UM SISTEMA ESPECIALISTA DE SUPORTE À DECISÃO PARA PLANEJAMENTO DE AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

Teresa Cristina de Aguiar

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

Aprovada por:

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

Prof. Jano Moreira de Souza, PhD

Prof. Sueli Mendes, PhD

Prof. Doris Ferraz de Aragón, D.Sc.

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

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 1992

AGUIAR, TERESA CRISTINA DE

Um Sistema Especialista de Suporte à Decisão para Planejamento de Ambientes de Desenvolvimento de Software (Rio de Janeiro) 1992.

ix,229p,29.7cm(COPPE/UFRJ, D.Sc. Engenharia de Sistemas e Computação, 1992)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. ambientes 2. seleção de componentes 3. sistema especialista de suporte à decisão 4. ciclo de vida 5.ferramentas 6. métodos

I. COPPE/UFRJ II. Título (série)

Ao Carlos e Vinicius

AGRADECIMENTOS

À Prof[a] Ana Regina C. da Rocha, pela orientação, incentivo e apoio durante o desenvolvimento do trabalho.

Aos professores da COPPE, em especial à Prof[a] Sueli Mendes pelos ensinamentos e ao Prof. Jano de Souza por sua colaboração.

À Prof[a] Regina Pereira e à Prof[a] Dóris de Aragón por suas sugestões.

Aos colegas da COPPE pelo apoio e companheirismo. Em especial à Marta, Luis Carlos, Trotta e Claudia, sempre disponíveis e amigos.

Aos participantes do Projeto TABA, que muito contribuíram com suas sugestões e interesse. Em especial:

- à Vera por sua amizade e colaboração neste trabalho através de sua tese de Mestrado e ao Juan, sempre trazendo uma palavra amiga;

- à Emília e Edith cujas teses de Mestrado deram subsídios para o desenvolvimento deste trabalho;

- ao Luis e Alberto, do Laboratório de Computação Gráfica, pelo apoio no início da implementação;

- ao Jobson pelo interesse e apoio eficiente na construção da ferramenta.

À FAPERJ pelo apoio financeiro ao Projeto TABA.

A minha mãe por estar sempre presente e ao Carlos e Vinicius pelo amor e compreensão.

E por fim, àqueles amigos que mesmo não conhecendo o assunto deste trabalho tanto colaboraram para sua realização.

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

UM SISTEMA ESPECIALISTA DE SUPORTE À DECISÃO PARA PLANEJAMENTO DE AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

Teresa Cristina de Aguiar

Março de 1992

Orientadora: Ana Regina Cavalcanti da Rocha

Programa: Engenharia de Sistemas e Computação

Este trabalho faz parte do Projeto TABA. O TABA visa a construção de uma estação de trabalho configurável capaz de criar um ambiente adequado às necessidades de cada projeto de desenvolvimento de software. Os ambientes são criados a partir da união de componentes, que são o ciclo de vida, métodos, ferramentas e hardware.

A determinação de quais componentes devem ser escolhidos não é uma tarefa trivial, uma vez que há várias possibilidades de combinação, além de ser necessário o conhecimento sobre cada componente.

Esta tese apresenta uma solução para apoiar o planejador de ambientes na escolha dos componentes do ambiente de desenvolvimento de software adequados ao desenvolvimento de cada projeto. Foi especificado um sistema com este objetivo e desenvolvida uma ferramenta de apoio, um sistema especialista de suporte à decisão, que implementa este sistema.

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

A SYSTEM TO HELP IN THE PLANNING

OF SOFTWARE DEVELOPMENT ENVIRONMENTS

Teresa Cristina de Aguiar

March, 1992

Thesis Supervisors: Ana Regina Cavalcanti da Rocha

Department: Systems and Computer Engineering

This work is part of the TABA Project. The main purpose of this project is the construction of a configurable workstation able to create an environment tailored to meet specific demands of each software development project.

Planning a Software Development Environment (SDE) is a heuristic process and needs expert knowledge and judgement. It depends on data related to different models for the software life cycle and on methods, tools and other SDEs. Previous knowledge in developing software in this application domain and implementation of SDEs is also required.

In this work we present a solution to help the SDE planner to choose the most suitable components to develop a particular project. These components will be integrated to form a SDE.

UM SISTEMA ESPECIALISTA DE SUPORTE À DECISÃO PARA PLANEJAMENTO DE AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

I.1 OBJETIVOS DO TRABALHO

Este trabalho faz parte do Projeto TABA que visa a construção de uma estação de trabalho que atenda aos seguintes objetivos: (Rocha-90) (Rocha-91)

a) auxilie o engenheiro de software no planejamento e criação do ambiente mais adequado ao desenvolvimento de um produto específico;

b) auxilie o engenheiro de software a implementar as ferramentas necessárias ao ambiente definido em (a);

c) permita aos desenvolvedores do produto (software) o uso da estação através do ambiente especificado em (a) e gerado em (b);

d) permita a execução do software, dado que, eventualmente, o software produto poderá ser executado na própria estação para ele configurada (o que é sempre verdade pelo menos na fase de testes).

Ambiente de desenvolvimento de software (ADS) é considerado neste trabalho como um conjunto integrado de métodos, ferramentas e o hardware que suportam o

desenvolvimento de um software nas diferentes etapas de seu ciclo de vida. Consideramos então o ciclo de vida, métodos, ferramentas e o hardware como os componentes a partir dos quais o ADS é formado.

Esta estratégia de combinar componentes possibilita que os inúmeros recursos que vêm sendo projetados e utilizados para apoiar o desenvolvimento de software sejam aproveitados.

Essa vantagem se concretiza na medida em que hajam recursos adequados e suficientes para se fazer composições. E também na medida em que estejam disponíveis facilidades para que o engenheiro de software indique os componentes de acordo com as necessidades do projeto que se pretende desenvolver.

A determinação dos componentes de um ADS não é uma tarefa trivial, uma vez que há várias possibilidades de combinação, além de ser necessário o conhecimento sobre cada componente e como eles se relacionam entre si.

Esta tese apresenta uma solução que possibilita àquele que está planejando um ADS ter apoio ao tomar suas decisões. Foi especificado um sistema com este objetivo e desenvolvida a primeira versão de uma ferramenta que implementa este sistema. O produto resultante é fruto de uma série de trabalhos práticos e teóricos realizados através de teses, experimentos e monografias.

As facilidades oferecidas pelo planejador de ADSs estão relacionadas às seguintes questões, consideradas básicas:

- **A análise das características do projeto:** O primeiro passo para que se possa indicar os componentes que formarão um ADS é conhecer as características do projeto para o qual este ADS está sendo planejado. Desta forma são necessários recursos que facilitem a descrição dessas características.

- **A definição do ciclo de vida:** Como, dependendo do sistema a ser desenvolvido, tem-se um ciclo de vida particular, necessita-se de uma representação que possa expressar as particularidades de cada ciclo assim como seu relacionamento com os demais componentes do ADS.

- **O suporte à decisão:** O planejamento de um ADS não é uma tarefa trivial. É um processo heurístico, ou seja, os aspectos envolvidos na escolha de um determinado componente não são definidos explícita e claramente para todos os casos, necessitando do conhecimento de especialistas. Estes especialistas devem ter conhecimento da área de aplicação e experiência no uso de métodos e ferramentas, o que muitas vezes não pode ser encontrado em uma única pessoa. Estas questões nos levaram a pensar no sistema que auxilia ao planejamento de ADSs como um sistema de apoio à decisão.

- **A reutilização de informações:** De forma a diminuir os custos de desenvolvimento de software é essencial que durante o processo de sua construção seja aproveitada a experiência obtida em outros desenvolvimentos. É essencial, portanto, a existência de recursos que possibilitem armazenar informações de modo que depois possam ser recuperadas.

A ferramenta desenvolvida para implementar o sistema de planejamento de ADSs recebeu o nome **"XAMÃ:** Planejador de ADS".

Xamã é um termo da língua tupi e é sinônimo de pajé, o chefe espiritual dos indígenas, misto de sacerdote, profeta e médico-feiticeiro. A idéia é que da mesma forma que o xamã utiliza seu conhecimento para resolver os problemas da tribo, esta ferramenta possa também com o auxílio de suas bases de dados e conhecimento, apoiar o planejador de ambientes a tomar decisões.

Da mesma forma como qualquer apoio espiritual nem sempre obtém os resultados mais desejados pela tribo, o apoio dado pelas bases de dados e de conhecimento também pode não ser o ideal. A questão é que para o tipo de problema abordado nesta tese não existe para cada caso uma solução única que possa ser considerada como ideal. Assim, as bases de dados e de conhecimento oferecem apoio e não a palavra final sobre uma determinada questão.

"XAMÃ: Planejador de ADS", como um subsistema do TABA, foi construído de forma a poder ser integrado aos demais subsistemas com os quais tem interface.

I.2 CONTEÚDO DO TRABALHO

Este trabalho está organizado da seguinte forma:

No capítulo II são apresentadas taxonomias e exemplos de ambientes, que são classificados de acordo com as taxonomias descritas. É apresentada também a visão particular que o Projeto TABA tem de ADS. Para concluir, o TABA é caracterizado de acordo com as taxonomias apresentadas e os requisitos desejáveis dos ambientes do futuro.

O capítulo III aborda duas questões relacionadas à seleção de componentes de ambientes:

- a avaliação de componentes e ambientes de forma a se ter subsídios para a seleção;

- os métodos e ferramentas que vêm sendo desenvolvidos de forma a serem utilizados no processo de seleção.

Por fim o capítulo III apresenta uma comparação entre o que foi pesquisado e a proposta desta tese.

No capítulo IV é descrita a evolução dos sistemas de suporte à decisão e dos sistemas especialistas e são apresentados exemplos de sistemas especialistas de suporte à decisão. Por fim conclui-se apresentando as motivações para dar uma solução ao problema desta tese através do desenvolvimento de um sistema com características de um sistema especialista de suporte à decisão.

O capítulo V apresenta o sistema especialista de suporte à decisão para planejamento de ADSs proposto por esta tese. No item V.1 o TABA é apresentado de forma a se posicionar este trabalho em seu contexto. O item V.2 contém a descrição do sistema proposto. E o item V.3 descreve os componentes da ferramenta implementada nesta tese.

E, por fim, o capítulo VI conclui apresentando o que vem sendo realizado no sentido de continuar o desenvolvimento deste trabalho e a contribuição desta tese para a área de Engenharia de Software.

CAPÍTULO II

AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

(Dart-87) define "ambiente" como o conjunto de hardware e ferramentas de software que o desenvolvedor utiliza para construir os sistemas de software. Com este mesmo significado é utilizado também o termo CASE (Computer Aided Software Engineering).

No início da computação a construção de software era essencialmente um problema de programação. Assim, a linguagem de programação era a ferramenta principal utilizada pelos desenvolvedores de software. Associada a ela existiam outras ferramentas usadas para facilitar a programação. Ambientes desse tipo são denominados "ambientes de programação". (Dart-87)

Em meados dos anos 80 começaram a surgir os chamados "ambientes de desenvolvimento de software", que se caracterizam por suportar não só as atividades das fases de nível mais baixo do ciclo de vida, como codificação e testes, mas também as demais, como a especificação de requisitos.

Nesta seção são apresentadas taxonomias e exemplos de ambientes, que são classificados de acordo com as taxonomias descritas. É apresentada também a visão particular que o Projeto TABA tem de ambientes e para concluir a Estação de Trabalho TABA é caracterizada de

acordo com as taxonomias apresentadas e os requisitos desejáveis dos ambientes do futuro.

II.1 CLASSES

Ambientes podem ser vistos de diferentes ângulos. A análise depende do aspecto que se deseja enfocar. A seguir são apresentadas três taxonomias que apresentam ambientes sob diferentes pontos de vista: a tecnologia adotada, a escala do problema a ser tratado e as diferentes possibilidades do ambiente gerar outros ambientes.

II.1.1 CLASSIFICAÇÃO DE (Penedo-88)

(Penedo-88) apresenta quatro classes de ambientes que se distinguem pela tecnologia que adotam.

a) Ambientes baseados na tecnologia de sistemas operacionais

Nesses ambientes as ferramentas são invocadas como programas, os dados são organizados e acessados através de um sistema de arquivos e o controle do usuário é realizado através de linguagem de comando. Exemplo é o UNIX.

Como extensões desses sistemas encontram-se os ambientes dirigidos à sintaxe e os ambientes baseados em linguagem.

A motivação inicial dos ambientes dirigidos à sintaxe foi dar ao usuário uma ferramenta interativa, um editor dirigido à sintaxe. Esta capacidade foi estendida para prover ambientes de programação com suporte à análise semântica interativa, execução de programa e depuração. O termo dirigido à sintaxe tem sido gradualmente substituído por orientado à estrutura. Exemplo é o Projeto Gandalf (Haberman-86).

Ambientes baseados em linguagens são construídos em torno de uma linguagem, provendo um conjunto de ferramentas adequado àquela linguagem. São altamente interativos e oferecem um suporte limitado para programação em larga escala (Dart-87). Exemplos são o Interlisp, para Lisp, o Smalltalk para Smalltalk, o Cedar para Mesa e o Rational para Ada.

b) Ambientes baseados em sistemas de gerência de dados

Nestes ambientes dados são estruturados e organizados através do uso de esquemas de bases de dados, ferramentas tomam o papel de transformadores de dados e o controle de dados é realizado através do fluxo de dados entre as ferramentas com a base de dados servindo como um repositório centralizado e lógico de informações. O controle do usuário pode ser realizado através de uma linguagem de comando ou de uma interface mais complexa. A base de dados é utilizada para definir produtos de trabalho

como por exemplo a especificação de requisitos e projeto e dados de teste. Exemplos são os ambientes Aspect Software Engineering (Hitchcock-86) e o Project Master Data Base (Penedo-86).

c) Ambientes baseados na tecnologia de sistemas especialistas

Nestes ambientes as bases de conhecimento são usadas para armazenar informações sobre os produtos que estão sendo gerados e informações de como gerar esses produtos. Utilizam técnicas da inteligência artificial. Exemplo é o projeto SAFE (Balzer-85).

d) Ambientes baseados em tecnologias orientadas a objetos

Esta tecnologia pode ser caracterizada posicionando-a entre as tecnologias baseadas em sistemas de gerência de dados e tecnologias de sistemas especialistas. Bases de objeto armazenam informações numa forma mais estruturada e mais compatível com os mecanismos de estrutura de dados das modernas linguagens de alto nível. Exemplo é o Arcadia Consortium (Taylor-86).

A partir da taxonomia apresentada acima pode-se fazer a seguinte classificação (Aguiar-90): as duas primeiras tecnologias podem ser consideradas convencionais,

enquanto a terceira utiliza Inteligência Artificial e a quarta apresenta aspectos de ambos os tipos.

Os ambientes convencionais se caracterizam por tratar de problemas que possuem soluções bem definidas e organizadas. Já os que utilizam inteligência artificial são aqueles que tratam de problemas que não têm um algoritmo sistemático e direto.

Problemas em que cada ação adotada abre novas possibilidades de ação geram árvores. Se uma árvore é muito grande nenhum computador pode explorar as consequências de todas as possibilidades. Há necessidade, então, de regras heurísticas: os caminhos mais promissores são analisados e os demais são deixados de lado. Nesses casos é utilizada a Inteligência Artificial.

A programação convencional se caracteriza pela existência de um algoritmo sistemático e direto, pelos dados e controles estarem interligados, por haver dificuldades de modificação e pela necessidade de respostas sempre corretas e ótimas.

Já na programação com Inteligência Artificial tem-se a busca heurística, o controle encontra-se separado do conhecimento, há teoricamente por este motivo facilidade de modificação, são toleradas algumas respostas incorretas além de serem aceitas respostas apenas satisfatórias.

II.1.2 CLASSIFICAÇÃO DE (Perry-88)

Em (Perry-88) é apresentada uma classificação de ADSs relativa à escala do problema a ser tratado. Pode ser observado, através desta taxonomia, como vêm sendo resolvidas pelos ambientes as questões relacionadas ao gerenciamento de projetos de software.

Nesta taxonomia são distinguidas quatro classes de modelos: indivíduo, família, cidade e estado.

No modelo indivíduo estão inseridos aqueles ADSs frequentemente referenciados como ambientes de programação. Estes ambientes fornecem um conjunto mínimo de ferramentas de implementação necessárias para construir o software. São ADSs dos seguintes tipos:

. **Ambientes "toolkits":** provêm uma coleção de ferramentas como compiladores, editores, "assemblers", "linkers" e depuradores. Ex: UNIX.

. **Ambientes interpretados:** consistem de um conjunto integrado de ferramentas centrado num interpretador para uma linguagem única como Lisp ou Smalltalk.

. **Ambientes orientados à linguagem:** são uma combinação dos ambientes "toolkits" e interpretados. Provêm ferramentas para construção de programas integradas por uma estrutura

complexa - uma "decorated syntax tree". Uma política única é realizada pelas ferramentas: detecção e notificação de erros o mais cedo possível. Ex: Garden e Synthesizer Generator.

. **Ambientes transformacionais:** suportam uma linguagem de largo espectro que possui objetos e estruturas de controle que vão de um nível mais abstrato a um nível mais concreto. Ex: Ergo, PDS, Refine.

Já no modelo família estão incluídos os ADSs que possuem além das ferramentas de construção do modelo indivíduo, facilidades para suportar as interações de um pequeno grupo de programadores. A diferença entre este e o modelo indivíduo é que no indivíduo nenhuma regra é necessária porque não existe interação. No família algumas regras são necessárias para regular interações críticas entre os programadores. O ambiente provê meios de se dar a interação dos desenvolvedores, com o objetivo que informação e esforço não sejam nem perdidas nem duplicadas como resultado de atividades simultâneas de programadores. São ADSs dos seguintes tipos:

. **Ambientes "toolkit" estendidos:** aos ambientes "toolkit" são acrescentadas estruturas de controle de versões e mecanismos de controle de configuração. Ex: SCCS database e RCS database, ambos do Unix e APSE (Ambiente de suporte a programação em ADA)

. **Ambiente integrado:** é uma extensão aos ambientes orientados a linguagem. A estrutura deste modelo é a de uma base de dados de propósito geral. O suporte vai desde simples versões "backup" a versões paralelas e sequenciais. Ex: Gandalf Prototype, Smile e Toolpack.

. **Ambiente distribuído:** estende o ambiente integrado através de máquinas conectadas por uma rede local. Ex: Mercury.

. **Ambiente de administração de projeto:** provê suporte adicional para mudanças de coordenação. Ex: DSEE (estruturas e mecanismos são providos para atribuir tarefas que são compostas de subtarefas e atividades).

As classes indivíduo e família são o estado corrente da arte mas os ambientes pertencentes a elas são inadequados para construir sistemas grandes.

No modelo cidade estão incluídos os ambientes em que interagem mais de 20 pessoas. Os autores argumentam que o modelo família vem sendo usado onde se necessita do modelo cidade, não sendo apropriado para tratar problemas desta escala. Pouco trabalho vem sendo realizado em ambientes que implementam o modelo cidade, isto é, ambientes que favoreçam a cooperação entre desenvolvedores. Dois ambientes que vêm sendo desenvolvidos são o Inscape/Infuse (Perry-89) e o ISTAR (Dixon-88).

Quanto ao modelo estado, os autores não conhecem nenhum ambiente que possa ser incluído nesta classe. Os autores consideram, no entanto, que ambientes do tipo estado deverão:

- prover um modelo genérico com suas ferramentas e estruturas de suporte, de forma a poder ser usado por uma companhia em particular;
- instanciar um modelo para cada projeto, adaptando cada instância dinamicamente às necessidades particulares de cada projeto;
- gerenciar as diferenças entre as várias instâncias de forma a suportar o deslocamento entre projetos.

II.1.3 CLASSIFICAÇÃO DE (Lucena-87)

Existe um grupo de ambientes que se caracteriza por poder gerar outros ambientes. São denominados meta-ambientes.

Meta-ambientes que têm sido desenvolvidos podem englobar uma ou mais das seguintes estratégias:

a) Estrutura puramente gerativa

A arquitetura do gerador de ambientes é baseada em um editor dirigido à sintaxe, permitindo a criação de ambientes em torno de uma determinada linguagem de

programação. Exemplos são os projetos GANDALF (Habermann-86) e o MENTOR (Donzeau-84) .

b) Configuradores de ambientes

Considerando-se um ambiente para produção de software, uma configuração de software é um conjunto de componentes (base de software, interfaces, ferramentas, etc) que são vistos como uma unidade para os propósitos de identificação, coordenação ou controle. Meta-ambientes configuradores geram ambientes a partir da junção de componentes. Exemplos são os trabalhos de (Hoffnagle-85) e (Giavitto-90).

c) Meta-ambientes exploratórios

São ambientes que utilizam estratégias como a transformação de programa utilizando técnicas de inteligência artificial. Exemplo é o projeto CHI (Smith-85) (Refine).

II.2 EXEMPLOS

Nesta seção são dados exemplos de ambientes que de acordo com as taxonomias apresentadas acima podem ser classificados como:

- Ambientes que utilizam a inteligência artificial;
- Ambiente com suporte gerencial;
- Meta-ambientes configuradores e exploratórios.

II.2.1 AMBIENTES QUE UTILIZAM A INTELIGÊNCIA ARTIFICIAL

A programação automática é uma das estratégias de desenvolvimento de software em que a Inteligência Artificial vem sendo utilizada. Pesquisas em Programação Automática envolvem:

a) Síntese de programas através de exemplos;

b) Síntese de programas através de especificações formais;

c) Síntese de programas utilizando a representação do conhecimento.

Neste trabalho será enfocado somente o último item, apresentado a seguir.

Um sistema de programação automática baseado no conhecimento tem como objetivo a construção de implementações eficientes de programas a partir de especificações formais de alto nível. Um sistema que provê assistência automatizada ao desenvolvimento de programas pode fazê-lo utilizando:

- Conhecimento do processo de programação:

Este tipo de conhecimento tem relação com os objetos da ciência da computação (como os construtores de controle, arranjos e grafos) e seus algoritmos associados e relações de implementação. É o conhecimento encontrado em textos de ciência da computação, como os de (Knuth-73).

- Conhecimento do domínio da aplicação:

Este é o conhecimento sobre o mundo no qual o software vai operar. É o conhecimento que o sistema manipula para, a partir da descrição do problema (ou especificação do programa), realizar o processo que permitirá chegar ao objetivo do sistema. Este conhecimento inclui como estruturar os conceitos na área do problema e como encontrar relações entre eles, além do conhecimento sobre como obter certos resultados na área do problema.

Em (Prieto-Diaz-87) é explorada a questão da análise do domínio de forma a dar subsídios para a reutilização.

O conhecimento do domínio tem recebido também muita atenção na pesquisa de programação automática. Uma questão pesquisada é a da interação dos dois tipos de conhecimento, domínio e programação, durante a síntese de programas.

Atualmente não há nenhum sistema de programação automática de propósito geral que tenha como entrada os requisitos na linguagem do domínio do problema. Há, no entanto, dois sistemas de propósito especial que operam pelo menos perto deste nível. Cada um desses sistemas contém extensivo conhecimento do domínio: Draco (Neighbors-86) e ONIX System (Barstow-86).

A abordagem dominante atualmente para representar conhecimento de programação é usar transformação de programas. Exemplos do uso desta abordagem podem ser encontrados em (Cheatham-86), (Darlington-86). De acordo com esta idéia, uma especificação formal é compilada (manual ou automaticamente) numa implementação eficiente através da aplicação repetitiva de transformações que preservam a correção. A diferença aqui entre especificação e implementação é o grau de quanto o programa está mais direcionado ao problema ou à máquina.

A transformação pode ser categorizada por sua granularidade, que é o grau de alteração na estrutura do programa. Transformações com baixa granularidade fazem modificações locais e de pequeno porte no programa. Transformações com alta granularidade provocam modificações globais. Transformações de baixa granularidade são utilizadas para "massagear" o programa de forma que uma transformação de alta granularidade se aplique.

Um assistente de software baseado no conhecimento que utilize a abordagem transformacional aumenta a produtividade através:

- de sua capacidade de prototipação;
- da redução dos custos de desenvolvimento e manutenção;
- do aumento da confiabilidade.

Neste tipo de assistente os programas podem ser representados por suas árvores sintáticas abstratas na base de conhecimentos. As transformações de programas poderiam ser representadas por exemplo, por regras de produção.

O fato de que há várias implementações possíveis de um programa significa que num determinado momento várias regras, cada uma refletindo uma escolha diferente de implementação, podem ser aplicadas. Substituindo a especificação da raiz da árvore e recursivamente considerando todas as possibilidades de transformação que podem ser aplicadas a cada nó da árvore, o espaço de programas equivalentes que o sistema pode derivar é gerado.

A estratégia de controle navega nesta árvore de forma a encontrar uma implementação eficiente para a especificação. Para guiar a busca há o acompanhamento do usuário, além da busca heurística.

Bases de conhecimento provêm um mecanismo uniforme para representar dados. Isto contrasta com o que ocorre em compiladores, nos quais diferentes estruturas de dados são usadas para representar a árvore sintática abstrata, a tabela de símbolos, o código intermediário, etc.

O acesso uniforme provido pela base de conhecimentos permite a integração de diversas ferramentas que podem ser usadas durante a transformação, como provadores de teoremas e ferramentas de análise de programa. Além disso, informações como dados de administração de projeto, documentação e dados de teste podem ser armazenadas na base de conhecimentos.

Usar a base de conhecimentos fornece flexibilidade enquanto por outro lado sacrifica a eficiência que determinadas estruturas de dados oferecem.

Funcionalmente, um sistema de transformação de programas e um compilador são idênticos, já que ambos traduzem uma especificação em alto nível para uma implementação em baixo nível. Compiladores, no entanto, são adequados para tarefas de tradução nas quais cada construtor tem uma única e natural implementação na linguagem objeto. Essa estratégia não funciona se a linguagem contém construtores que têm várias implementações possíveis com características de eficiência que variam e quando a seleção da implementação apropriada requer uma análise global. Nesses casos, a possibilidade de interação com o usuário e a

flexibilidade e modificabilidade dos sistemas de transformação são mais desejados do que a eficiência de um compilador.

A tecnologia desenvolvida para compiladores - "parsers", geradores de "parser", "pretty printers", "type checks", etc - é componente essencial de assistentes baseados no conhecimento voltados para a programação. Entretanto, técnicas de otimização e análise de programa necessitam ser expandidas e melhoradas. Especialmente úteis são as ferramentas de análise incremental que eficientemente atualizam dados de análise de programa quando um programa sofre uma modificação como resultado da aplicação de uma transformação.

Uma opção para o desenvolvimento transformacional é utilizar uma linguagem de largo-espectro, contendo construtores usados tanto na especificação como na implementação (Goldberg-86). Na literatura observa-se a frequente utilização de linguagens lógicas ou funcionais.

- A abordagem da lógica para desenvolvimento de programas: Nesta abordagem começa-se com uma especificação escrita no cálculo de predicados de 1^{a} ordem e é derivada por transformação uma implementação na linguagem restrita de fórmulas de Horn. A implementação resultante é executada como um programa Prolog. Pelo fato da especificação e implementação serem ambas programas lógicos, as regras de transformação são simples regras de dedução.

Mas, uma vez que a dedução não é realizada automaticamente, um sistema de dedução natural é usado.

- **A abordagem da programação funcional no desenvolvimento de programas:** Nesta abordagem a especificação é um programa funcional escrito como uma lista de equações recursivas. A implementação é outro programa funcional com melhores características de eficiência mas com perda em termos de clareza.

Apesar da abordagem transformacional ser uma solução bastante atrativa, não há referência de nenhum sistema criado que demonstre para uma razoável classe de problemas, que o software possa ser gerado com um custo menor do que usando métodos tradicionais.

A seguir são apresentadas duas pesquisas realizadas na área de transformação de programa: os projetos SAFE e REFINE.

PROJETO SAFE (Balzer-85)

Este projeto está sendo desenvolvido no Instituto de Ciências da Informação da Universith of Southern California. Tem como objetivo o desenvolvimento de uma especificação formal de altíssimo nível, que é transformada para uma especificação formal de baixo nível, a qual é

depois traduzida para o código fonte. A manutenção do programa é realizada através da revisão da especificação seguida da repetição da sequência de transformações.

Gist é a linguagem formal de altíssimo nível utilizada. Foi projetada de forma a utilizar as construções da linguagem natural.

Algumas das características desta linguagem são::

- **Base de dados global:** o mundo é composto por objetos, suas relações entre si e o conjunto de operações que podem ser realizadas sobre eles. No nível da especificação os objetos são acessíveis uniformemente.

- **Método operacional:** este método utiliza um interpretador para definir a semântica de uma linguagem. Assim, a computação que transforma entradas em saídas é descrita explicitamente através dos estados pelos quais passa o interpretador. A semântica operacional permite que as especificações sejam vistas como programas, embora possam ser extremamente ineficientes.

- **Conhecimento perfeito:** no nível de especificação todas as informações implícitas e explícitas são representadas. As informações deriváveis são incluídas na base de dados global através de regras de inferência.

- **Referência descritiva:** é a habilidade de se ter acesso a objetos através de suas relações com outros objetos.

- **Referência histórica:** permite acesso a objetos a partir de estados anteriores aos atuais.

A linguagem Gist foi utilizada para especificar várias aplicações reais.

O objetivo dos projetistas de Gist era de que especificações nesta linguagem fossem lidas facilmente. No entanto isto não ocorreu. Para superar este problema foi construído um parafraseador que recebe como entrada uma especificação em Gist e tem como saída sua tradução em linguagem natural. Além de permitir que as especificações sejam facilmente lidas e entendidas, permite que erros sejam descobertos.

REFINE (Refine)

O Reasoning Systems, Inc desenvolveu um ambiente de software baseado nos princípios dos protótipos do projeto CHI (Smith-85). Está implementado em Sun 3 e Sun 4. A idéia do Projeto CHI era o desenvolvimento de uma linguagem de altíssimo nível, baseada na lógica e tipos de dados da teoria de conjuntos, que fosse transformada para Lisp de acordo com a abordagem transformacional.

Refine é uma linguagem fortemente tipada de largo espectro que possui como componentes a teoria de conjuntos, a lógica, regras de transformação, "pattern matching", orientação a objetos (subclasses, herança e atributos) e procedimentos (iteração, seleção, atribuição, etc).

O compilador Refine é implementado como um sistema de transformação de programa. Ele gera Common Lisp a partir de Refine, transformando descrições de programa armazenadas na base de conhecimento, através de refinamentos sucessivos. O compilador tem como características:

- as definições podem ser compiladas individualmente sem que outras definições precisem ser re-compiladas ou re-encadeadas;

- o compilador detecta definições não alteradas, mesmo quando mudanças de formato ou outras modificações "sem importância" são realizadas.

O sistema sintático permite ao programador definir uma linguagem específica do domínio, isto é, uma notação textual para dados na base de conhecimento. Gera, também, ferramentas de processamento da linguagem, como:

- "parser" (traduz um texto para a estrutura da base de conhecimento);
- "printer" (traduz uma estrutura da base de conhecimento para um texto);

- analisador léxico.

II.2.2 AMBIENTES COM SUPORTE GERENCIAL

Suporte para o gerenciamento do processo de desenvolvimento de software torna-se imprescindível quando os sistemas atingem uma complexidade que torna necessário o envolvimento de vários profissionais empenhados na tarefa de desenvolver o sistema.

(Sathi-88) cita algumas questões que exemplificam a complexidade da tarefa de gerenciamento de grandes programas:

- um grande número de atividades (possivelmente maior que 10000) torna impossível a um gerente adquirir informações atuais sobre todas as atividades.

- um grande número de departamentos está envolvido no projeto com diferentes objetivos e cooperando entre si.

- a natureza tecnológica desses programas torna difícil um planejamento sem necessidade de futuras alterações. Modificações são frequentes e necessitam ser aprovadas.

O principal objetivo da gerência é dar condições, a um grupo de pessoas, de trabalhar em torno de um objetivo comum.

Gerenciar um projeto consiste na gerência do produto que está sendo desenvolvido assim como na gerência do processo para desenvolvê-lo. (Dart-87) (Sathi-88)

Fazendo uma síntese a partir de (Ghezzi-91) e (Hamid-86) podemos dividir a gerência do processo de desenvolvimento nas seguintes atividades (não estão incluídas aqui atividades como a gerência de recursos financeiros e outras que são utilizadas independente do produto que se queira obter, software ou qualquer outro):

- **Planejamento:** o gerente deve decidir que objetivos devem ser alcançados, que recursos são necessários para se alcançar os objetivos, como e quando os recursos devem ser adquiridos e como os objetivos devem ser alcançados. Envolve a determinação do fluxo de informações, pessoas e produtos envolvidos no projeto.

- **Gerência de recursos humanos:** envolve a determinação de autoridades e responsabilidades para grupos de atividades visando atender aos objetivos do projeto e a alocação de pessoal para as posições identificadas na estrutura organizacional. Tem também como objetivo guiar aqueles que estão envolvidos no projeto no sentido de entenderem e se identificarem com a estrutura organizacional e com os objetivos do projeto.

- **Controle:** consiste em medir e adaptar atividades de forma a assegurar que os objetivos sejam alcançados. Requer a medição do desempenho comparando com o que era esperado no planejamento e a tomada de decisões no sentido de corrigir os desvios.

A gerência do produto envolve facilidades para:

- Gerência de versões e
- Gerência de modificações propostas e avaliação do seu impacto, atribuição de pessoal para realização das modificações e tudo o mais que está envolvido com o processo de alterar um software.

A seguir é apresentado o exemplo de um ambiente que vem sendo desenvolvido enfatizando o apoio gerencial. É um ambiente que pode ser classificado como do tipo cidade pela taxonomia de (Perry-88).

ISTAR

(Integrated Project Support Environment)

(Dell-86) (Dixon-88)

No ISTAR todas as tarefas/atividades do projeto são vistas como um contrato, que é um bem definido pacote de trabalho que pode ser realizado independentemente por um desenvolvedor para um cliente (um gerente ou outro desenvolvedor).

Ao ser criado um novo projeto é gerada uma base de dados e a medida que novos sub-contratos são definidos são criadas bases de dados específicas para eles. É formada uma hierarquia de contratos. Há tarefas que são específicas de um contrato e as ferramentas alocadas a essas tarefas agem especificamente na base de dados do contrato.

Considerando que o ISTAR dispõe de um conjunto de ferramentas que podem ser alocadas a cada contrato formando portanto um ambiente, podemos considerá-lo como um meta-ambiente configurador.

ISTAR parte do princípio que há necessidade de suporte gerencial em 3 áreas:

- **Administração do projeto**, que envolve:

. planejamento, estimativa e escalonamento de atividades necessárias para se obter os níveis desejados de qualidade, funcionalidade e desempenho dentro das restriç^{ll}es especificadas de custo, escalonamento e recursos.

. atribuição dessas atividades como tarefas à equipe do projeto

. aplicação de controles de projeto.

- **Administração de recursos**, que envolve a identificação de recursos disponíveis ao projeto, sua

alocação a tarefas apropriadas e a resolução de quaisquer conflitos sobre o uso de um recurso particular.

- **Administração de custos**, que envolve:

. o confronto de informações sobre o consumo atual e previsto de recursos dentro do projeto

. a estimativa do progresso e desenvolvimento de controles de projeto

II.2.3 META-AMBIENTES CONFIGURADORES E EXPLORATÓRIOS

Os dois ítens anteriores mostraram exemplos de ambientes que também podem ser classificados como meta-ambientes. O REFINE (Refine) é um meta-ambiente exploratório e o ISTAR (Dixon-88) é um meta-ambiente configurador.

Outros meta-ambientes configuradores que devem ser citados são o PACT (Thomas-89) e o Adage (Giavitto-90).

