVIDADE:

O método utiliza linguagens com notação acessível para os usuários, gerando modelos fáceis de serem entendidos.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.19 AVALIAÇÃO DO MÉTODO SARS

MÉTODO: "System for Application-Oriented Requirements Specification"

DESENVOLVEDOR: M. Hagemann - "Institute for Real-Time Control Systems and Robotics", Universidade de Karlsruhe, República Federal da Alemanha

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida baseado em protótipos, onde o protótipo é desenvolvido de forma rápida pelo gerador de código de SARS.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento em tempo real.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Sistemas embutidos, em particular sistemas de controle de processos.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de decomposição funcional, pois a estrutura do sistema é definida a partir das funções a serem executadas como respostas a estímulos.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down".

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza como instrumento a linguagem LARS que pode ser classificada como:

- Instrumento para especificação textual de funções;
- Instrumento para modelagem de estados;
- Instr. para especificação de concorrência e sincronização.

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem natural com sintaxe e semântica restritas:
  . linguagem LARS
- Linguagem natural sem restrições.

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Formal (linguagem LARS) e informal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(EPPLE, 1983), (HAGEMANN, 1987)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

Conceitos sobre controle de processos.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Totalmente automatizado.

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

O grau de formalidade da linguagem LARS dificulta as alterações do modelo.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

Analisador da linguagem LARS.

- VALIDABILIDADE -

COMPREENSIVIDADE:

A dificuldade de entender as especificações formais geradas pelo método é amenizada pelo protótipo, que pode ser experimentado pelo usuário para ver se atende às suas necessidades.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

O próprio protótipo.

V.2.20 AVALIAÇÃO DO MÉTODO TAGS

MÉTODO: "Technology for the Automated Generation of Systems"

DESENVOLVEDOR: Teledyne Brown Engineering

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida baseado em protótipos. O protótipo é desenvolvido de forma incremental e utilizado como produto final.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

- Processamento seqüencial;
- Processamento em tempo real.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Pode ser aplicado no desenvolvimento de qualquer tipo de aplicação, inclusive em sistemas de hardware e em sistemas embutidos.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de decomposição funcional, pois o fluxo de dados e o fluxo de controle do sistema são especificados a partir da identificação das funções.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down".

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza como instr. a linguagem "Input Output Requirements Language" (IORL), que pode ser classificada como:

- Instrumento para modelagem da estrutura de funções (através do Diagrama de Bloco Esquemático (DBE));
- Instrumento para modelagem de fluxo de controle (através do Diagrama de Tempo e Relacionamento de Entrada/Saída (DTRES) e do Diagrama de Processo Pré-definido (DPP));
- Instr. para especificação de concorrência e sincronização;
- Instrumento para modelagem de estrutura de dados (através da Tabela de Parâmetros Internos (TPI) e da Tabela de Parâmetros de Processo Pré-definido (TPP)).

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

A linguagem IORL pode ser classificada como:

- Linguagem com gráficos em forma livre e texto em linguagem natural (por causa do diagrama DBE);
- Ling. c/ gráficos derivados de princípios teóricos e texto em linguagem natural (por causa dos diagramas DTRES e DPP);
- Ling. com sintaxe e semântica posicionais (por causa das tabelas TPI e TPP).

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(SIEVERT, 1985)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de formação específica para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Totalmente automatizado.

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

As alterações no produto são facilitadas pelo apoio automatizado oferecido pelo método.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

Analisador de diagnósticos, responsável pela análise estática dos diagramas gerados em IORL.

- VALIDABILIDADE -

COMPREENSIVIDADE:

O usuário pode experimentar o protótipo do sistema, a fim de avaliar se ele atende às suas necessidades.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

Compilador de simulação, que produz código fonte na linguagem Ada e executa o protótipo, detectando erros dinâmicos.

V.2.21 AVALIAÇÃO DO MÉTODO TDPD

MÉTODO: "Two-Dimensional Program Design"

DESENVOLVEDOR: Shmuel Rotenstreich

William Howden

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico (fase de projeto).

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento seqüencial.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

O método tem sido utilizado em aplicações comerciais e admnistrativas, podendo ser usado em outros tipos de aplicações.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolver software em pequena e larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

Abordagem de decomposição funcional, pois as funções do sistema são modeladas em uma estrutura hierárquica.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down", para construir o Diagrama de Estruturas Bidimensional.

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza como instrumento a linguagem para Diagrama de Estruturas Bidimensional (DBE) que pode ser classificada como instrumento para modelagem de estrutura de funções.

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

A linguagem para DEB pode ser classificada como linguagem com gráficos em forma livre e texto em linguagem natural.

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Semi-formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(ROTENSTREICH, 1986)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de um nível de formação específico para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

NP

AUTOMATIZABILIDADE:

A linguagem para DEB pode ser automatizada.

EXTENSIBILIDADE:

As alterações do modelo gerado podem ser dificultadas pela ausência de apoio automatizado.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

Como utiliza instrumento similar ao Gráfico de Estruturas do método "Structured Design", supõe-se a aplicação das normas de qualidade utilizadas por esse método.

