ESTUDO E CLASSIFICAÇÃO DE MODELOS DE CICLO DE VIDA

Edith Chitman

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

Aprovada por:

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

Prof. Jano Moreira de Souza, PhD.

Prof. Fernando Manso, PhD.

CHITMAN, EDITH

Estudo e Classificação de Modelos de Ciclo de Vida [Rio de Janeiro] 1991.

XI, 149 p. 29,7cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1991)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Ciclo de Vida 2. Modelo 3. Teses

I. COPPE/UFRJ II. Título (série).

Ao Julio

Resumo da Tese Apresentada à COPPE / UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

ESTUDO E CLASSIFICAÇÃO DE MODELOS DE CICLO DE VIDA

Edith Chitman

Maio de 1991

Orientadora : Prof. Ana Regina Cavalcanti da Rocha

Programa : Engenharia de Sistemas e Computação

Boa qualidade no processo de desenvolvimento e no produto final de um software está relacionada com a escolha do modelo de ciclo de vida a ser adotado.

Um modelo de ciclo de vida é usado para sugerir uma ordenação das atividades existentes no desenvolvimento de software e também como um poderoso instrumento de planejamento, permitindo melhores condições para gerenciar o desenvolvimento, com qualidade, obtendo-se confiabilidade no trabalho que está sendo desenvolvido.

Esta tese tem como objetivo realizar um estudo da literatura sobre os diferentes modelos de ciclo de vida propostos e uma posterior classificação dos mesmos tendo em vista sua adequação ao desenvolvimento de produtos com diferentes características.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirement for the degree of Master of Science (M.Sc.).

STUDY AND CLASSIFYING OF MODEL OF SOFTWARE LIFE CYCLE

Edith Chitman

May, 1991

Thesis Supervisor : Prof. Ana Regina Cavalcanti da Rocha

Department : Systems and Computer Engineering

The quality of the software development process and the software final product are related with the choice of the software life cycle model adopted.

A model of software life cycle is used to suggest an order in the activities found in the software development process and also to be a powerful planning tool that allows quality and good conditions to the management of the development, increasing the reliability of the work done in the software development.

This thesis presents a study about different models of software life cycle propposed in the literature and a classification of those models considering their suitability to the development of products with different features.

INDICE

Pag.

INDICE DE FIGURAS

CAPITULO I

INTRODUÇÃO

I.1 - Conceituação de Ciclo de Vida de Software

O **Ciclo de Vida** é usado para sugerir uma ordenação das atividades existentes no desenvolvimento e manutenção de software.

É considerado o intervalo de tempo decorrido desde o momento da concepção de um software até sua obsolescência. Inclue uma série de transformações que resultam em produtos como resultado do processo.

É identificado um conjunto de fases ou etapas que representam sua evolução desde o nascimento da necessidade de criação de um software até a sua descontinuidade ou morte.

Ultimamente essa identificação tem recebido diversas denominações, tais como: "Processo de Desenvolvimento"; "Ciclo de Desenvolvimento" e "Ciclo de Vida".

O termo "processo de desenvolvimento" foi definido como um processo formal através do qual objetivos são transformados sucessivamente em requisitos, em especificações de projeto, sendo posteriormente implementados em código, testados e finalmente entregues e mantidos em operação. Este conceito identifica-se, portanto com o de "ciclo de vida".

O termo "ciclo de desenvolvimento" é usado para caracterizar o subconjunto das fases componentes do ciclo de

vida, das quais se exclui as fases de operação e manutenção.

A definição de um ciclo de vida, além de servir como um poderoso instrumento de planejamento, permite que se tenha melhores condições para gerenciar o desenvolvimento, para gerência de qualidade e na obtenção de confiabilidade do próprio trabalho que está sendo desenvolvido.

Segundo O'Neill em [O'NEILL80] Ciclo de Vida é o conjunto de fases que descreve o processo do software a partir da definição do sistema até o término do suporte operacional. As fases que compõem o processo de software são mais definidas em termo de componentes de trabalho, no qual identifica as tarefas e incumbências que devem ser realizadas. As fases podem sobrepor-se temporariamente, mas cada uma deve ser indicada para um complemento prévio ou para a conclusão subsequente das fases dependentes.