O ADAGE (Adaptable Graph-Based Environment) tem como objetivo cobrir todas as fases do ciclo de vida e possibilitar o gerenciamento de todo o processo pelo usuário. Tem como princípio ser incremental, ou seja, extensível.

É um ambiente baseado no sistema operacional UNIX, que opera em estações SUN e DEC 3100, sob o ULTRIX.

Utiliza um modelo de dados que é uma extensão do modelo de entidades e relacionamentos, representado através de uma rede de grafos. Através deste modelo é representada a integração entre ferramentas.

II.3 AMBIENTE NO CONTEXTO TABA

No Projeto TABA, ambiente de desenvolvimento de software é considerado como um conjunto integrado de métodos, ferramentas e hardware que suportam o desenvolvimento de um software nas diferentes etapas de seu ciclo de vida. Consideramos o ciclo de vida, métodos, ferramentas e o hardware como os componentes a partir dos quais o ADS é formado.

A seguir é descrito o que entende-se por ciclo de vida, métodos e ferramentas.

CICLO DE VIDA

Um ciclo de vida define o processo de desenvolvimento de software, especificando todas as atividades que devem ser realizadas durante este processo.

MÉTODOS (Rocha-87)

Métodos 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.

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:

- **técnicas construtivas:** guiam o processo de construção do 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 métodos "Structured System Analysis" (Gane-84), observa-se que este método possui os seguintes instrumentos: linguagem para diagrama de fluxo de dados, linguagem para dicionário de dados, linguagem para descrição da lógica dos processos e linguagem para diagram de acesso imediato aos dados.

Os instrumentos estão divididos em:

- **instrumentos para geração:** dão apoio às técnicas construtivas. Os instrumentos 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 rpocesso 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. O item seguinte aborda a questão das ferramentas.

FERRAMENTAS

Ferramentas são recursos automatizados que permitem a realização das atividades envolvidas no processo de desenvolvimento de software. Com base em (Werner-91), (Ghezzi-91), (Ghezzi-85), (Fairley-85), e (Weiss-90) podemos fazer a seguinte classificação das ferramentas de software que vêm sendo utilizadas isoladamente ou em ambientes:

- **Editores:** São utilizados para gerar, corrigir e atualizar documentos. Podem ser textuais e/ou gráficos. Podem seguir uma sintaxe formal ou não e podem ser dirigidos à sintaxe de uma só linguagem ou de mais de uma.

- **Linkers:** São utilizados para combinar fragmentos de código objeto num grande sistema. Ligam nomes que aparecem num módulo a nomes externos de outros módulos.

- **Interpretadores:** Realizam ações específicas sobre um código, escrito com uma notação formal. Este código pode

ser até mesmo uma especificação de alto nível, desde que esteja escrita numa linguagem formal.

- **Tradutores:** Enquanto na interpretação as ações indicadas pelos comandos da linguagem são diretamente executadas, na tradução os programas escritos em linguagens de alto nível são traduzidos para versões equivalentes em linguagem de máquina, antes de serem executados.

- **Geradores de código:** São linguagens de quarta geração que geram código executável a partir das informações fornecidas pelo usuário. Estas informações são inseridas através da interface com o usuário que pode ser de diferentes tipos, dependendo da aplicação.

- **Depuradores:** Executam um programa com o objetivo de auxiliar a isolar e corrigir erros conhecidos. Podem ser utilizados também por outras razões, como observar o comportamento dinâmico de um programa. Neste caso são úteis, por exemplo, para o entendimento de programas escritos por outros programadores.

- **Ferramentas para teste:** Podem dar suporte de diferentes maneiras: suporte à documentação dos testes, suporte à derivação dos casos de teste, suporte à avaliação da atividade de teste e suporte a testes relacionados a verificação de requisitos de qualidade como o desempenho.

- **Analisadores estáticos:** Verificam propriedades desejadas em um programa sem a sua execução. Verificam por exemplo, se estão sendo aplicados os princípio para desenvolver um projeto estruturado ou se estão sendo utilizados os melhores conceitos de programação ao se construir um programa numa determinada linguagem.

- **Ferramentas para gerência da configuração:** Provêm as seguintes facilidades:

. controle do acesso a bibliotecas de componentes, evitando que, por exemplo, dois membros da equipe façam alterações no mesmo componente ao mesmo tempo;

. definição, armazenamento e recuperação de versões ou revisões de um programa;

. manutenção do sistema num estado consistente, automaticamente gerando versões derivadas após a modificação de algum componente.

- **Ferramentas para gerência:** Estão incluídas as ferramentas para programação de atividades e para controle, baseadas em diagramas PERT e gráficos de GANTT, além das ferramentas para estimativa de custos.

- **Tutores e assistentes:** Os assistentes dão suporte durante o desenvolvimento e os mais comuns são os do tipo "help". Tutores auxiliam aqueles que não têm experiência, provendo um tutorial com exercícios.

- **Ferramentas para aquisição do conhecimento:** Têm como objetivo adquirir o conhecimento de especialistas de forma a desenvolver sistemas especialistas.

- **Geradores de documentação:** Geram os documentos que são utilizados no processo de desenvolvimento. O UNIX provê comandos que geram documentos como o man (de manual) e o cmp, que apresenta de uma forma documentada o resultado da comparação de arquivos.

II.4 CONCLUSÃO

Ambientes completos e integrados que suportem todo o ciclo de vida ainda não são uma realidade. De forma a dar este tipo de suporte, aumentando a produtividade do processo de desenvolvimento, (Ghezzi-91) aponta as seguintes características que os ambientes do futuro deverão possuir:

- Todas as fases do processo de desenvolvimento devem ser assistidas pelo computador de forma integrada;

- A formalidade deve ser suportada e não imposta a cada nível do processo de geração de software. Da mesma forma o nível de formalidade apropriado deve ser decidido pelo usuário do ambiente;

- As ferramentas devem ser capazes de aceitar objetos parcialmente especificados e possibilitar a

realização de modificações "on-line", verificando incrementalmente sua consistência e tolerando as informações incompletas.

- O ambiente deve ser capaz de incorporar progressivamente novas ferramentas.

- A modularização deve ser suportada como um mecanismo para estruturar objetos complexos, não somente para a fase de projeto, mas para todo o processo de produção.

- Deve haver um suporte educacional no sentido de treinar os usuários antes que uma ferramenta seja introduzida.

- O ambiente deve ser de uso "amigável", provendo uma interação natural entre homem e computador.

De acordo com o ponto de vista dos autores o ambiente deve suportar diferentes linguagens e paradigmas. São identificados dois tipos de usuários: o administrador do ambiente, que é quem manipula o ambiente adicionando ou modificando funções, e o usuário do ambiente, que interage com o ambiente no sentido de desenvolver uma aplicação.

Consideram, ainda, que as ferramentas deveriam ser baseadas em métodos e que o ambiente deveria suportar trabalho cooperativo.

Os ambientes do futuro deverão obviamente explorar os recursos tornados disponíveis a partir dos avanços do hardware, o que sem dúvida irá torná-los mais poderosos.

O projeto TABA é uma tentativa no sentido de dar suporte a todo o ciclo de vida do software, aumentando a produtividade do processo de desenvolvimento e atendendo a todas as características, citadas acima, apresentadas por (Ghezzi-91) como essenciais aos ambientes do futuro.

A Estação de Trabalho TABA possui um meta-ambiente. O recurso que o meta-ambiente utiliza para gerar ADSs é reunir uma série de componentes, que juntos possibilitarão o desenvolvimento de um sistema de software. De acordo com a taxonomia de (Lucena-87) a Estação de Trabalho TABA possui um meta-ambiente configurador. Em (Hoffnagle-85) e (Giavitto-90) são apresentados exemplos de meta-ambientes que vêm sendo desenvolvidos utilizando esta mesma abordagem.

Neste trabalho será desenvolvida uma primeira versão de uma ferramenta pressupondo que os ADSs a serem gerados serão baseados na tecnologia de sistemas operacionais, de acordo com a taxonomia de (Penedo-88). Nesses ambientes as ferramentas são invocadas como programas, os dados são organizados e acessados através de um sistema de arquivos e o controle do usuário é realizado através de uma linguagem de comando.

O motivo desta escolha é o fato de que o sistema indicará componentes de ambientes a serem usados. Mas não haverá necessariamente uma ligação bem definida entre os componentes. Esta limitação será superada quando o GEOTABA (Rocha-91), o gerente extensível de objetos do TABA, estiver implementado.

No Projeto TABA estão sendo realizados três sub-projetos em paralelo no sentido de atender a três grupos de aplicações, provendo facilidades para o desenvolvimento de software desses grupos, que são os seguintes:

- software científico (Souza-91);
- software educacional (Stahl-91);
- software que utiliza como paradigma de desenvolvimento a orientação a objetos (Mattoso-91).

O TABA tem também como objetivo gerar ADSs que permitam o desenvolvimento de software considerando a possibilidade de um grupo de pessoas interagirem entre si. Desta forma deverá ter recursos de apoio gerencial que permitam inserir os ADSs configurados no modelo cidade da taxonomia de (Perry-88).

Esta tese contribui no sentido de oferecer ao meta-ambiente do TABA um suporte ao planejamento de ambientes que o distingue dos demais meta-ambientes configuradores. Estes meta-ambientes (Hoffnagle-85)

(Giavitto-90) possuem recursos que possibilitam a configuração de um ADS mas não oferecem facilidades para planejar esta configuração.

CAPÍTULO III

SELEÇÃO DE COMPONENTES DE AMBIENTES

A seleção dos componentes de um ADS não é uma tarefa trivial, uma vez que há várias possibilidades de combinação, além de ser necessário o conhecimento sobre cada componente e como eles se relacionam entre si.

Duas questões são abordadas neste capítulo relacionadas à seleção de componentes:

- A avaliação de componentes e ambientes de forma a se ter subsídios para a seleção.
- Os métodos e ferramentas utilizados no processo de seleção.

Os ítens III.1 e III.2 abordam essas duas questões. E em III.3 é apresentada uma comparação entre o que foi pesquisado e a proposta desta tese.

III.1 AVALIAÇÃO DE AMBIENTES

O processo de seleção de um ambiente ou de um componente de ambiente deve pressupor que tenha havido antes uma avaliação. A avaliação e seleção são duas atividades, portanto, que não podem ser separadas.

A avaliação pode ser realizada de diferentes maneiras, uma não excluindo a outra:

- consulta a fornecedores, técnicos responsáveis pela avaliação e usuários;

- consulta a resultados de experiências relacionadas;

- realização de experiências controladas;

A avaliação pode ser classificada, também, de acordo com os objetos a serem avaliados:

- **avaliação de componentes particulares:** são úteis mas falham por não considerar aspectos globais do ambiente ou como os componentes interagem.

- **avaliação de ambientes:** consideram a interação entre os componentes e possibilitam a comparação entre ambientes.

A avaliação de ambientes é muito mais difícil do que a avaliação de um componente. Ao se avaliar um compilador, por exemplo, o interesse está num conjunto de medidas que expressem o desempenho, como o tempo de compilação em termos de instruções traduzidas por minuto, a velocidade de execução ou o tamanho do código objeto resultante. No caso da avaliação de um ambiente há muitas outras questões a considerar e não existe normalmente nenhum mapeamento do conjunto de ferramentas de um ambiente para o conjunto de ferramentas de outro. Por exemplo, é difícil comparar o ambiente que realiza o ciclo tradicional de edição-compilação-montagem-execução com o ADS que realiza compilação incremental durante a edição. (Weiderman-86)

Com relação a questão da avaliação são apresentados a seguir:

- Dois métodos que realizam avaliações: O trabalho de (Gomes-87), apresentado em III.1.1, tem como objetivo além de avaliar, selecionar um tipo de componente e utiliza como estratégia a consulta a pessoas e a realização de experiências controladas, mas não dá detalhes de como seriam realizadas essas experiências. Já o trabalho de (Weiderman-86), apresentado em III.1.2, visa a avaliação de ambientes a partir da realização de experiências controladas, detalhando como estas deveriam ser conduzidas. Maiores informações sobre experiências controladas podem ser obtidas em (Basili-86) onde são relatados uma série de experimentos realizados na Engenharia de Software.

- Uma experiência em avaliação: É apresentado, no item III.1.3, um trabalho sobre avaliação de diferentes aspectos do desenvolvimento de sistemas especialistas.

III.1.1 UM MÉTODO PARA AVALIAÇÃO E ESCOLHA DE LINGUAGENS DE 4ª GERAÇÃO (Gomes-87)

O trabalho foi desenvolvido e aplicado na avaliação de software de quarta geração. O objetivo é a escolha de uma ferramenta adequada à tarefa a ser realizada como também adequada àqueles que irão utilizá-la.

O processo de escolha é composto pelas seguintes etapas:

- Levantamento das necessidades;
- Seleção de softwares;
- Avaliação dos softwares;
- Conclusões.

a) Levantamento das necessidades

Compreende a realização do diagnóstico do ambiente ideal através do levantamento detalhado das principais características dos usuários e suas aplicações.

O levantamento é realizado através de entrevistas, nas quais um questionário é submetido aos usuários finais e aos técnicos de informática (diferentes questionários), onde são abordados os seguintes tópicos:

- arquivos usados;
- atendimento atual e desejado;
- características dos usuários.

b) Seleção dos softwares

São selecionados aqueles que estão mais de acordo com as características levantadas na fase anterior.

c) Avaliação dos softwares

Para permitir que os diferentes softwares sejam comparados são estabelecidos seis grupos, apresentados a seguir, cada qual com uma lista de ítens a serem observados.

Grupo A: Premissas

Foram considerados os pré-requisitos exigidos para a viabilidade de instalação do produto.

Grupo B: Características gerais

Foram consideradas as características relacionadas às principais qualidades necessárias.

- Integração com dicionário de dados dos SGBD diretamente com o dicionário próprio do software de quarta geração;
- Facilidade de documentação;
- Manipulação de determinados arquivos;
- Segurança e integridade;
- Validação automática dos dados;
- Interface gráfica;
- Interface com outras linguagens;
- Geração de protótipos;
- Gerador de relatórios padrão;
- Gerador de telas;
- Ferramentas para análise estatística;
- Ferramentas diversas;

- Interconexão com micros;
- Facilidades de consulta.

Grupo C: Características de operação

- Métodos de acesso suportados;
- Desempenho "on-line";
- Desempenho em "batch";
- Conversacional.

Grupo D: Características específicas da linguagem

- Facilidade de uso;
- Geração de código, interpretação ou compilação;
- Funções "built-in";
- Definição de arquivos e tabelas;
- Definição de procedimentos;
- Limitação do tamanho de registros e registros variáveis;
- Geração de arquivos com formato específico e arquivos intermediários;
- Tratamento de matrizes.

Grupo E: Suporte

Relaciona-se às facilidades que dão apoio aos usuários da linguagem.

- Depuração "on-line";

- "Help";
- Editores inteligentes de telas, relatórios e linguagem;
- Opção de "scrolling" nos arquivos;
- Opção de divisão em qualquer ponto da tela;
- Análise sintática dos comandos a cada entrada;
- Funções que fornecem estatísticas das aplicações indicando alternativas de procedimentos para o melhor desempenho.

Grupo F: Características de comercialização

Relaciona-se ao comportamento do produto no mercado e ao apoio técnico do fornecedor.

- Número de instalações no Brasil e no exterior;
- Tempo no mercado;
- Estrutura de suporte, manutenção, documentação e treinamento;
- Aceitação no mercado;
- Perpectiva de expansão.

Para realizar-se a avaliação obtém-se a opinião para cada item de:

- **Fornecedores:** através de palestras e perguntas objetivas;
- **Técnicos responsáveis pela avaliação:** através de consultas aos manuais dos produtos;
- **Técnicos-usuários de outras empresas:** através de literatura e entrevistas.

Para cada grupo é atribuído um percentual e para os ítens, pesos.

Esses critérios, percentuais e pesos são definidos baseados no levantamento das necessidades e no ambiente de hardware e software existente na instalação, podendo assim ser alterados de acordo com o ambiente pesquisado.

Somente o grupo de avaliação deve saber dos pesos atribuídos.

A atribuição de notas obedece a seguinte escala:

- O software não atende: 0
- O software atende precariamente: 1
- O software atende bem: 2
- O software atende muito bem: 3

d) Conclusões

Para se chegar a uma conclusão objetiva são realizadas saídas gráficas (barras, setores, etc) baseadas na tabela de notas e pesos, podendo constar os pontos alcançados em cada grupo de critérios.

A idéia é que aqueles softwares que foram considerados os mais adequados sejam instalados e sejam desenvolvidos um ou mais projetos piloto de forma a testar seu desempenho.

III.1.2 UM MÉTODO PARA AVALIAÇÃO DE AMBIENTES (Weiderman-86)

Este método é composto por seis fases. As primeiras três fases são independentes do ambiente que será avaliado e são realizadas somente uma vez para todos os ambientes. Enquanto as outras três são dependentes do ambiente e são realizadas uma vez para cada ambiente avaliado. Cada uma das fases se subdivide em passos.

FASE 1: Identificação e classificação de atividades de desenvolvimento de software

Esta fase responde à questão: "O que os desenvolvedores de software, administradores de sistema, gerentes de projeto e outros envolvidos no processo de desenvolvimento de software devem fazer para realizar seu trabalho?"

Consiste de três passos:

1. Identificar as classes de atividades que serão exercitadas pelo conjunto de experimentos genéricos.
2. Refinar cada classe de atividades do passo anterior numa lista de atividades específicas.
3. Classificar como primária e secundária a lista refinada de atividades.

FASE 2: Estabelecer critérios de avaliação

Consiste de dois passos:

1. Estabelecer os critérios pelos quais cada atividade deve ser julgada, como funcionalidade, desempenho, interface com o usuário e interface com o sistema.

2. Criar um conjunto de perguntas aplicadas aos critérios.

FASE 3: Desenvolvimento de experimentos genéricos

Os experimentos são genéricos no sentido que não usam ferramentas específicas em ambientes específicos, mas sim referem-se a tarefas genéricas que devem ser realizadas em sequência.

Estes experimentos devem ser construídos com muito cuidado. Devem ser detalhados o suficiente para permitir ao implementador traduzi-los em experimentos específicos, mas gerais o suficiente para não implicar num conjunto específico de ferramentas.

FASE 4: Desenvolvimento de experimentos específicos

Nesta fase os experimentos genéricos são instanciados em experimentos a serem realizados num determinado ambiente. Esta atividade deve ser realizada por

um especialista no ambiente específico de forma que a tradução aproveite as melhores facilidade dos recursos disponíveis no ambiente.

Os resultados do processo de tradução incluem a determinação da dificuldade de tradução e as respostas a questões sobre funcionalidade (o que pode e o que não pode ser realizado facilmente).

FASE 5: Execução do experimento

O produto desta fase são os dados obtidos a partir da realização do experimento.

FASE 6: Análise dos resultados

Envolve a análise dos dados obtidos na fase anterior.

Depois que o método tiver sido aplicado a vários ambientes será possível comparar os resultados obtidos pelo mesmo experimento nos diferentes ambientes. Poderão então ser feitas recomendações para seleção de ambientes existentes ou para a construção de outros mais.

Esse método vem sendo aplicado a vários ambientes ADA no "Software Engineering Institute" da Carnegie Mellon University, de forma que os ambientes possam ser comparados objetivamente de acordo com os mesmos critérios.

III.1.3 UMA EXPERIÊNCIA NA AVALIAÇÃO DE DIFERENTES ASPECTOS DO DESENVOLVIMENTO DE SISTEMAS ESPECIALISTAS (Roberts-90)

Na Inglaterra há um projeto no sentido de facilitar a transferência de tecnologias para a indústria coordenado pelo "Alvey Directorate" que cobre vários campos como Arquitetura de Sistemas, Engenharia de Software, CAD, Interface Homem-Máquina, VLSI e sistemas inteligentes baseados no conhecimento.

O artigo (Roberts-90) refere-se ao trabalho de três grupos denominados "Alvey Expert Systems Community Clubs", que têm como objetivo a transferência da tecnologia de sistemas especialistas para a indústria. A intenção é encorajar firmas com interesses comuns a trabalharem juntas no sentido de aplicar novas tecnologias a problemas em suas áreas. O objetivo de cada clube é desenvolver um ou mais sistemas especialistas em seu campo de interesse, subvencionados pelo governo. Assim os membros podem compartilhar a experiência de construir um sistema por uma fração do custo de um desenvolvimento independente.

Os clubes incluem membros acadêmicos e da indústria assim como "software houses" que são contratadas para construir os sistemas. Os membros da indústria pagam para se juntar ao clube enquanto os membros da academia

contribuem com o conhecimento e o esforço de desenvolvimento.

Espera-se que os membros ganhem experiência no desenvolvimento de sistemas especialistas assim como nas técnicas, métodos, ferramentas e hardware utilizados. Quanto às "software houses" são encorajadas a formular métodos de desenvolvimento.

III.2 MÉTODOS E FERRAMENTAS PARA SELEÇÃO DE COMPONENTES DE AMBIENTES

A partir dos trabalhos que foram selecionados da literatura, (Davis-86) (Kitto-91) (Zucconi-89) (Gomes-87) (Encarnação-86) (Lustman-87), podemos fazer a seguinte classificação dos métodos e ferramentas para seleção de componentes de ambientes:

- Métodos que apóiam a seleção a partir do levantamento dos componentes de ambientes existentes no mercado, adequados às necessidades pré-estabelecidas;

- Ferramentas de software que já possuem o conhecimento das características de componentes e que então, a partir destas informações, dão apoio ao processo de seleção.

Quanto à primeira classe, pode ser exemplificada pelo método de (Gomes-87) apresentado no item III.2, que tem como objetivo a seleção de linguagens de quarta geração.

Quanto à segunda classe, são apresentados a seguir dois projetos: O primeiro visa a seleção de métodos para especificação de requisitos e projeto. Apesar desses métodos não necessariamente serem automatizadas, a idéia pode ser estendida para seleção de ferramentas. O segundo projeto tem como objetivo a seleção de ferramentas para aquisição do conhecimento.

III.2.1 "THE SPECIFICATION TECHNOLOGY GUIDELINES" (Davis-86)

Este projeto teve como objetivo organizar informações existentes sobre tecnologia de requisitos e projeto num "guidebook" de forma que elas pudessem ser utilizadas pelos gerentes técnicos da Força Aérea dos EUA para selecionar métodos apropriadas em futuros projetos.

O processo de seleção de uma metodologia é composto pelos seguintes passos:

- Calcular o nível de importância (NI);
- Determinar a categoria de software que mais está de acordo com o projeto;
- Selecionar os métodos mais apropriados;
- Escolher um método.

1. Calcular o nível de importância (NI)

O conceito de nível de importância (NI) visa examinar o projeto a partir de três pontos de vista:

- considerações de projeto;
- considerações relativas ao software;
- considerações de qualidade.

a) Considerações de projeto

a.1) Custo: o NI de um projeto aumenta na mesma proporção do relaxamento de restrições relacionadas ao custo de desenvolvimento.

a.2) Segurança: o NI de um projeto aumenta na mesma proporção da necessidade de segurança do sistema.

a.3) Planejamento: o NI de um projeto aumenta na mesma proporção da possibilidade de relaxamento de restrições de planejamento do projeto.

b) Considerações do software

b.1) Complexidade: o NI de um projeto cresce na mesma proporção da complexidade da solução.

b.2) Formalidade de desenvolvimento: o NI de um projeto cresce na mesma proporção do nível desejado de controle por parte do contratante.

b.3) Utilidade do software: o NI de um projeto cresce na mesma proporção da utilidade da aplicação.

c) Considerações de qualidade

c.1) Confiança: o NI de um projeto cresce na mesma proporção do nível de confiança desejado.

c.2) Correção: o NI de um projeto cresce na mesma proporção do nível de correção desejado.

c.3) Manutenção: o NI de um projeto cresce na mesma proporção do nível de manutenção necessário.

c.4) Verificação: o NI de um projeto cresce na mesma proporção do nível de verificação necessário.

O NI varia de 0 a 3 e cada projeto é uma combinação de vários níveis.

São exemplos de projetos com NI igual a:

- 0: geradores de teste, programas de conversão de tabelas;
- 1: editores, compiladores, simuladores de ambientes;
- 2: vigilância aérea, sistemas de aviso antecipado;
- 3: controle nuclear, software que envolve perigo de vida.

Calcula-se o NI de um projeto seguindo-se os passos:

- Avaliar o NI para cada consideração;
- Atribuir um fator de peso para cada consideração;
- Calcular a média ponderada para chegar ao nível de importância global (NIG).

Para calcular o NI é utilizada uma tabela onde constam os 10 ítens a serem considerados, os quatro NI de cada um e uma lista de softwares representativos para cada NI.

2. Determinar a categoria de software que mais está de acordo com o projeto

Chegou-se à conclusão que a maioria dos softwares encontram-se entre 18 categorias, as quais foram descritas numa tabela a fim de poder ser realizado este passo.

3. Selecionar os métodos mais apropriados

Neste passo são utilizadas uma série de tabelas, duas para cada fase do processo de desenvolvimento.

Deve-se dizer que o modelo de ciclo de vida adotado é único e é composto pelas seguintes fases:

- análise de requisitos;

- projeto preliminar;
- projeto detalhado;
- codificação e teste de unidades;
- integração e testes.

Para cada fase há as seguintes tabelas:

1ª tabela: as linhas são compostas pelas 18 categorias de software e as colunas pelas capacidades desejadas por um método que atenda à fase do processo de desenvolvimento em questão. Se houver na interseção de uma linha com uma coluna uma marca, então é porque a capacidade é desejada naquela categoria de software.

2ª tabela: as linhas são compostas pelos métodos que atendem a fase do processo de desenvolvimento em questão e as colunas pelas capacidades desejadas por um método, da mesma forma que a tabela anterior. Se houver na interseção de uma linha com uma coluna uma marca então é porque o método suporta aquela capacidade.

A escolha dos métodos mais adequados se dá através da observação destas duas tabelas. Os métodos escolhidos serão aqueles que demonstrarem atender da melhor forma possível os requisitos da categoria do software a ser desenvolvido.

Cada método recebe um valor (SUM) que corresponde à soma do valor de cada capacidade do método para a fase que

ela atende. Cada capacidade pode receber um valor que varia de 0 a 3, onde 0 é nenhum suporte e 3 corresponde a muito suporte.

4. Escolher um método

Neste passo, a partir dos métodos selecionados no passo anterior, escolhe-se o mais indicado. Esse processo de escolha é realizado de duas formas: nível de suporte uniforme e nível de suporte não uniforme.

Cada um dos métodos candidatos é comparado com um método ideal. Essa comparação se dá através da comparação de valores.

O melhor método terá o valor MS o mais próximo possível de 0. Um valor de MS maior que 0 significa que o método provê mais suporte que o desejado e um valor menor que 0 significa que o suporte provido é menor que o desejado.

- Nível de suporte uniforme:

MS = SUM - NI * COUNT, onde:

SUM é o valor encontrado ao ser classificada o método (descrito no passo 4),
NI é o nível de importância e

COUNT é a cardinalidade do conjunto de capacidades.

- Nível de suporte não-uniforme:

$$MS = \sum_{\text{todas as capacidades desejadas}} (R_{\text{método}} - R_{\text{ideal}}), \text{ onde:}$$

$R_{método}$: é o valor encontrado para uma capacidade particular da metodologia,

R_{ideal}: é o valor desejado da capacidade na metodologia ideal.

Dessa forma só são consideradas as capacidades desejadas. E os métodos não são penalizadas por prover outras capacidades além das desejadas, o que é razoável uma vez que as não desejadas podem ser ignoradas.

Segundo o autor este método é adequado a gerentes experientes uma vez que ele permite pesar seletivamente as capacidades.

III.2.2 SELECIONANDO FERRAMENTAS PARA AQUISIÇÃO DO CONHECIMENTO EM FUNÇÃO DAS CARACTERÍSTICAS DA APLICAÇÃO (Kitto-89)

O Dialog Manager é um dos recursos automatizados do AQUINAS, uma estação de trabalho para aquisição do conhecimento. Tem como objetivo dar recomendações sobre estratégias e ferramentas para aquisição do conhecimento a partir das características da aplicação a ser desenvolvida.

O Dialog Manager é um sistema baseado no conhecimento consistindo de heurísticas e estratégias codificadas em regras e procedimentos. O domínio de seu conhecimento inclui o próprio AQUINAS e as características de outras ferramentas para aquisição do conhecimento. Ele examina o conjunto de regras de forma a selecionar a técnica de aquisição do conhecimento mais apropriada para ser utilizada baseado em heurísticas que consideram:

- as tarefas da aplicação;
- o método de resolução de problemas;
- o estado corrente da base de conhecimento;
- a experiência do usuário;
- considerações de tempo;
- preferências do usuário.

Num diálogo inicial com o especialista do domínio ou engenheiro do conhecimento o Dialog Manager auxilia a identificar o tipo de tarefa da aplicação, o paradigma de resolução do problema e a ferramenta de aquisição do conhecimento apropriada. Se AQUINAS for a ferramenta para aquisição do conhecimento escolhida, o Dialog Manager recomenda técnicas específicas do AQUINAS para elicitação e análise do conhecimento especialista.

Recomendações na escolha de estratégias para aquisição do conhecimento são oferecidas em dois estágios:

Primeiro, numa entrevista inicial com o especialista do domínio, o Dialog Manager pergunta ao especialista sobre a natureza da tarefa da aplicação. Baseado nas características da tarefa e similaridades a tipos de tarefas de aplicação com as quais o Dialog Manager tem familiaridade, o Dialog Manager classifica a tarefa de aplicação, seleciona um método de resolução de problema apropriado para a tarefa e recomenda uma ferramenta específica para aquisição do conhecimento.

Segundo, para as tarefas de aplicação para as quais AQUINAS é o sistema de aquisição do conhecimento sugerido, o Dialog Manager guiará o especialista ou engenheiro do conhecimento na aquisição do conhecimento necessário para resolução do problema. O Dialog Manager considera características da aplicação e métodos para

resolução de problemas para selecionar estratégias adequadas de aquisição do conhecimento no AQUINAS.

III.3 CONCLUSÃO

A proposta desta tese é a de elaborar uma ferramenta que permita a seleção de componentes de forma a compor um ambiente de desenvolvimento de software.

Dentre as ferramentas para seleção pesquisadas e referenciadas neste capítulo, a de (Kitto-91) é a que mais se assemelha à solução adotada nesta tese, uma vez que também é um sistema baseado no conhecimento.

Entretanto, como poderá ser observado no capítulo V desta tese, nossa proposta é mais ampla. O capítulo VI desta tese apresenta com detalhes o que a diferencia.

Quanto à questão da avaliação de componentes, abordada no item III.1, está incluída na proposta desta tese mas não foi definido um método para sua realização. Faz parte de um trabalho a ser iniciado a partir da conclusão desta tese.

CAPÍTULO IV

SISTEMAS ESPECIALISTAS DE SUPORTE À DECISÃO

Sistemas de Suporte à Decisão (SSD) são sistemas desenvolvidos para suportar processos de tomada de decisão de gerentes em situações de decisões complexas e mal estruturadas. Já um Sistema Especialista (SE) se caracteriza por conter o conhecimento de um campo particular de forma a dar assistência aos especialistas ou prover informações ou auxílio àqueles que não têm acesso a um especialista numa determinada área.

Sistemas Especialista de Suporte à Decisão (SESD) incorporam características de SEs e SSDs. São adequados a problemas que requerem o armazenamento de conhecimento de um especialista, como num SE, mas nos quais não é possível deixar que todas as decisões sejam tomadas pelo sistema. São problemas nos quais é essencial a intervenção daquele que está utilizando o sistema pelo fato do conhecimento não estar suficientemente claro a ponto de ser incluído no sistema.

Neste capítulo é descrita a evolução dos SSDs e dos SEs e são apresentados exemplos de SESDs. Por fim conclui-se apresentando as motivações para dar uma solução ao problema desta tese através do desenvolvimento de um sistema com características de um SESD.

IV.1 EVOLUÇÃO DOS SISTEMAS DE SUPORTE À DECISÃO

SSDs são sistemas desenvolvidos para suportar processos de tomada de decisão de gerentes em situações que envolvem decisões complexas e mal estruturadas. (Stabell-87)

Situações ou problemas totalmente estruturados são computáveis e pode-se decidir se a computação é justificável dada a quantidade de tempo e recursos computacionais envolvidos.(Luconi-86)

Pode-se dizer que um problema é totalmente estruturado quando:

- os objetivos são bem estabelecidos;
- é possível se especificar os dados de entrada necessários;
- existem procedimentos padrões através dos quais a solução pode ser calculada;
- não existe necessidade de estratégias complexas para geração e avaliação de alternativas.

Esses problemas são adequados ao uso de técnicas de programação convencionais nas quais virtualmente tudo sobre o problema é bem definido, ou seja, o problema já foi resolvido. (Luconi-86)

Em problemas desestruturados, os procedimentos padrões auxiliam mas não são suficientes por si próprios, uma vez que os dados podem estar incompletos e os objetivos

e restrições podem ser apenas parcialmente entendidos. Nestes casos o computador pode realizar as partes melhor entendidas da resolução do problema enquanto ao mesmo tempo o usuário utiliza seus objetivos, intuições e conhecimento geral para formular problemas, modificar e controlar a resolução de problemas e interpretar os resultados. O usuário pode prover ou modificar dados, procedimentos ou objetivos e pode usar seu conhecimento de todos esses fatores para decidir as estratégias de resolução de problemas. (Luconi-86)

Do início dos anos 70 até hoje os aspectos que são enfatizados ao se desenvolver um SSD têm sofrido modificações. Como em outras áreas a evolução dos SSD é um reflexo da exploração de tecnologias intelectuais e relacionadas ao computador. No caso de SSD, direcionada para melhorar a criatividade nas decisões. A seguir é apresentado um resumo desta evolução. (Sol-87) (Coelho-88)

- No início dos anos 70, SSD foram descritos como sistemas baseados em computador para auxiliar na tomada de decisão, devido principalmente ao trabalho da "Carnegie-Mellon School" de Herbert Simon e Allen Newell sobre psicologia e administração. O ponto de início foi encontrado na aplicação de tecnologia interativa para tarefas gerenciais de forma a usar computadores para melhorar a tomada de decisão. Havia uma forte ênfase no conceito de um único tomador de decisão.

- Do meio para o fim dos anos 70 a ênfase foi em sistemas baseados em computador interativos que auxiliassem os tomadores de decisão a usar bases de dados e de modelos para resolver problemas mal estruturados. A ênfase estava não só no processo de decisão mas também no suporte à computação pessoal com ferramentas para tornar mais rápido o desenvolvimento de aplicações e pacotes de planejamento financeiro. O foco passou de processo de decisão para computação pessoal ou do usuário final.

- Em fins dos anos 70 e início dos 80 o movimento de SSD gerou sistemas utilizando tecnologia adequada e disponível para melhorar a eficiência de atividades gerenciais e profissionais. Software amigável foi produzido com o rótulo de SSD. Disciplinas como pesquisa operacional e psicologia aderiram ao movimento. Conceitos como centro de informações e prototipação foram associados a SSD.

- Do meio ao fim dos anos 80 novas tecnologias continuaram a emergir, por exemplo telecomunicações, permitindo mais esforços nas questões organizacionais, em oposto à computação pessoal e sistemas distribuídos. Outra mudança é a passagem da tecnologia de tomada de decisão "stand-alone" (um terminal de um computador de tempo compartilhado ou um computador pessoal) para tomada de decisões a nível de grupo. Vê-se também uma nova base técnica para SSD: a convergência de estações de trabalho inteligentes. E uma nova tecnologia está emergindo: a de sistemas especialistas.