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

A linguagem gráfica utilizada pelo método é uma linguagem simples, composta por um número limitado de símbolos, não oferecendo dificuldades de ser entendida pelos usuários.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.2.22 AVALIAÇÃO DO MÉTODO USE

MÉTODO: "User Software Engineering"

DESENVOLVEDOR: Anthony Wasserman

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida baseado em protótipos. O protótipo gerado pode ser descartado ou expandido.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento "on line".

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Sistemas interativos.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

- Abordagem de fluxo de dados;
- Abordagem de modelagem da informação.

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

- Técnica "outside-in" pois a especificação das funções do sistema é feita a partir da visão externa do usuário, considerando a definição dos diálogos como o ponto de partida para a construção do produto;

- Técnica "top-down" p/ decompor as funções e as subconversações.

TIPO DE INSTRUMENTOS NOTACIONAIS:

- Instrumento para modelagem de fluxo de dados:
  - linguagem para Diagrama de Fluxo de Dados (DFD)

- Instr. para modelagem de relacionamentos entre dados:
  - linguagem para Diagrama de Entidade-Relacionamento (DER)

- Instrumento para modelagem de estados:
  - linguagem para Diagrama de Transição USE (DT/USE)

- Instrumento para modelagem de fluxo de controle:
  - ling para DT/USE

- Instrumento para definição de elementos do modelo:
  - ling. para Dicionário de Dados (DD)

- Instrumento para especificação textual de funções:
  - linguagem para especificação informal de operações

- Instrumento para modelagem de diálogo:
  - linguagem para DT/USE
  - linguagem p/ especificação de diálogo

- Instrumento para definição de banco de dados:
  - linguagem para manipulação de dados do Sistema Troll/USE

- Instrumento para elaboração de código:
  - "Programming LAnguage for INteraction" (PLAIN)

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

- Linguagem natural sem restrições:
  . linguagem para especificação informal de operações
- Linguagem natural acrescida de simbologia
  . ling. para especificação de diálogo
  . ling. para manipulação de dados do Sistema Troll/USE e ling. PLAIN
- Ling. c/ gráficos derivados de princípios teóricos e texto em linguagem natural:
  . linguagem para DT/USE
- Linguagem com gráficos em forma livre e texto em linguagem natural:
  . linguagem para DFD
  . linguagem para DER
- Linguagem natural com sintaxe/semântica restritas: linguagem para DD

As linguagens para DFD e DT/USE têm ainda capacidade de dar um "zoom".

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Formal, semi-formal e informal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

- (WASSERMAN, 1982) e (WASSERMAN, 1986);
- (MILL, 1984) para o EDT;
- (WASSERMAN, 1985) para o DT;
- (KERSTEN, 1981) e (WASSERMAN, 1981) para PLAIN.

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

O método não necessita de formação específica para ser utilizado.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

Totalmente automatizado.

AUTOMATIZABILIDADE:

NA

EXTENSIBILIDADE:

As alterações do produto são facilitadas:
- pela independência de dados alcançada através do uso de banco de dados e de operações abstratas;
- pela independência de diálogo;
- pelo uso de bibliotecas de operações;
- pelo uso de protótipos.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

NP

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

RAPID/USE para verificação do projeto de diálogos do sistema.

- VALIDABILIDADE -

COMPREENSIVIDADE:

O protótipo facilita o entendimento e a validação do produto, pois pode ser utilizado para simular os diálogos e as funções do sistema. Permite a definição de vários tipos de diálogos do mesmo sistema para vários tipos de usuários.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

- Interpretador de Diagrama de Transição, que simula os diálogos do sistema;
- O próprio protótipo.

V.2.23 AVALIAÇÃO DO MÉTODO VDM

MÉTODO: "Vienna Development Method"

DESENVOLVEDOR: "IBM Vienna Research Laboratories"

- UTILIZABILIDADE PARA CONSTRUÇÃO

- APLICABILIDADE -

ADEQUAÇÃO AO CICLO DE VIDA:

Modelo de ciclo de vida clássico, em particular na fase de análise, podendo, entretanto, ser aplicado durante todo o processo de desenvolvimento.

ADEQUAÇÃO AO TIPO DE PROCESSAMENTO:

Processamento seqüencial.

ADEQUAÇÃO AO DOMÍNIO DA APLICAÇÃO:

Aplicações que requerem um tratamento formal, destacando-se o seu uso na área industrial.

ADEQUAÇÃO AO NÍVEL DE COMPLEXIDADE:

Desenvolvimento de software em larga escala.

- EXPRESSIVIDADE -

ABORDAGEM DE DESENVOLVIMENTO UTILIZADA:

NA

TÉCNICAS CONSTRUTIVAS UTILIZADAS:

Técnica "top-down", pois o modelo abstrato do sistema é detalhado até chegar a um nível implementável.

TIPO DE INSTRUMENTOS NOTACIONAIS:

Utiliza como instrumento a linguagem Meta-IV que pode ser classificada como:

- Instrumento para especificação textual de funções;
- Instrumento para modelagem de estrutura de dados.

FORMA DE EXPRESSÃO DOS INSTRUMENTOS NOTACIONAIS:

A linguagem Meta-IV pode ser classificada como linguagem natural acrescida de simbologia.