Através da visão de conjunto propiciada pela definição de um ciclo de vida fica evidenciada a estruturação existente entre as suas diversas fases, ou seja, o encadeamento das fases.

Torna-se claro que se nos ativermos com maior atenção ou maior emprego de recursos técnicos, recursos financeiros, ou recursos humanos, em determinada fase do ciclo de vida de software poderemos alcançar benefícios consideráveis em termo de obtenção de melhor qualidade ou de menores custos em fases subsequentes.

Ciclo de vida de software enseja uma base para

categorização e controle das atividades do projeto. A idéia básica associada ao ciclo de vida é a quebra do ciclo em fases. Cada fase inclui atividades específicas que resultam em produtos visíveis e aceitáveis. Estas fases, normalmente, são definidas em termos dos quatro elementos a seguir [TRIPP81]:

- Produtos resultantes;

- Recursos necessários;

- Produtos aceitos (verificados e validados) das fases anteriores, e;

- Métodos de desenvolvimento utilizados.

Os produtos resultantes de uma fase permitem identificar a conclusão ou não desta fase. Tipicamente, estes produtos incluem documentos, tais como:

- Padrões a serem usados no projeto;

- Especificações (de requisitos, de projeto preliminar, etc);

- Documentação interna (por exemplo, dicionário de dados);

- Resultados experimentais de simulações, protótipos, etc.;

- Dados de teste e/ou relatórios;

- Dados/resultados de verificação/validação;

- Critérios de aceitação, resultados de teste, etc.;

- Documentos externos (por exemplo, manuais do usuário, manuais de referência, etc.);

- Listagens de programa/módulo.

Os recursos necessários (pessoal, tempo, máquina,etc.) precisam ser estimados dentro de níveis razoáveis antes da tomada de decisão de investimento.

Qualquer método de desenvolvimento de software prevê a preparação de vários documentos ao longo do processo de construção do produto. A conclusão de um documento (especialmente no final de uma fase) atesta que uma etapa do projeto foi cumprida e torna visível à gerência, o estágio do processo de desenvolvimento. Além disso, representa um elemento de inestimável valor para comunicação entre os membros da equipe do projeto.

No final do desenvolvimento, o conjunto de documentos produzidos constitui a documentação do projeto, registrando a solução adotada em cada passo. Em suma, é a história do desenvolvimento do projeto.

I.2 - Motivação

Este trabalho está inserido no contexto do Projeto TABA [ROCHA89A] [ROCHA90], cujo objetivo é a construção de uma Estação de Trabalho para desenvolvimento de software. A Estação TABA será configurável para atender às características das diferentes áreas de aplicação ou mesmo de

projetos.

Assim sendo, o projeto TABA visa construir uma estação de trabalho para o engenheiro de software, que permita a implementação de ambientes adequados ao desenvolvimento de software em diferentes áreas de aplicação, possibilitando também sua execução.

Na Estação TABA, a especificação do ambiente de desenvolvimento poderá ser feita diretamente pelo usuário, através de uma linguagem para definição de ambientes ou através de um sistema especialista de apoio à decisão que auxilia na tomada de decisões sobre os componentes desejáveis para um ADS, tendo em conta as características do produto a ser desenvolvido [ROCHA89].

Este sistema [AGUIAR89], possui uma base de conhecimentos que contém:

- Conhecimento sobre métodos de apoio ao desenvolvimento de software;

- Conhecimento sobre modelos de ciclo de vida;

- Conhecimento sobre aspectos gerenciais, de fatores humanos e produtividade;

- Conhecimento sobre diferentes domínios de aplicação.

A construção desta base de conhecimentos supôs vários estudos, entre eles a elaboração de uma classificação de modelos de ciclo de vida com o objetivo de identificar que modelos de ciclo de vida são mais apropriados para o

desenvolvimento de um produto, considerando suas características particulares. Este trabalho de tese foi realizado com vistas a elaborar esta classificação.

I.3 - Objetivo da Tese

O objetivo desta tese é, portanto, a aquisição e organização de conhecimentos sobre modelos de ciclo de vida. O Projeto TABA usará essa informação para identificar, a partir das características do produto de software a ser desenvolvido o processo de desenvolvimento mais adequado.