O termo SSD abrange três aspectos principais. (Keen-87)

- **Decisão:** associada com aspectos não-técnicos e analíticos e o critério para selecionar opções;
- **Suporte:** dirigido à implementação e entendimento da forma como as pessoas agem e como auxiliá-las;
- **Sistema:** ligado a projeto, desenvolvimento e tecnologia.

Ao longo dos anos tem havido um balanço entre a ênfase que é atribuída a cada um desses aspectos. Hoje, no entanto, a atenção está mais voltada para o aspecto decisão.

O aspecto **decisão** tem sido objeto de muitas pesquisas, uma vez que a habilidade de gerar e analisar mais alternativas poderá melhorar a eficácia de todo o processo de decisão. Duas questões abordadas na literatura sobre decisão são as decisões com consequências catastróficas e as decisões em grupo, que são apresentadas a seguir.

- **Decisões com consequências catastróficas** (Fox-90) (Lehner-90)

Decidir envolve escolher entre soluções, ações ou componentes alternativos, levando em consideração a

incerteza que surge da imprecisão e/ou falta de confiança na informação.

Nos últimos anos SSD baseados em computador começaram a ser utilizados intensivamente em áreas como medicina e controle de processos industriais onde decisões podem ter consequências críticas.

A utilização de SEs para suporte à decisão tem causado polêmicas sobre o potencial de erros catastróficos criados por estes sistemas. A consciência sobre os riscos associados com SEs é atualmente tão grande que a questão foi levada a um programa de televisão britânico no qual usuários e críticos americanos e ingleses argumentaram sobre o perigo de se usar SEs em aplicações médicas, industriais e militares, dentre outras.

O artigo de (Fox-89) aponta que tanto na pesquisa em Inteligência Artificial ou engenharia do conhecimento, não tem havido suficiente ênfase nos procedimentos de decisão. Além de ser necessária a pesquisa na área, o autor diz que é necessário que a teoria da decisão estatística e a teoria da decisão comportamental façam parte do treinamento básico de engenheiros do conhecimento e pesquisadores da área de Inteligência Artificial. Aponta também como necessário o conhecimento de psicologia.

A teoria da decisão comportamental é o ramo da psicologia que realiza pesquisas sobre o julgamento e a

tomada de decisão humanas. O artigo de (Lehner-90) aborda esta questão que lida com a validade do conhecimento especialista humano e quando este conhecimento deve ser modelado.

- Decisões em grupo

Nos anos 80, SSD para grupos surgiram de forma a auxiliar aqueles que tomam decisões. A "IFIP WG 8.3 Conference" realizada em 1988 na Itália, se concentrou nas questões que afetam o suporte de decisões organizacionais que não se repetem. Essas decisões são tipicamente realizadas por grupos, ou depois de intensas consultas entre diferentes unidades organizacionais, em vez de serem realizadas individualmente. (Coelho-88)

Apesar de atualmente a maioria dos encontros para se tomar decisões em grupo ser realizada face-a-face, a tecnologia está começando a ser aplicada no sentido de tornar possível que os participantes fiquem separados no espaço e no tempo.

SSD em grupo incluem pacotes para suportar trabalhos individuais e pacotes para suporte a trabalho em grupo. Para permitir que cada um realize seu próprio trabalho, o conjunto para criação de textos e arquivos, gráficos, planilhas, bases de dados e rotinas de "help" são providas para cada estação de trabalho. Para o grupo este tipo de SSD provê software para manipular a opinião do

público. Assim, a tela pública pode ser usada para apresentar uma lista cumulativa de todas as sugestões e mostrar os resultados de votações entre alternativas. (Gray-87)

A idéia é que as estações de trabalho e os produtos gerados do trabalho de todos os usuários sejam interconectados.

Segundo (Engelbart-88):

- É inevitável que haverá uma combinação de redes de trabalho locais de alta velocidade juntamente com redes de alto nível (privadas ou públicas) que farão a interconexão de estações de trabalho e das várias ferramentas e serviços numa organização.

- As estações de trabalho devem ter acesso a várias ferramentas e serviços, providos por um número de fontes distribuídas na rede.

- A coleção de ferramentas e serviços para cada usuário deve ser integrada num todo coerente.

- Cada usuário deve ter acesso a ferramentas, serviços e arquivos de trabalho pessoal a partir de outras estações de trabalho, de forma que mesmo não estando no seu local de trabalho possa realizar suas atividades.

IV.2 EVOLUÇÃO DOS SISTEMAS ESPECIALISTAS

A tecnologia de SEs vem contribuindo para solucionar uma série de falhas que podem ocorrer com os especialistas de uma área: permite que o conhecimento, uma vez adquirido, não seja esquecido, a não ser que haja algum acidente com a memória em que está armazenado; facilita a transferência de conhecimentos; possibilita a geração de resultados mais consistentes, uma vez que o SE não está sob o efeito de fatores emocionais; a geração de documentação é facilitada, já que existe um mapeamento entre a forma que o conhecimento é representado no sistema e a descrição em linguagem natural dessa representação. Os SEs têm ainda a grande vantagem do baixo custo de operação, apesar do custo de desenvolvimento ser ainda bastante alto.

Entretanto não é possível dispensar-se totalmente o especialista pois ainda não há nenhum recurso que o supere em:

- criatividade;
- adaptabilidade a novas situações;
- capacidade sensorial (Nos SEs os dados sensoriais devem ser transformados em símbolos entendíveis pelo sistema, o que os torna lentos);
- conhecimento geral do mundo e portanto dos mais diferentes aspectos que podem afetar a geração de soluções a partir do conhecimento do domínio do SE.

Um SE é definido como um programa que tem as seguintes propriedades:

- tem o mesmo nível de desempenho que um especialista no domínio tratado;
- aplica seus conhecimentos para gerar soluções efetivas e eficientes, utilizando a estratégia que os especialistas usam para eliminar cálculos desnecessários;
- utiliza um raciocínio simbólico;
- tem profundidade, ou seja, dá soluções a problemas reais;
- possui um meta-conhecimento, ou seja, um conhecimento próprio que lhe permite raciocinar sobre suas próprias operações, além de uma estrutura que simplifica o processo de raciocínio.

O conhecimento num SE é organizado de forma a separar o conhecimento do domínio do problema de outros conhecimentos, como o conhecimento de como interagir com o usuário. O conhecimento do domínio é chamado base de conhecimento, enquanto o conhecimento geral de resolução de problemas é chamado máquina de inferência. Um programa que tem o conhecimento organizado desta forma é chamado sistema baseado no conhecimento.

Os SEs diferem dos programas convencionais pela forma como são organizados, pelo modo como incorporam o conhecimento, pela maneira de execução e a impressão que criam através de suas interações (Hayes-Roth-83). Entretanto

os SEs diferem dos programas tradicionais principalmente pelo uso de raciocínio heurístico. As aplicações tradicionais empregam algoritmos, isto é, regras precisas que, quando seguidas, levam a uma conclusão correta. Por exemplo, a quantia paga a um empregado mensalmente é calculada de acordo com um conjunto de regras precisas. Já os SEs frequentemente tratam de problemas dos quais não é possível obter-se uma única resposta precisa. É necessário, para resolvê-los, utilizar técnicas heurísticas que provêm para um problema uma boa resposta, mas não necessariamente a melhor. (Luconi-86)

Em (Gevarter-87) são apresentados os componentes utilizados em ferramentas para construção de SEs. A figura 4.1 mostra a estrutura de uma ferramenta para construção de SEs e a seguir são apresentadas as diferentes possibilidades de cada um desses componentes.

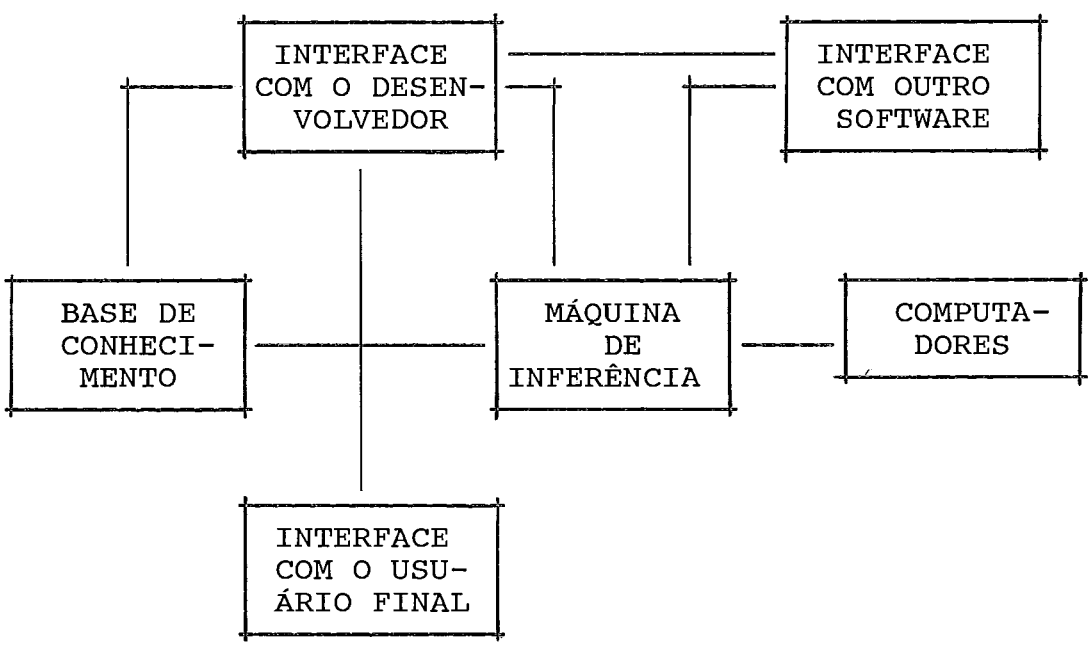


Figura 4.1: Estrutura de uma ferramenta para construção de SEs (Gevarter-87)

a) Representação do conhecimento

Há 3 aspectos a serem considerados na representação do conhecimento:

a.1) Descrições dos objetos que constituem o conhecimento declarativo.

Estes objetos podem ser descritos através de: "frames" (com ou sem herança), objetos (da programação orientada a objetos), pares parâmetro-valor, notação lógica e regras.

a.2) Ações que mudam situações e/ou modificam a base de dados.

Ações podem ser representadas por: regras (grupadas ou não), exemplos, notação lógica e procedimentos.

a.3) Representação de certeza, ou grau no qual o conhecimento ou dado é considerado correto.

Formas de representar a incerteza são: fatores de certeza (Mycin), "fuzzy logic" e probabilidade.

b) Máquina de inferência

Pode ser implementada através de: "backward chaining", "forward chaining", raciocínio hipotético, orientação a objetos, "blackboard", lógica, indução,

"demons", meta-controle, gerenciamento de incerteza e combinação de padrões, dentre outras.

c) Interface com o desenvolvedor

Prevê o uso de facilidades para: criação da base de conhecimento, depuração, criação da interface com o usuário e controle da inferência.

d) Interface com o usuário final

Prevê o uso de facilidades para: responder a consultas, aceitar múltiplas e incertas respostas do usuário, gráficos, responder a questões "como", responder a questões "porquê", responder a questões "e se", poda inicial pelo usuário, soluções múltiplas e salvar exemplos.

Em (Gevarter-87) há a seguinte classificação de SEs por aplicação:

a) Classificação

Classificação refere-se a selecionar uma resposta de um conjunto fixo de alternativas numa base de conhecimento. Algumas das subcategorias de classificação são:

- Interpretação de medidas: refere-se à seleção de hipóteses realizada na base de dados de medidas.

- **Diagnose:** o sistema não só interpreta dados para determinar a dificuldade, mas também busca dados adicionais quando esses dados são requeridos de forma a auxiliar uma determinada linha de raciocínio.

- **Depuração, tratamento ou reparo:** essas funções se referem à tomada de ações ou de medidas recomendadas para corrigir uma situação adversa que foi diagnosticada.

- **Consultor:** um SE como uma interface do programa de computador com o usuário é muito útil para usuários inexperientes. Esses sistemas dependem dos objetivos do usuário e da situação atual de forma a sugerir o que fazer posteriormente. Assim, os avisos evoluem conforme o estado do mundo real modifica. Podem ser úteis também guiando usuários em outros domínios (ex: auto-reparo).

b) Projeto e síntese: refere-se à configuração de um sistema com base num conjunto de alternativas.

c) Assistente inteligente: a ênfase é ter um sistema que dependendo das necessidades do usuário possa dar avisos, fornecer informações ou realizar outras subtarefas.

d) Predição: refere-se a prever o que acontecerá no futuro com base nas informações atuais. Essa previsão pode depender somente da experiência ou pode envolver o uso de modelos e

fórmulas. Os sistemas mais dinâmicos podem usar simulação para auxílio à previsão.

e) Escalonamento: refere-se à ordenação no tempo de um conjunto de tarefas de forma que elas possam ser executadas com os recursos disponíveis sem que uma interfira na outra.

f) Planejamento: é a seleção de uma série de ações de um conjunto complexo de alternativas de forma a atender os objetivos do usuário. É mais complexo que o escalonamento no sentido de que as tarefas são escolhidas e não fornecidas.

g) Monitoramento: refere-se à observação de uma situação de forma a alertar o usuário caso haja algum fato anormal. Aplicações típicas são vôos espaciais e processos industriais.

h) Controle: é a combinação de monitorar um sistema e tomar ações apropriadas em resposta ao monitoramento. Em vários casos, como na operação de veículos ou máquinas, o atraso tolerável da resposta pode ser da ordem de milisegundos.

i) Síntese de informações: um sistema com esta função tem como entrada informações e retorna uma nova organização ou uma síntese.

j) Descoberta: é similar a sintetizar informações exceto pelo fato de que a ênfase é em encontrar novas relações, ordens ou conceitos. Ainda é uma área de pesquisa. Exemplos

incluem encontrar novos conceitos matemáticos e leis elementares da física.

Alguns SEs têm sido desenvolvidos para a área de Engenharia de Software. Exemplo é o apresentado em (Ramsey-89) que tem como objetivo ajudar a detectar tão cedo quanto possível os problemas que podem ocorrer durante a fase de codificação e teste de um projeto. O sistema primeiro determina se um projeto de software está seguindo os padrões de desenvolvimento normais comparando medidas como horas de programação por linha de código fonte contra dados históricos dessas medidas específicas do ambiente. O que for detectado através dessas comparações, como um índice anormal de horas de programação por linha de código, serve de entrada a cada sistema especialista e cada sistema tenta, então, determinar as razões como alta complexidade ou baixa produtividade, para cada padrão anormal de desenvolvimento de software. A detecção prévia de problemas potenciais pode prover uma assistência de grande valor ao gerente do projeto de desenvolvimento de software.

É um fato, entretanto, que pelo fato da Engenharia de Software ser uma área muito nova, com muito de seu conhecimento ainda não claro o suficiente, os sistemas especialistas desenvolvidos nesta área devem ser considerados protótipos exploratórios.

Apesar disso a Engenharia de Software pode certamente se beneficiar do uso de SEs pelo fato de se poder aprender através deles. O desenvolvimento de qualquer SE requer a organização do conhecimento. E assim o engenheiro do conhecimento pode adquirir uma compreensão maior da área estudada uma vez que ele é forçado a desenvolver, entender e organizar relacionamentos entre várias partes do conhecimento (Ramsey-89).

Os SEs para a Engenharia de Software podem também ser utilizados para treinar ou auxiliar pessoas incluindo gerentes de software. Eles contém vários princípios de Engenharia de Software assim como informações históricas que podem ser muito úteis para gerentes e desenvolvedores inexperientes. (Ramsey-89)

Apesar da tecnologia de SEs ainda estar em sua infância, nos últimos anos tem havido um crescente interesse no projeto e implementação de SEs em vários domínios de aplicação (Jain-89). Há, no entanto, uma série de questões que vêm sendo enfrentadas por aqueles que desenvolvem SEs. A seguir são apresentados os problemas que a área de desenvolvimento de SEs enfrenta atualmente.

a) Viabilidade do SE

É essencial que antes de se iniciar o desenvolvimento de um SE, seja realizada uma avaliação do

problema de forma a se determinar as vantagens e necessidades de sua implementação. Esta avaliação impede que mais tarde se chegue à conclusão que o SE não pode ser desenvolvido com os recursos disponíveis, ou ainda, que não compensa o custo de seu desenvolvimento. Evita-se, também, que seja desenvolvido um sistema geral e complexo, que necessite de um número excessivo de regras e objetos na base de dados, o que impedirá sua implementação ou tornará o sistema muito lento.

(Waterman-86) considera que para esta avaliação devem ser considerados três aspectos: se o desenvolvimento do SE é possível, justificado e apropriado.

(Jain-89) aborda também essa questão da viabilidade. Diz que na literatura há pouca discussão sobre os fatores que influenciam o sucesso ou a falha de um determinado sistema. Ele considera que as características do domínio do problema tratado pelo SE desempenham um importante papel no sucesso do sistema. Diz também que há outros fatores responsáveis pelo sucesso mas neste seu artigo se concentra nas características do domínio. Sua pesquisa tenta identificar características básicas de domínios de problemas de SEs que poderiam contribuir para o sucesso de um SE.

b) Estimativa do tamanho

Um problema sério é a estimativa do número de regras necessárias a um SE. A melhor forma de realizar uma estimativa razoável é construir um protótipo durante o estudo de viabilidade. O tamanho deste protótipo poderá dar uma idéia do tamanho do sistema completo. (Waterman-86)

c) Especialista do domínio

Outras dificuldades que surgem estão relacionadas ao especialista do domínio:

- falta de tempo do especialista;
- o especialista não está familiarizado com o problema;
- problemas gerados pelo fato de vários especialistas estarem sendo consultados.

As duas primeiras dificuldades podem ser evitadas através de uma escolha mais criteriosa do especialista. O especialista deve ser competente, capaz de encontrar boas soluções para problemas difíceis e ser capaz de transmitir suas idéias a outras pessoas. Deve, também, ter interesse por computadores, além de dispor regularmente de tempo para interagir com o engenheiro do conhecimento.

Quanto à terceira dificuldade (Waterman-86) considera que no início do projeto deve-se envolver somente um ou dois especialistas, no máximo. Deve-se interagir com

eles até que um protótipo inicial seja desenvolvido. Isto porque gastar pouco tempo com muitos especialistas leva a um entendimento parcial de como o especialista resolve os problemas do domínio. O processo de aquisição do conhecimento envolve uma interação entre o especialista e o engenheiro do conhecimento de forma que este último se torne quase um especialista e o especialista se familiarize com as idéias e métodos da Inteligência Artificial. Para que isto ocorra é necessário paciência e tempo, o que torna difícil a interação com vários especialistas. Outros especialistas podem ser úteis nas fases de teste e revisão.

Já no projeto descrito em (Roberts-90) são utilizados grupos de especialistas. Diz que o uso de vários especialistas é recomendado uma vez que evita-se o risco de se produzir um sistema centrado numa única opinião além de reduzir a dependência a um único indivíduo.

d) Processo de desenvolvimento

Há na literatura diferentes propostas de ciclo de vida para desenvolvimento de SEs (Roberts-90) (Weitzel-89) (Cupello-88). Em todas é comum a realização de um protótipo demonstrável.

e) Ferramentas

Ocorrem, também, uma série de problemas relacionados às ferramentas que são utilizadas para o

desenvolvimento do SEs. A má escolha de uma ferramenta pode levar à representação do sistema de uma forma diferente daquela vista pelo especialista, o que tornará extremamente difícil a tarefa de correção. Tanto o especialista como o engenheiro do conhecimento devem ser capazes de ler, na linguagem utilizada, as regras do SE que está sendo desenvolvido, para que possam interagir entre si. Algumas linguagens para construção de SE permitem um melhor entendimento do que outras. Por exemplo, de acordo com (Waterman-86), os especialistas entendem melhor as regras escritas em ROSIE do que as escritas em LISP.

Deve-se ter em mente, também, a limitação da tecnologia atual para desenvolvimento de SE. É frequentemente difícil ou impossível representar todo o conhecimento do domínio num sistema, mesmo quando o especialista é capaz de comunicá-lo ao engenheiro do conhecimento. Isto porque poucas ferramentas de suporte de alto nível foram completamente desenvolvidas ou são confiáveis. Na verdade, muitas delas são novas e ainda não foram testadas. As limitações atuais das ferramentas fazem com que a tecnologia para desenvolvimento de SE apresente problemas para:

- Incorporar conhecimentos difíceis de serem representados: Certas representações, como as de conhecimento temporal ou espacial podem requerer grandes quantidades de memória para controlar o estado de elementos em vários momentos ou para

armazenar as relações espaciais de diferentes grupos ou objetos. (Waterman-86) (Hayes-Roth-84)

- **Realizar um raciocínio de senso comum:** Senso comum é um conhecimento que todos possuem e utilizam. Por causa do grande número de informações contidas neste conhecimento, não é fácil armazená-lo num programa.

- **Reconhecer os limites de sua capacidade:** Quando é exigido algo além de seus limites ou quando são dados problemas diferentes daqueles para os quais foram designados, os SEs podem falhar de modo inesperado.

- **Manipular grandes quantidades de conhecimentos rapidamente:** É uma dificuldade que surge ao se tentar solucionar problemas de comando e controle em tempo-real.

- **Adquirir conhecimento:** O processo de aquisição do conhecimento envolve a interação do especialista, do engenheiro do conhecimento e da ferramenta em que está sendo desenvolvido o SE. Este processo interativo é problemático pois inicialmente os participantes não falam na mesma linguagem, o que torna a tarefa tediosa além de consumir muito tempo.

Pode-se apontar também como problemas na área de aquisição do conhecimento: a falta de métodos e ferramentas adequadas e de conceitos que permitam a análise de dados. (Breuker-88)

- **Refinar as bases de conhecimento, manipulando conhecimentos inconsistentes:** Muitas ferramentas para construir SEs possuem um editor de bases de conhecimento. Entretanto, ainda não foi desenvolvida nenhuma técnica geral para analisar logicamente a completeza e consistência do sistema. Os engenheiros do conhecimento utilizam um ou dois especialistas para criar a base inicial de conhecimentos do sistema, reduzindo assim as chances de se introduzir inconsistências na base. No entanto, a utilização de um único especialista pode levar a problemas de avaliação nos domínios, dos quais os especialistas frequentemente discordam.

- **Manipular esquemas múltiplos de representação:** As linguagens para construção de SE não são tão flexíveis e gerais como os engenheiros do conhecimento poderiam desejar. Frequentemente, tipos particulares de conhecimento (por exemplo, temporal ou espacial) não podem ser representados facilmente ou esquemas de representação diferentes (por exemplo, regras e "frames") não podem ser representados natural e eficientemente na mesma linguagem. Da mesma forma, muitas linguagens não provêem mecanismos para construção de interfaces adequadas com o usuário. O engenheiro do conhecimento necessita de linguagens naturais gráficas com as quais possa contar, de forma a tornar a interação usuário-sistema tranquila e eficiente.

- **Gerar explicações:** As facilidades de explicação ainda estão num estágio primitivo. No entanto elas têm se mostrado bastante importantes para a operação do sistema pois aumentam a velocidade de desenvolvimento do sistema e facilitam sua aceitação por parte do usuário, uma vez que inspiram confiança no desempenho do sistema e nos processos de raciocínio.

f) Pessoal especializado

A área de SE é ainda recente e não familiar à maioria dos especialistas em computação, tornando-se difícil de ser entendida e aplicada.

Desenvolver um SE, portanto, implica na necessidade de pessoal treinado tecnicamente (engenheiros e cientistas) com capacidade gerencial e treinamento adequado em Inteligência Artificial (Cupello-88). Pelo fato desses sistemas envolverem altos custos e muito tempo de desenvolvimento, é necessário que o engenheiro do conhecimento tenha conhecimento sobre as técnicas para construção de SEs de forma a determinar quando são apropriadas. (Roberts-90)

g) Custo e tempo de desenvolvimento

Com a tecnologia atual e o restrito número de pessoal especializado, é impossível desenvolver-se um SE

rapidamente. Para agilizar o processo de nada adianta aumentar o número de pessoas alocadas para desenvolver o sistema. É também muito difícil estruturar subprojetos independentes, de forma que alguns passos possam ser realizados em paralelo.

Apesar de ser difícil generalizar (Cupello-88) afirma que para se desenvolver um SE leva-se aproximadamente de três a seis meses para se construir um protótipo demonstrável, de seis meses a um ano para se desenvolver todo o protótipo e de dezoito meses a dois anos para concluir o sistema a ser implantado. O sistema final não é estático podendo ser alterado ao longo do tempo.

Com relação a grandes sistemas os custos atuais são proibitivos a menos que seja essencial o desenvolvimento e não possam ser utilizados métodos convencionais.

A experiência relatada em (Robers-90) de unir empresas e instituições acadêmicas num esforço comum de desenvolvimento parece ser uma idéia interessante para rapidamente se obter conhecimento sobre os diferentes aspectos envolvidos no processo de desenvolvimento e a curto prazo se reduzir o custo e o tempo de construção de um SE.

IV.3 ESTRUTURA DE ALGUNS SISTEMAS ESPECIALISTAS DE SUPORTE À DECISÃO

Sistemas Especialista de Suporte à Decisão (SESDs) incorporam características de SEs e SSDs. (figura 4.2) São adequados a problemas que requerem o armazenamento de conhecimento de um especialista, como num SE, mas nos quais não é possível deixar que todas as decisões sejam tomadas pelo sistema. São problemas nos quais é essencial a intervenção daquele que está utilizando o sistema pelo fato do conhecimento não estar suficientemente claro a ponto de ser incluído no sistema. Segundo (Luconi-86) os SESD necessitam de poderosas interfaces com o usuário que permitam a ele inspecionar e controlar facilmente o processo de resolução do problema. (figura 4.3)

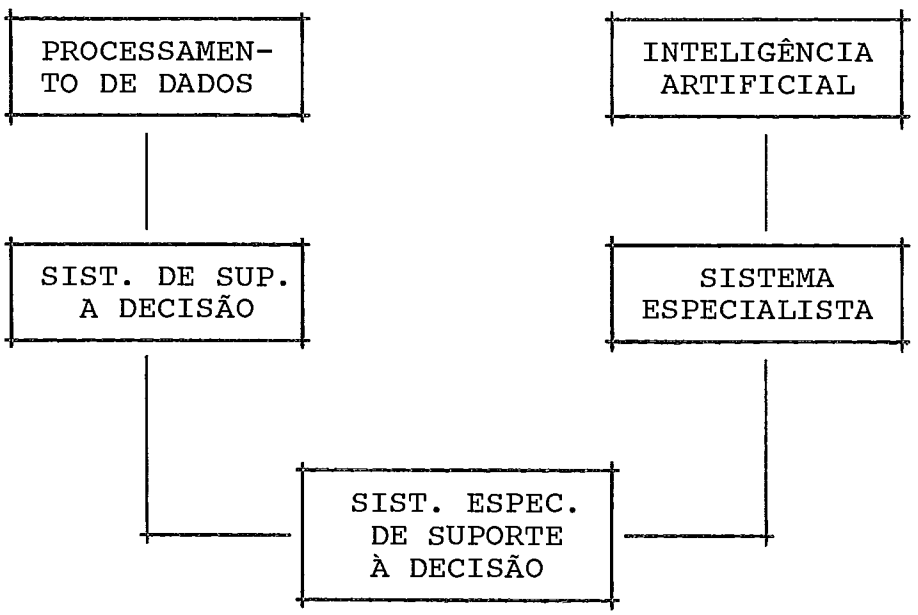


Figura 4.2: Origem dos SESD (Luconi-86)

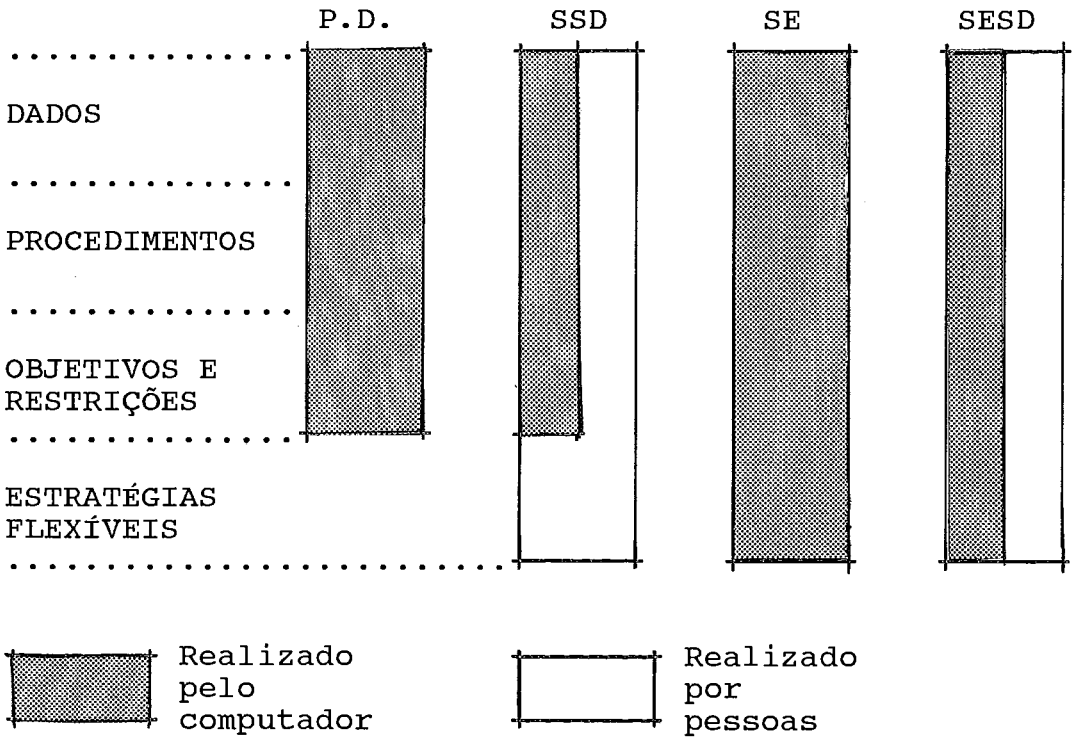


Figura 4.3: Distinção entre diferentes tipos de sistemas (Luconi-86)

O sistema Dipmeter Advisor, aplicado à área de geologia é um exemplo de SESD uma vez que é incapaz por si próprio de automaticamente detectar certos tipos de padrões geológicos complexos. Ele apresenta, graficamente, os dados básicos e permite que o próprio especialista detecte o padrão. O especialista informa ao sistema seu diagnóstico e o sistema continua, utilizando esta informação. (Luconi-86)

A seguir são apresentados exemplos de SESD:

MARBLE

(An Expert System for Managing and Recommending Business Loan Evaluation) (Shaw-87)

O processo de resolução de problema deste sistema consiste de uma sequência de operações utilizando informações da base de conhecimento, base de dados externa, base de dados dinâmica (chamada também de memória de trabalho) e a base de modelos.

Base de modelos: contém módulos de programas para análises financeiras, estimativas, simulações ou regressões.

Base de dados externa: contém dados históricos de empréstimos e informações financeiras de companhias que fazem empréstimos.

Base de conhecimento: a decisão de avaliação de empréstimo é tradicionalmente analisada por modelos lineares estatísticos. Como colocado por Haslen e Longbrake, Kaplan e Dietrich, análises estatísticas com modelos lineares não capturam o julgamento subjetivo e a avaliação qualitativa tão importante na decisão de empréstimo. MARBLE aplica, então, o julgamento realizado por especialistas da área para chegar às suas decisões de empréstimo. A base de conhecimento é composta por regras de produção.

O MARBLE tem, ainda, recursos para raciocínio impreciso, explicação e capacidade de refinamento incremental.

OASES

(Operations AnalySis Expert System) (Biswas-88)

Foi projetado para lidar com problemas especificamente na fase de estado estacionário de processos de fabricação como montagem de automóveis ou produção de fibra de vidro.

Foi designado para desempenhar o papel de um assistente inteligente.

1. Processador "Front-end"

Possui 3 componentes:

a) Um simples processador de linguagem natural para interagir com o usuário;

b) Um seletor de perguntas que dirige o diálogo usuário-sistema selecionando questões relevantes de uma grande base de dados de perguntas armazenadas.

O sistema evita incomodar o usuário com muitas perguntas irrelevantes ou redundantes. Adota, também, um formato de interação de modo que as respostas do usuário não são restritas ao contexto da pergunta. O processador de

linguagem é flexível o suficiente para aceitar informações adicionais e não limitar os usuários a respostas específicas. Assim, os usuários podem prover informações sobre outros aspectos do problema que não estão conectados com a pergunta.

c) um mecanismo de explicação ("help")

2. Base de conhecimento

O conhecimento é representado em regras de produção. Cada partição da base de conhecimento contém regras baseadas em um dos vários tipos de processo. Essa estrutura espelha o processo de raciocínio especialista e torna fácil implementar o diálogo e o esquema de inferência.

3. Mecanismo de inferência

Tem 4 componentes principais:

a) o mecanismo de combinação de evidência;

b) o procedimento para selecionar a hipótese que está na posição topo;

c) o mecanismo de seleção de pergunta que dirige o diálogo usuário-sistema baseado nas hipóteses que estão na posição topo;

d) um controlador de nível topo para selecionar partições na base de regras.

IV.4 CONCLUSÃO

Sistemas Especialistas de Suporte à Decisão incorporam características de Sistemas Especialistas e Sistemas de Suporte à Decisão. São adequados a problemas que requerem o armazenamento de conhecimento de um especialista, como num Sistema Especialista, mas nos quais não é possível deixar que todas as decisões sejam tomadas pelo sistema. São problemas nos quais é essencial a intervenção daquele que está utilizando o sistema pelo fato do conhecimento não estar suficientemente claro a ponto de ser incluído no sistema.

Este é o caso do conhecimento que está sendo manipulado neste trabalho. O planejamento de um ambiente de desenvolvimento de software não é uma tarefa trivial. É um processo heurístico, ou seja, os aspectos envolvidos na escolha de um determinado componente do ADS não são definidos explícita e claramente para todos os casos, necessitando do conhecimento de especialistas (Blum-86) (Janse-87) (Ramsey-89). Estes especialistas devem ter conhecimento da área de aplicação e experiência no uso de métodos e ferramentas, o que muitas vezes não pode ser encontrado em uma única pessoa.

Assim, optou-se pelo desenvolvimento de um sistema com características de um Sistema Especialista de Suporte à Decisão. Desta forma, o sistema proposto nesta tese é orientado no sentido de oferecer apoio à decisão e não a palavra final sobre alguma questão.

CAPÍTULO V

PLANEJAMENTO DE ADSs NO TABA

Este capítulo apresenta o sistema proposto com o objetivo de planejar ADSs e a ferramenta desenvolvida para implementá-lo.

No item V.1 o Projeto TABA é apresentado de forma a se posicionar este trabalho em seu contexto. O item V.2 contém a descrição do sistema proposto. E o item V.3 descreve os componentes da ferramenta implementada.

V.1 PROJETO TABA

O Projeto TABA visa a construção de uma estação de trabalho que atenda aos seguintes objetivos: (Rocha-90)

a) auxilie o engenheiro de software no planejamento e criação do ambiente mais adequado ao desenvolvimento de um produto específico;

b) auxilie o engenheiro de software a implementar as ferramentas necessárias ao ambiente definido em (a);

c) permita aos desenvolvedores do produto (software) o uso da estação através do ambiente especificado em (a) e gerado em (b);

d) permita a execução do software, dado que, eventualmente, o software produto poderá ser executado na própria estação para ele configurada (o que é sempre verdade pelo menos na fase de testes).