GRAU DE FORMALIDADE DOS INSTRUMENTOS NOTACIONAIS:

Formal.

- FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO -

DOCUMENTAÇÃO PARA CONSTRUÇÃO:

(COHEN, 1986)

EXIGÊNCIA DE FORMAÇÃO PARA CONSTRUÇÃO:

Conceitos sobre sistemas formais e lógica dos predicados.

EXISTÊNCIA DE INSTRUMENTOS COGNITIVOS:

NP

- FACILIDADE DE USO -

EXISTÊNCIA DE APOIO AUTOMATIZADO:

SD

AUTOMATIZABILIDADE:

A linguagem Meta-IV pode ser automatizada.

EXTENSIBILIDADE:

O grau de formalidade da linguagem Meta-IV e a ausência de apoio automatizado dificultam as alterações no produto.

- UTILIZABILIDADE PARA AVALIAÇÃO

- VERIFICABILIDADE -

EXISTÊNCIA DE NORMAS DE QUALIDADE:

A cada etapa do desenvolvimento são realizadas verificações por meio de regras de prova p/ avaliar se estruturas de dados e operações correspondem àquelas definidas no nível anterior.

EXISTÊNCIA DE FERRAMENTAS PARA VERIFICAÇÃO:

NP

- VALIDABILIDADE -

COMPREENSIVIDADE:

A compreensão dos modelos gerados é dificultada pelo grau de formalidade da linguagem utilizada pelo método.

EXISTÊNCIA DE FERRAMENTAS PARA VALIDAÇÃO:

NP

V.3 COMPARAÇÃO DOS MÉTODOS AVALIADOS

Algumas comparações entre os métodos avaliados na seção anterior são apresentadas nesta seção sob a forma de tabelas. Essas comparações foram realizadas com o objetivo de fornecer subsídios para a seleção de métodos, no contexto do Meta-Ambiente da Estação TABA. Assim sendo, foram feitas as seguintes comparações:

- tipo de processamento x ciclo de vida (tabela V.3.1);
- abordagem de desenvolvimento x ciclo de vida (tabela V.3.2);
- abordagem de desenvolvimento x técnica construtiva (tabela V.3.3);
- métodos x instrumentos notacionais x automatização (tabela V.3.4).

Na tabela V.3.4 os instrumentos notacionais são referenciados através das seguintes abreviaturas:

MEF - instrumento para modelagem da estrutura de funções;
ETF - instrumento para especificação textual de funções;
EDF - instrumento para especificação detalhada de funções;
MFD - instrumento para modelagem de fluxo de dados;
MFC - instrumento para modelagem de fluxo de controle;
MED - instrumento para modelagem de estrutura de dados;

MRD - instrumento para modelagem de relacionamentos entre dados;

DE - instrumento para definição de elementos do modelo;

ME - instrumento para modelagem de estados;

MO - instrumento para modelagem de objetos;

MCS - instrumento para modelagem de concorrência e sincronização;

DBD - instrumento para definição de banco de dados;

EC - instrumento para elaboração de código;

RC - instrumento para representação do conhecimento;

MD - instrumento para modelagem de diálogo.

TIPO PROCESS/ CICLO DE VIDA	CLÁSSICO ANÁLISE	CLÁSSICO PROJETO	PROTÓTIPO	SÍNTESE	REUTILIZ
SEQUENCIAL	ESA JSD OOA OOS OOSA SADT SSA VDM	MBSD SD TDPD	SBRE TAGS		OOA OOS
"ON LINE"	ESA OOA	MBSD	BOP PUI RIPL SBRE USE		OOA
TEMPO REAL	DARTS JSD OOA		SBRE SREM SARS TAGS		OOA

Tabela V.3.1: Tabela de Comparação de Tipo de Processamento x Ciclo de Vida

ABORD.DESENV/ CICLO DE VIDA	CLÁSSICO ANÁLISE	CLÁSSICO PROJETO	PROTÓTIPO	SÍNTESE	REUTILIZ
DECOMPOSIÇÃO FUNCIONAL	DARTS	SD TDPD	KBSDLC SBRE SREM SARS TAGS		
FLUXO DE DADOS	DARTS ESA OOS OOSA SADT SSA		KBSDM USE		OOS
ESTRUTURA DE DADOS	JSD SADT	MBSD			
MODELAGEM DA INFORMAÇÃO	ERM ESA OOS OOSA		BOP USE		OOS
ORIENTAÇÃO A OBJETO	OOA OOS				OOA OOS
BASEADA EM CONHECIMENTO			KBSDLC KBSDM		

Tabela V.3.2: Tabela de Comparação de Abordagem de Desenvolvimento x Ciclo de Vida

AB.DES/TECN.	TOP-DOWN	BOTTOM-UP	OUTS-IN	OUTS-OUT	MOST-CRITC
DECOMPOSIÇÃO FUNCIONAL	DARTS KBSDLC SBRE SREM SD SARS TAGS TDPD				
FLUXO DE DADOS	DARTS ESA KBSDM OOS OOSA SADT SSA USE				
ESTRUTURA DE DADOS	JSD SADT		MBSD		JSD
MODELAGEM DA INFORMAÇÃO			BOP USE		BOP ERM ESA OOS OOSA
ORIENTAÇÃO A OBJETO	OOA OOS	OOA			OOS
BASEADA EM CONHECIMENTO	KBSDLC KBSDM				