Para atingirmos este objetivo são apresentados modelos de ciclo de vida de software encontrados em literaturas e é feita uma análise comparativa dos mesmos, identificando-se suas semelhanças e peculiaridades.

Vários autores definem modelos de ciclo de vida de software, onde a diferença está em um número distinto de fases. Essas diferenças residem na liberdade existente para se subdividir uma determinada fase em um número variável de atividades que por sua vez podem ser decompostas ou agregadas para obtenção de tarefas indivisíveis ou atividades globais.

O problema crítico, portanto, não é dar nome a cada fase, mas sim definir como estas serão realizadas e quais os produtos a serem gerados por cada uma delas, de forma a atender às necessidades do ambiente em que o desenvolvimento e operação se processam.

I.4 - Organização da Tese

Esta tese está organizada em seis capítulos.

CAPITULO I

Contém a introdução do trabalho apresentando a conceituação de ciclo de vida de software, os fatores que motivaram a elaboração da tese, a definição dos objetivos da tese e sua organização como um todo.

CAPITULO II

Contém uma revisão bibliográfica dos diversos enfoques sobre ciclo de vida de software. Primeiramente identifica-se a problemática relacionada à escolha de um modelo de ciclo de vida adequado, e a seguir descrevem-se os diferentes modelos propostos.

CAPITULO III

Neste capítulo são mostradas algumas comparações entre os diferentes modelos de ciclo de vida de software mencionados no capítulo anterior. É apresentada a classificação dos modelos de ciclo de vida e uma posterior seleção .

CAPITULO IV

Neste capítulo são apresentados subsídios para a base de conhecimento da Estação Taba. Com os modelos de ciclo de vida selecionados no capítulo anterior é apresentado um conjunto de regras que orientem a escolha do modelo adequado.

CAPÍTULO V

Contém os modelos de ciclo de vida de software organizados através de um diagrama de atividades. Associado a este diagrama são apresentados os produtos a serem gerados em cada atividade bem como os recursos utilizados.

CAPÍTULO VI

Este capítulo apresenta as conclusões do trabalho. Algumas observações e sugestões relacionadas a possibilidades de futuras pesquisas são abordadas.

CAPÍTULO II

REVISÃO DA LITERATURA

Este capítulo apresenta um estudo da literatura técnica atual específica sobre modelos de ciclo de vida de software.

Discute-se a problemática relacionada à escolha de um ciclo de vida adequado ao desenvolvimento de um dado projeto e descrevem-se os diferentes modelos propostos.

II.1 - A Problemática da Seleção de um Modelo de Ciclo de Vida

II.1.1 - Caracterização de Ambientes de Desenvolvimento de Software

A problemática de seleção de um modelo de ciclo de vida insere-se no contexto de seleção de um Ambiente de Desenvolvimento de Software (ADS).

Segundo Rocha [ROCHA87a], Ambientes de Desenvolvimento de Software são compostos de um modelo de ciclo de vida, métodos instrumentos e ferramentas, de forma a viabilizar a construção de produtos com elevado índice de qualidade no menor tempo possível.

Segundo Hausen [HAUSEN81], os Ambientes de Desenvolvimento de Software devem proporcionar um apoio completo a todo o ciclo de vida, apoiando a transição de uma fase para outra e aos ciclos subentendidos.

Ambientes de Desenvolvimento de Software devem cobrir

os seguintes tópicos:

- O tópico central, que é a produção do software, atinge desde a fase de análise de requisitos até a fase de manutenção, cobrindo simultaneamente tanto a verificação como a validação do produto;

- O tópico, muitas vezes, negligenciado em ambientes é o de gerenciamento do software que engloba o planejamento do projeto (criação e controle do Plano do Projeto como também o monitoramento do processo de desenvolvimento);

- O tópico mais importante e que, em geral, não é mencionado em ambientes que é a participação do usuário no processo de desenvolvimento do software.

O conceito de ciclo de vida, no contexto de Ambientes de Desenvolvimento de Software, é sempre utilizado no sentido de sugerir uma ordenação de atividades tendo associado a cada uma delas um método de desenvolvimento adequado a sua realização.