Essas funções determinam quatro ambientes na estação TABA: (figura 5.1)

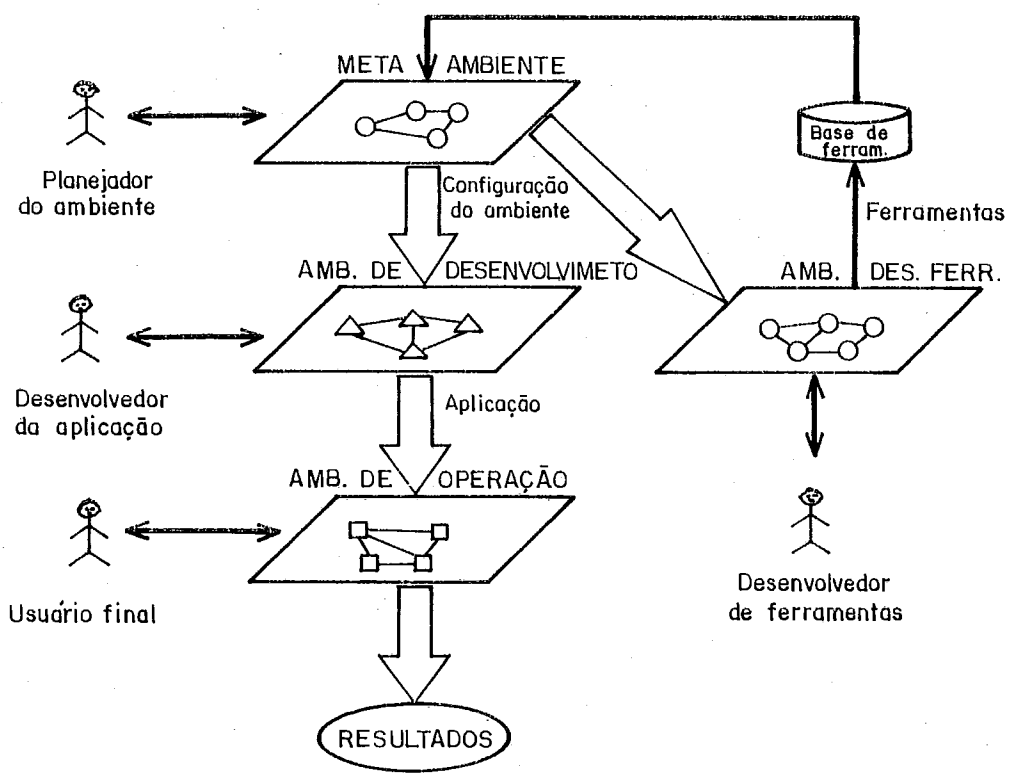


Figura 5.1: Ambientes do TABA (Souza-91)

a) Meta-ambiente

O Meta-ambiente do TABA tem como objetivos principais planejar e gerar o ADS necessário ao desenvolvimento de um software particular.

O ADS é considerado no Projeto TABA como um conjunto integrado de métodos, instrumentos, ferramentas e o hardware que suportam o desenvolvimento de um software nas diferentes etapas de seu ciclo de vida. Consideramos neste trabalho o ciclo de vida, métodos e ferramentas, além do hardware, como os componentes a partir dos quais o ADS é formado.

Planejar o ADS envolve determinar uma configuração adequada ao desenvolvimento de um projeto.

O próximo item desta tese (V.2.2) detalha o meta-ambiente.

b) Ambiente de desenvolvimento

O ambiente de desenvolvimento tem como objetivo possibilitar que, a partir do ADS gerado pelo meta-ambiente, seja desenvolvido o software desejado.

c) Ambiente de operação

O ambiente de operação é aquele no qual irá operar o software desenvolvido no ambiente de desenvolvimento.

V.2 PLANEJAR ADS

Como já mencionado, o meta-ambiente tem como objetivos principais planejar e gerar o ADS necessário ao desenvolvimento de um software particular.

"Planejar ADS" é um dos processos do meta-ambiente que atende ao objetivo de realizar o planejamento de um ADS. A seguir, no item V.2.1, são apresentados os princípios nos quais "Planejar ADS" está baseado. O item V.2.2 apresenta três trabalhos desenvolvidos com o objetivo de dar subsídios para a construção das bases de dados e conhecimento que dão suporte à decisão. E, por fim, o item V.2.3 apresenta a especificação de "Planejar ADS".

V.2.1 PRINCÍPIOS

O processo de planejamento de ADSs está fundamentado em quatro questões:

- a análise das características do projeto a ser desenvolvido;
- a definição do ciclo de vida;
- o suporte à decisão, e,
- a reutilização de informações.

A seguir cada uma dessas questões é detalhada.

V.2.1.1 A ANÁLISE DAS CARACTERÍSTICAS DO PROJETO

O primeiro passo para que se possa indicar os componentes que formarão um ADS é conhecer as características do projeto para o qual este ADS está sendo planejado. Para possibilitar a descrição destas características o usuário planejador de ADS pode recorrer a:

- Descrição das características do sistema atual, caso exista este sistema:

A partir desta decrição, realizada pelo próprio planejador, fica mais simples descrever as características do sistema atual.

- Consulta a uma base de dados com informações sobre características de diversas aplicações:

O planejador analisa essas características e copia para a base de dados do projeto que está sendo desenvolvido as caracterísitcas que considera válidas para o seu caso. A análise de projetos semelhantes será útil para apoiar o usuário principalmente no início do planejamento quando ainda não estão claras as características do sistema a ser desenvolvido. Como será visto mais adiante, esta lista de características é utilizada quando em seu processo de inferência o sistema pergunta sobre a existência de uma determinada característica do sistema a ser desenvolvido. O usuário responde com base nessas informações que tem disponíveis.

V.2.1.2 A DEFINIÇÃO DO CICLO DE VIDA

Como, dependendo do sistema a ser desenvolvido, tem-se um ciclo de vida particular, necessita-se de uma representação que possa expressar as particularidades de cada um, assim como o seu relacionamento com os demais componentes do ADS.

Optou-se, então, por uma representação do ciclo de vida que fosse genérica, possibilitando a modelagem de qualquer processo.

Tentativas de encontrar uma representação genérica para o ciclo de vida podem ser observadas em (Dixon-88) e (Willians-88). Em (Dixon-88) o modelo considera as atividades que fazem parte do projeto de software como um conjunto estruturado de contratos. Esses contratos especificam os resultados visíveis externamente de cada atividade assim como os relacionamentos entre pares de atividades.

Na representação adotada neste trabalho cada ciclo de vida é visto como um grafo em que os nós são as atividades do projeto. A determinação do objetivo da atividade é livre. De maneira geral as atividades atendem à construção do software e sua documentação, à verificação da qualidade, testes, depuração e conserto de erros e a objetivos gerenciais. (figura 5.2)

Para cada atividade consta o relacionamento com as demais atividades e também as informações a ela associadas. O relacionamento entre as atividades é provido pelos arcos do grafo que ligam os nós. As informações associadas a cada atividade são sua identificação, objetivo, data prevista para o início, tempo de duração e os demais componentes do ADS (métodos, ferramentas e hardware) que estão relacionados a atividade.

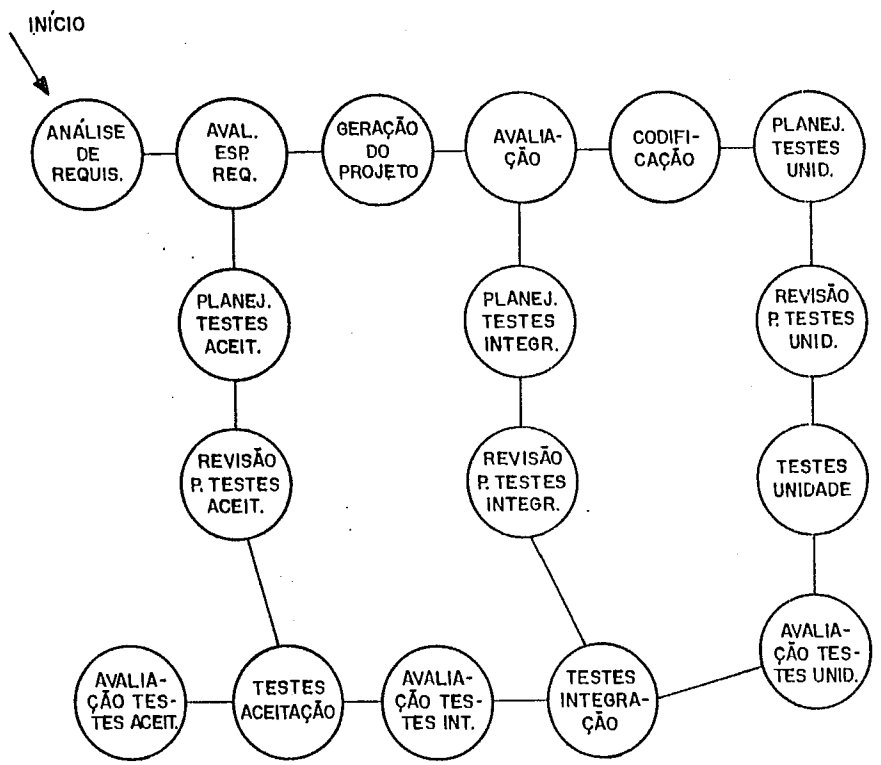


Figura 5.2: Exemplo de um diagrama de atividades.

Para cada atividade ser realizada necessita de métodos e/ou ferramentas independente de métodos. Um método,

por sua vez, está ou não associado a uma ferramenta. Cada ferramenta pode estar associada a um hardware específico para seu uso. E há um hardware (denominado neste trabalho, genérico) que é utilizado para o desenvolvimento de todas as atividades.

Para a elaboração do ciclo de vida o usuário tem dois tipos de apoio:

- O primeiro é uma base formada por ciclos de vida representados como grafos. A idéia é possibilitar a composição de um ciclo de vida a partir da composição de ciclos de vida que estão armazenados nesta base. O ciclo de vida gerado a partir destas composições deverá ainda sofrer alterações pelo usuário de forma a se adaptar às particularidades do projeto em questão.

- O segundo é prover uma base de conhecimento com informações que auxiliem o usuário a escolher um desses ciclos de vida que estão armazenados.

V.2.1.3 O SUPORTE À DECISÃO

O planejamento de um ADS não é uma tarefa trivial. É um processo heurístico, ou seja, os aspectos envolvidos na escolha de um determinado componente do ADS não são definidos explícita e claramente para todos os casos, necessitando do conhecimento de especialistas (Blum-86). Estes especialistas devem ter conhecimento da área de

aplicação e experiência no uso de métodos e ferramentas, o que muitas vezes não pode ser encontrado em uma única pessoa.

Estas questões nos levaram a pensar no sistema que auxilia ao planejamento de ADSs como um sistema de apoio à decisão. Sistemas com este objetivo vêm sendo desenvolvidos de forma a reduzir a incerteza na área gerencial.

Um sistema de apoio à decisão é um sistema interativo que auxilia o indivíduo que toma decisões no seu processo decisório quando é necessária a resolução de problemas pouco estruturados.

Uma outra área que tem sido explorada na atividade decisória é a da Inteligência Artificial, através do desenvolvimento de sistemas especialistas. Utilizar estes sistemas significa dar ao gerente assistência ou prover informações ou auxílio àqueles que não têm acesso a um especialista numa determinada área. Os sistemas especialistas diferem dos programas tradicionais principalmente pelo uso de raciocínio heurístico. Tratam de problemas nos quais não é possível obter-se uma única resposta precisa. É necessário para resolvê-los utilizar técnicas heurísticas que provêm para um problema uma boa resposta, mas não necessariamente a melhor.

Ainda não há a possibilidade de substituir-se um planejador através do uso de sistemas especialistas. No

entanto, já ficou constatada a possibilidade de aproveitar-se essa tecnologia para apoiar a atividade gerencial, utilizando-se técnicas de sistemas especialistas para estender a capacidade dos sistemas de suporte à decisão tradicionais (Ramsey-89).

Sistemas Especialistas de Suporte à Decisão incorporam características de Sistemas Especialistas e Sistemas de Suporte à Decisão. São adequados a problemas que requerem o armazenamento de conhecimento de um especialista, como num Sistema Especialista, mas nos quais não é possível deixar que todas as decisões sejam tomadas pelo sistema. São problemas nos quais é essencial a intervenção daquele que está utilizando o sistema pelo fato do conhecimento não estar suficientemente claro a ponto de ser incluído no sistema.

Este é o caso do conhecimento que está sendo manipulado neste trabalho. Assim, optou-se pelo desenvolvimento de um sistema com características de um Sistema Especialista de Suporte à Decisão. Tem-se desta forma conhecimento que pode ser utilizado para apoiar na indicação de componentes do ADS, assim como três bases de dados.

A primeira base de dados tem informações sobre métodos e ferramentas. Nesta base de dados para cada componente há dados como nome, desenvolvedor, empresa que

comercializa, projetos desenvolvidos que utilizaram o componente, etc.

A segunda base de dados, já mencionada acima no item que trata da definição do ciclo de vida, contém ciclos de vida representados como grafos.

A terceira base de dados, já mencionada acima no item de análise das características do projeto, contém informações sobre características de diferentes aplicações.

Quanto à base de conhecimento deverá conter informações que indiquem as opções possíveis para cada componente do ADS (ciclo de vida, métodos e ferramentas).

V.2.1.4 A REUTILIZAÇÃO DE INFORMAÇÕES

De forma a diminuir os custos de desenvolvimento de software é essencial que durante o processo de sua construção seja aproveitada a experiência obtida em outros desenvolvimentos.

Esta experiência está contida nas especificações que são geradas ao longo do desenvolvimento assim como em tudo o que está envolvido no planejamento do projeto. No caso das especificações ainda há como recuperá-las apesar de ser na maioria dos casos inviável. Quanto às informações de planejamento costumam residir na memória das pessoas envolvidas no processo.

De forma a permitir a reutilização de informações de planejamento de ADSs, "Planejar ADS" deverá gerar um histórico durante todo o desenvolvimento no qual constarão as diferentes versões de ADSs utilizadas no projeto e o que motivou o descarte de cada umas das versões.

Ao fim do projeto as informações do histórico serão analisadas, de modo a se avaliar o comportamento dos componentes do ADS e também do ADS como um todo.

A partir dessa avaliação as bases de dados e de conhecimento, que são utilizadas no suporte à decisão, poderão então ser atualizadas. A consulta dessas bases permite o aproveitamento da experiência obtida no planejamento de ADSs.

V.2.2 SUBSÍDIOS PARA O SUPORTE À DECISÃO

De modo a gerar informações para as bases de dados e de conhecimento que dão suporte à decisão, foram realizados trabalhos que deram subsídios para a construção das seguintes bases:

- a base de dados de características de aplicações;
- a base de dados de ciclos de vida;
- a base de conhecimento com informações sobre ciclos de vida e métodos de especificação.

A seguir é apresentada uma síntese desses trabalhos e são descritos os resultados que foram utilizados para compor a primeira versão da ferramenta implementada nesta tese.

V.2.2.1 CARACTERÍSTICAS DE APLICAÇÕES

A tese (Werneck-90a) teve como motivação a determinação de características de aplicações. As informações obtidas a partir deste trabalho permitiram a construção da base de dados de características de aplicações.

Para obter essas informações foi realizada uma pesquisa de campo em empresas brasileiras situadas no Rio de Janeiro tendo como público alvo os especialistas no desenvolvimento de software de diferentes áreas.

O universo da pesquisa de campo foi limitado a três áreas: administrativa, científica (ciências exatas) e industrial. Essas áreas foram escolhidas considerando-se aspectos de maturidade no uso do computador, da tecnologia empregada e de interesse ao projeto TABA.

Foram consultadas empresas dos seguintes setores: a) distribuição de petróleo, b) serviços de transportes e c) química e petróleo.

Quanto às aplicações pesquisadas foram as seguintes: controle de estoque, cálculo numérico, controle de materiais, controle patrimonial, contas a pagar e a receber, controle de custos, contabilidade, engenharia de controle, estatística, faturamento, financeiro, folha de pagamento, gestão de compras, gestão de produção, gerenciamento de pessoas, gestão de vendas, manutenção de equipamentos, gerência, orçamento, planejamento e controle da produção, pesquisa operacional e simulação.

O método de pesquisa utilizado foi uma adaptação do método utilizado em (Coura-86).

O questionário, os critérios de seleção das empresas e o processo de aplicação dos questionários foram submetidos a uma apreciação para validação de todo o método.

Após esta fase alguns ajustes foram feitos e o questionário foi testado em duas empresas de forma a verificar sua aplicabilidade e praticidade como ferramenta.

Ao fim desse processo foi iniciada a pesquisa de campo, enviando-se uma carta de apresentação aos responsáveis pelas áreas de informática em cada empresa. Em seguida, foram realizadas as entrevistas com os gerentes de desenvolvimento de software e os responsáveis pelo desenvolvimento ou manutenção dos sistemas. O questionário foi submetido a 197 sistemas espalhados em 17 empresas.

Concluída a fase de pesquisa foi feita uma análise dos dados e a aplicação estatística, utilizando-se a Análise Estatística do pacote SPSS/PC.

As aplicações foram descritas a partir de uma série de atributos, incluídos nas classes de características básicas, requisitos de qualidade e características de implementação.

A seguir são descritos os atributos de cada uma dessas classes:

a) Características básicas

- Complexidade do trabalho realizado
- Composição dos tipos de transações
    - . consulta
    - . atualização
    - . emissão de relatórios
    - . tomada de decisão
    - . diagnóstico
    - . cálculo
    - . ação mecânica
- Grau de interação com outras aplicações
- Cálculos (quantidade de fórmulas e complexidade)
- Grau de interação com o usuário

- Composição do tipo de entrada de dados
    . número
    . texto
    . imagem
- Composição da saída de dados
    . número
    . texto
    . imagem

b) Requisitos de qualidade

- Confiança
- Desempenho
- Facilidade no uso do sistema
- Tempo de aprendizagem no uso do sistema
- Disponibilidade
- Satisfação do usuário final
- Precisão
- Dependência do hardware
- Nível de segurança
- Administração de erros
- Dependência do software operacional

c) Características de implementação

- Volume de dados de entrada
- Volume de dados de saída
- Volume de dados armazenados em memória permanente

- Porte do sistema (em linhas de código de programa fonte)
- Tipo de processamento
    . sequencial
    . interativo
    . tempo real
- Tipo de armazenamento
- Uso de técnias especiais de métodos numéricos
- Uso de banco de dados
- Uso de rede de teleprocessamento
- Uso de comunicação de dados entre computadores

As informações que constam da base de características de aplicações da versão desenvolvida nesta tese de "XAMÃ: Planejador de ADS" são as que foram obtidas a partir de análises de dados considerando:

- as características básicas e
- os requisitos de qualidade.

A seguir são apresentados os resultados obtidos a partir de cada uma dessas análises. (Werneck-90b)

CARACTERÍSTICAS BÁSICAS

Considerando as características básicas as aplicações foram classificadas em seis grupos:

Grupo 1: Controle de custos, Engenharia do controle, estatística e pesquisa operacional.

Grupo 2: Gestão de compras, planejamento e controle da produção.

Grupo 3: Controle de estoque, controle patrimonial, contas a pagar e a receber, contabilidade, faturamento e gestão de vendas.

Grupo 4: Financeiro, gerenciamento de pessoas, gestão de produção e orçamento.

Grupo 5: grupo 3 + grupo 4

Grupo 6: Folha de pagamento e gerência.

A tabela 5.1 apresenta os valores encontrados para cada um desses grupos.

TABELA DE CARACTERISTICAS BASICAS

		G1	G2	G3	G4	G5	G6
COMPLEXIDADAE:		ME/A	ME	ME/A	ME/A	ME/A	A
C O M P O S I Ç Ã O	.Atualização (%)	20,2	26,8	25,8	22,7	24,6	24,3
	.Consulta (%)	13,1	29,3	17,2	16,6	17	9,5
	.Relatório (%)	18,7	20	25,5	24,3	25	24,4
	.Cálculos (%)	20,7	13,9	19,5	20,3	19,8	27,6
	.Decisão (%)	15,5	2,7	6,2	8,9	7,3	6,8
	.Diagnóstico (%)	8,7	7,4	5,8	7,3	6,4	7,7
	.Ação Mecânica (%)	3,2	0	0	0	0	0
Grau Integração outras Aplicações:		MB/B	ME/A	ME/A	B/ME	ME	ME/A
CÁLCULOS:							
. Quantidade de Fórmulas:		P	MP	MP/P	P	P	ME/A
. Complexidade		B	B	B	B	B	B
GRAU INTERAÇÃO DO SISTEMA COM O USUÁRIO:		B/ME	A/MA	ME/A	A	ME/A	B/ME
COMPOSIÇÃO DA ENTRADA	.Número (%)	72,8	59	76,6	78	77,6	75
	.Texto (%)	27,2	41	23,4	20,6	22,4	23,2
	.Imagens (%)	0	0	0	1,4	0,6	1,8
COMPOSIÇÃO DA SAÍDA	.Número (%)	69,6	56,8	68,4	64,3	66,8	66,3
	.Texto (%)	30,4	41,8	31,4	33,1	32	32,1
	.Imagens (%)	0	1,4	0,2	2,6	1,2	1,6

Escala de Valores :
5 - EA - Extremamente Alto
4,5 - MA - Muito Alto
4 - A - Alto
3 - ME - Médio
2 - B - Baixo
1 - MB - Muito Baixo
0 - NE - Nenhum ou Ausente

Quantidade de Fórmulas:
Maior 200 - MA -Muito Alto
entre 70 e 200 - A - Alto
entre 70 e 30 - ME - Médio
entre 30 e 10 - B - Baixo
entre 1 e 10 - MB - Muito Baixo
0 - NE - Nenhuma Fórmula

Tabela 5.1: Características básicas X grupos de aplicação (Werneck-90b)

REQUISITOS DE QUALIDADE

Com relação aos requisitos de qualidade foram encontrados 7 grupos:

Grupo 1: Folha de pagamento, gerência e orçamento

Grupo 2: Controle de estoque, contabilidade, engenharia de controle, financeiro, gestão de compras, gerenciamento de pessoal e gestão de vendas.

Grupo 3: Controle patrimonial, contas a pagar e a receber e faturamento.

Grupo 4: Controle de materiais, manutenção de equipamentos, planejamento e controle da produção e pesquisa operacional.

Grupo 5: Controle de custos, estatística e gestão de produção.

Grupo 6: grupo 1 + grupo 2

A tabela 5.2 apresenta os valores encontrados para cada um desses grupos.

TABELA DE ATRIBUTOS DE QUALIDADE

	G1	G2	G3	G4	G5	G6
CONFIANÇA:	A	MA/EA	MA/EA	EA	MA	MA/EA
DESEMPENHO:	B/ME	ME	A	MA	ME	ME/A
FACILIDADE NO USO:	ME/A	A	A	A	A	A
TEMPO DE APRENDIZAGEM:	MB/B	B/ME	ME	ME/A	B/ME	ME
DISPONIBILIDADE:	ME/A	A	A/MA	MA/EA	ME/A	A
SATISFAÇÃO USUÁRIO FINAL:	ME	MA	MA	A/EA	MA	MA
PRECISÃO:	A	MA	A/MA	MA/EA	MA	MA
DEPENDÊNCIA NO HARDWARE:	MB/B	MB	B	B/ME	ME	B
NÍVEL DE SEGURANÇA:	B/ME	A/MA	A	A	ME/A	A
ADMINISTRAÇÃO DE ERROS	ME/A	ME/A	ME/A	A/MA	ME/A	ME/A
DEPEND. SOFTWARE OPERAC.	B/ME	ME	A	A	A	ME/A

Escala de Valores:

5 - EA - Extremamente Alto
4,5 - MA - Muito Alto
4 - A - Alto
3 - ME - Médio
2 - B - Baixo
1 - MB - Muito Baixo
0 - NE - Nenhum ou Ausente

Tabela 5.2: Atributos de qualidade X grupos de aplicação (Werneck-90b)

V.2.2.2 CICLOS DE VIDA

A base de ciclos e a base de conhecimento que contém informações de forma a indicar ciclos de vida foram construídas com base nos seguintes trabalhos:

- Uma tese de mestrado da COPPE, relacionada a este trabalho (Chitman-91) onde foram estudados os diferentes ciclos de vida apresentados na literatura;

- O artigo (Davis-88), onde é feita uma comparação dos diferentes modelos de ciclo de vida;

- O artigo (Carey-90), onde são apresentadas experiências com o uso da prototipagem;

- O artigo (Bersoff-91), onde são apresentados os impactos dos diferentes ciclos de vida na gerência de configuração;

- Nossa própria experiência obtida através do acompanhamento de projetos de desenvolvimento de software.

A seguir é apresentado:

- o conhecimento que levou à formulação das regras que indicam ciclos possíveis de serem utilizados na versão da ferramenta implementada nesta tese;

- a descrição dos diferentes ciclos que constam da versão da ferramenta implementada nesta tese.

CONHECIMENTO SOBRE CICLOS:

De forma a atender às necessidades do usuário o ciclo de vida deve ser organizado considerando que existe um grande grau de incerteza por parte dos usuários projetistas e programadores sobre exatamente o que necessita ser realizado. Principalmente quando se trata de um sistema nunca antes desenvolvido.

Esta incerteza leva a um desenvolvimento lento, vários reinícios, produtividade baixa e altos custos de manutenção (Jones-86). Este problema pode no entanto ser melhorado dependendo do ciclo de vida adotado.

Há vários fatores que determinam o perfil do ciclo de vida, como o o tipo de linguagem utilizada, o suporte gerencial que se pretende adotar, recursos que possibilitem a reutilização, o tipo da aplicação (Waterman-86) e o fato do ciclo ser para desenvolvimento a partir do início da vida do software ou para manutenção (Yau-88). Neste trabalho, no entanto, foram consideradas somente as implicações relacionadas ao tipo de linguagem utilizada.

Em (Davis-88) é apresentado um trabalho comparando os diferentes ciclos de vida que vêm sendo adotados como

alternativa ao ciclo cascata: prototipagem rápida descartável, desenvolvimento incremental, prototipagem evolutiva, reutilização de software e síntese automática de software. Considerando-se a questão do atendimento às necessidades do usuário, todos os ciclos apresentaram melhores resultados que o cascata.

O ciclo cascata é característico de uma época em que as linguagens de terceira geração eram os recursos utilizados para codificação e especificações de mais alto nível não eram executáveis. O advento das linguagens de quarta geração propiciaram a possibilidade de prototipar-se o sistema. Prototipagem é o processo que permite ao desenvolvedor criar o modelo executável do software a ser desenvolvido.

O fato da prototipagem ter começado a difundir-se somente em 1985 está relacionado à limitação das linguagens antes existentes: para se construir um protótipo com linguagens como Assembler, Cobol ou Fortran era necessário codificar talvez 50% do sistema o que tornava inviável seu desenvolvimento. Linguagens interpretadas como APL, Basic e FORTH foram utilizadas para prototipagem com algum sucesso. No entanto, o esforço de concluir um protótipo significativo foi maior que o desejado quando o sistema era grande e complexo. A disponibilidade das poderosas linguagens não procedurais, como as que suportam vários produtos de banco de dados comerciais, causaram o interesse pela prototipagem.

Essas novas linguagens permitem um desenvolvimento muito rápido, frequentemente medido em horas ou dias. (Jones-86)

A prototipagem vem sendo utilizada para:

- simulação funcional total: contém todas as funções do sistema, podendo ser utilizada pelo usuário como um produto final;

- simulação funcional parcial: o protótipo visa testar hipóteses formuladas sobre o funcionamento do sistema.

Há dois tipos principais de prototipagem que implementam estas idéias: rápida descartável e evolutiva.

a) Prototipagem rápida descartável:

Na prototipagem rápida descartável a idéia é construir-se uma rápida implementação parcial do sistema que depois é descartada. Essa implementação pode ser realizada em diferentes estágios do processo de desenvolvimento, dependendo das necessidades:

- durante a especificação de requisitos: para entendimento de aspectos pouco conhecidos;

- em estágios finais do desenvolvimento: para avaliar o desempenho de componentes do sistema a fim de garantir que serão mantidas as restrições de desempenho especificadas.

A idéia neste tipo de prototipagem é fazer o protótipo de um aspecto pouco entendido do sistema. O protótipo é desenvolvido numa linguagem de quarta geração e é utilizado como um modelo para o sistema final.

b) Prototipagem evolutiva:

Na prototipagem evolutiva inicia-se o desenvolvimento pelos aspectos mais bem entendidos do sistema, ao contrário da prototipagem descartável. Não é realizada uma análise global do sistema.

O objetivo nesta abordagem é utilizar o protótipo como o sistema final depois de uma série de modificações realizadas a partir da interação com o usuário. Partes do sistema desenvolvidas numa linguagem de quarta geração, que se apresentam ineficientes, podem ser substituídas por uma linguagem de terceira geração. (Carey-90)

O ideal seria poder desenvolver o sistema utilizando a prototipagem evolutiva. Suas limitações no entanto estão diretamente relacionadas ao estado da arte das linguagens. Numa artigo de 1988, Davis (Davis-88) afirma que

sistemas complexos desenvolvidos de acordo com esta abordagem são caros além de consumirem muito tempo.

Em (Carey-90) são relatadas duas experiências em que foi adotada a prototipagem evolutiva. A primeira não obteve sucesso enquanto a segunda, apesar de alguns problemas, pode ser considerada um sucesso. A explicação está relacionada à época em que foram desenvolvidos os sistemas. O primeiro caso entre 83 e 84 e o segundo em 88. Além de ter havido um aprendizado quando foi realizada a segunda experiência, as ferramentas de software para prototipagem já estavam mais sofisticadas, evitando uma série de problemas.

(Carey-90) enfatiza a necessidade de se ter um ambiente para prototipagem assim como treinar a equipe de desenvolvimento e antes de se partir para uma prototipagem avaliar se ela é necessária e possível. Ele cita os cinco passos sugeridos por Klinger, gerente de um laboratório de sistemas e programação do "Ortho Pharmaceutical Corporation", para utilizar a prototipagem:

- avaliar cada aplicação de forma a verificar se a prototipagem trará benefícios.
- utilizar um ciclo de vida adequado.
- adquirir ferramentas de software apropriadas e treinar a equipe.
- decidir como o processo de desenvolvimento de software será administrado e controlado.

- treinar os usuários finais nos procedimentos a serem seguidos durante o ciclo de prototipagem.

(Pressman-87) enfatiza também a necessidade de se verificar se a prototipagem é o caminho indicado para o software a ser desenvolvido, uma vez que nem todo software é adequado para prototipagem. Em geral qualquer aplicação que cria "displays" visuais dinâmicos, interage bastante com o usuário ou exige processamento algorítmico ou combinatorial é um candidato a prototipagem. No entanto, além da área de aplicação deve ser pesada a complexidade da aplicação. Se uma aplicação como as citadas acima requer o desenvolvimento de dezenas de milhares de linhas de código antes que qualquer função demonstrável possa ser realizada, torna-se muito complexa para a prototipagem. Se, no entanto, a complexidade pode ser particionada pode ser possível prototipar partes do software.

Outras questões devem ser levantadas para se indicar a prototipagem, como:

- a gerência do projeto deseja e é capaz de trabalhar com o método de prototipagem?
- as ferramentas de prototipagem necessárias estão disponíveis?
- os desenvolvedores têm experiência com os métodos de prototipagem?

Para pequenas aplicações pode ser possível a obtenção dos requisistos e diretamente partir-se para a implementação. No entanto, para esforços maiores é necessária a existêncica de uma fase de projeto mesmo com a utilização de uma técnica de quarta geração. O uso destas técnicas sem esta fase para grandes sistemas causará as mesmas dificuldades encontradas no desenvolvimento de software que usa abordagens convencionais: baixa qualidade e manutenibilidade além da reprovação daquele que encomendou o sistema.

Existem dois tipos de ambientes de prototipagem. Um é o ambiente gerador de aplicação integrado e completo, capaz de produzir "menus" integrados, relatórios e janelas e que seja ligado a uma base de dados. Exemplos são o R:Base 5000 ou System V para microcomputador e NOMAD2 para mainframe. (Carey-90)

Outro ambiente é o formado por uma coleção de ferramentas não integradas que auxiliam na rápida construção de partes separadas do sistema. Juntas, essas ferramentas são comumente chamadas de "analysts' workbench" ou "programmers' workbench". Essas ferramentas são dos seguintes tipos:

- editores de texto;
- geradores de tela;
- geradores de relatório;
- bancos de dados relacionais;
- linguagens de quarta geração;

- planilhas eletrônicas;
- dicionários de dados acoplados a sistemas de gerência de dados;
- linguagens de consulta;
- pacotes estatísticos;
- rotinas back-up;
- geradores de documentação;
- help on-line;
- sistema de teste interativo.

Analisadas separadamente, essas ferramentas são a princípio caras se comparadas com os métodos tradicionais de codificação em linguagens de terceira geração como Cobol.(Carey-90)

Linguagens que permitem a síntese automatizada de programas podem ser utilizadas tanto na prototipagem como no ciclo cascata.

O uso de linguagens que permitem a síntese automatizada de software pode se dar das seguintes formas:(Bersoff-91)

- usuários e/ou desenvolvedores escrevem uma especificação de requisitos formal e uma ferramenta de software é utilizada para traduzir os requisitos em software operacional. Para aplicações de Banco de Dados essa tecnologia tem atendido bem através das linguagens de quarta geração. Para outras aplicações a tecnologia é

geralmente disponível mas em instituições de pesquisa. Linguagens de requisitos que devem ser utilizadas são muito formais e o software resultante tende a não satisfazer as restrições de desempenho.

- usuários e/ou desenvolvedores de software escrevem uma especificação de requisitos formal e uma ferramenta de software é utilizada para automaticamente gerar o projeto de alto nível. Então os engenheiros de software completam o desenvolvimento através de meios mais convencionais. Assim como no caso anterior esta tecnologia está disponível mas linguagens de requisitos são tão formais e o projeto resultante está longe de ser ótimo.

- usuários e/ou desenvolvedores de software:

a) realizam um projeto de alto nível, normalmente como hierarquias de diagramas de fluxo de dados mostrando o movimento de dados e controle através do sistema, ou como hierarquias de máquinas hipotéticas como gráficos de estado ("statecharts").

b) especificam algoritmos para a execução de cada função primitiva, usualmente como inglês estruturado, tabelas de decisão, árvores de decisão, máquinas de estado finito, etc.

c) usam a ferramenta de software para simular ou gerar a maior parte do código de baixo nível. Este tipo de síntese de software é o mais fácil atualmente e está disponível nos vários distribuidores de ferramentas CASE.

Nesses três casos (a, b e c), o software está sendo desenvolvido em linguagens de alto nível e ferramentas de software são usadas para automaticamente gerar as instruções de baixo nível necessárias.

Neste estudo da influência das linguagens sobre o ciclo pode-se perceber como cada linguagem determina sequências de atividades próprias num ciclo. A base de conhecimento sobre ciclos foi desenvolvida a partir deste conhecimento e indica um dos ciclos apresentados a seguir.

Esta base de conhecimento, no entanto, terá mais qualidade na medida em que forem desenvolvidos sistemas e o conhecimento de utilização de linguagens for então incorporado à base. Além disso outras questões deverão ainda ser analisaddas como a influência do suporte gerencial que se pretende adotar, o tipo da aplicação, a existência de recursos que possibilitem a reutilização, dentre outras.

DESCRIÇÃO DOS CICLOS

A seguir são apresentados os seis ciclos de vida que constam da base de ciclos da versão da ferramenta implementada nesta tese:

- ciclo do tipo cascata onde não é realizado nenhum tipo de síntese automatizada;

- ciclo do tipo cascata onde é realizada síntese automatizada de análise para projeto;

- ciclo do tipo cascata onde é realizada síntese automatizada de projeto para código executável;

- ciclo do tipo cascata onde é realizada síntese automatizada de análise para código executável;

- ciclo evolutivo;

- ciclo de prototipagem rápida descartável.