Tabela V.3.3: Tabela de Comparação de Abordagem de Desenvolvimento x Técnica Construtiva

MT/IT	MEF	ETF	EDF	MFD	MFC	MED	MRD	DE	ME	MO	MCS	DBD	EC	RC	MD
BOP								X					X		
DASTS	*			*				X	*		X				
ERM							X					X			
ESA		*	*	*		*	*	*							
JSD	*	*				*									
KBDLC														*	
KBSDM				*				*						*	
MBSD	*	*			*	*									
OOA		*	*							X					
OOS				*			X	*		X					
OOSA		*	*	*			*		*						
PUI															X
RIPL									X						X
SBRE															
SREM					X						X				
SADT				X		X									
SD	X														
SSA		X	*	X		*	X	X	*						
SARS		X							X		X				
TAGS	X				X	X					X				
TDPD	*														
USE		X		X			X	X	X			X	X		X
VDM		*				*									

X - instrumentos automatizados

* - instrumentos não automatizados

Tabela V.3.4: Tabela de Comparação de Métodos x Instrumentos Notacionais x Automatização

V.4 CONCLUSÃO

Neste capítulo foi feita uma avaliação dos métodos descritos no Capítulo VI, considerando-se os atributos de qualidade descritos no Capítulo III. A partir dessa avaliação, foram feitas algumas comparações entre os métodos tendo como objetivo obter subsídios para a seleção de métodos no contexto do Projeto TABA.

CAPÍTULO VI

CONCLUSÃO

O presente trabalho teve como objetivo avaliar métodos para o desenvolvimento de software, visando identificar critérios que permitissem a seleção de métodos mais adequados para uma determinada aplicação.

O trabalho está inserido no projeto TABA e os seus resultados fornecem subsídios para a construção da base de conhecimentos da Estação TABA, no que se refere a métodos para o desenvolvimento de software.

Para atingir os objetivos propostos foi realizada uma pesquisa bibliográfica sobre características de métodos. A partir dessas características foi proposta uma estrutura para avaliação, com base no Método para Avaliação da Qualidade de Software proposto em (ROCHA, 1983).

Foram estudados uma série de métodos, sendo selecionados um conjunto de vinte e três para serem avaliados. Os métodos foram escolhidos buscando atender a requisitos para a construção da Base de Conhecimentos da Estação TABA e procurando dar uma visão geral das categorias de métodos existentes.

O número de métodos existentes é muito grande e a tarefa de avaliar/selecionar métodos não é trivial. É importante, portanto, a existência de propostas que auxiliem nesses processos.

Como continuação deste trabalho prevê-se o estudo de outros métodos e sua conseqüente avaliação para inclusão na base de conhecimentos da Estação TABA. Outra possibilidade de continuação do trabalho é a realização de uma pesquisa de campo, entre profissionais atuantes no desenvolvimento de software, com o objetivo de adquirir conhecimentos sobre sua experiência no uso de métodos.

REFERÊNCIAS BIBLIOGRÁFICAS

AARAM, Jarle (1984); "The BOP Prototyping Concept"; **In: Approaches to Prototyping**; Germany; Springer-Verlag; 1984.

AGUIAR, Teresa; BLASCHEK, José; NOGUEIRA, Durval e ROCHA, Ana Regina (1987); "Ferramentas Automatizadas para Apoiar as Fases de Análise e Projeto"; **In: XX Congresso Nacional de Informática**; São Paulo, vol. 2, pp. 1064-1070, 1987.

AGUIAR, Teresa (1986); **Ferramentas para Apoio à Análise Estruturada**; Tese de Mestrado; IME; Rio de Janeiro; 1986.

AGUIAR, Teresa (1989); **Protótipo do Especificador de Ambientes da Estação TABA**; Relatório Técnico ES-208/89; COPPE/UFRJ; Rio de Janeiro; Julho 1989.

ALFORD, Mack (1977); **A Requirements Engineering Methodology for Real-Time Processing Requirements**; IEEE Transactions on Software Engineering; vol. SE-3, no. 1, pp. 60-69, Janeiro 1977.

ALFORD, Mack (1980); **Software Requirements Engineering Methodology (SREM) at the Age**

of Four; Computer Software and Applications Conference; pp. 866-874, Outubro 1980.

ANGULO, Edith (1988); **Dicionário de Dados: Uma Ferramenta Automatizada de Apoio ao Método DARTS**; Tese de Mestrado, COPPE/UFRJ; Rio de Janeiro; 1988.

ARARIBÓIA, G. (1988); **Inteligência Artificial: Um Curso Prático**; Rio de Janeiro; Editora LTC; 1988.

ARTHUR, Lowell (1985); **Measuring Programmer Productivity and Software Quality**; New York; John Wiley & Sons; 1985.

BAILIN, Sidney (1989); **An Object-Oriented Requirements Specification Method**; Communications of the ACM; vol. 32, no. 5, pp. 608-623, Maio 1989.