II.1.2 - Problemática dos Diferentes Modelos de Ciclo de VIda

Numa conferência realizada em 1980 [McCRACKEN82] para estudar o Tópico Análise e Projeto de Software: Um plano para os anos 80, a maior parte dos integrantes da conferência estavam de acordo com o conceito de ciclo de vida apresentado anteriormente numa pré-conferência que dizia que o desenvolvimento de sistema consistia dos dez passos: Análise

organizacional, avaliação do sistema, análise de possibilidade, plano do projeto, projeto lógico, projeto físico, projeto de programa, implementação, operação, revisão e operação.

Concluiu-se que existem várias variações no tema básico, mas que todos os conceitos de ciclo de vida se direcionam a uma só definição.

Nesta conferência considerou-se que o conceito de ciclo de vida é prejudicial, pois qualquer forma de ciclo de vida é um projeto gerencial estruturado imposto ao desenvolvimento do software. Pequenas variações podem ser aplicadas a todos os desenvolvimentos de software e ao mesmo tempo o ciclo de vida se adapta a essas novas condições.

Estas considerações negativas quanto ao conceito de ciclo de vida foram divulgadas rapidamente, havendo algumas defesas a seu respeito.

Hall [HALL82] defende o conceito de ciclo de vida, mostrando que este conceito capacita o planejamento de uma meta, de forma a se poder conduzir e controlar o desenvolvimento de software.

Essas idéias iniciaram-se após inúmeras interrogações sobre a utilização, validade e prazo. Tais perguntas são:

1) Qual a definição de ciclo de vida?

2) Há um ciclo de vida padrão?

3) Pode-se prognosticar um ciclo de vida de software?

4) Qual o objetivo de predizer o ciclo de vida de software?

Pode-se concordar com um conceito único para ciclo de vida todavia não há concordância generalizada com a natureza, sequência ou número de fases [CAPRON86].

As diferentes apresentações nos modelos de ciclo de vida baseiam-se na liberdade existente para subdividir-se cada fase em um número variável de atividades que por sua vez podem ser decompostas ou agregadas para obtenção de tarefas indivisíveis ou atividades globais, respectivamente [STAA79].

O problema crítico portanto não é dar nome a cada fase, mas sim definir como estas fases serão realizadas e quais os produtos que serão gerados pelas mesmas, de forma a atender às necessidades do ambiente em que o desenvolvimento e operação se processam.

Segundo Blum [BLUM82] o modelo clássico de ciclo de vida originou-se do modelo de hardware composto de: requisitos, projeto, fabricação, teste, operação e manutenção. Grande ênfase é dada à correção do projeto porque para uma grande produção, o alto custo de corrigir os erros quando a fabricação começa é muito elevado. Neste modelo a manutenção é a reposição de alguma parte falha. No modelo clássico de ciclo de vida, as fases de codificação e depuração substituem a fabricação.

A seleção de um modelo de ciclo de vida deve-se basear em dois fatores: a compreensão do problema e a facilidade de

implementação.

O primeiro fator refere-se a completude da compreensão do problema antes que a implementação se inicie.

O segundo fator que deve influenciar a escolha do modelo, refere-se à disponibilidade de ferramentas que dão apoio ao modelo de ciclo de vida.

É necessário categorizar os diferentes tipos de softwares a serem desenvolvidos, considerar modelos de ciclo de vida alternativos que sejam aplicados a cada caso, identificar e desenvolver ferramentas e métodos que facilitem a implementação dessas diferentes classes de aplicação.

Segundo Rocha [ROCHA87a] os modelos de ciclo de vida são utilizados para orientar a gerência do processo de desenvolvimento, onde aspectos de custo, recursos alocados, tempo dispendido, metas e subprodutos de cada fase terão uma grande importância.

Não existe um modelo único que seja adequado à produção e manutenção de qualquer software. A escolha de um modelo de ciclo de vida depende basicamente do propósito do ciclo de vida e das característica do software a ser produzido.

Assim sendo, para selecionar o modelo de ciclo de vida mais apropriado ao desenvolvimento de um determinado projeto devemos adequar as características do modelo às características do produto. Para identificar as características inerentes a diferentes domínios de aplicação foi elaborado, no contexto do projeto TABA, um trabalho

[WERNECK90], que a partir de um estudo da literatura e de uma pesquisa de campo, definiu uma Taxonomia de Domínios de Aplicação.