Para cada ciclo é apresentado o grafo que o representa e a descrição de suas atividades.

CICLO 1:

Ciclo do tipo cascata onde não é realizado nenhum tipo de síntese automatizada (figura 5.3)

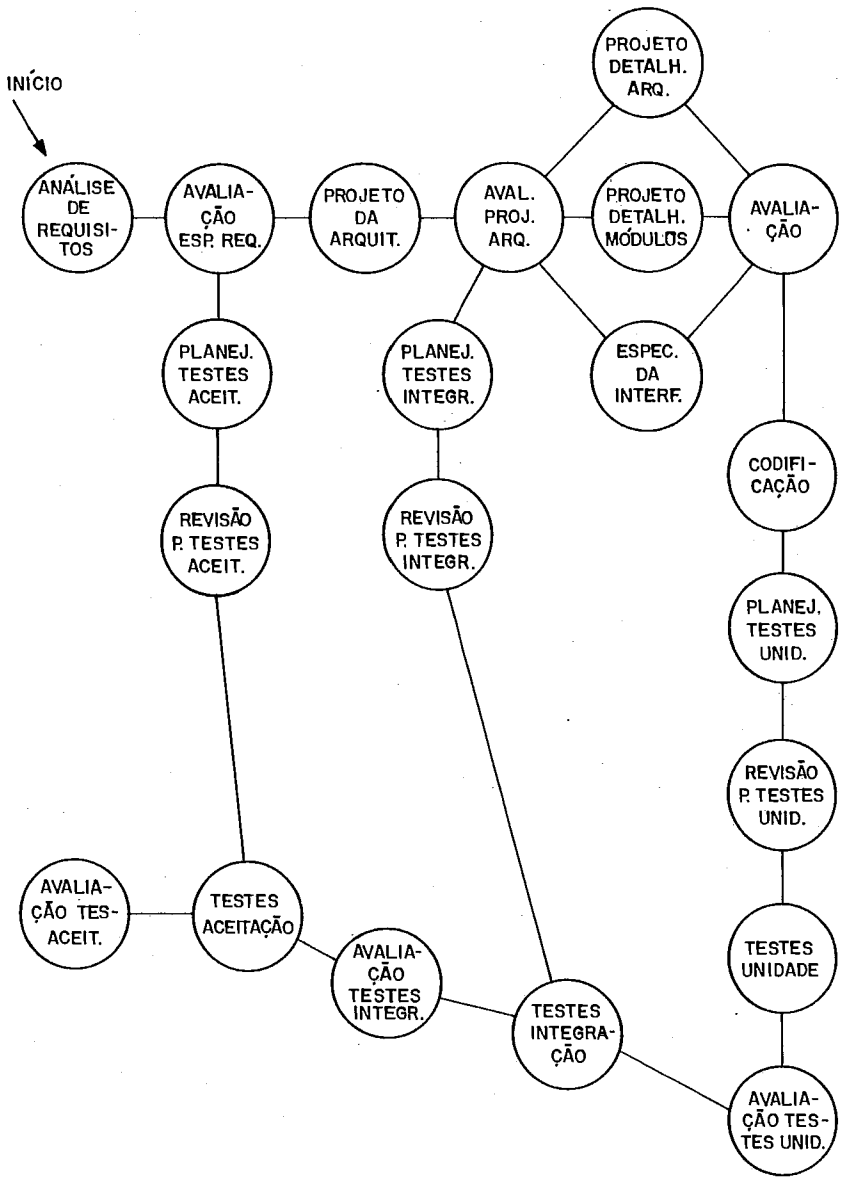


Figura 5.3: Ciclo 1

- Análise de requisitos: Construir uma especificação de alto nível do sistema a partir de consultas às pessoas envolvidas.

- **Avaliação da especificação de requisitos:** Avaliar os produtos gerados na atividade "Análise de Requisitos".

- **Planejamento de testes de aceitação:** Planejar os testes de aceitação de acordo com o que foi especificado nas atividades anteriores.

- **Revisão do plano de testes de aceitação:** Analisar se os testes planejados cobrem todos os aspectos a serem verificados.

- **Projeto da arquitetura:** Construir uma especificação modular do sistema, na qual são representados os módulos e seus relacionamentos.

- **Avaliação do projeto da arquitetura:** Avaliar os produtos gerados na atividade "Projeto da arquitetura".

- **Planejamento de testes de integração:** Planejar os testes de integração.

- **Revisão do planejamento de testes de integração:** Analisar se os testes de integração planejados cobrem todos os aspectos a serem verificados.

- **Projeto detalhado de arquivos:** Descrever o conteúdo dos arquivos e todas as demais informações de forma que seja possível na atividade "codificação" construir toda a parte relacionada aos dados do projeto.

- **Projeto detalhado de módulos:** Especificar cada módulo definido na atividade "Projeto da Arquitetura".

- **Especificação da interface:** São especificados os documentos de entrada e saída de dados, os relatórios e as telas de entrada e saída de dados.

- **Avaliação:** Avaliar em conjunto os produtos gerados pelas atividades "Projeto detalhado de arquivos", "Projeto detalhado de módulos" e "Especificação da interface".

- **Codificação:** Traduzir para as linguagens "finais", executáveis, as especificações geradas nas atividades "Projeto da arquitetura", "Projeto detalhado de arquivos", "Projeto detalhado de módulos" e "Especificação da interface".

- **Planejamento de testes de unidades:** Planejar os testes de unidades.

- **Revisão do plano de testes de unidades:** Verificar se os testes de unidades planejados são suficientes para testar cada unidade.

- **Testes de unidades:** Os testes de cada módulo, planejados na atividade "Planejamento de testes de unidades", são realizados sobre os módulos do produto gerado

em "codificação". É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de unidades: Os resultados dos testes realizados em "Testes de unidades" são avaliados e tomadas as decisões que se fizerem necessárias.

- Testes de integração: Os testes de integração planejados na atividade "Planejamento de testes de integração", são realizados sobre o produto gerado em "codificação". É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de integração: Os resultados dos testes realizados em "Teste de integração" são avaliados e tomadas as decisões que se fizerem necessárias.

- Testes de aceitação: Os testes de aceitação planejados na atividade "Planejamento de testes de aceitação" são realizados no ambiente onde o software irá operar. É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de aceitação: Os resultados dos testes realizados em "Testes de aceitação" são avaliados e tomadas as decisões que se fizerem necessárias.

CICLO 2:

Ciclo do tipo cascata onde é realizada síntese automatizada de análise para projeto (figura 5.4)

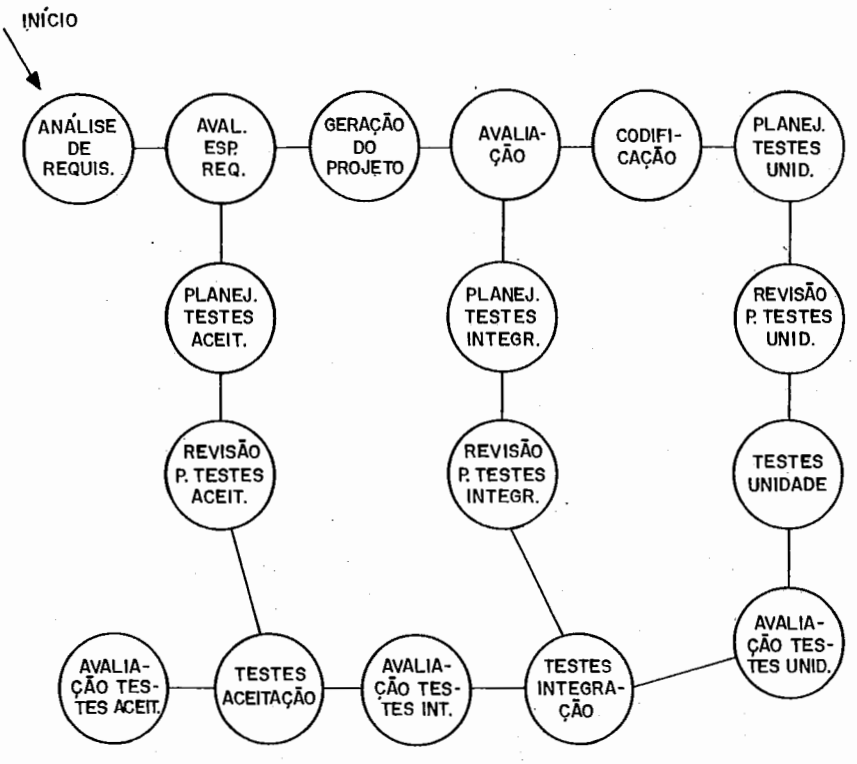


Figura 5.4: Ciclo 2

- Análise de requisitos: Construir uma especificação de alto nível do sistema a partir de consultas às pessoas envolvidas.

- Avaliação da especificação de requisitos: Avaliar os produtos gerados na atividade "Análise de Requisitos".

- Planejamento de testes de aceitação: Planejar os testes de aceitação de acordo com o que foi especificado nas atividades anteriores.

- **Revisão do plano de testes de aceitação:** Analisar se os testes planejados cobrem todos os aspectos a serem verificados.

- **Geração do projeto:** Realizar a tradução da análise para o projeto através de uma ferramenta que permita realizar esta operação automaticamente. Acrescentar ao produto gerado novas informações.

- **Avaliação:** Avaliar em conjunto os produtos gerados pela atividade "Geração do projeto".

- **Planejamento de testes de integração:** Planejar os testes de integração.

- **Revisão do planejamento de testes de integração:** Analisar se os testes de integração planejados cobrem todos os aspectos a serem verificados.

- **Codificação:** Traduzir para as linguagens "finais", executáveis, as especificações geradas na atividade "Geração do Projeto".

- **Planejamento de testes de unidades:** Planejar os testes de unidades.

- **Revisão do plano de testes de unidades:** Verificar se os testes de unidades planejados são suficientes para testar cada unidade.

- Testes de unidades: Os testes de cada módulo, planejados na atividade "Planejamento de testes de unidades", são realizados sobre os módulos do produto gerado em "codificação". É gerado um relato do que ocorreu em cada teste.

- Avaliação dos teste de unidades: Os resultados dos testes realizados em "Testes de unidades" são avaliados e tomadas as decisões que se fizerem necessárias.

- Testes de integração: Os testes de integração planejados na atividade "Planejamento de testes de integração", são realizados sobre o produto gerado em "codificação". É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de integração: Os resultados dos testes realizados em "Teste de integração" são avaliados e tomadas as decisões que se fizerem necessárias.

- Testes de aceitação: Os testes de aceitação planejados na atividade "Planejamento de testes de aceitação" são realizados no ambiente onde o software irá operar. É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de aceitação: Os resultados dos testes realizados em "Testes de aceitação" são avaliados e tomadas as decisões que se fizerem necessárias.

CICLO 3:

Ciclo do tipo cascata onde é realizada síntese automatizada do projeto para código executável (figura 5.5)

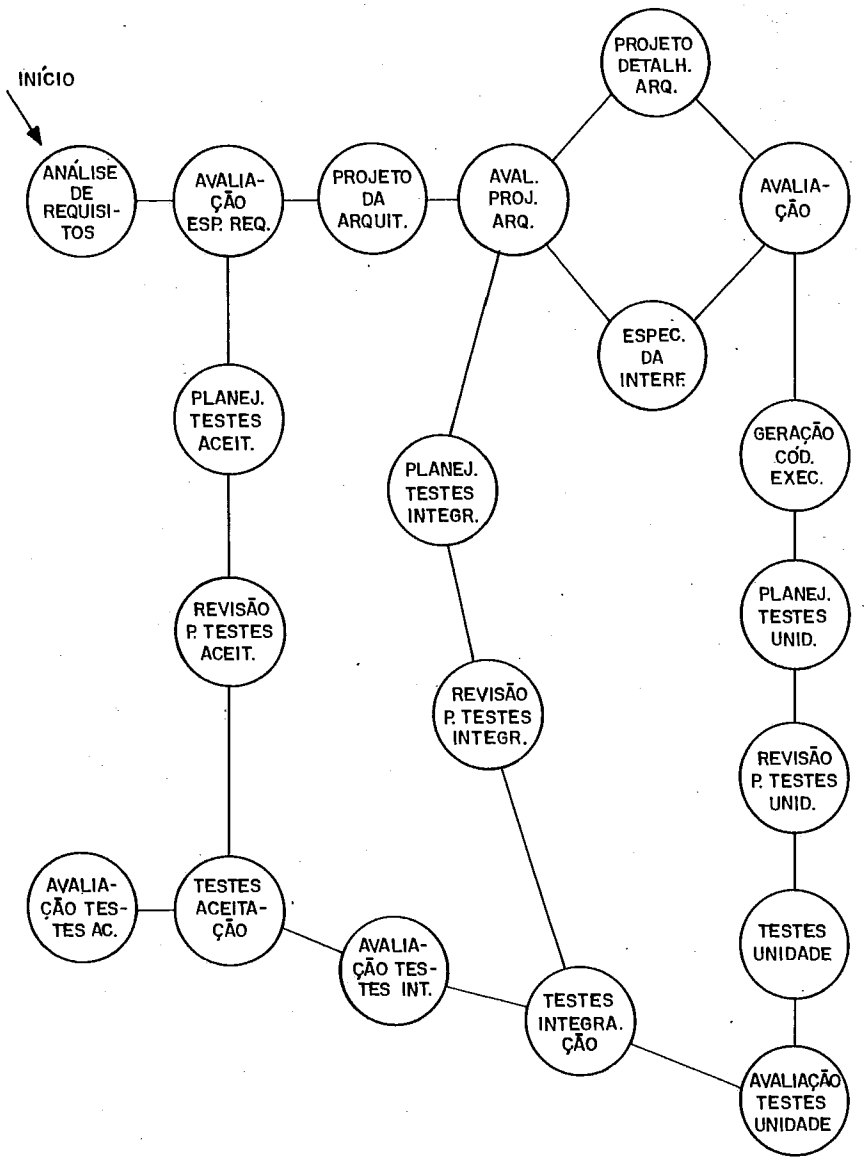


Figura 5.5: Ciclo 3

- **Análise de requisitos:** Construir uma especificação de alto nível do sistema a partir de consultas às pessoas envolvidas.

- **Avaliação da especificação de requisitos:** Avaliar os produtos gerados na atividade "Análise de Requisitos".

- **Planejamento de testes de aceitação:** Planejar os testes de aceitação de acordo com o que foi especificado nas atividades anteriores.

- **Revisão do plano de testes de aceitação:** Analisar se os testes planejados cobrem todos os aspectos a serem verificados.

- **Projeto da arquitetura:** Construir uma especificação modular do sistema, na qual são representados os módulos e seus relacionamentos.

- **Avaliação do projeto da arquitetura:** Avaliar os produtos gerados na atividade "Projeto da arquitetura".

- **Planejamento de testes de integração:** Planejar os testes de integração.

- **Revisão do planejamento de testes de integração:** Analisar se os testes de integração planejados cobrem todos os aspectos a serem verificados.

- **Projeto detalhado de arquivos:** Descrever o conteúdo dos arquivos e todas as demais informações de forma que seja possível na atividade "geração de código executável" construir toda a parte relacionada aos dados do projeto.

- **Especificação da interface:** São especificados os documentos de entrada e saída de dados, os relatórios e as telas de entrada e saída de dados.

- **Avaliação:** Avaliar em conjunto os produtos gerados pelas atividades "Projeto detalhado de arquivos" e "Especificação da interface".

- **Geração de código executável:** Realizar a tradução do projeto para código executável através de uma ferramenta que permita realizar esta operação automaticamente. Incluir no produto gerado novas informações se necessário.

- **Planejamento de testes de unidades:** Planejar os testes de unidades.

- **Revisão do plano de testes de unidades:** Verificar se os testes de unidades planejados são suficientes para testar cada unidade.

- **Testes de unidades:** Os testes de cada módulo, planejados na atividade "Planejamento de testes de unidades", são realizados sobre os módulos do produto gerado

em "Geração do código executável". É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de unidades: Os resultados dos testes realizados em "Testes de unidades" são avaliados e tomadas as decisões que se fizerem necessárias.

- Testes de integração: Os testes de integração planejados na atividade "Planejamento de testes de integração", são realizados sobre o produto gerado em "geração do código executável".

- Avaliação dos testes de integração: Os resultados dos testes realizados em "Teste de integração" são avaliados e tomadas as decisões que se fizerem necessárias.

- Testes de aceitação: Os testes de aceitação planejados na atividade "Planejamento de testes de aceitação" são realizados no ambiente onde o software irá operar. É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de aceitação: Os resultados dos testes realizados em "Testes de aceitação" são avaliados e tomadas as decisões que se fizerem necessárias.

CICLO 4:

Ciclo do tipo cascata onde é realizada síntese automatizada da análise para o projeto e do projeto para o código executável (figura 5.6)

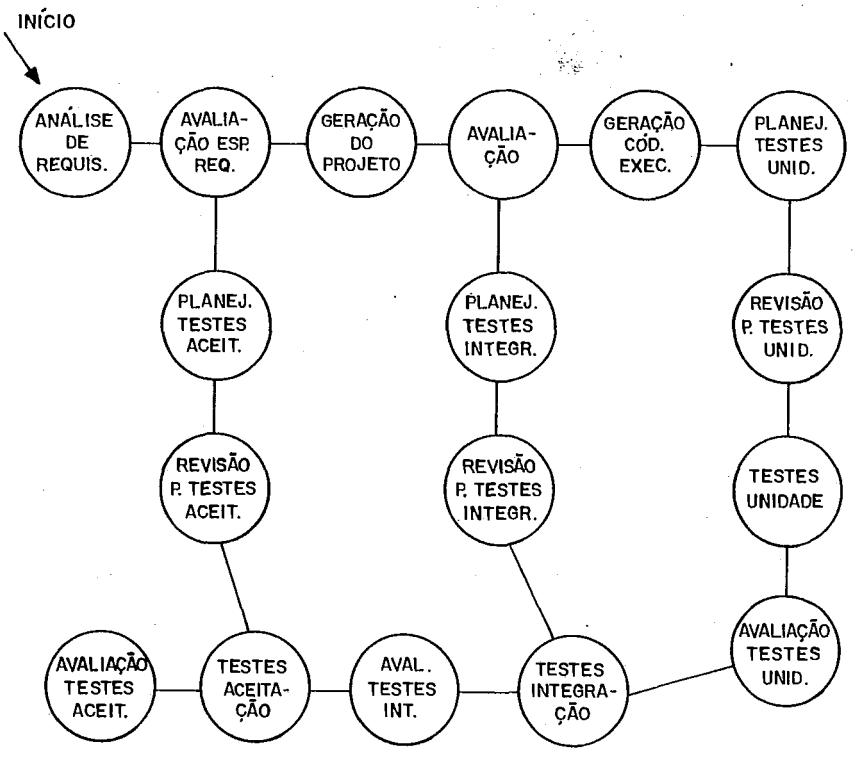


Figura 5.6: Ciclo 4

- Análise de requisitos: Construir uma especificação de alto nível do sistema a partir de consultas às pessoas envolvidas.

- Avaliação da especificação de requisitos: Avaliar os produtos gerados na atividade "Análise de Requisitos".

- **Planejamento de testes de aceitação:** Planejar os testes de aceitação de acordo com o que foi especificado nas atividades anteriores.

- **Revisão do plano de testes de aceitação:** Analisar se os testes planejados cobrem todos os aspectos a serem verificados.

- **Geração do projeto:** Realizar a tradução da análise para o projeto através de uma ferramenta que permita realizar esta operação automaticamente. Acrescentar ao produto gerado novas informações.

- **Avaliação:** Avaliar em conjunto os produtos gerados pela atividade "Geração do projeto".

- **Planejamento de testes de integração:** Planejar os testes de integração.

- **Revisão do planejamento de testes de integração:** Analisar se os testes de integração planejados cobrem todos os aspectos a serem verificados.

- **Geração de código executável:** Realizar a tradução do projeto para código executável através de uma ferramenta que permita realizar esta operação automaticamente. Incluir no produto gerado novas informações se necessário.

- **Planejamento de testes de unidades:** Planejar os testes de unidades.

- **Revisão do plano de testes de unidades:** Verificar se os testes de unidades planejados são suficientes para testar cada unidade.

- **Testes de unidades:** Os testes de cada módulo, planejados na atividade "Planejamento de testes de unidades", são realizados sobre os módulos do produto gerado em "Geração do código executável". É gerado um relato do que ocorreu em cada teste.

- **Avaliação dos testes de unidades:** Os resultados dos testes realizados em "Testes de unidades" são avaliados e tomadas as decisões que se fizerem necessárias.

- **Testes de integração:** Os testes de integração planejados na atividade "Planejamento de testes de integração", são realizados sobre o produto gerado em "codificação".

- **Avaliação dos testes de integração:** Os resultados dos testes realizados em "Teste de integração" são avaliados e tomadas as decisões que se fizerem necessárias.

- **Testes de aceitação:** Os testes de aceitação planejados na atividade "Planejamento de testes de aceitação" são realizados no ambiente onde o software irá operar. É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de aceitação: Os resultados dos testes realizados em "Testes de aceitação" são avaliados e tomadas as decisões que se fizerem necessárias.

CICLO 5:

Prototipagem evolutiva (figura 5.7)

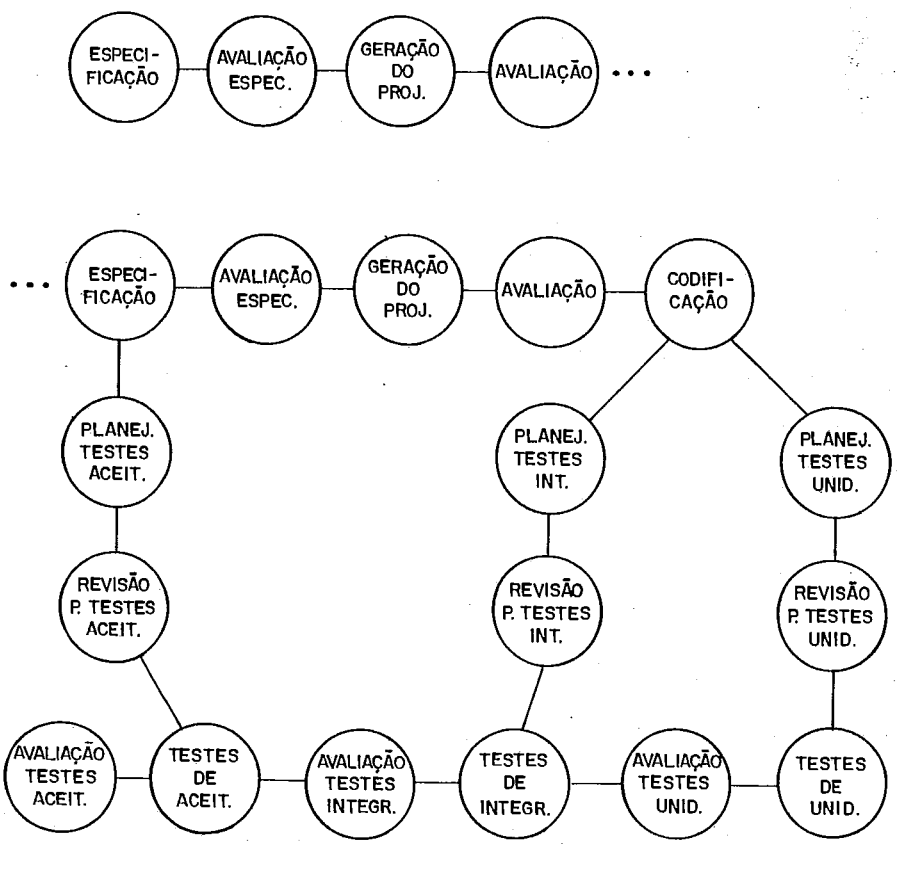


Figura 5.7: Ciclo 5

Observação: A utilização do símbolo ... significa que no ponto em que aparece no grafo haverá necessariamente a inserção de atividades. No caso deste ciclo são repetidas

quantas vezes necessário a sequência de atividades que antecedem o símbolo.

- **Especificação:** Construir uma especificação de alto nível do sistema a partir de consultas às pessoas envolvidas, representando os aspectos mais bem entendidos. Aspectos não esclarecidos poderão ser analisados na atividade "Avaliação".

- **Avaliação da especificação de requisitos:** Avaliar os produtos gerados na atividade "Especificação".

- **Geração do protótipo:** O protótipo é construído de acordo com a especificação realizada.

- **Avaliação:** O protótipo desenvolvido é avaliado.

- **Codificação:** Uma vez tendo sido aceito o protótipo este é melhorado de forma a se tornar operacional.

- **Planejamento de testes de aceitação:** Planejar os testes de aceitação de acordo com o que foi especificado nas fases anteriores.

- **Revisão do plano de testes de aceitação:** Analisar se os testes planejados cobrem todos os aspectos a serem verificados.

- **Planejamento de testes de integração:** Planejar os testes de integração.

- **Revisão do planejamento de testes de integração:** Analisar se os testes de integração planejados cobrem todos os aspectos a serem verificados.

- **Planejamento de testes de unidades:** Planejar os testes de unidades.

- **Revisão do plano de testes de unidades:** Verificar se os testes de unidades planejados são suficientes para testar cada unidade.

- **Testes de unidades:** Os testes de cada módulo, planejados na atividade "Planejamento de testes de unidades", são realizados sobre os módulos do produto gerado em "codificação". É gerado um relato do que ocorreu em cada teste.

- **Avaliação do teste de unidades:** Os resultados dos testes realizados em "Testes de unidades" são avaliados e tomadas as decisões que se fizerem necessárias.

- **Testes de integração:** Os testes de integração planejados na atividade "Planejamento de testes de integração", são realizados sobre o produto gerado em "codificação". É gerado um relato do que ocorreu em cada teste.

- Avaliação dos testes de integração: Os resultados dos testes realizados em "Testes de integração" são avaliados e tomadas as decisões que se fizerem necessárias.

- Testes de aceitação: Os testes de aceitação planejados na atividade "Planejamento de testes de aceitação" são realizados no ambiente onde o software irá operar.

- Avaliação dos testes de aceitação: Os resultados dos testes realizados em "Testes de aceitação" são avaliados e tomadas as decisões que se fizerem necessárias.

CICLO 6:

Prototipagem rápida descartável (figura 5.8)

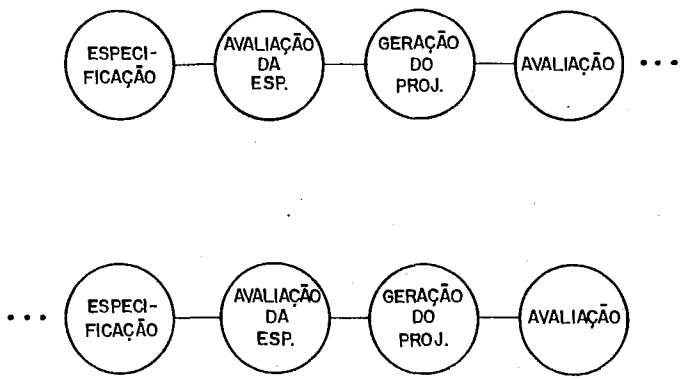


Figura 5.8: Ciclo 6

Observação: A utilização do símbolo ... significa que no ponto em que aparece no grafo haverá necessariamente a

inserção de atividades. No caso deste ciclo são repetidas quantas vezes necessário a sequência de atividades que antecedem o símbolo.

- **Especificação:** Construir uma especificação de alto nível de algum aspecto não compreendido.

- **Avaliação da especificação:** Avaliar a especificação construída em "Especificação".

- **Geração do protótipo:** O protótipo é construído de acordo com a especificação realizada.

- **Avaliação:** O protótipo desenvolvido é avaliado.

V.2.2.3 MÉTODOS PARA ESPECIFICAÇÃO

A base de conhecimento de métodos de especificação foi elaborada a partir dos seguintes trabalhos:

- A tese de mestrado (Crispim-91), que avalia uma série de métodos para desenvolvimento de software voltados basicamente para a modelagem de sistemas. Utiliza um método para a avaliação proposto na própria tese.

- O livro (Cohen-86), que aborda como devem ser especificados os sistemas complexos.

- O livro (Mendes-89), sobre métodos semi-formais e formais para a especificação de sistemas.

- Nossa própria experiência na utilização de métodos de especificação.

Os métodos de especificação avaliados na tese (Crispim-91) foram os seguintes:

- Bop prototyping system - BOP (Aaram-84)
- Design approach for real-time systems - DARTS (Gomaa-84)
- Entity-relationship model - ERM (Chen-90)
- Essential Systems Analysis - ESA (Mcmenamin-91)
- Jackson system development - JSD (Pressman-88)

- Knowledge-based system development life cycle KBSDLC (Weitzel-89)
- Knowledge-based system development methodology KBSDM (Keller-87)
- Methodology for business system development - MBSD (Mathur-87)
- Object-oriented analysis - OOA (Coad-90)
- Object-oriented specification - OOS (Bailin-89)
- Object-oriented systems analysis - OOSA (Shlaer-88)
- Prototyping of user interfaces - PUI (Christensen-84)
- Rapid intelligent prototyping language - RIPL (Cochran-85)
- Scenario-based requirements elicitation - SBRE (Holbrook-90)
- Software requirements engineering methodology SREM (Alford-80)
- Structured analysis and design technique - SADT (Davis-90)
- Structured design - SD (Stevens-88)
- Structured systems analysis - SSA (Yourdon-90)
- System for application-oriented requirements specification - SARG (Hagemann-87)
- Tecnology for the automated generation of systems - TAGS (Sievert-85)
- Two-dimensional program design - TDPD (Rotenstreich-86)

- User software engineering - USE (Wasserman-86)
- Vienna development method - VDM (Cohen-86)

A avaliação dos métodos acima citados foi realizada com base num método proposto com este objetivo. Os conceitos básicos deste método de avaliação fundamentam-se na estrutura do Método de Avaliação da Qualidade de Software definido em (Rocha-83) e em um conjunto de características relacionadas a métodos, instrumentos e ferramentas.

Na avaliação de métodos de especificação foram considerados os seguintes aspectos:

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

Estes aspectos são detalhados a seguir: (Crispim-91)

UTILIZABILIDADE PARA CONSTRUÇÃO DO PRODUTO

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

O fator **aplicabilidade** avalia o campo de utilização de um método.

O fator **expressividade** avalia o método quanto aos aspectos relacionados à modelagem do problema.

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

O fator **facilidade de uso** avalia o grau de facilidade para construir o produto, considerando que o desenvolvedor conheça o método e entenda o problema.

A figura 5.9 apresenta os critérios relacionados a cada um desses fatores.

UTILIZABILIDADE PARA AVALIAÇÃO DO PRODUTO

Este objetivo é avaliado de acordo com os seguintes fatores: verificabilidade e validabilidade.

A **verificabilidade** preocupa-se com a qualidade do produto gerado, observando se a linguagem utilizada para esta representação foi utilizada corretamente.

A **validabilidade** está relacionada ao grau em que o produto gerado atinge os objetivos que motivaram seu desenvolvimento.

A figura 5.9 apresenta os critérios relacionados a cada um desses fatores.

UTILIZABILIDADE PARA GERÊNCIA DO PROCESSO DE DESENVOLVIMENTO

Este objetivo é avaliado através dos fatores: suporte para planejamento do projeto e suporte para acompanhamento do projeto.

O **suporte para planejamento do projeto** avalia um método quanto a sua capacidade de auxiliar no planejamento das tarefas a serem realizadas durante o processo de desenvolvimento de software.

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

A figura 5.9 apresenta os critérios relacionados a cada um desses fatores.

OBJETIVO UTILIZABILIDADE PARA CONSTRUÇÃO

FATOR: Aplicabilidade
Critérios: Adequação ao ciclo de vida
Adequação ao tipo de processamento
Adequação ao domínio da aplicação
Adequação ao nível de complexidade

FATOR: Expressividade
Critérios: 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
Critérios: Documentação para construção
Exigência de formação para construção
Existência de instrumentos cognitivos

FATOR: Facilidade de uso
Critérios: Existência de apoio automatizado
Automatizabilidade
Extensibilidade

OBJETIVO UTILIZABILIDADE PARA AVALIAÇÃO

FATOR: Verificabilidade
Critérios: Existência de normas de qualidade
Existência de ferramentas para verificação

FATOR: Validabilidade
Critérios: Compreensividade
Existência de ferramentas para validação

OBJETIVO UTILIZABILIDADE PARA GERÊNCIA

FATOR: Suporte para planejamento do projeto
Critérios: Suporte para estimativa de custos
Suporte para definição do cronograma
Suporte para estimativa de recursos

FATOR: Suporte para acompanhamento do projeto
Critérios: Suporte para acompanhamento de custos
Suporte para acompanhamento de cronograma
Suporte para controle de versões

Figura 5.9: Objetivos, fatores e critérios do modelo para avaliação de métodos (Crispim-91)

As tabelas 5.3 e 5.4 apresentam alguns dos resultados obtidos a partir da avaliação realizada. Na tabela I constam métodos pesquisados X instrumentos notacionais X automatização. 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 - instrumeto 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.

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 5.3: Comparação de métodos X instrumentos notacionais X automatização (Crispim-91)

Na tabela 5.4 consta o tipo de processamento X o ciclo de vida.

TIPO PROCESS/ CICLO DE VIDA	CLÁSSICO ANÁLISE	CLÁSSICO PROJETO	PROTÓTIPO	SÍNTESE	REUTILIZ
SEQUENCIAL	ESA JSD 00A OOS OOSA SADT SSA VDM	MBSD SD TDPD	SBRE TAGS		00A OOS
"ON LINE"	ESA 00A	MBSD	BOP PUI RIPL SBRE USE		00A
TEMPO REAL	DARTS JSD 00A		SBRE SREM SARS TAGS		00A

Tabela 5.4: Comparação de tipo de processamento X ciclo de vida (Crispim-91)

O conhecimento que consta da versão de XAMÃ implementada nesta tese, para indicar métodos de

especificação, leva em consideração os seguintes critérios abordados na tese de (Crispim-91):

- Com relação a aplicabilidade:
  - . tipo de processamento
  - . nível de complexidade

- Com relação a expressividade:
  - . abordagem de desenvolvimento utilizada
  - . grau de formalidade que os instrumentos notacionais do método possibilitam.

Na próxima versão de XAMÃ outros critérios deverão ser considerados.

A seguir cada um dos critérios relacionados acima é detalhado:(Crispim-91) (Mendes-89)

A) ABORDAGEM DE DESENVOLVIMENTO

Indica os princípios, os mecanismos e a forma de visualização utilizadas pelo método para a modelagem do problema.

- **Decomposição funcional:** O que se deseja representar é estruturado de acordo com suas funções e sub-

funções, sendo especificado o relacionamento entre cada uma das funções.

- **Fluxo de dados:** O que se deseja representar é modelado em termos dos fluxos de dados de entrada e saída e dos processos necessários para gerar os fluxos de dados.

- **Orientada a estrutura de dados:** Os métodos que seguem esta abordagem identificam inicialmente os objetos chaves do problema. A partir daí são identificadas as operações a serem realizadas.

- **Modelagem da informação:** O que se deseja representar é modelado em termos de entidade e relacionamentos.

- **Orientação a objetos:** A idéia desta abordagem é mapear o que se deseja representar para um mundo composto por objetos. Um objeto é uma entidade que consiste de uma estrutura de dados (atributos) e de todas as operações (métodos) que podem manipular esses dados.