BLASCHEK, José (1987); **Dicionário de Dados para a Análise Estruturada de Sistemas**; Tese de Mestrado; IME; Rio de Janeiro; 1987.

BOEHM, Barry (1980); **Software Engineering - AS IT IS**; Academic Press; 1980.

BOEHM, Berry (1981); **Software Engineering Economics**; Englewood Cliffs, New Jersey; Prentice-Hall, Inc.; 1981.

BOLDYREFF, Cornelia (1989); **Reuse, Software Concepts, Descriptive Methods and the Practitioner Project**; ACM SIGSOFT Software Engineering Notes; vol. 14, no. 2, pp. 25-31, Abril 1989.

BOOCH, Grady (1983); **Software Engineering with Ada**; 1st. ed. Benjamin/Cummings; Menlo Park, Califórnia; 1983.

BOOCH, Grady (1986); **Object-Oriented Development**; IEEE Transaction on Software Engineering; vol. SE-12, no. 2, pp. 211-221, Fevereiro 1986.

CAMERON, John (1983); **Tutorial - JSP & JSD: The Jackson Approach to Software Development**; IEEE Computer Society; New York; The Institute of Electrical and Electronics Engineering, Inc.; 1983

CAVALCANTE, Sueli (1988); **Uma Ferramenta Automatizada de Apoio ao Método DARTS**; Tese de Mestrado; COPPE/UFRJ; Rio de Janeiro; 1988.

CHARETTE, Robert (1986); **Software Engineering Environments: Concepts and Technology**; New York; McGraw-Hill Inc.; 1986.

CHEN, Peter (1976); **The Entity-Relationship Model: Towards a Unified View of Data**; ACM

Transaction on Database Systems; vol. 1, no. 1, pp. 9-36, Março 1976.

CHEN, Peter (1977); **The Entity-Relationship Approach to Logical Data Base Design**; Information Sciences; Massachusetts; 1977.

CHEN, Peter (1990); **Gerenciando Banco de Dados: A Abordagem Entidade-Relacionamento para Projeto Lógico**; São Paulo; Editora McGraw-Hill, Ltda. e Newstec Editora Ltda.; 1990.

CHORAFAS, Dimitris (1988); **Sistemas Especialistas: Aplicações Comerciais**; Rio de Janeiro; McGraw Hill Ltda.; 1988.

CHRISTENSEN, Niels e KREPLIN, Klaus-Dieter (1984); "Prototyping of User Interfaces"; **In: Approaches to Prototyping**; Germany; Springer-Verlag; 1984.

CLUNIE, Clifton (1987); **Verificação e Validação de Software na Fase de Especificação de Requisitos**; Tese de Mestrado; COPPE/UFRJ; Rio de Janeiro; 1987.

COAD, Peter e YOURDON, Edward (1990); **Object-Oriented Analysis**; Englewood Cliffs; New Jersey; Prentice-Hall, Inc.; 1990.

COCHRAN, Duane e STOCKER, Frederick (1985); **RIPL: An Environment for Rapid Prototyping with**

Intelligent Support; ACM SIGCHI Bulletin; vol. 17, no. 2, pp. 29-35, Outubro 1985.

COHEN, Bernard; HARWOOD, William e JACKSON, Melvyn (1986); **The Specification of Complex Systems**; Great Britain; Addison-Wesley Publishing Company, Inc.; 1986.

DAVIS, Carl (1977); **The Software Development System**; IEEE Transactions on Software Engineering; vol. SE-3, no. 1, pp. 69-84, Janeiro 1977.

DAVIS, Margaret e ADDLEMAN, David (1986); **A Practical Approach to Specification Technology Selection**; The Journal of Systems and Software; no. 6, pp. 285-294, 1986.

DAVIS, Alan; BERSOFF, Edward H. e COMER, Edward R. (1988); **A Strategy for Comparing Alternative Software Development Life Cycle Models**; IEEE Transactions on Software Engineering; vol. 14, no. 10, pp. 1453-1460, Outubro 1988.

DAVIS, Alan (1990); **Software Requirements: Analysis and Specification**; Englewood Cliffs, New Jersey; Prentice-Hall, Inc.; 1990.

DEMARCO, Tom (1978); **Structured Analysis and System Specification**; New York; Yourdon Pess; 1978.

DEMARCO, Tom (1984); "A Meta-Methodology for Systems Development", In: **International Workshop on Models and Languages for Software Specification and Design**; Orlando, Florida; Março de 1984.

DART, Susan (1987) et alli; **Software Development Environments**; Computer; pp. 18-28; Novembro 1987.

DEREMER, Frank e KRON, Hans (1980); "Programming-in-the-Large versus Programming-in-the-Small", In: **Tutorial on Software Design Techniques**; Freeman, P. e Wasserman A. (eds.); pp. 237-243, Abril 1980.

DICKOVER, M.; MCGOWAN, C. e ROSS, D. (1978); "Software Design Using SADT"; In: **Structured Analysis and Design**; vol. II, pp. 101-114, 1978.

EPPLE, W. (1983); "SARS - A System for Application oriented Requirements Specification"; In: **Workshop on Real-Time Programming**; Hatfield, U.K.; Março 1983.