Neste trabalho completamos o estudo necessário para possibilitar a seleção de modelos de ciclo de vida descrevendo os diferentes modelos de ciclo de vida e identificando seu propósito e adequação.

II.2 - Modelos de Ciclo de Vida de Software Propostos na Literatura

Nesta seção descrevemos as várias propostas, presentes na literatura técnica, relativas a modelos de ciclo de vida.

II.2.1 - Modelo Tradicional

Em todas as propostas de ciclo de vida que podem ser agrupadas no contexto de "Modelo Tradicional" é dada uma visão de todo o processo de desenvolvimento de software, como um evento iterativo onde as principais tarefas são análise, projeto, implementação, testes, operação e manutenção.

Neste modelo, o processo de desenvolvimento pode ser visto como uma sequência de tarefas onde pode ser notado as modificações ocorrerem, repetidamente, de forma a se entender melhor o software que se quer obter.

O modelo clássico do tipo cascata foi definido primeiramente por Royce [ROYCE70] e mais tarde aperfeiçoado

por Boehm [BOEHM76]. O uso desse modelo :

- Encoraja a especificação do sistema, isto é, definir os requisitos, antes de projetá-lo e construí-lo;

- Encoraja o planejamento de como os componentes irão interagir antes de construí-los;

- Capacita o gerente do projeto a caminhar progressivamente e descobrir possíveis erros antecipadamente;

- Demanda a produção de uma série de documentos no decorrer do processo de desenvolvimento, que mais tarde podem ser utilizados para testar e manter o sistema;

- Capacita a organização que desenvolverá o sistema a ser mais estruturada e controlável.

As principais características do modelo cascata são [BOEHM81]:

- Cada fase atinge o ponto culminante através da atividade de verificação e validação, cujo objetivo é identificar e eliminar o maior número possível de problemas no produto de cada fase;

- Os produtos das fases anteriores são utilizados como base para as fases subsequentes.

O modelo cascata reconhece a realimentação dos laços entre as fases, obtém normas para minimizar o trabalho considerado de custo elevado.

O esquema básico desse modelo conduziu a formulação de

modelos de processos alternativos. O principal tema que deu origem a muitas dificuldades foi a ênfase numa completa elaboração de documentos como critério de conclusão das fases de especificação de requisitos e projeto.

Para algumas classes de software, como por exemplo compiladores ou sistemas operacionais, essa é uma maneira eficaz de desenvolvimento. Entretanto, para outras classes de software, tais como aplicações envolvendo grande interação com o usuário, esse modelo de desenvolvimento não é adequado.

A seguir apresentamos a título de exemplo, algumas variações deste modelo.

II.2.1.1 - Modelo Proposto por Boehm [BOEHM81]

Número de fases : 8

Nome das fases : Viabilidade do Sistema
Requisitos e Plano do Software
Projeto do Produto
Projeto Detalhado
Código
Integração
Implementação
Operação e Manutenção

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Viabilidade do Sistema

Objetivos :

- Definir conceitos preferenciais do produto de software;

- Determinar um ciclo de vida viável e as vantagens de conceitos alternativos.

2) Fase de Requisitos e Plano de Software

Objetivo :

- Aprovar a especificação dos requisitos funcionais, de interface e de desempenho do produto de software;

3) Fase de Projeto do Produto

Objetivo :

- Verificar a especificação global da arquitetura de hardware e software, da estrutura de controle e da estrutura de dados do produto.

4) Fase de Projeto Detalhado

Objetivo :

- Verificar a especificação da estrutura de controle, estrutura de dados, relação de interface, dimensão, algoritmos padrões e suposição de cada componente do programa.

5) Fase de Código

Objetivo :

- Verificar o conjunto de componentes do programa.

6) Fase de Integração

Objetivo :

- Integrar funcionalmente o produto de software.

7) Fase de Implementação

Objetivo :

- Efetuar uma completa operação funcional do sistema hardware e software, incluindo como objetivos os programas e dados convertidos, instalados e treinados.