- **Baseada em conhecimento:** O que se deseja representar é modelado a partir de recursos que representam o conhecimento. Não são representados os aspectos que envolvem processamento.

- **Abstração axiomática:** O que se deseja representar é modelado através de asserções que devem ser verdadeiras antes e depois da execução de qualquer comando.

- **Abstração operacional:** O que se deseja representar é modelado através de um interpretador para definir a semântica de uma linguagem. Assim, a computação que transforma entradas em saídas é descrita explicitamente através dos estados pelos quais passa uma máquina abstrata (interpretador).

- **Abstração baseada em modelo abstrato:** O que se deseja representar é mapeado diretamente no seu significado que é chamado denotação. Uma denotação é um valor matemático tal como um número ou uma função.

B) GRAU DE FORMALIDADE

A diferença básica entre especificações semi-formais e formais está relacionada à forma como é definida a linguagem na qual a especificação é desenvolvida. Desta forma é interessante que haja um entendimento de dois dos diferentes aspectos de uma linguagem: estrutura e significado.

O aspecto estrutural da linguagem é chamado "sintaxe". Trata da combinação dos signos sem considerar seu significado.

O aspecto do significado é conhecido como "semântica". Trata do significado dos signos.

Uma especificação é semi-formal quando utiliza uma linguagem que pode ou não possuir uma sintaxe bem definida mas certamente não tem uma semântica precisa, permitindo que uma série de erros não sejam detectados.

Uma especificação é formal quando a linguagem utilizada possue a sintaxe e a semântica rigorosamente definidas.

C) NÍVEL DE COMPLEXIDADE

A programação de um sistema grande e complexo é muito diferente da programação de um problema específico que pode ser representado num único programa. Desta forma recursos que são adequados para desenvolvimento de programas não necessariamente podem ser utilizados quando se trata de um sistema complexo. Consideramos quanto ao nível de complexidade: desenvolvimento de programas e desenvolvimento de sistemas.

D) TIPO DE PROCESSAMENTO

Consideramos como tipos de processamento: com concorrência e sem concorrência.

Um sistema concorrente deve ser visto como um conjunto de processos, cada processo sendo representado por uma unidade de programa.

Processos são concorrentes se suas execuções podem (conceitualmente) se sobrepor no tempo.

Frequentemente processos são interativos. A interação resulta de uma das duas razões seguintes:

- **Competição:** processos competem pelo acesso a um determinado recurso compartilhado, que deve ser usado em exclusão mútua (por exemplo, uma impressora de linhas).

- **Cooperação:** processos cooperam para atingir um objetivo comum.

Processos interagem corretamente somente se certa relação de precedência vale entre suas ações elementares, isto é, determinadas ações precisam preceder outras.

Competição correta exige que nunca duas atualizações de uma variável compartilhada "t" sejam executadas ao mesmo tempo.

Linguagens para programação concorrente provêm mecanismos linguísticos para definição de processos concorrentes e instruções de sincronização adequadas para garantir a ordem parcial requerida entre as ações.

CONHECIMENTO SOBRE MÉTODOS

As regras da base de conhecimento sobre métodos para a especificação foram elaboradas a partir das seguintes informações:

a) Caso o usuário planejador do ADS esteja pensando em utilizar a abordagem orientada a objetos, pode-se indicar alguns métodos independentemente de qualquer outra avaliação dos critérios que estão sendo analisados. Métodos como OOA (Coad-90), OOS (Bailin-89) e o método proposto em (Mattoso-90) podem ser indicados.

Deve-se deixar claro, no entanto, que métodos para especificação orientados a objeto ainda não estão suficientemente maduros.

b) Caso o usuário planejador do ADS esteja pensando em utilizar uma abordagem baseada no conhecimento pode-se indicar métodos como KBSDLC (Weitzel-89) e KBSDM (Keller-87).

c) Para sistemas com abordagens diferentes da orentação a objetos e baseada no conhecimento, deve-se avaliar o nível de complexidade. Ou seja, se o que se deseja especificar é um programa ou um sistema.

d) Deve-se avaliar então se é indicada a utilização de um método formal ou semi-formal. Métodos formais são indicados quando:

- a especificação é de um programa e há experiência com o uso do método ou o método já foi avaliado e foi considerada a sua adequação.

- a especificação é de um programa e não há experiência com o uso do método mas o método foi avaliado e considerada sua adequação.

- a especificação é de um sistema e há experiência com o uso do método e com o tipo de aplicação.

- a especificação é de um sistema e o sistema a ser desenvolvido é de um tipo onde não podem ocorrer falhas após a sua implantação.

- a especificação é de um sistema e o método de especificação a ser utilizado já foi avaliado e considerada sua adequação e há interesse em se fazer uma experiência independente dos custos.

Se nenhuma das condições acima for verdadeira deve ser utilizado um método semi-formal.

e) Alguns dos métodos formais para especificação de sistemas possuem recursos para tratar concorrência. Redes

de Petri (Davis-90) e Estelle (ISO-85)(New-90) são exemplos de métodos formais que tratam concorrência. VDM (Cohen-86) e OBJ (Goghen-86) são exemplos de métodos formais que não tratam concorrência.

f) Métodos formais para especificação de programas podem ter como abordagem de desenvolvimento:

- abstração axiomática. Exemplo é o método apresentado em (Hoare-69).

- abstração operacional. Exemplo é o método apresentado em (McCarthy-63).

- abstração baseada em modelo abstrato. Exemplo é o VDM (Cohen-86).

g) Métodos semi-formais para a especificação de programas podem ter as seguintes abordagens de desenvolvimento:

- decomposição funcional textual com restrição: Exemplos são o pseudo-código e o português estruturado (Gane-84).

- decomposição funcional gráfica livre: Exemplo é o SD (Stevens-88).

- posicionais: Exemplo é a árvore de decisão e a tabela de decisão (Gane-84).

h) Alguns dos métodos semi-formais para especificação de sistemas possuem recursos para tratar

concorrência. DARTS (Gomaa-84), SREM (Alford-80) e JSD (Pressman-88) são exemplos.

i) Métodos semi-formais que não tratam a concorrência podem ter uma abordagem de desenvolvimento de fluxo de dados, como SSA (Yourdon-90) ou uma abordagem orientada a estrutura de dados como o ERM (Chen-90).

V.2.3 ESPECIFICAÇÃO

A solução proposta nos ítens V.2.1 e V.2.2 para a realização de planejamento de ADSs é descrita a seguir de modo mais formal. A especificação foi realizada utilizando como método a Análise Estruturada (Gane-84).

O item V.2.3.1 apresenta o meta-ambiente no qual está inserido o processo "Planejar ADS" de forma a se observar o relacionamento deste processo com os demais elementos do meta-ambiente. E o item V.2.3.2 apresenta a especificação de "Planejar ADS".

V.2.3.1 META-AMBIENTE

O Meta-ambiente do TABA tem como objetivo gerar o ADS necessário ao desenvolvimento de um software particular.

O recurso que utiliza para gerar ADSs é reunir uma série de componentes, que juntos possibilitarão o

desenvolvimento de um sistema de software. Em (Hoffnagle-85) e (Giavitto-90) são apresentados exemplos de meta-ambientes que vêm sendo desenvolvidos utilizando esta mesma abordagem.

O meta-ambiente é composto por seis processos principais (figura 5.10), descritos a seguir:

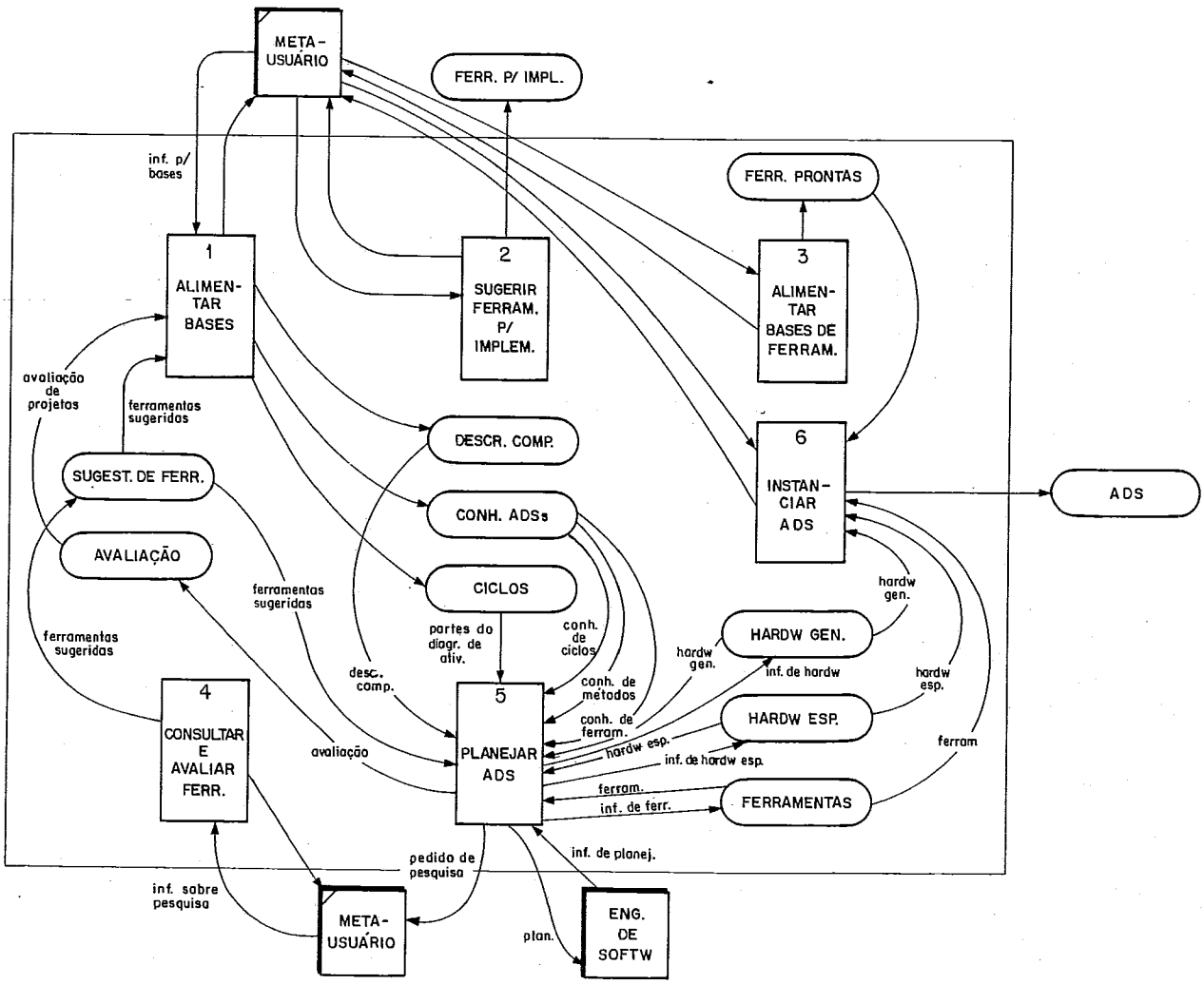


Figura 5.10: Diagrama de fluxo de dados do meta-ambiente

Proc.1: Alimentar bases

Tem como objetivo armazenar em bases de dados e de conhecimento as informações que serão utilizadas ao se planejar um ADS. Há duas bases de dados e uma de conhecimento:

- descrição de componentes: base de dados com informações sobre as características de métodos e ferramentas;

- ciclos de vida: base de dados com sequências de atividades, cada uma representando um ciclo de vida;

- conhecimento para indicação de elementos do ADS: base de conhecimento com informações que indicam a adequação de se utilizar determinados ciclos de vida, métodos e ferramentas.

As informações são armazenadas nas bases pelo meta-usuário depois que houver um refinamento dos dados obtidos a partir:

- de avaliações do ADS e seus componentes, realizadas ao fim do desenvolvimento de cada projeto;

- do resultado de consulta a empresas seguida de avaliação dos componentes ou ADSs sugeridos para um caso particular de desenvolvimento.

Proc.2: Sugerir ferramentas para implementação

Tem como objetivo armazenar os nomes de ferramentas a serem incluídas no TABA. Essas idéias podem vir a partir de:

- métodos que foram utilizados, se mostraram adequados e poderiam se tornar melhores se fossem automatizados;

- idéias do meta-usuário.

Proc.3: Alimentar base de ferramentas

Tem como objetivo incluir no TABA ferramentas de software. Pode ser uma ferramenta adquirida no mercado como também gerada pelo próprio TABA em seu ambiente de desenvolvimento.

Proc.4: Consultar e avaliar ferramentas

Quando no planejamento do ADS não se tem subsídios suficientes (por parte do engenheiro de software ou das bases de dados e de conhecimento) para indicar uma ou mais ferramentas de software, é necessário então consultar o

mercado de forma a obter um conjunto de ferramentas adequadas ao caso. Neste processo está incluído o apoio a esta pesquisa assim como uma posterior avaliação.

Proc.5: Planejar ADS

Tem como objetivo gerar a especificação do ADS a ser utilizado na construção de um determinado software. Esta tarefa é realizada a partir da consulta a "descrição de componentes", "ciclos de vida", "conhecimento para indicação de elementos do ADS" e "sugestão de ferramentas".

A especificação do ADS inclui o hardware e as ferramentas utilizadas, informações estas que estão armazenadas em "hardware genérico", "hardware específico" e "ferramentas da atividade".

Como durante o desenvolvimento o ADS pode mudar, são armazenadas versões dos diferentes ADSs juntamente com as informações que caracterizam o projeto, levando à indicação da versão e uma explicação do que motivou a mudança. Ao final do desenvolvimento do projeto é realizada uma avaliação com base neste histórico gerado durante a realização do projeto.

Proc.6: Instanciar ADS

Tem como objetivo gerar a partir da especificação do ADS, o ambiente operacional (armazenado em "ADS"). De forma a dar apoio ao uso deste ADS é gerado um assistente, que será incorporado ao ADS.

O ADS é composto a partir das ferramentas que estão disponíveis no TABA.

V.2.3.2 PLANEJAR ADS

DESCRIÇÃO DOS PROCESSOS

"Planejar ADS" é detalhado nos diagramas das figuras 5.11, 5.12 e 5.13. A seguir são descritos os processos que são expansão de "Planejar ADS".

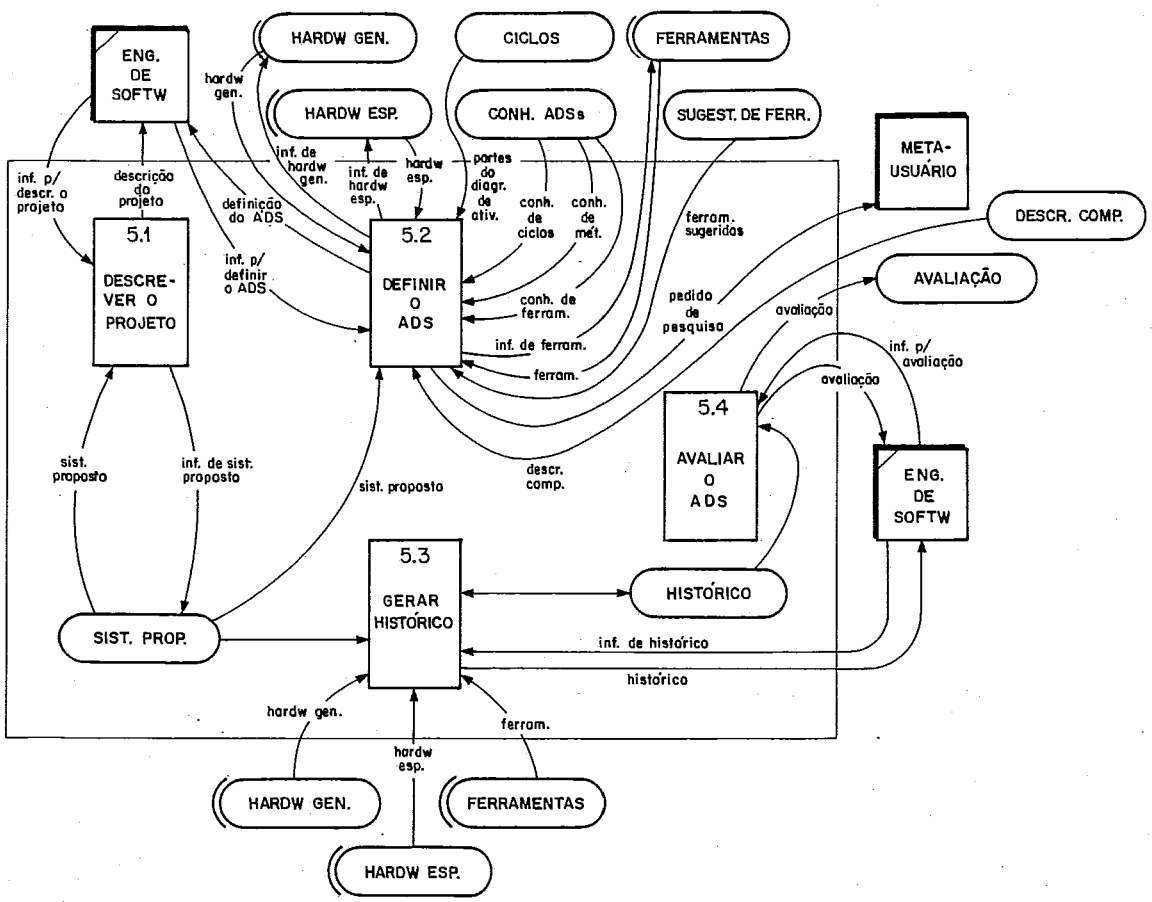


Figura 5.11: Diagrama de fluxo de dados "Planejar ADS".

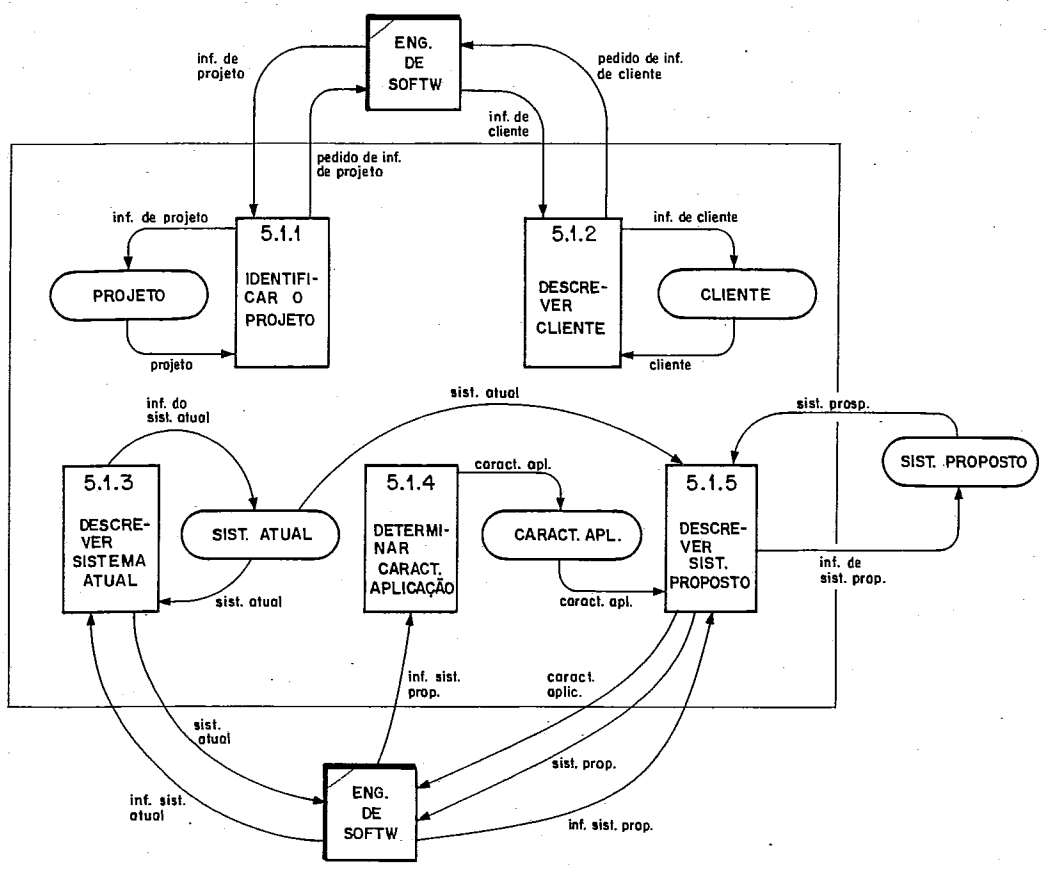


Figura 5.12: Diagrama de fluxo de dados "Descrever o projeto".

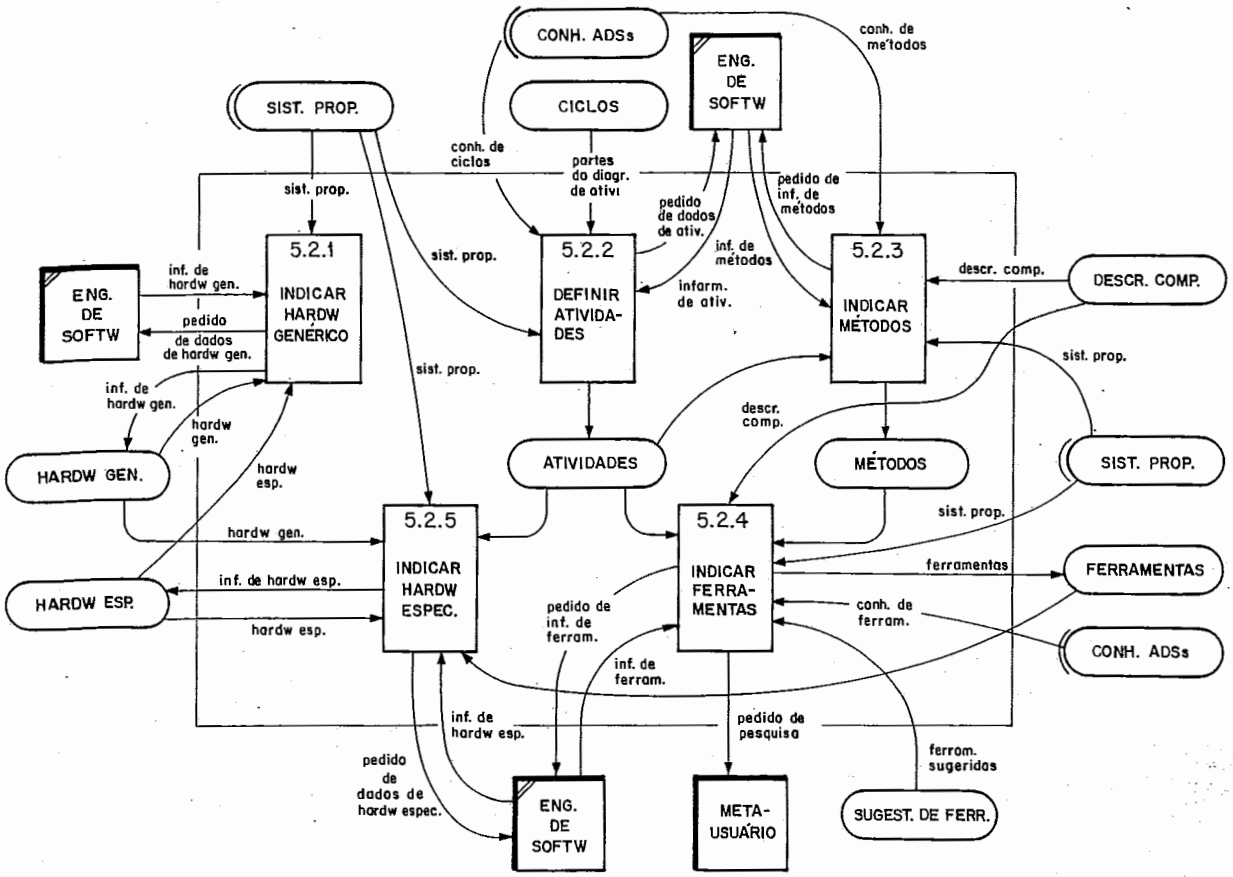


Figura 5.13: Diagrama de fluxo de dados "Definir o ADS".

Proc. 5.1: Descrever o projeto.

É composto por:

Proc. 5.1.1: Identificar projeto:

Tem como objetivo a descrição das informações como nome do projeto e um resumo de seu objetivo.

Proc. 5.1.2: Descrever cliente:

Tem como objetivo a descrição do nome do cliente, seu endereço, atividade da empresa, telefone, fax e endereço eletrônico.

Proc. 5.1.3: Descrever sistema atual:

Tem como objetivo a descrição do sistema que venha sendo utilizado atendendo mesmo que parcialmente aos objetivos do sistema proposto.

Proc. 5.1.4: Determinar características da aplicação

Tem como objetivo determinar as características da aplicação a partir das características de sistemas semelhantes.

Proc. 5.1.5: Descrever sistema proposto

A partir das informações obtidas como a análise do sistema atual e sabendo-se as características de sistemas do mesmo tipo, é realizada a descrição do sistema que se pretende desenvolver.

Proc. 5.2: Definir o ADS

É composto por:

Proc. 5.2.1: Indicar hardware genérico

Tem como objetivo a descrição do hardware que independe de ferramentas específicas. Deve ser consultado "sistema proposto".

Proc. 5.2.2: Definir atividades

Tem como objetivo a definição das atividades do projeto. Para isso pode-se consultar "conhecimento de ADSs" de forma a se ter a indicação de uma sequência de atividades que seja adequada ao problema que está sendo tratado. Pode-se também copiar de "ciclos" uma sequência de atividades adequada ao caso. Deve ser consultado "sistema proposto".

Proc. 5.2.3: Indicar métodos

Para cada atividade há métodos que podem ser utilizados. Este processo tem como objetivo a indicação de métodos para cada atividade. Deve ser consultado "sistema proposto". Pode-se consultar "conhecimento de ADSs" assim como se necessário pode ser consultado "descrição de componentes". É possível que uma atividade não seja apoiada por um método.

Proc. 5.2.4: Indicar ferramentas de software

Tem como objetivo a indicação das ferramentas de cada atividade. As ferramentas podem ou não estar relacionadas a um método. Por este motivo "métodos" deve ser consultado. Deve, também, ser consultado "sistema proposto". Pode-se consultar "conhecimento de ADSs". Caso seja necessário é pedido ao meta-usuário que pesquise ferramentas possíveis de serem usadas. Estas informações são armazenadas em "ferramentas sugeridas" que depois é consultado. É possível que uma atividade não utilize nenhuma ferramenta. Neste caso pode ser apoiada por um método sem nenhum apoio automatizado.

Proc. 5.2.5: Indicar hardware específico

Tem como objetivo permitir a indicação do hardware que está diretamente relacionado a uma ferramenta indicada. Por este motivo devem ser consultadas todas as bases de dados que contenham indicações de ferramentas, além de "hardware genérico".

Proc. 5.3: Gerar o histórico

Durante o desenvolvimento do projeto são geradas várias versões de ADSs. Este processo tem como objetivo

armazenar cada versão juntamente com a explicação do que motivou a sua substituição.

Proc. 5.4: Avaliar ADS

Tem como objetivo gerar, ao final do projeto, uma avaliação do que ocorreu durante o desenvolvimento a partir da observação dos dados de "histórico", "sistema proposto" e "características do projeto".

DESCRIÇÃO DOS DEPÓSITOS DE DADOS

- Atividades

Para cada atividade há as seguintes informações: nome, objetivo, data de início, duração, nomes das atividades anteriores.

- Avaliação

Contém a avaliação, uma informação textual.

- Características da aplicação

Contém uma relação de características da aplicação.

- Ciclos de vida

Para cada ciclo de vida há as seguintes informações: nome e o grafo que representa as atividades do ciclo e seus relacionamentos.

- Cliente

Contém as seguintes informações relacionadas ao cliente: nome, endereço, atividade da empresa, telefone, fax e endereço eletrônico.

- Conhecimento para indicação de elementos do ADS

Contém informações no formato de regras que indicam a adequação de se utilizar determinados ciclos de vida, métodos e ferramentas.

- Descrição de componentes

Contém informações sobre as características de métodos e ferramentas. Para cada um desses componentes são dadas informações como: nome, desenvolvedor, empresa que comercializa, projetos desenvolvidos utilizando o componente, etc.

- Ferramentas

Para cada ferramenta há as seguintes informações: nome, tipo, condição (já adquirido, vai ser comprado, vai ser alugado) e detalhamento (uma breve informação que se deseje armazenar).

- Hardware específico

Para cada hardware há as seguintes informações: nome, quantidade, condição (já adquirido, vai ser comprado, vai ser alugado) e detalhamento (uma breve informação que se deseje armazenar).

- Hardware genérico

Para cada hardware há as seguintes informações: nome, quantidade, condição (já adquirida, vai ser comprada, vai ser alugada) e detalhamento (uma breve informação que se deseje armazenar).

- Histórico

Para cada versão de ADS há as seguintes informações: nome, a própria versão, as informações que caracterizam o projeto e que levaram à indicação da versão, o motivo da substituição.

- Métodos

Para cada método há as seguintes informações: nome, tipo e detalhamento (uma breve informação que se deseje armazenar).

- Projeto

Contém o nome e o objetivo do projeto.

- Sistema atual

Contém uma descrição textual do sistema atual.

- Sistema proposto

Contém uma descrição textual do sistema que se pretende desenvolver.

- Sugestões de ferramentas

Para cada ferramenta há as seguintes informações: nome, características (conjunto de dados sobre as ferramentas não detalhado aqui), avaliação.

V.3 ASPECTOS DA IMPLEMENTAÇÃO

"XAMÃ: Planejador de ADSs" é o nome da ferramenta desenvolvida nesta tese que implementa parte do que foi proposto para "Planejar ADS". As limitações são as seguintes:

- Quanto a definição do ciclo de vida: Não foi desenvolvido o editor gráfico que possibilita a edição dos diagramas de atividade. O diagrama é apresentado ao usuário como uma lista de atividades com todas as informações necessárias para que o usuário, caso deseje, desenhe o diagrama à mão.

- Quanto à reutilização de informações de planejamento de ADSs: não foram implementados os recursos necessários para possibilitá-la.

- Quanto ao suporte à decisão: Não foi incluído nas bases de dados e de conhecimento todo o conhecimento já pesquisado sobre métodos de especificação, ciclos de vida e características de aplicação. Não foi implementada a base de dados com informações sobre métodos e ferramentas.

A figura 5.14 apresenta a arquitetura do que foi implementado e a seguir os componentes dessa arquitetura são descritos.

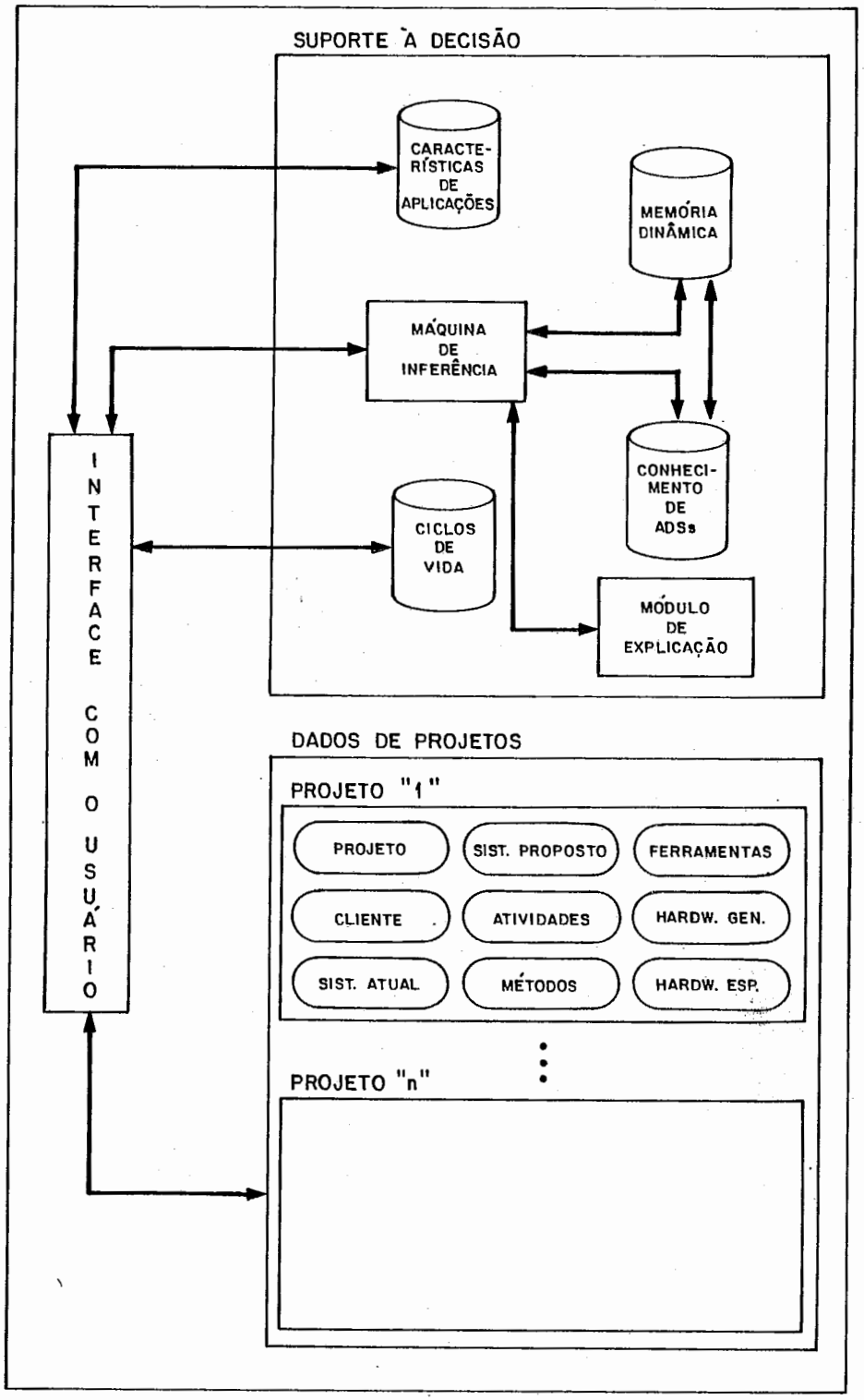


Figura 5.14: Arquitetura de "Planejar ADS".

V.3.1 INTERFACE COM O USUÁRIO

Foi desenvolvida utilizando-se o Xview e C. A figura 5.15 apresenta o diagrama de transição de estados que possibilita a descrição das principais janelas e seus relacionamentos. Cada estado neste diagrama representa uma janela. As janelas são apresentadas nas figuras 5.16 a 5.24.