FAIRLEY, Richard (1986); **Software Engineering Concepts**; McGraw-Hill, Inc.; 1986.

FERNANDES, Antônio e LAENDER, Alberto (1989); **MER+: Uma Extensão do Modelo de Entidades e Relacionamentos para Projeto Conceitual de Banco de Dados**; Revista Brasileira de Computação; Rio de Janeiro; vol. 5, no. 1, pp. 5-14, Julho/Setembro 1989.

FERREIRA, Rogério (1987); **A Relação entre a Natureza da Aplicação e os Fatores de Qualidade do Software**; Tese de Mestrado; IME; Rio de Janeiro; 1987.

FISHER, Alan S. (1990); **CASE: Utilização de Ferramentas para Desenvolvimento de Software**; Rio de Janeiro; Editora Campus Ltda.; 1990.

FLOYD, Christiane (1984); "A Systematic Look at Prototyping"; **In: Approaches to Prototyping**; Germany; Springer-Verlag; 1984.

FLOYD, Christiane (1986); "A Comparative Evaluation of System Development Methods"; **In: Information Systems Design Methodologies: Improving Practice**; Olle, T., Sol, H. e Verrijn-Stuart, A. (eds.); North-Holland; 1986.

FREEMAN, Peter (1980); "The Nature of Design"; **In: Tutorial on Software Design Techniques**;

Freeman, P. e Wasserman, A. (eds.); IEEE Computer Society; 1980.

GANE, Chris e SARSON, Trish (1979), **Structured Systems Analysis: Tools and Techniques**; Englewood Cliffs, New Jersey; Prentice-Hall, Inc.; 1979.

GANE, Chris (1988); **Desenvolvimento Rápido de Sistemas**; Rio de Janeiro; LTC - Livros Técnicos e Científicos Editora Ltda.; 1988.

GANE, Chris (1990); **CASE: O Relatório Gane**; Rio de Janeiro; LTC - Livros Técnicos e Científicos Editora Ltda.; 1990.

GHEZZI, Carlo e JAZAYERI, Mehdi (1985); **Conceitos de Linguagem de Programação**; Editora Campus Ltda.; 1985.

GOMAA, Hassen (1984); **A Software Design Method for Real-Time Systems**; Communications of the ACM; vol. 27, no. 9, pp. 938-949, Setembro 1984.

GOMAA, Hassen (1986); **Software Development of Real-Time Systems**; Communications of the ACM; vol. 29, 1986.

GOMAA, Hassen (1987); **Using the DARTS Software Design Method for Real-Time Systems**; Proceedings of the 12th Structured Methods Conference; Chicago; Agosto 1987.

GOMAA, Hassen (1989); **A Software Design Method for Distributed Real-Time Applications**; The Journal of Systems and Software; no. 9, pp. 81-94, 1989.

HAGEMANN, M. (1987); **Formal Requirements Specification of Process Control Systems**; ACM SIGSOFT Software Engineering Notes; vol. 12, no. 4, pp. 36-42, Outubro 1987.

HAMILTON, M. e ZELDIN, S. (1976); **"Higher Order Software - A Methodology for Defining Software"**; IEEE Transactions on Software Engineering; vol. SE-2, no. 1, pp. 9-32, Março 1976.

HOARE, Charles (1972); **Proof of correctness of data representations"**; Acta Information; vol. 1, no. 3, pp. 271-281, 1972.

HOLBROOK, Hilliard (1990); **A Scenario-Based Methodology Conducting Requirements Elicitation**; ACM SIGSOFT Software Engineering Notes; vol. 15, no. 1, pp. 95-104, Janeiro 1990.

JACKSON, Michael (1975); **Principles of Program Design**; Academic Press; 1975.

JACKSON, Michael (1983); **System Development**; Prentice-Hall; 1983.

JONES, Capers (1979); **An Overview of Programming Architectures and Design Languages**; Relatório Técnico; IBM General Products Division; Santa Tereza Laboratory; 1979.

JONES, Capers (1981); **Software Language Categories**; Relatório Técnico; ITT Programming Technology Center; Stratford; Connecticut; 1981.

JORDANGER, I. (1981a); **BOP Prototyping User's Guide**; Trondheim: SINTEF-report STF17 181012; Fevereiro 1981.

JORDANGER, I. (1981b); **BOP Prototyping, Expert's Guide**; Trondheim: SINTEF-report STF17 A81013; Fevereiro 1981.

JORDANGER, I. (1981c); **Prototyping of EDP Systems, The BOP Concept**; Trondheim: SINTEF-report STF17 A81075; Agosto 1981.

KELLER, Robert (1987); **Expert System Technology: Development and Application**; Englewood Cliffs, New Jersey; Prentice-Hall, Inc.; 1987.

KELLY, John (1987); **A Comparison of Four Design Methods for Real-Time Systems**; Proceedings of the Ninth International Conference on Software Engineering; Monterey, California; pp 238-252, Março 1987.

KERSTEN, M. e WASSERMAN, Anthony (1981); **The Architecture of the PLAIN Data Base Handler**; Software Practice and Experience; vol. 11, no. 2, pp. 175-186, Fevereiro 1981.