8) Fase de Manutenção

Objetivo :

- Efetuar uma completa atualização funcional do sistema (hardware e software).

O processo para alcançar essas metas dá-se através das atividades de verificação, validação e gerência de configuração, que devem estar presentes em cada fase do ciclo de vida de software.

A figura 1 dá uma visão geral deste modelo.

II.2.1.2 - Modelo Proposto por Capron [CAPRON86]

Número de fases : 5

Nome das fases : Investigação Preliminar
Análise
Projeto
Desenvolvimento
Implementação

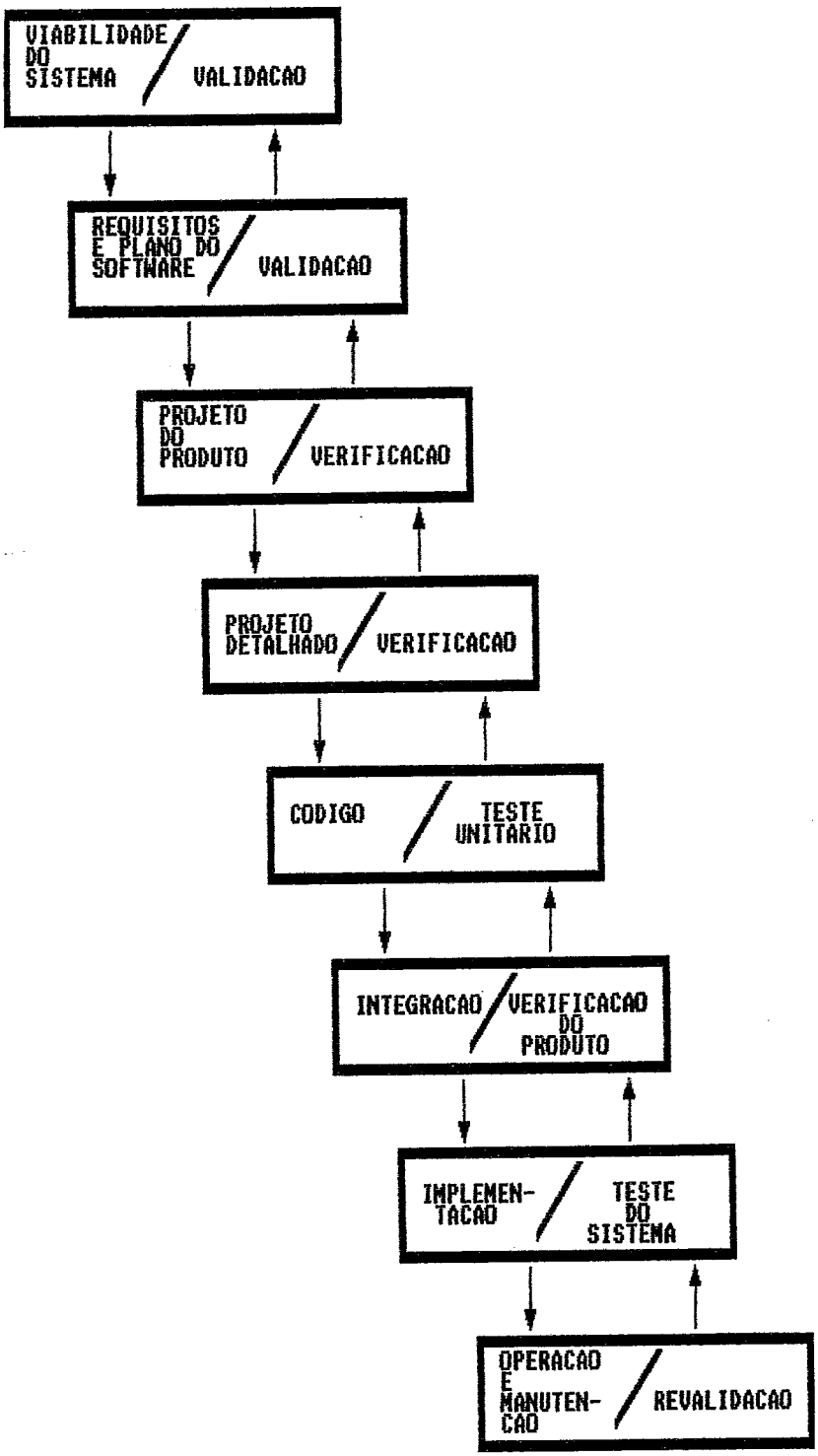


FIGURA 1 - Modelo Cascata de Ciclo de Vida [BOEHM81]

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Investigação Preliminar

Objetivos :

- Determinar o problema, definindo seu escopo do problema e a área específica para estudo;

- Definir um conjunto de objetivos abrangendo as expectativas do usuário.