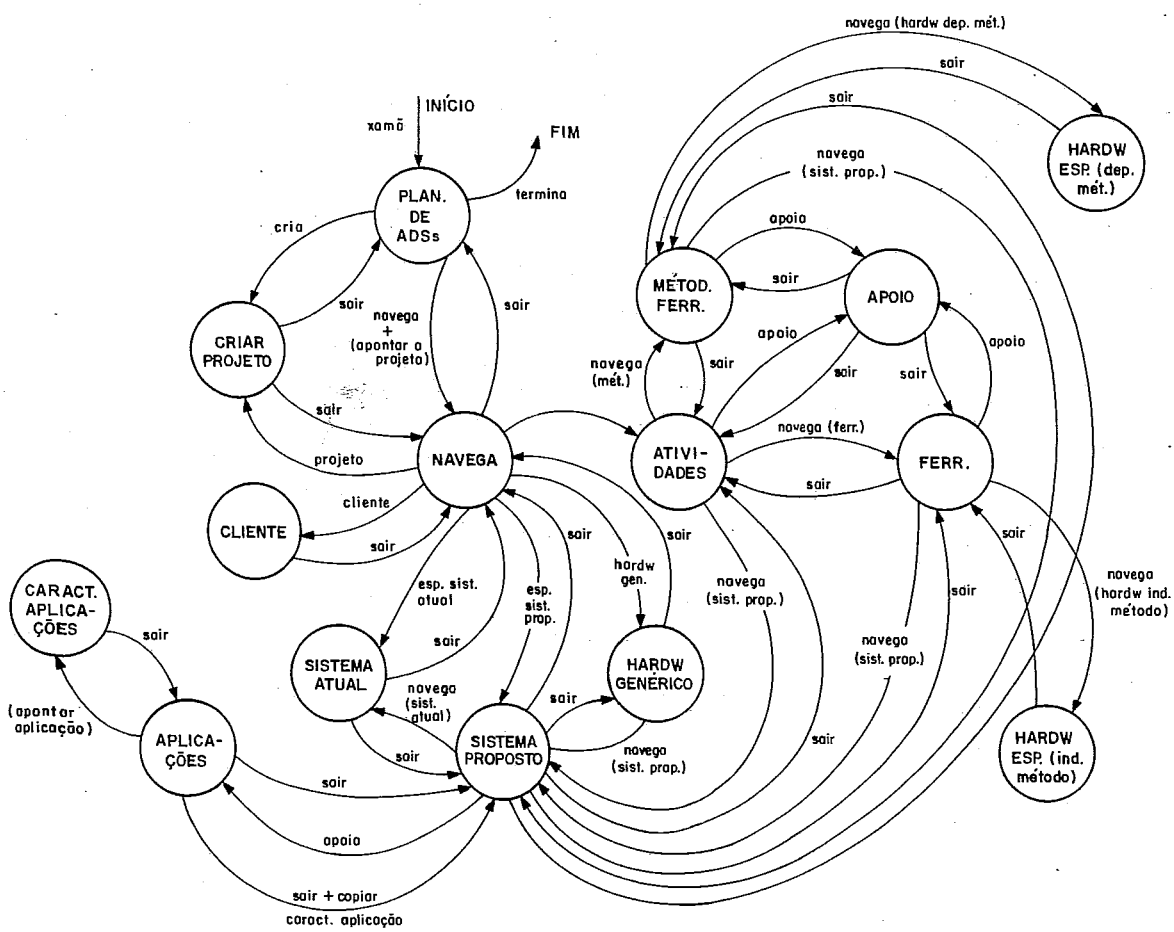


Figura 5.15: Diagrama de transição de estados.

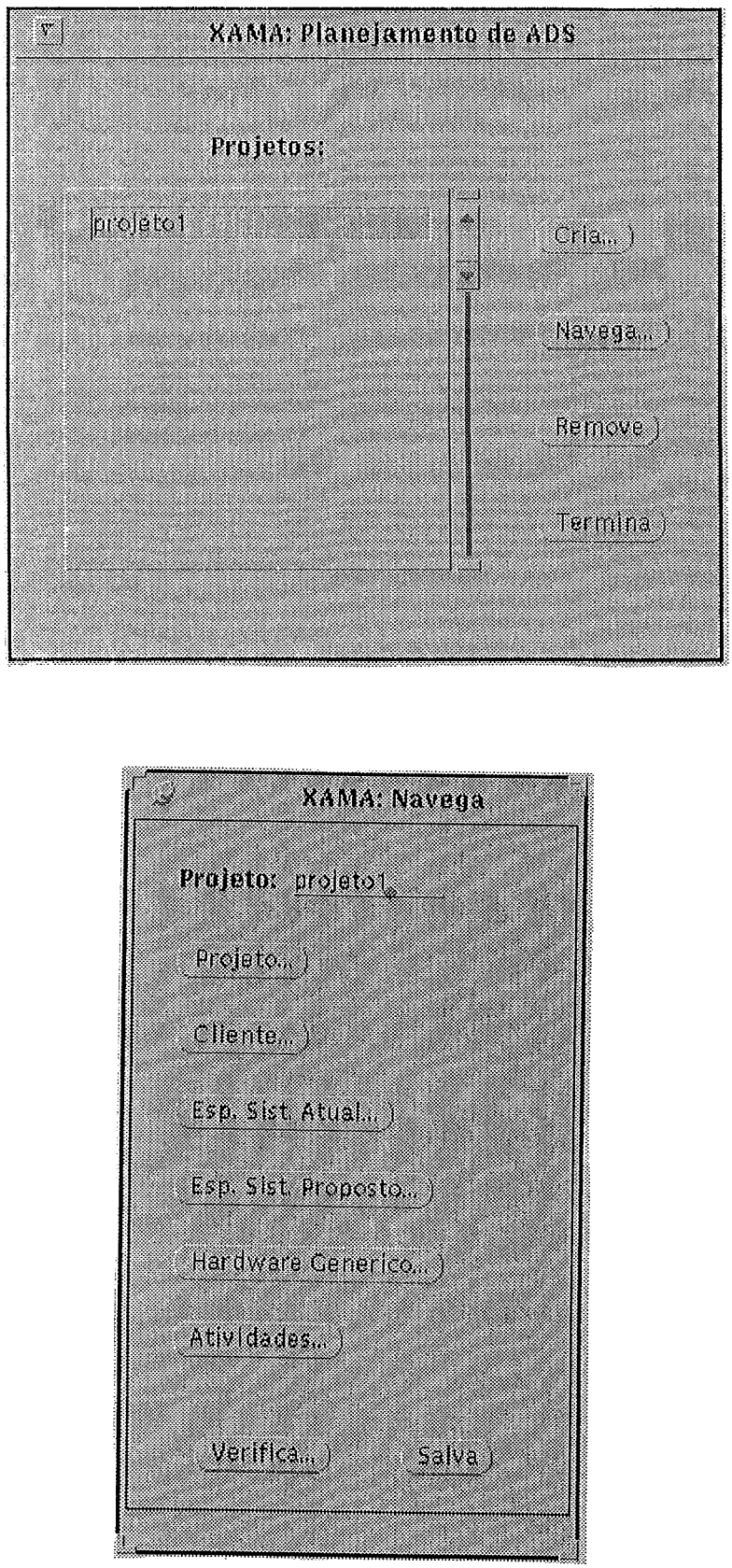
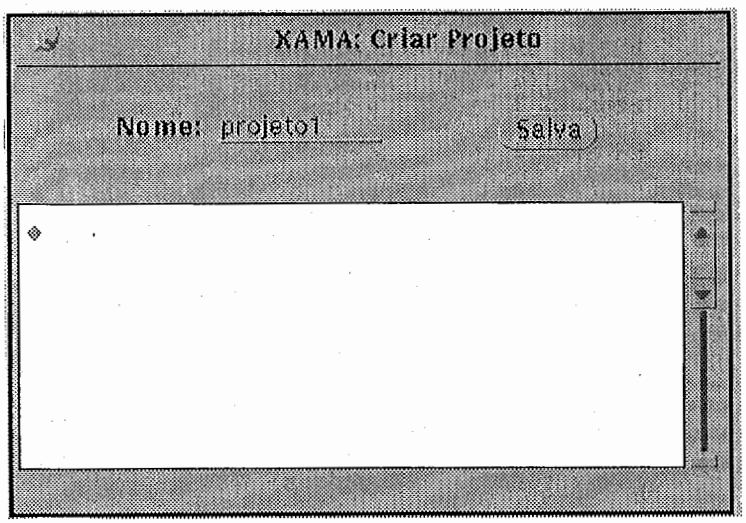


Figura 5.16: Janelas Navega e Planejamento de ADS



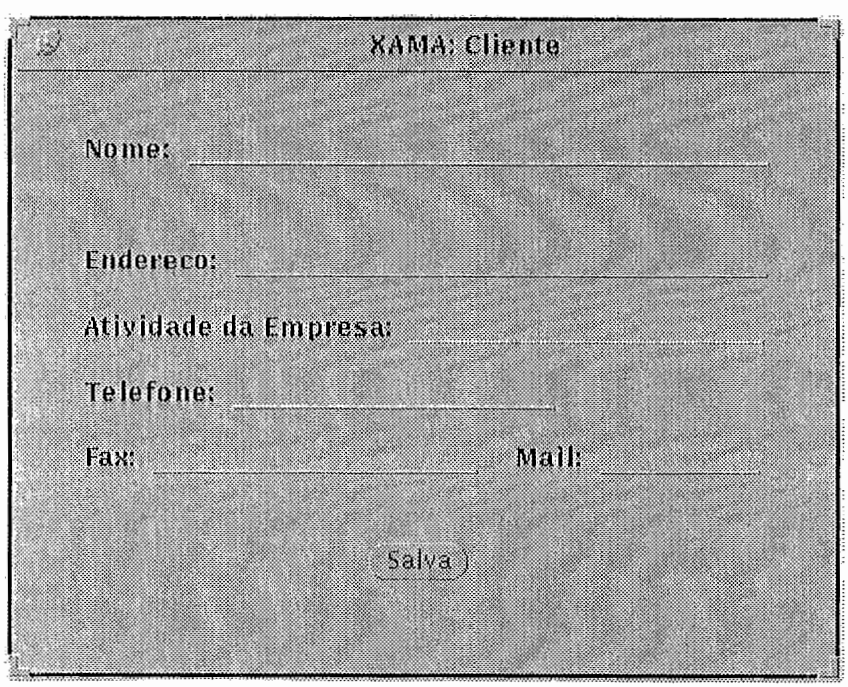
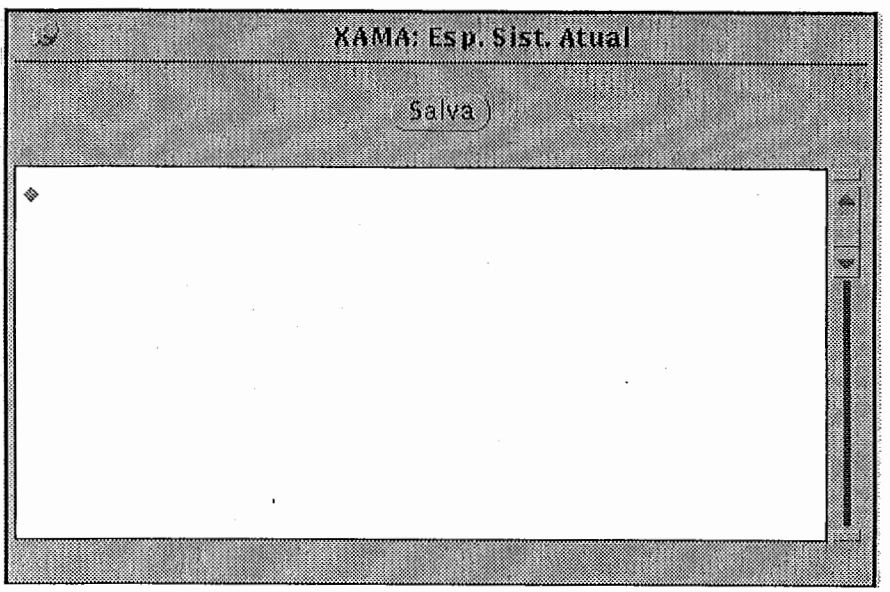


Figura 5.17: Janelas Criar Projeto e Cliente



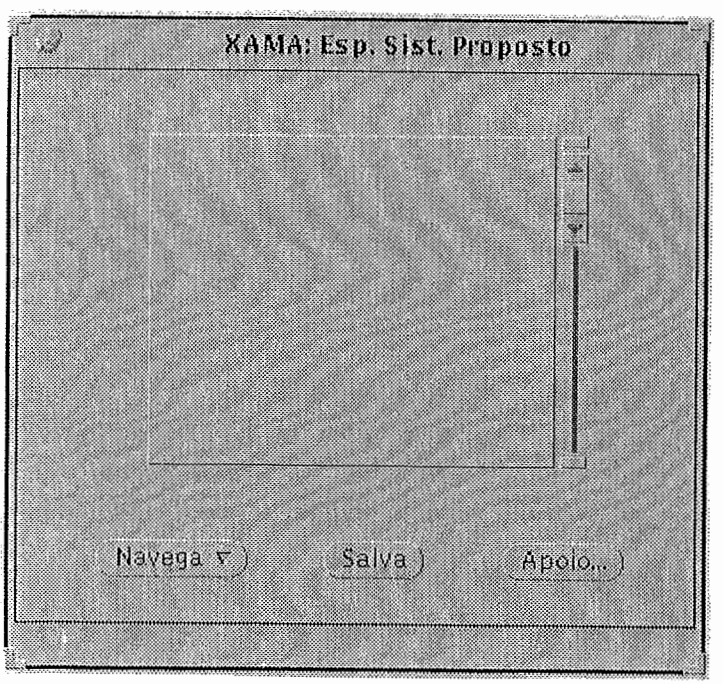
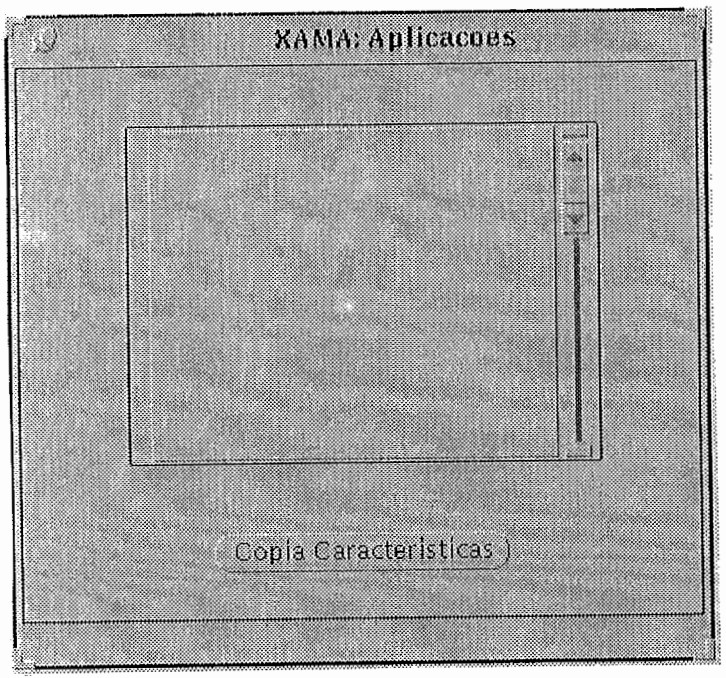


Figura 5.18: Janelas Sistema Atual e Sistema Proposto



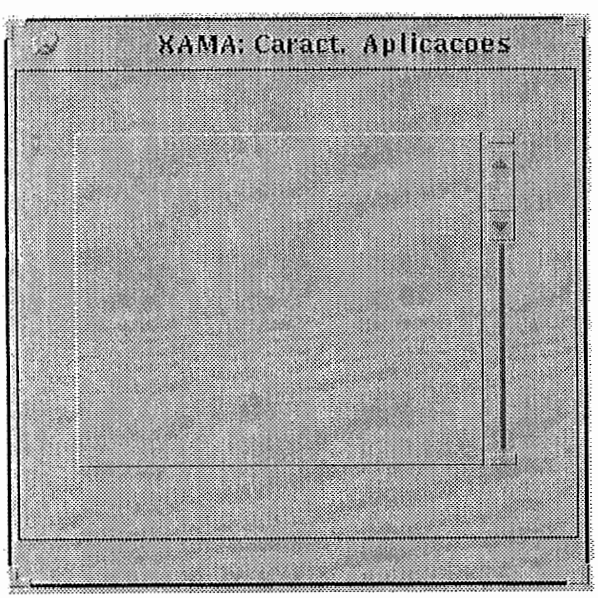


Figura 5.19: Janelas Aplicações e Características de Aplicações

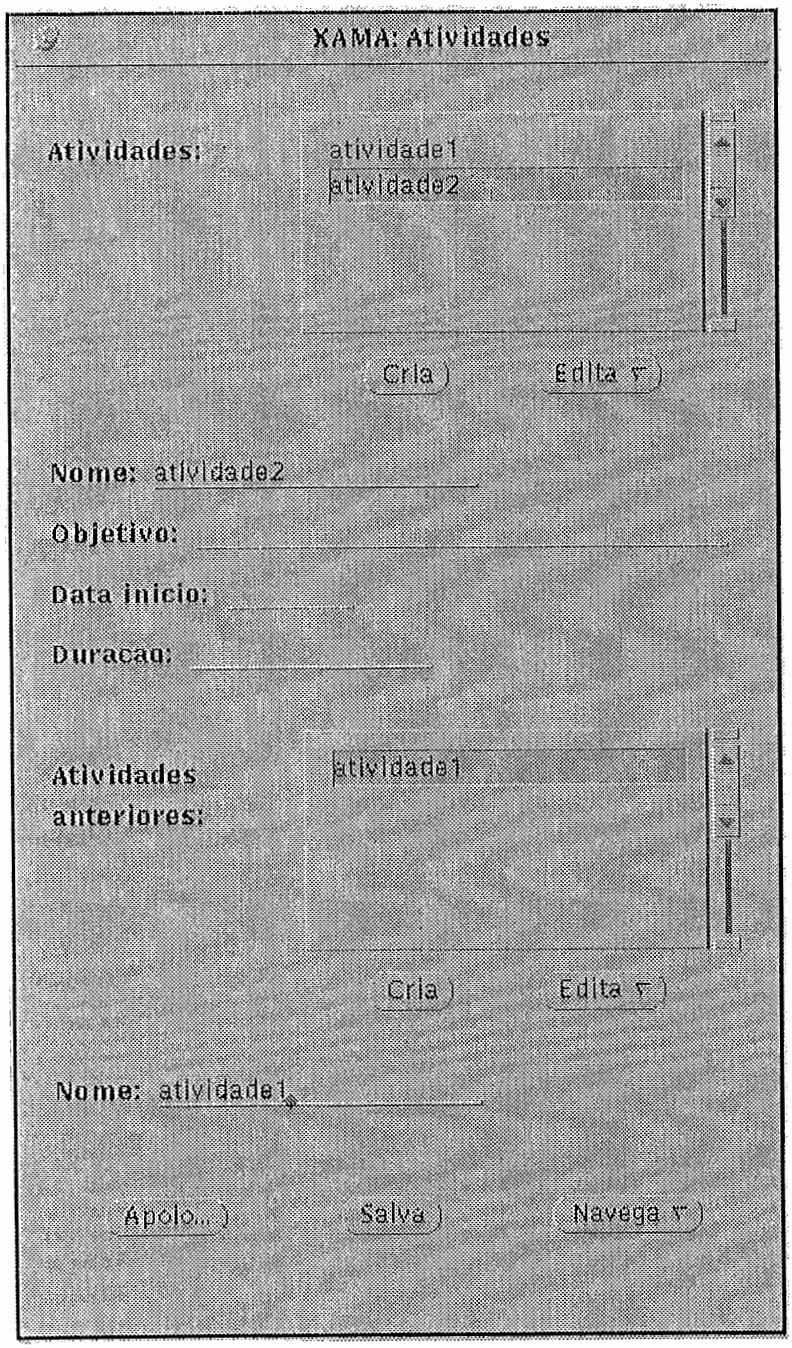


Figura 5.20: Janela Atividades

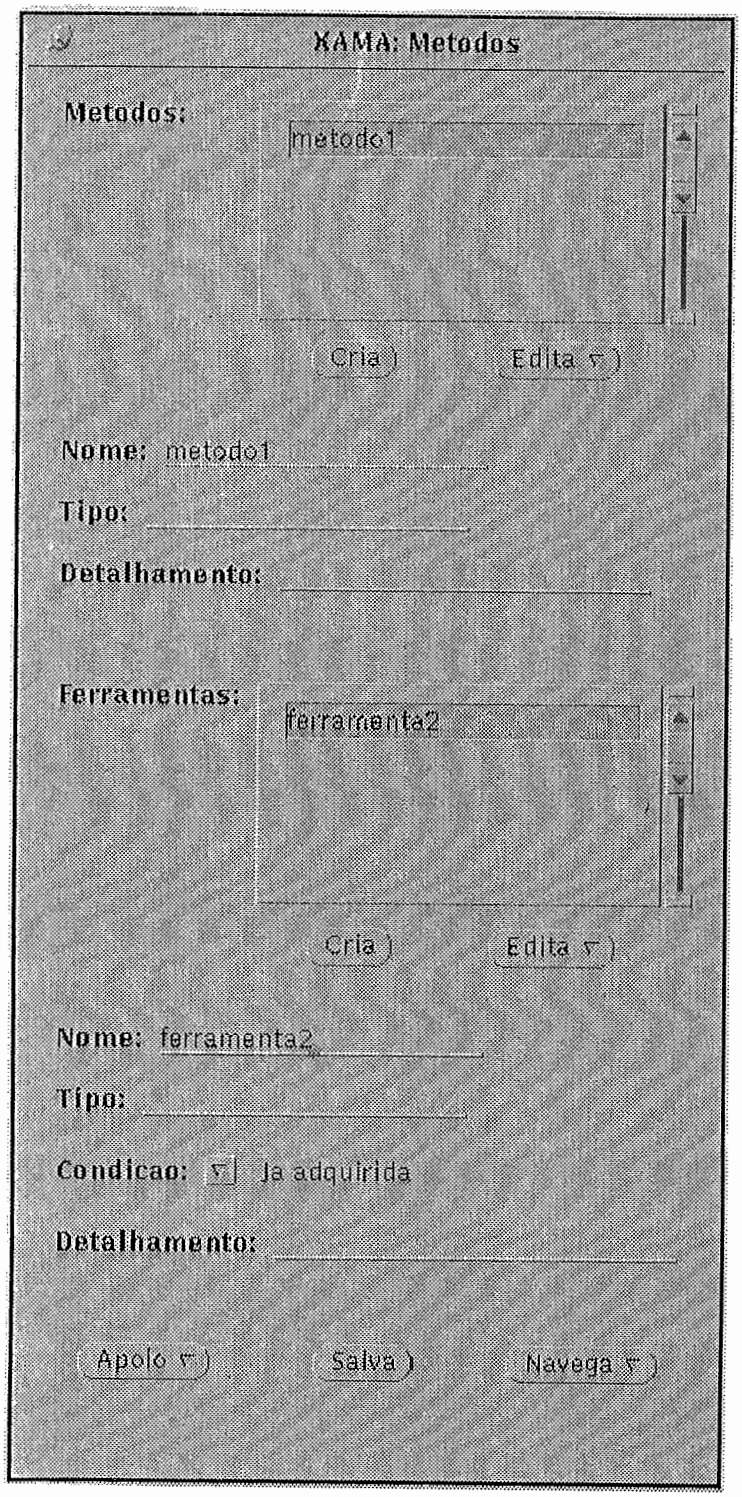


Figura 5.21: Janela Métodos

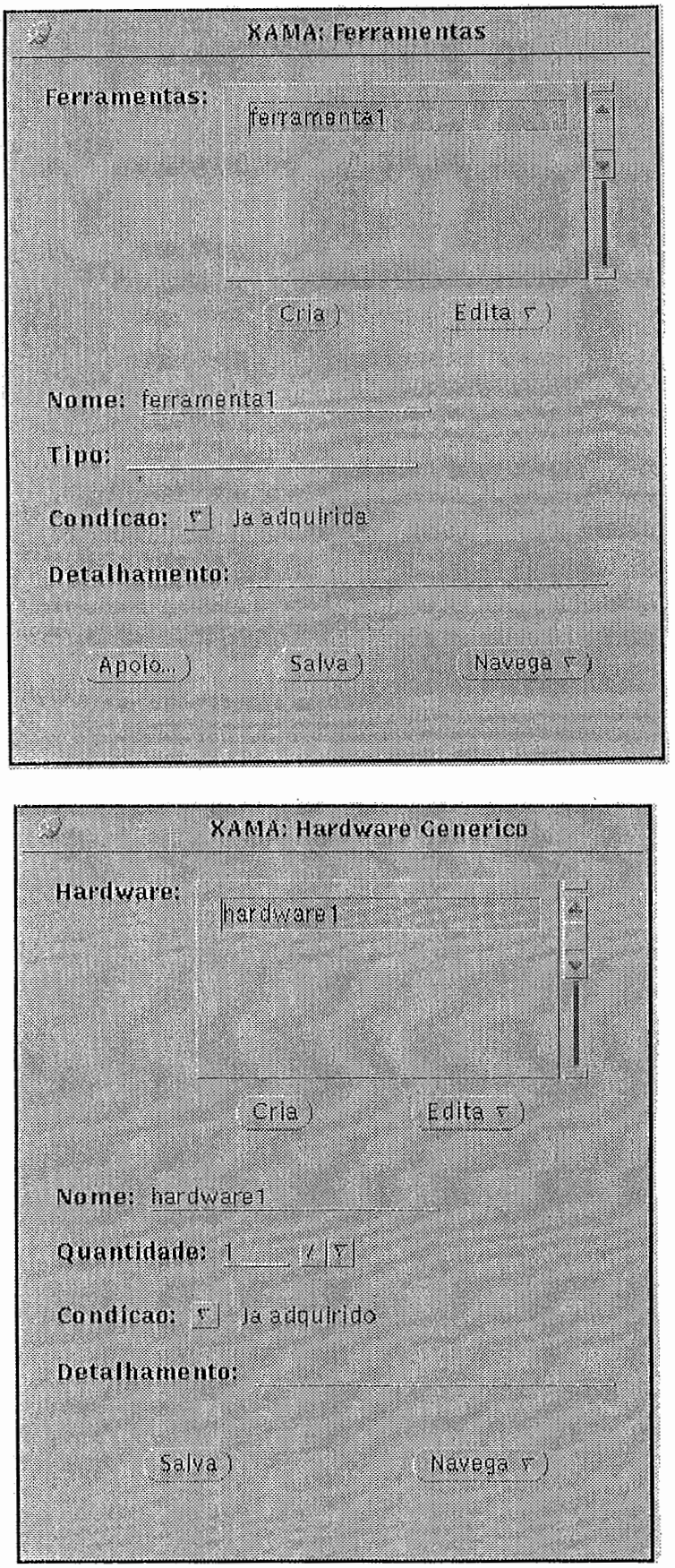


Figura 5.22: Janelas Hardware genérico e Ferramentas

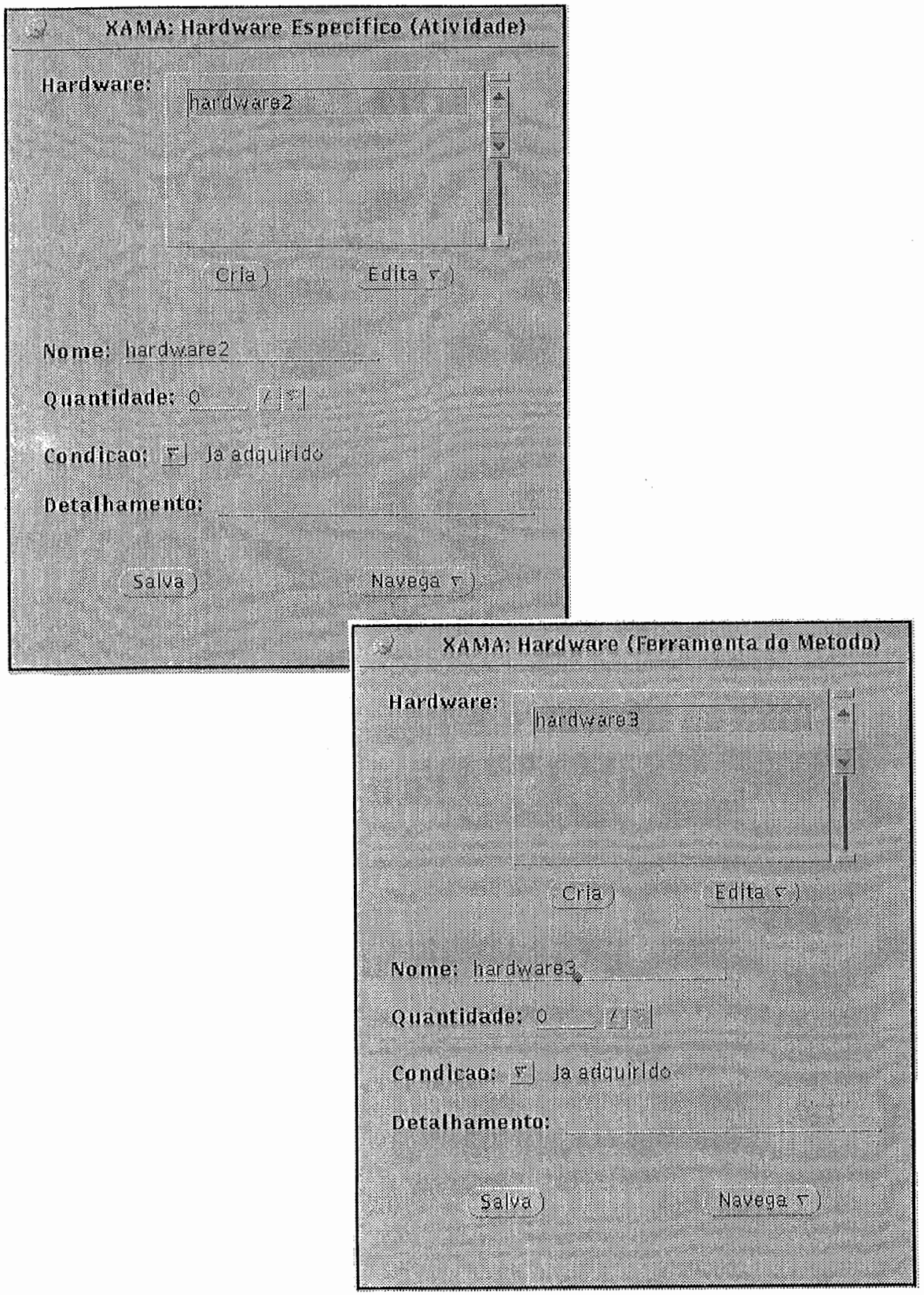


Figura 5.23: Janelas Hardware específico dependente de método e Hardware específico independente de método

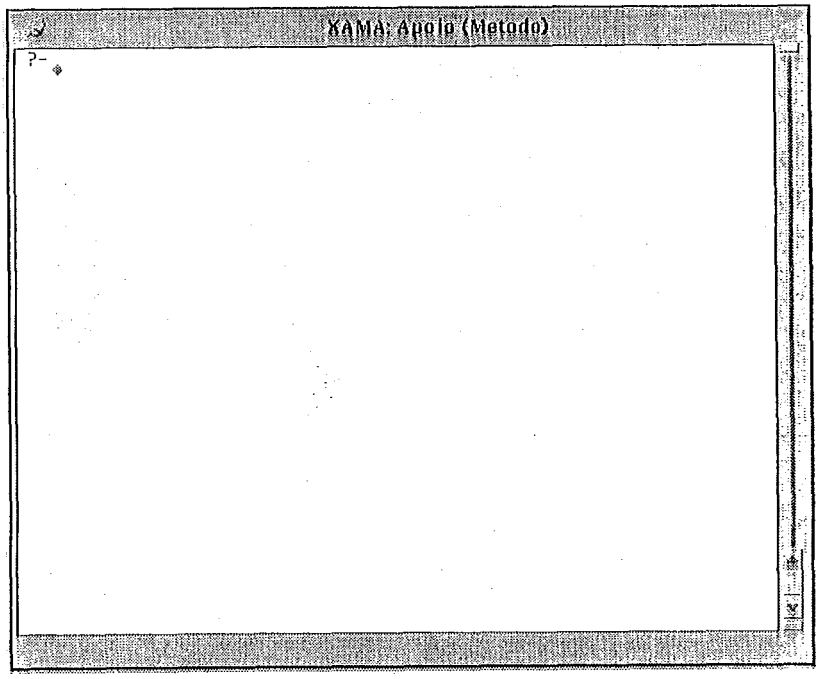


Figura 5.24: Janela Apoio

V.3.2 MÁQUINA DE INFERÊNCIA

Contém os métodos de raciocínio utilizados na solução de problemas. Como foi utilizado o Prolog, a máquina de inferência está baseada na lógica - cálculo de predicados de primeira ordem.

V.3.3 MÓDULO DE EXPLICAÇÃO

Após o sistema ter dado uma resposta a uma determinada consulta é possível saber os "fatos" que levaram àquela resposta. O módulo de explicação é o responsável pela apresentação ao usuário dessas informações.

V.3.4 MEMÓRIA DINÂMICA

É composta pelas respostas fornecidas pelo usuário às perguntas realizadas pelo sistema ao longo de uma consulta. Estas respostas são os "fatos" que podem ser reutilizados durante uma consulta à base.

Nesta versão de XAMÃ as informações que constam da memória dinâmica não são guardadas para posteriores consultas, uma vez que o usuário pode acessar a base de conhecimento, em momentos diferentes, enfocando aspectos diversos de um mesmo projeto.

V.3.5 BASE DE CARACTERÍSTICAS DE APLICAÇÕES

As informações contidas na base de características de aplicações foram obtidas a partir do trabalho apresentado no item V.2.2.1 desta tese.

V.3.6 BASE DE CICLOS DE VIDA

A idéia desta base é possibilitar a composição de um ciclo de vida a partir de um ou mais ciclos de vida que estão armazenados nesta base. O ciclo gerado deverá ainda sofrer alterações de forma a se adaptar às particularidades do projeto em questão.

Pretende-se que cada ciclo tenha o formato de um diagrama de atividades, representado como um grafo. Nesta versão de XAMÃ, no entanto, não foi desenvolvido o editor gráfico que permite a manipulação dos diagramas de atividade. Os recursos disponíveis permitem no entanto a geração deste diagrama manualmente, uma vez que todas as informações para sua construção são armazenadas, como por exemplo a identificação das atividades anteriores a cada atividade.

Os ciclos de vida armazenados são os descritos no item V.2.2.2 desta tese.

V.3.7 BASE DE CONHECIMENTOS PARA INDICAÇÃO DE ELEMENTOS DE ADSs

É na base de conhecimento que está contido o conhecimento que auxilia o usuário a especificar o ADS.

A obtenção das informações que fazem parte da base de conhecimento foi realizada a partir dos trabalhos

descritos no item V.2.2 desta tese além de terem sido consultados especialistas.

O conhecimento que está descrito em regras-Prolog, pertence as seguintes áreas da Engenharia de Software:

- ciclos de vida;
- métodos de especificação.

Na pesquisa deste conhecimento foram identificados tipos de informações que atuam sobre a escolha de componentes de ADSs. Aqui, estas informações são chamadas parâmetros e podem ser grupadas nas seguintes classes:

- Parâmetros das características do sistema: estão relacionados diretamente às características do software a ser desenvolvido, como por exemplo o seu tamanho, o grau de inovação do sistema, o grau de confiabilidade desejado e a função da aplicação.

- Parâmetros do computador: estão relacionados ao computador que será utilizado para desenvolver o software. Os recursos de máquina disponíveis irão determinar o software que poderá ser utilizado.

- Parâmetros do pessoal: estão relacionados as questões que envolvem as pessoas que trabalharão com o sistema. Por exemplo, a experiência profissional da equipe de desenvolvimento com o uso de determinados componentes do

ADS como também com a aplicação irá influenciar na determinação de componentes.

- Parâmetros de desempenho e custo: estão relacionados ao que se espera do software em termos de seu desempenho e custo. O limite, por exemplo, do custo de desenvolvimento irá influenciar na compra de ferramentas e hardware.

- Parâmetros relacionados a outros recursos que serão utilizados: por exemplo, a necessidade de se utilizar um determinado método irá envolver a escolha de ferramentas que apóiem este método.

- Parâmetros tecnológicos e arquiteturais do ambiente: estão relacionados à tecnologia e arquitetura adotadas pelo ADS. A preferência pelo desenvolvimento do sistema através de orientação a objetos, por exemplo, irá determinar que métodos e ferramentas sejam escolhidos utilizando esta abordagem.