LEITE, Júlio (1983); "SADT Datagrams, a Powerful Tool for Requirements Analysis"; **In: Anais do XVI Congresso Nacional de Informática**; São Paulo; SUCESU; Outubro 1983.

LEVERSON, N. (1983); **Applying behaviorial abstraction to information system design and integrity**; Ph.D. Dissertation; University California; Los Angeles, 1980.

LUCENA, Carlos (1987); **Inteligência Artificial e Engenharia de Software**; Rio de Janeiro; Jorge Zahar Editor Ltda.; 1987.

MARTIN, Jaimes (1977); **Computer Data-Base Organization**; Prentice-Hall; 1977.

MARTIN, Jaimes e CARMA, McClure (1985); **Diagramming Techniques for Analysts and Programmers**; Englewood Cliffs, New Jersey; Prentice-Hall, Inc.; 1985.

MATHUR, Raghubin (1987); **Methodology for Business System Development**; IEEE Transactions on Software Engineering; vol. SE-13, no. 5, pp. 593-601, Maio 1987.

MATTOSO, Adriana (1990); **TABA_OBJ: Um Ambiente de Desenvolvimento de Software com Orientação a Objetos**; Tese de Mestrado; COPPE/UFRJ; Rio de Janeiro; Agosto 1990.

MCCLURE, Carma (1989); **The Case Experience**; BYTE; pp. 235-246; Abril 1989.

MCMENAMIN, Stephen e PALMER, John (1991); **Análise Essencial de Sistemas**; São Paulo; Editora McGraw-Hill, Ltda. e Makron Books Ltda; 1991.

MENDES, Sueli e AGUIAR, Teresa (1989); **Métodos para Espeficação de Sistemas**; São Paulo; Editora Edgard Blucher Ltda.; 1989.

MILLS, C. e WASSERMAN, Anthony (1984); "A Transition Diagram Editor"; **In: Proceedings 1984 Summer Usenix Meeting**; pp. 287-296; Junho 1984.

MYERS, Glenford (1978); **Composite/Structured Design**; New York; Van Nostrand Reinhold; 1978.

NOGUEIRA, Durval (1988); **Ferramentas Automatizadas para Apoio ao Projeto Estruturado**; Tese de Mestrado; COPPE/UFRJ; Rio de Janeiro; Abril 1988.

ORR, Ken; GANE, Chris; YOURDON, Edward; CHEN, Peter e CONSTANTINE, Larry (1989);

Methodology: The Experts Speak; BYTE; pp. 221-233, Abril 1989.

PAGE-JONES, Meiler (1980); **The Practical Guide to Structured Systems Design**; New York; Yourdon Press, Inc; 1980.

PASSOS, Cristina (1991); **Avaliação da Estrutura Modular de Programas na Fase de Projeto**; Tese de Mestrado; COPPE/UFRJ; Rio de Janeiro; Maio 1988.

PRESSMAN, Roger (1988); **Software Engineering: A Practitioner's Approach**; Singapore; McGraw-Hill International Editions; 1988.

PRICE, Roberto (1987); "Ferramentas para Desenvolvimento de Software"; **In: XX Congresso Nacional de Informática**; São Paulo; vol. 2, pp. 1071-1077, 1987.

RIBEIRO, Carlos (1989); **Um Sistema de Apoio à Avaliação de Custos de Software**; Tese de Mestrado; COPPE/UFRJ; Rio de Janeiro; Dezembro 1988.

RICH, Elaine (1983); **Artificial Intelligence**; Singapore; McGraw-Hill International Editions; 1983.

ROCHA, Ana Regina (1981); "Um Estudo Comparativo de Métodos de Especificação de Sistemas Automatizados"; **In: Primeira Conferência**

Internacional en Ciencia de la Computacion; Pontifícia Universidad Catolica; Chile; Agosto 1981.

ROCHA, Ana Regina (1983); **Um Modelo para Avaliação da Qualidade de Especificações**; Tese de Doutorado; PUC; Rio de Janeiro, 1983.

ROCHA, Ana Regina (1985); "Controle da Qualidade de Software: Uma Necessidade Atual"; **In: VIII Congresso Nacional de Informática**; São Paulo; pp. 639-642, 1985.

ROCHA, Ana Regina (1987); **Análise e Projeto Estruturado de Sistemas**; Rio de Janeiro; Editora Campus, Ltda.; 1987.

ROCHA, Ana Regina; AGUIAR, Teresa e BLASCHEK, José (1987a); **Ambientes para Desenvolvimento de Software: Definição de Termos**; Relatório Técnico ES-137/87; COPPE/UFRJ; 1987.

ROCHA, Ana Regina e AGUIAR Teresa (1988); "Ambientes de Desenvolvimento de Software"; **In: Ambientes de Desenvolvimento de Software e o Projeto TABA**; Rocha, Ana Regina e Souza, Jano (eds.); Relatório Técnico ES-179/88; COPPE/UFRJ; Rio de Janeiro; Dezembro 1988.

ROCHA, Ana Regina; AGUIAR, Teresa; SOUZA, Jano; D'IPOLITTO, Cláudio (1989); **O Meta-Ambiente da Estação TABA**; Relatório Técnico; COPPE/UFRJ; Rio de Janeiro; Março 1989.