2) Fase de Análise

Objetivos :

- Entender o sistema existente, coletar e analisar dados;

- Estabelecer os requisitos do sistema.

3) Fase de Projeto

Objetivos :

- Determinar o projeto preliminar com a estrutura do sistema;

- Planejar detalhadamente o novo sistema (projeto detalhado).

4) Fase de Desenvolvimento

Objetivos :

- Codificar as especificaçÕes preparadas na fase de projeto;

- Testar o sistema.

5) Fase de Implementação

Objetivos :

- Transformação para o novo sistema;
- Avaliar o sistema.

A figura 2 dá uma visão geral deste modelo.

II.2.1.3. - Modelo Proposto por Enger [ENGER80]

Número de fases : 6

Nome das fases : Análise dos Requisitos
Projeto Lógico
Projeto Físico
Definição de Programa
Implementação do Sistema
Operação do Sistema

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Análise dos Requisitos

Objetivos :

- Avaliar o pedido do usuário;
- Conduzir estudo de viabilidade;
- Definir os requisitos do usuário;
- Preparar o plano do projeto.

2) Fase de Projeto Lógico

Objetivos :

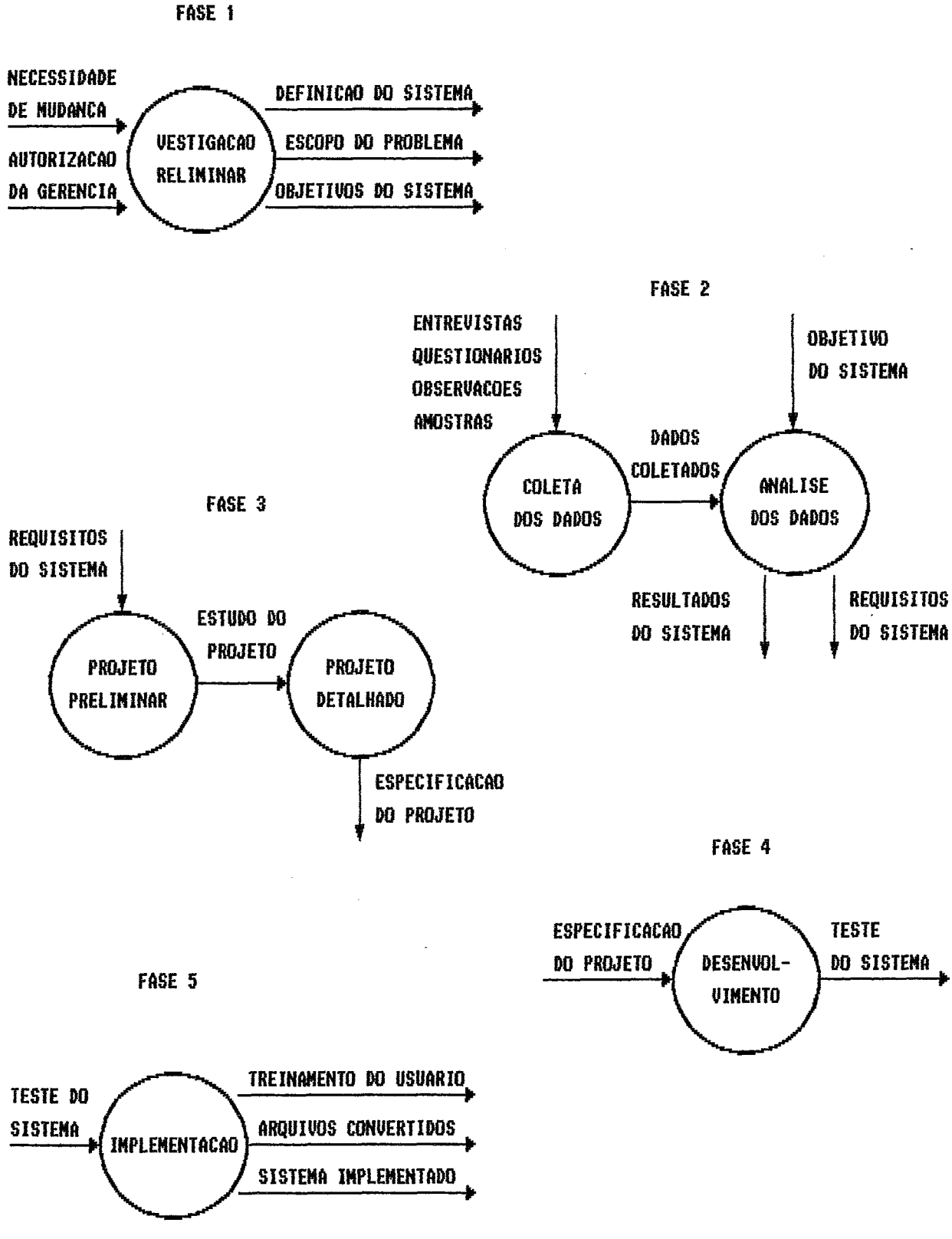


FIGURA 2 - Fases do Ciclo de Vida [CAPRON86]

- Preparar a especificação geral do projeto;
- Refinar os requisitos do sistema do usuário.

3) Fase de Projeto Físico

Objetivos :

- Preparar a especificação detalhada do projeto;
- Definir os subsistemas;
- Projetar a estrutura da base de dados.

4) Fase de Definição de Programa

Objetivos :

- Codificar os programas;
- Testar os programas isoladamente;
- Documentar os programas.

5) Fase de Implementação do Sistema

Objetivos :

- Executar o teste do subsistema;
- Executar o teste do sistema;
- Realizar o treinamento dos usuários;
- Estabelecer controles de conversão;
- Executar a conversão dos dados.

6) Fase de Operação do Sistema

Objetivos :

- Operar o sistema final;
- Manter o novo sistema;
- Avaliar o sistema.