- Parâmetros relacionados a outros componentes de software: a determinação de um componente pode implicar que outros mais sejam escolhidos de forma a apoiar o uso do primeiro. É o caso de um compilador, que determina o uso de um ou mais depuradores.

- Parâmetros relacionados diretamente ao componente: estão relacionados a questões que têm a haver

com o próprio componente, como por exemplo, o fato de um método de custo só ser recomendado se tiver sido calibrado adquirindo o perfil do ambiente onde será utilizado.

O capítulo seguinte apresenta os projetos que já vêm sendo realizados e as propostas de trabalhos para construção da próxima versão de XAMÃ.

CAPÍTULO VI

CONCLUSÃO

Este trabalho faz parte do Projeto TABA (Rocha-89) (Rocha-90) (Rocha-91). O TABA visa a construção de uma estação de trabalho configurável capaz de criar um ambiente adequado às necessidades de cada projeto de desenvolvimento de software. Ambiente de Desenvolvimento de Software (ADS) é considerado neste trabalho como um conjunto integrado de métodos, ferramentas e o hardware que suportam o desenvolvimento de um software nas diferentes etapas de seu ciclo de vida. Consideramos o ciclo de vida, métodos, ferramentas e o hardware como os componentes a partir dos quais o ADS é formado.

A determinação de quais componentes devem ser escolhidos não é uma tarefa trivial, uma vez que há várias possibilidades de combinação, além de ser necessário o conhecimento sobre cada componente.

Esta tese apresenta uma solução para apoiar o usuário planejador de ambientes na escolha dos componentes de software adequados ao desenvolvimento de cada projeto. Foi especificado um sistema com este objetivo e desenvolvida a primeira versão de uma ferramenta que o implementa: "XAMÃ: Planejador de ADS", um sistema especialista de suporte à decisão.

Esta ferramenta foi desenvolvida de forma a atender a um dos objetivos do meta-ambiente do TABA, o planejamento de ADS, e de maneira a poder se integrar com os demais sub-sistemas com os quais tem interface.

O desenvolvimento desta tese teve duas fases principais. Na primeira fase, com o objetivo de testar idéias, foi desenvolvido o protótipo de um sistema para apoiar a indicação dos componentes do ADS (Aguiar-89). Este protótipo foi construído em Prolog, para micro-computadores compatíveis com o IBM-PC, utilizando o "shell" desenvolvido na tese (Ribeiro-89). A ênfase, nesta fase, foi no conhecimento dos sistemas especialistas de suporte à decisão e nas formas de desenvolvimento desse tipo de sistema. (Werneck-89) apresenta o resultado da fase de identificação do problema e (Aguiar-89) apresenta a etapa de construção do protótipo.

A partir da experiência obtida nesta primeira fase, foi realizada uma nova especificação do sistema, descrita resumidamente no capítulo V e construída a ferramenta "XAMÃ: Planejador de ADS" (Aguiar-91), para SUN escrita em Prolog e C. XAMÃ foi gerado, apoiado nos resultados de uma série de trabalhos práticos e teóricos realizados através de teses, experimentos e monografias, descritos ao longo desta tese.

O que distingue o sistema proposto nesta tese não é o fato de ser uma ferramenta de seleção de componentes

baseada no conhecimento, uma vez que em (Kitto-91), por exemplo, é apresentada uma ferramenta com este objetivo, descrita no item III.2.

O que distingue o sistema proposto é, primeiro, o fato de ser uma ferramenta de seleção dos componentes necessários a todo o processo de desenvolvimento de software, atendendo às atividades de gerência, modelagem e avaliação da qualidade. Utiliza como estratégia a definição do ciclo de vida juntamente com a definição dos demais componentes do ADS. Considera que existe uma forte relação entre os componentes do ADS, uma vez que ferramentas têm sido desenvolvidas para serem utilizadas em atividades específicas do ciclo de vida e por outro lado o uso de um método ou ferramenta tem implicações no ciclo adotado. A definição do ciclo de vida tem também a grande vantagem de possibilitar a organização do processo de seleção, permitindo uma visualização do conjunto e a sequência de uso das ferramentas e métodos.

Para representar o ciclo de vida foi provida uma representação, através de um grafo, que se caracteriza por ser genérica, possibilitando a modelagem de qualquer ciclo. Cada ferramenta ou método é, então, relacionado a uma ou mais atividades deste ciclo, assim como cada atividade pode ter um ou mais métodos ou ferramentas associadas.

Em segundo lugar, este sistema se distingue pelo seu suporte à decisão, que inclui bases de dados e de

conhecimento. A primeira base de dados contém informações sobre métodos e ferramentas. São informações do tipo desenvolvedores, descrição, projetos desenvolvidos que as utilizaram, etc. A segunda base de dados contém ciclos de vida descritos na linguagem criada para representá-los, com o objetivo de compor ciclos de vida do projeto a partir desses ciclos. A terceira base de dados contém as características de diferentes aplicações. Quanto à base de conhecimento, suas informações são no sentido de indicar um componente a partir das características do projeto a ser desenvolvido.

Em terceiro lugar, este sistema se distingue por sua proposta de reutilização de informações de planejamento de ADSs, através do armazenamento das diferentes versões de ADSs geradas pela ferramenta e ao final do projeto a avaliação deste produto. A partir desta avaliação as bases de dados e de conhecimento poderão ser atualizadas. A reutilização das informações de planejamento é um aspecto bastante positivo do sistema, uma vez que possibilita que as experiências no planejamento de ambientes sirvam de base a outros planejamentos.

"XAMÃ: Planejador de ADS", a ferramenta construída, implementou parcialmente o sistema proposto. A ausência de um sistema de banco de dados e de recursos mais eficientes para a construção de um editor gráfico foram responsáveis por algumas das limitações da implementação. A próxima versão deverá incluir:

a) Quanto a definição do ciclo de vida:

Deverá ser construído um editor gráfico que possibilite a edição dos diagramas de atividades que representam os ciclos de vida.

b) Quanto à reutilização de informações:

Deverão ser implementados os recursos propostos nesta tese que possibilitem a reutilização das informações de planejamento de ADSs.

c) Quanto ao suporte à decisão:

Deverá ser construída a base com informações sobre métodos e ferramentas utilizando-se um hipertexto. Já foi concluída uma experiência em PC utilizando o hipertexto desenvolvido na tese de mestrado (Faria-91) no qual foram armazenadas informações de métodos para especificação de sistemas. Pretende-se estender essa experiência para SUN utilizando-se uma extensão, já especificada, do hipertexto desenvolvido pela tese (Netto-91).

No sentido de aumentar a base de conhecimentos para indicação de elementos de ADSs deverão ser geradas mais regras a partir dos trabalhos de (Crispim-91), sobre métodos de especificação, e (Glitz-92) sobre métodos para aquisição do conhecimento de engenheiros de software.

Os recursos providos pelo sistema de planejamento de ADSs dão ao meta-ambiente da estação TABA uma faceta que o distingue de outros meta-ambientes configuradores. Estes meta-ambientes possuem recursos que possibilitam a configuração de um ADS mas não oferecem facilidades para planejar esta configuração.

Quanto à configuração do ADS está em fase de desenvolvimento um outro trabalho de doutoramento (Travassos-91) que permitirá que o ADS planejado seja instanciado e colocado disponível aos usuários desenvolvedores de software da estação.

De forma a possibilitar que o planejamento e a geração de ADSs seja realizada de maneira eficiente são previstos recursos no meta-ambiente para:

a) Avaliação e refinamento das informações geradas ao longo do processo de criação dos ADSs de forma que possam ser armazenadas nas bases de dados e de conhecimento que são consultadas para apoio à criação de novos ADSs.

b) Armazenamento de sugestões de ferramentas a serem implementadas para que no futuro façam parte do conjunto de ferramentas disponíveis na estação TABA.

c) Inclusão de ferramentas no conjunto de ferramentas disponíveis na estação TABA. Podem ser

ferramentas adquiridas no mercado como também geradas na própria estação TABA.

d) Consulta e avaliação de ferramentas existentes no mercado.

Para estes quatro recursos deverão ser realizados a partir de agora projetos de pesquisa de forma a especificá-los com mais detalhes e desenvolver o suporte automatizado para apoiá-los.

Uma série de outros trabalhos já vêm sendo realizados em paralelo aos já citados e no futuro deverá haver uma integração de todos eles. Os principais são citados a seguir:

a) Estão sendo construídos ADSs específicos para as áreas de software educacional (Stahl-91), software científico (Souza-91) e software orientado a objetos (Mattoso-90). Estes ADSs farão parte do conjunto de ferramentas disponíveis na estação TABA. Quanto à base de conhecimento para indicação de componentes de ADSs, poderá indicar para determinados casos o uso de um desses ambientes ou então de algum componente que faça parte dos mesmos. XAMÃ já foi desenvolvida de forma a incorporar o resultado destes trabalhos.

b) Está previsto que ao se gerar um ADS na estação TABA se forneça um auxílio para seu uso sob a forma de um

assistente especialista. Um primeiro experimento nesta direção foi realizado por (Falcão-92) onde é implementado um assistente para a Análise Estruturada (Gane-84), especificamente para apoio ao desenvolvimento de diagramas de fluxo de dados. Está previsto que este trabalho será estendido e generalizado.

c) O projeto "Controle da Qualidade de Software", submetido ao PROTEM (Projetos Temáticos Multi-institucionais) do CNPq, visa o desenvolvimento das ferramentas para avaliação da qualidade que farão parte da estação TABA. Quanto à base de conhecimento para indicação de componentes de ADSs, esta terá informações de forma a indicar para cada caso o uso dessas ferramentas.

d) A tese (Glitz-92) fez um estudo de métodos para aquisição do conhecimento de especialistas em Engenharia de Software. O resultado deste trabalho deverá ser utilizado de forma a se obter o conhecimento a ser colocado nas bases de conhecimento das próximas versões de "XAMÃ: Planejador de ADS".

e) Determinar o ADS significa estimar os recursos de métodos, software e hardware que serão necessários ao desenvolvimento de um sistema. Esta atividade faz parte do planejamento de um projeto. O objetivo do planejamento é prover uma base que permita ao gerente realizar estimativas razoáveis de recursos (humanos, de software e de hardware), de custos e de cronograma.

Essas estimativas são realizadas no início do projeto e são atualizadas durante o desenvolvimento do projeto. O fato é que existem relacionamentos entre recursos, o que torna problemático planejá-los separadamente. Desta forma o sistema de planejamento de ADSs deverá ser integrado ao sistema de planejamento de projetos do TABA, permitindo que todas as informações necessárias ao planejamento sejam compartilhadas e vistas como parte de um todo homogêneo. Para atingir este objetivo já está sendo desenvolvido um trabalho sobre planejamento de projetos. (Sardinha-91)

f) Uma das limitações desta versão de XAMÃ foi originada por não se dispor de um sistema de gerência de banco de dados. No entanto, está sendo desenvolvido o GEOTABA, o gerente extensível de objetos do TABA. Trabalhos têm sido realizados na área de sistemas extensíveis e de orientação a objetos na gerência de aplicações não convencionais para banco de dados (Rocha-91).

g) Ao fim do desenvolvimento da próxima versão de XAMÃ, pretende-se fazer experiências em ambientes que desenvolvam software, tanto em empresas quanto no meio acadêmico. Avaliações deste tipo são a única forma de validar a ferramenta, uma vez que a eficácia dos recursos de suporte à decisão só poderá ser medida se as bases de conhecimento e de dados contiverem informações adequadas ao local que as está utilizando. E essas informações terão um

alto grau de confiança se puderem ser frequentemente atualizadas, a partir da experiência obtida no desenvolvimento de projetos.

Esta tese apresenta, portanto, o trabalho desenvolvido de forma a possibilitar ao meta-ambiente do TABA o planejamento de ADSs. E conclui apresentando os trabalhos em desenvolvimento ou propostos que permitirão que as atividades de planejamento e geração de ADSs no meta-ambiente sejam realizadas eficientemente.

REFERÊNCIAS BIBLIOGRÁFICAS

(Aaram-84) Jarle Aaram, "The BOP Prototyping Concept" em Approaches to Prototyping, Germany, Springer-Verlag, 1984.

(Alford-80) M. Alford, "Software Requirements Engineering Metodology at the age or four", Computer Software and Applications Conference, outubro 1980.

(Aguiar-89) T.C. Aguiar, "Protótipo do Especificador de Ambientes da Estação TABA", Relatório Técnico ES-208/89, COPPE/UFRJ, 1989.

(Aguiar-90) T.C. Aguiar, Ambiente de desenvolvimento de software: a questão da administração de seus dades, Exame de qualificação em Banco de Dados, COPPE-SISTEMAS/UFRJ, 1990.

(Aguiar-91) T.C. Aguiar, A.R. da Rocha, "XAMÃ: Um assistente especialista de apoio à decisão na especificação de ADSs", Rel. Técnico TABA-RT-5/91, 1991.

(Arango-84) G. Arango, "La Estructuracion de Dominio y un Nuevo Modelo de Desarrollo de Software", Anais do 2º Congr. Latinoamericano de Investigacion Operativa e Ingenieria de Sistemas, Buenos Aires, agosto 1984.

(Bailin-89) Sidney Bailin, "Object-Oriented Requirements Specification Method", Communication of ACM, vol.32, no. 5, pp. 608-623, Maio 1989.

(Balzer-85) Robert Balzer, "A 15 Year Perspective on Automatic Programming", IEEE Transaction on Software Engineering, vol SE-11, novembro 1985.

(Barstow-86) D. Barstow, "A Perspective on Automatic Programming", em: Readings in Artificial Intelligence and Software Engineering, editado por Charles Rich, Richard Waters, Morgan Kaufmann Publishers, 1986.

(Basili-86) V. Basili e outros, "Experimentation in Software Engineering", IEEE Trans. on Softw. Engineering, vol SE-12, no 7, julho 1986.

(Bersoff-91) E.H. Bersoff, A.M. Davis, "Impacts of Life Cycle Models on Software", Communications of the ACM, vol. 34, nº 8, agosto 91.

(Biswas-88) G. Biswas, M. Oliff, A. Sen, "An Expert Decision Support System for Production Control", Decision Support Systems 4, 235-248, 1988.

(Blanning-85) R.W. Blanning, "Expert Systems for Management: Research and Applications", Journal of Information Science 9, 153-162, 1985.

(Blum-86) B.I. Blum, V.G. Sigillito, "An expert system for designing information systems", John Hopkins APL Technical Digest, vol.7, nº 1, 1986.

(Boehm-86) B. Boehm, "A spiral model of software development and enhancement", Software Engineering Notes, vol. 11, nº 4, 1986.

(Bonczek-81) R.H. Bonczek, C.W. Holsapple, A.B. Whinston, "The Evolution from MIS to DSS: extention of data management to model management", em: M.J. Ginzberg, W. Reitman e E.A. Stohr, eds., Decision Support Systems, North-Holland, New York, 1982.

(Breuker-88) J.B. Breuker, B. Wielinga, "Models of Expertise in Knowledge Acquisition", em: G. Guida e C. Tasso (eds) Topics in Expert System Design: Methodologies and tools - North Holland P. Company, Amsterdan, 1988.

(Carey-90) J.M.Carey, "Prototyping: alternative systems development methodology", Information and Software Technology, vol.32, no 2, março 1990.

(Cheathan-86) T.C. Cheathan, "Reusability Through Program Transformations", em Readings in Artificial Intelligence and Software Engineering, editado por Charles Rich, Richard Waters, Morgan Kaufmann Publishers, 1986.

(Chen-90) Peter Chen, Gerenciando Banco de Dados: A Abordagem Entidade-Relacionamento para Projeto Lógico, São Paulo, Editora McGraw-Hill, Ltda. e Newstec Editora Ltda., 1990.

(Chitman-91) Edith Chitman, "Estudo e classificação de modelos de ciclo de vida", Tese de mestrado - COPPE/UFRJ, 1991.

(Christensen-84) Niels Christensen, Klaus-Dieter Kreplin, "Prototyping of User-Interfaces", em: Approaches to Prototyping, Germany, Springer-Verlag, 1984.

(Coad-90) Peter Coad, Edward Yourdon(1990), Object-Oriented Analysis, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1990.

(Cochran-85) Duane Cochran, Frederick Stocker, "RIPL: An Environment for Rapid Prototyping with Intelligent Support", ACM SIGCHI Bulletin, vol. 17, no. 2, pp. 29-35, Outubro 1985.

(Cohen-86) B. Cohen e outros, The Specification of Complex Systems, Addison-Wesley Publishing Company, 1986.

(Coelho-88) H. Coelho, "State of the Art of DSS's in 1988", 1st International Informatics Congress of Rio de Janeiro, Brasil, 22 a 26 de agosto de 1988.

(Coura-86) E.C.C. Coura, O estágio de desenvolvimento do planejamento em empresas no Brasil - Uma metodologia de estudo - Tese de mestrado - COPPEAD/UFRJ, 1986.

(Crispim-91) Emilia Crispim, Avaliação de métodos para o desenvolvimento de software, Tese de mestrado - COPPE/UFRJ, 1991.

(Cupello-88) J.M. Cupello, D.J. Mishelevich, "Managing Prototype Knowledge Expert System Projects", Communication ACM, vol. 31, no 5, maio 1988.

(Darlington-86) J.D. Darlington, "An Experimental Program Transformation and Syntesis System", em Readings in AI and DB, editado por J. Mylopoulos, M. Brodie, Morgan Kaufmann Publishers, California, 1988.

(Dart-87) S.A. Dart e outros, "Software Development Environments", IEEE Computer, novembro 1987.

(Davis-86) M.J. Davis, D.R. Addleman, "A Practical Approach to Specification Technology Selection", Journal of Systems and Software 6, 285-294, 1986.

(Davis-88) A.M Davis, E.H. Bersoff, E.R. Comer, "A strategy for comparing alternative software development life cycle models", IEEE Transaction on Software Engineering, vol 14, no 10, outubro 88.

(Davis-90) A.M. Davis, Software Requirements - Analysis and Specification, Prentice Hall, New Jersey, 1990.

(Dell-86) P.W. Dell, "Early experience with an IPSE", Software Engineering Journal, nov. 1986.

(Dixon-88) D. Dixon, "Integrated Support for Project Management", IEEE, 1988.

(Donzeau-84) G.V. Donzeau, e outros, "Programming Environments Based on Structured Editors: The Mentor Experience", em: Interactive Programming Environments, ed. Barstow, Mc Graw Hill, 1984.

(Duarte-91) M.A. Duarte, Um sistema para representação do conhecimento de métodos de desenvolvimento de software, Tese de mestrado, COPPE-SISTEMAS/UFRJ, janeiro 1991.

(Encarnação-86) J.L. Encarnação, L.A. Messina, Z.I. Markov, "Models and methods for the implementation of decision support systems aiming at the evaluation and selection of CAD-Systems", Design Theory of CAD Systems, North Holland, Amsterdam, 1986.

(Engelbart-88) D.C. Engelbart, "Toward High-Performance Knowledge Workers", em Computer-Supported Cooperative Work: A Book of Reading, Editado por Irene Greif, Lotus Development Corporation, Cambridge, Massachusetts, 1988.

(Fairley-85) R.E. Fairley, "Software Engineering Concepts", McGraw Hill, 1985.

(Falcão-92) João F. Falcão, Um assistente baseado em conhecimento para ambientes de desenvolvimento de software, Tese de mestrado a ser defendida em abril 1992, COPPE-SISTEMAS/UFRJ.

(Faria-91) Eduardo Faria, Marte: Um meta-gerador de aplicações baseado na reutilização de templates, Tese de mestrado, COPPE-SISTEMAS/UFRJ, setembro de 1991.

(Fox-90) J. Fox, "Safe expert systems: simulating experts or building formal theories?", The Knowledge Engineering Review, vol. 5, 1, 1990.

(Gane-84) C. Gane, T. Sarson, Análise Estruturada de Sistemas, LTC, Rio de Janeiro, 1984.

(Gevarter-87) W.B. Gevarter, "The Nature and Evaluation of Commercial Expert System Building Tools", Computer, 24-41, maio de 1987.

(Ghezzi-85) C. Ghezzi, M. Jazayeri, Conceitos de linguagens de programação, Ed. Campus, 1985.

(Ghezzi-91) C. Ghezzi e outros, Fundamentals of Software Engineering, Prentice-Hall International Inc., 1991.

(Giavitto-90) J.L. Giavitto e outros, "Design Decisions for the Incremental Adage Framework", IEEE, 1990.

(Ginzberg e Stohr-82) M.J. Ginzberg, E.A. Stohr, "Decision Support Systems: Issues and Perspectives" em: M.J. Ginzberg, W. Reitman e E.A. Stohr, eds., Decision Support Systems, North-Holland, New York, 1982.

(Glitz-92) S. Glitz, Aquisição do Conhecimento, Tese de mestrado a ser defendida em março de 1992, COPPE-SISTEMAS/UFRJ.

(Goghen-86) J.A. Goghen, J.J. Tardo, "An Introduction to OBJ: A language for writing and testing formal algebraic programs specifications", em Software Specification Techniques, N. Gehani e Mc Gettrick (eds), Addison-Wesley, 1986.

(Goldberg-86) Allen T. Goldberg, "Knowledge-based Programming: A survey of program design and construction tecniques", em IEEE Transaction on Software Eng., vol SE-12, no 7, julho 1986.

(Gomaa-84) H. Gomaa, "A Software Design Method for Real-Time Systems", Communication of ACM, vol. 27, no. 9, pp. 938-949, Setembro 1984.

(Gomes-87) A.G. Gomes, H. Ribeiro, L.P. Teixeira, M.F. Castro, "Um Metodologia para Avaliação e Escolha de Linguagens de 4ª Geração", Anais XIX Congresso Nacional de Informática, 1987.

(Gray-87) P. Gray, "Group Decision Support Systems", Decision Support Systems 3, 1987.

(Haberman-86) N. Haberman, D. Notkin, "Gandalf: Software Development Environments", IEEE Trans. on Softw. Eng., vol SE-12, no 12, dezembro 1986.

(Hagemann-87) A. Hagemann, "Formal Requirements Specification of Process Control Systems", ACM SIGSOFT Software Engineering Notes, vol. 12, nº 4, pp. 36-42, Outubro 1987.

(Hamid-86) T.K. Abdel-Hamid e outros, "Impacts of Schedule Estimation on Software Project Behavior", IEEE Software, julho 1986.

(Hayes-Roth-83) F. Hayes-Roth, D. A. Waterman, D.B. Lenat, "Building Expert Systems", Addison-Wesley Publishing Company,Inc, Massachusetts, 1983.

(Hayes-Roth-84) F. Hayes-Roth, "Knowledge-Based Expert Systems", IEEE Computer, outubro de 1984.

(Hitchcock-86) P. Hitchcock e outros, "The use of databases for software engineering", Proc. 5th British National Conference on Database, julho 1986.

(Hix-91) D. Hix, R.S. Schulman, "Human-computer Interface Development Tools: A Methodology for their evaluation", Communication of the ACM, nº 3, vol. 34, março 1991.

(Hoare-69) C. Hoare, "An axiomatic basis for computer programming", CACM, v. 12, 1969.

(Hoffnagle-85) G.F. Hoffnagle, W.E. Beregi, "Automating the Software Development Process", IBM Systems Journal, vol. 24, n. 2, 1985.

(Holbrook-90) H. Holbrook, "A Scenario-Based Methodology Conducting Requirements Elicitation", ACM SIGSOFT Software Engineering Notes, vol. 15, nº 1, pp. 95-104, Janeiro 1990.

(ISO-85) Estelle - A formal description technique based on extended state transaction model, ISO/TC, 97/SC, 21 nº 422, maio de 1985.

(Jain-89) H.K. Jain, A.R. Chaturvedi, "Expert System Problem Selection: A Domain Characteristics Approach", Information & Management 17, 1989.

(Janse-87) M.D. Brouwer-Janse, M. Grunes, "Design of an intelligent interface for software planning", Information and Software Technology, 1987.

(Jones-86) Capers Jones, "Programming Productivity", McGraw Hill, New York, 1986.

(Keen-87) P.G.W. Keen, "Decision Support Systems: The Next Decade", Decision Support Systems 3, 1987.

(Keller-87) R. Keller, Expert System Technology - Development and Application, Yourdon Press, 1987.

(Kitto-89) C.M. Kitto, J.H. Boose, "Selecting knowledge acquisition tools and strategies based on application characteristics", Int. J. Man-Machine Studies(1989)31.

(Knuth-73) D. Knuth, The Art of Computer Programming, Addison-Wesley, 1973.

(Kraiem-89) Z.M.Kraiem, D.P. Dagli, J.E. Diekmann, "DISCON: An expert system for the analysis of differing site conditions claims", Knowledge-based Systems, vol 2, nº 3, setembro 1989.

(Lehner-90) P.E. Lehner, L. Adelman, "Behavioural decision theory and it's implication for knowledge engineering", The Knowledge Engineering Review, vol. 5, 1, 1990.

(Lucena-87) C. Lucena, Inteligência Artificial e Engenharia de Software, Jorge Zahar Editor, 1987.

(Luconi-86) F.L. Luconi, T.W. Malone, M.S.S. Morton, "Expert Systems: The Next Challenge for Managers", Sloan Management Review, 1986.

(Lustman-87) F. Lustman, "Managing Computer Projects", Reprinted in Management & Administration, abril 1987.

(Mathur-87) Raghubin Mathur, "Methodology for Business System Development", IEEE Transactions on Software Engineering, vol. SE-13, nº 5, pp. 593-601, Maio 1987.

(Mattoso-90) Adriana Mattoso, TABA-OBJ: Um Ambiente de Desenvolvimento de Software com Orientação a Objetos, Tese de Mestrado, COPPE-SISTEMAS/UFRJ, agosto de 1990.

(Mc Carthy-63) J. Mc Carthy, A basis for a mathematical theory of computation, em: Computer Programming and Formal Systems, Braffor e Hirschber (eds), North Holland Publishing Company, 1963.

(Mcmenamin-91) Stephen Mcmenamin, PALMER, John Palmer, Análise Essencial de Sistemas, São Paulo, Editora McGraw-Hill, Ltda. e Makron Books Ltda, 1991.

(Mendes-89) Sueli Mendes, Teresa C. Aguiar, Métodos para Especificação de Sistemas, Ed. Edgard Blucher, 1989.

(Mishkoff-85) H.C. Mishkoff, "Understanding Artificial Intelligence", Texas Instruments Information Publishing Center, Dallas, Texas, 1985.

(Neighbors-86) J.Neigbors, "The Draco Approach to Constructing Software from Reusable Components", em Readings in Artificial Intelligence and Software Engineering, editado por Charles Rich e Richard Waters, Morgan Kaufmann Publishers, 1986.

(Netto-91) Luiz L. Netto, O Desenvolvimento de um sistema hipertexto em ambiente UNIX, Tese de mestrado, COPPE-SISTEMAS/UFRJ, setembro 1991.

(New-90) D. New, P.D. Amer, "Adding graphics and animation to Estelle", Information and Software Tecnology, vol 32, nº 2, março 1990.

(Penedo-86) M.H. Penedo, "Prototyping a Project Master Data Base for Software Engineering Environment", ACM SIGPLAN Notices, vol 22, nº 1, 1986.

(Penedo-88) Maria Heloisa Penedo, Willian Riddle, "Guest Editor's Introduction-Software Engineering Environment Architectures", em IEEE Trans. on Softw. Eng., vol 14, nº 6, junho 1988.

(Perry-88) D.E. Perry, G.E. Kaiser, "Models of Software Development Environments", Proc. of the 10th ICSE, 1988.

(Perry-89) D.E. Perry, "The Inscape Environment", ACM Sigsoft, maio 1989.

(Pressman-87) R. Pressman, Software Engineering: A Practioner's Approach, McGraw-Hill, 1987.

(Prieto-Diaz-87) Ruben Prieto-Diaz, "Domain Analysis for Reusability", IEEE 11th Compsac, outubro 1987.

(Ramsey-89) C.L. Ramsey, V. Baili, "An evaluation of expert systems to software engineering management", IEEE Transaction on Software Engineering, vol 15, nº 6, junho 1989.

(Refine) Apostila do "Refine Training Course"

(Ribeiro-89) C.A.S. Ribeiro, Um Sistema Especialista de Suporte à Decisão para Custos de Software, Tese de Mestrado, COPPE-SISTEMAS/UFRJ, 1989.

(Roberts-90) H.J. Roberts, "Expert Systems Clubs: Design Methods", The Computer Journal, vol. 33, n. 6, 1990.

(Rocha-83) A.R.C.da Rocha, Um Modelo para avaliação da qualidade de especificações, Tese de doutorado, PUC-RJ, 1983.

(Rocha-87) A.R.C.da Rocha, T.C. de Aguiar, J. Blaschek, Ambientes para desenvolvimento de software: definição de termos, Relatório técnico ES-137/87, COPPE-SISTEMAS/UFRJ, 1987.

(Rocha-88) A.R.C.da Rocha, T.C. de Aguiar, "Ambiente de desenvolvimento de software" em Ambiente de Desenvolvimento de Software e o Projeto TABA, A.R.C.da Rocha, J.M. de Souza, Relatório Técnico ES-178/88 - COPPE-SISTEMAS/UFRJ, dezembro de 1988.

(Rocha-89) A.R.C.da Rocha, J.M. de Souza, T.C. de Aguiar, C. d'Ipolitto, Meta-Ambiente TABA, Relatório Técnico - COPPE/Sistemas/UFRJ, 1989.

(Rocha-90) A.R.C. da Rocha, J.M. de Souza, T.C. de Aguiar, "TABA: A Heuristic Workstation for Software Development", COMPEURO, Tel-Aviv/Israel, maio 1990.

(Rocha-91) A.R.C. da Rocha, J.M. de Souza, "O Projeto TABA", Relatório Técnico RJ-7/91.

(Rotenstreich-86) Shmuel Rotenstreich, William Howden, "Two Dimensional Program Design", IEEE Transactions on Software Engineering, vol. SE-12, no. 3, pp. 377-384, 1986.

(Sardinha-91) Elisiane A.B. Sardinha, "Um gerenciador de projetos de desenvolvimento de software para a estação TABA, Anais do Workshop de Gerência de Projetos de Desenvolvimento de Software, COPPE-SISTEMAS/UFRJ, agosto 1991.

(Sathi-88) A. Sathi, e outros, "Callisto: An Intelligent Project Management System", Computer-Supported Cooperative Work: A Book of Reading, Edited by Irene Greif, Lotus Dev. Comp., Cambridge, Massachusetts, 1988.

(Shaw-87) M.J. Shaw, "Applying Inductive Learning to Enhance Knowledge-based Expert Systems", Decision Support Systems 3, 319-332, 1987.

(Sievert-85) Gene Sievert, Terrence Mizell, "Specification-Based Software Engineering with TAGS", Computer, pp. 56-65, Abril 1985.

(Shlaer-88) Sally Shlaer, Stephnen Mellor, Object-Oriented Systems Analysis, Englewood Cliffs, New Jersey, Prentice-Hall, Inc.

(Smith-85) D. Smith, G.B. Kotik, S.J.Westfold, "Research on Knowledge Based Software Environments at Kestrel Institute", IEEE Trans. on Soft. Eng., vol SE-11, nov. 1985.

(Sol-87) H.G. Sol, "Conflicting Experiences with DSS", Decision Support Systems 3, 203-211, 1987.

(Souza-91) Jano Souza, A.R. Rocha, "The Architecture of Taba-HEP Workstation", Proceedings of the International Conference on Computing in High Energy Physics '91, Tsukuba, Japão, março 1991.

(Sprague-80) R.H. Sprague, "DSS in Context", Decision Support Systems, 197-202, 1987.

(Stabell-87) C.B. Stabell, "Decision Support Systems: Alternative Perspectives and Schools", Decision Support Systems 3, 243-251, 1987.

(Stahl-91) Marimar Stahl, Ambiente de desenvolvimento de software educacional, Relatório técnico Taba-rt-4/91, COPPE/UFRJ.

(Stevens-88) W. Stevens, Projeto Estruturado de Sistemas, Ed. Campus, Rio de Janeiro, 1988.

(Taylor-86) R. Taylor e outros, "Arcadia: A software development environment research project", Prc. ACM/IEEE Symp. Ada Tools and Environments, Flórida, 1986.

(Thomas-89) Ian Thomas, "Tool Integration in the Pact Environment", ACM Sigsoft, maio 1989.

(Travassos-91) Guilherme H. Travassos, Um modelo para representação das informações manuseadas pelas ferramentas da estação TABA, Exame de qualificação de doutorado, COPPE-SISTEMAS/UFRJ, 1991.

(Wasserman-86) Anthony Wasserman, "The User Software Engineering Methodology: An Overview, Em Information Systems Design Methodologies: A Comparative Review, Olle, T., Sol,

H. e Verrijn-Stuart, A. (editores), North-Holiand Publishing Company, pp. 591-628, 1986.

(Waterman-86) D.A. Waterman, A Guide to Expert Systems, Addison-Wesley Publishing Company, Massachusetts, 1986.

(Weiderman-86) N.H. Weiderman, e outros,"A Methodology for Evaluating Environments", Second ACM Sigsoft/Sigplan Software Engineering Symposium on Practical Software Development Environments, 1986.

(Weiss-90) H. Fersko-Weiss, "CASE Tools for Designing your Applications", PC Magazine, Janeiro 1990.

(Weitzel-89) J.R. Weitzel, L. Kerschberg, "Developing Knowledge-Based Systems: Reorganizing the system development life cycle", Communications of the ACM, vol 32, no 1, abril 1989.

(Werneck-89) V. Werneck, S. Assis, J. Matos, T.C. de Aguiar, "Especificador de Ambientes da Estação TABA: Fase de Identificação", Relatórios Técnicos do Programa de Engenharia de Sistemas e Computação - UFRJ, 1989.

(Werneck-90a) Vera Werneck, Taxonomia de domínios de aplicação, Tese de mestrado, COPPE-SISTEMAS/UFRJ, 1990.

(Werneck-90b) Vera Werneck, T.C.Aguiar, A.R.Rocha, "Classificação de domínios de aplicação para especificação

de ambientes de desenvolvimento de software", XVII Conf. Latinoamericana de Informática, Assunção, Paraguai, setembro 1990.

(Werneck-91) V. Werneck, T.C. Aguiar, A.R.C. da Rocha, "Características Determinantes das Tecnologias de Desenvolvimento de Software", PANEL'91 - XVIII Conferencia Latinoamericana de Informatica, julho 1991, Caracas, Venezuela.

(Werner-91) Claudia Werner, "Um estudo sobre ambientes de programação", Relatório técnico, TABA-RT-1/91, COPPE-SISTEMAS/UFRJ, 1991.

(Wiederhold-86) Gio Wiederhold, "Knowledge versus Data", em On Knowledge Base Management Systems-Integrating AI and DB Technologies, editado por M. Brodie, J.Mylopoulos, Springer-Verlag, 1986.

(Williams-88) L.G. Williams, "Software Process Modeling: A Behavioral Approach", IEEE, 1988.

(Yau-88) S.S.Yau, R.A. Nicholl, I. Tsai, Sying-Suang Liu, "An integrated life-cycle model for software maintenance", IEEE Transaction on Software Engineering, vol 14, no 8, agosto 88.

(Yourdon-89) E. Yourdon, Revisões Estruturadas, Ed. Campus, 1989.

(Yourdon-90) Edward Yourdon, Análise Estruturada Moderna, Editora Campus Ltda., Rio de Janeiro, 1990.

(Zucconi-89) L. Zucconi, "Selecting a CASE Tool", ACM Sigsoft, Software Engineering Notes, vol 14, n. 2, abril 89.