ROMAN, Gruia-Catalin (1985); **A Taxonomy of Current Issues in Requirements Engineering**; Computer; pp. 14-21, Abril, 1985.

ROTENSTREICH, Shmuel e HOWDEN, William (1986); **Two-Dimensional Program Design**; IEEE Transactions on Software Engineering; vol. SE-12, no. 3, pp. 377-384,

ROSS, Douglas (1977); **Structured Analysis (SA): A Language for Communicating Ideas**; IEEE Transactions on Software Engineering; vol. SE-3, no. 1, pp. 16-34, Janeiro 1977.

ROSS, Douglas e SCHOMAN, Kenneth (1977a); **Structured Analysis for Requirements Definition**; IEEE Transactions on Software Engineering; vol. SE-3, no. 1, pp. 6-15, Janeiro 1977.

ROSS, Douglas (1980); "Removing the Limitations of Natural Language"; **In: Software Engineering**; New York; Academic Press; pp. 149-179, 1980.

ROSS, Douglas (1985); **Applications and Extentions of SADT**; IEEE Computer; vol. 18, no. 4, pp. 25-34, Abril 1985.

SIEVERT, Gene e MIZELL, Terrence (1985); **Specification-Based Software Engineering with TAGS**; Computer; pp. 56-65, Abril 1985.

SHAW, M. (1981); **ALPHARD: Form and Content.**; New York; Springer-Verlag; 1981.

SHLAER, Sally e MELLOR, Stephnen (1988); **Object-Oriented Systems Analysis**; Englewood Cliffs, New Jersey; Prentice-Hall, Inc.

SMITH, J. e SMITH, D. (1983); "Conceptual Database Design"; **In: Software Design Techniques**; Los Alamitos; IEEE Computer Society; 1983.

SOUZA, Jano (1989); **Projeto TABA: Proposta de Desenvolvimento, Informações Complementares e Orçamento**; Relatório Técnico; COPPE/UFRJ; Rio de Janeiro, Agosto de 1989.

STARK, M. e SEIDEWITZ, E. (1987); "Towards a general object-oriented Ada lifecycle"; **In: Proceedings of the Joint 4th Washington Area Ada Technology**; ACM NASA Goddard Space Center, et al.; Washington D.C.; 1987.

STEVENS, Wayne; MYERS, Glenford e CONSTANTINE, Larry (1974); **Structured Design**; IBM Systems Journal; vol. 13, no. 2, pp. 115-139, 1974.

STEVENS, Wayne (1988); **Projeto Estruturado de Sistemas**; Rio de Janeiro; Editora Campus Ltda.; 1988.

TRAVASSOS, Guilherme (1990); **FEGRES: FErramentas GRáficas para Engenharia de Software**; Tese de Mestrado, COPPE/UFRJ; Rio de Janeiro; Janeiro 1990.

TSICHRITZIS, D. e LOCHOVSKY, F. (1982); **Data Models**; Englewood Cliffs, New Jersey; Prentice-Hall, Inc.; 1982.

WASSERMAN, Anthony et. al. (1981); **Revised Report on the Programming Language PLAIN**; ACM SIGPLAN Notices; vol. 16, no. 5, pp. 59-80, Maio 1981.

WASSERMAN, Anthony (1982); "The User Software Engineering Methodology: An Overview; **In: Information Systems Design Methodologies: A Comparative Review**; Olle, T., Sol, H. e Verrijn-Stuart, A. (editores); North-Holland Publishing Company; pp. 591-628.

WASSERMAN, Anthony e FREEMAN, Peter (1983); **Ada Methodologies: Concepts and Requirements**; ACM SIGSOFT Software Engineering Notes; vol. 8, no. 1, pp. 33-50, Janeiro 1983.

WASSERMAN, Anthony (1985); **Extending State Transition Diagrams for the Specification of Human**

Computer Interaction; IEEE Transactions on Software Engineering; vol.SE-11, Agosto 1985.

WASSERMAN, Anthony; PIRCHER, Peter; SHEWMAKE, David e KERSTEN, Martin (1986); **Developing Interactive Information Systems with the User Software Engineering Methodology**; IEEE Transactions on Software Engineering; vol. SE-12, no. 2, pp. 326-345, Fevereiro 1986;

WEITZEL, John e KERSCHBERG, Larry (1989); **Developing Knowledge-Based Systems: Reorganizing the System Development Life Cycle**; Communications of the ACM; vol. 32, no. 4, pp. 482-488, Abril 1989.

WERNECK, Vera (1990); **Taxonomia de Domínios de Aplicação**; Tese de Mestrado, COPPE/UFRJ; Rio de Janeiro; Novembro 1990.

WING, J. e NIXON, M. (1989); **Extending Ina Jo with Temporal Logic**; IEEE Transactions on Software Engineering; Fevereiro 1989.

YOURDON, Edward e CONSTANTINE, Larry (1978); **Structured Design: Fundamentals of a Discipline of Computer Program and System Design**; New York; Yourdon Press; 1978.

YOURDON, Edward (1990); **Análise Estruturada Moderna**; Editora Campus Ltda.; Rio de Janeiro; 1990.