II.2.1.4 - Modelo de Houghton [HOUGHTON87]

Número de fases : 7

Nome das fases : Iniciação
Definição
Projeto de Alto Nível
Projeto Detalhado
Programação
Operação e Manutenção
Retirada

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Iniciação

Objetivos :

- Identificar a necessidade do software;
- Fazer uma definição preliminar de requisitos;
- Realizar o estudo de viabilidade;
- Definir o plano de desenvolvimento;
- Realizar análise de custo e benefício.

2) Fase de Definição

Objetivos :

- Desenvolver uma descrição do software incluindo os requisitos funcionais e de dados;
- Estabelecer os recursos de custo, hardware, execução, etc;
- Desenvolver os planos de validação, verificação e

teste;

- Gerar os requisitos básicos para os casos de teste.

3) Fase de Projeto de Alto Nível

Objetivos :

- Definir a arquitetura do software (parte principal do software e o fluxo de dados entre as partes);

- Estabelecer os algoritmos fundamentais e as principais representações de dados;

- Gerar os projetos básicos de teste;

- Verificar se o projeto está de acordo com os requisitos.

4) Fase de Projeto Detalhado

Objetivos :

- Definir os algoritmos e a representação dos dados;

- Definir os dados de teste;

- Verificar o projeto detalhado.

5) Fase de Programação

Objetivos :

- Implementar o projeto detalhado;

- Testar e integrar os componentes de código;

- Testar o sistema.

6) Fase de Operação e Manutenção

Objetivos :

- Colocar em uso o software;

- Realizar mudanças no software caso novos requisitos

sejam adicionados ou haja algum problema;

- Realizar testes após mudanças.

7) Fase de Retirada

Objetivo :

- Desativar o software em caso de pouca utilidade do software, quando as mudanças no produto começam a confundir os requisitos e projeto original ou quando a manutenção se torna muito cara.

II.2.1.5 - Modelo Proposto por Fairley [FAIRLEY85]

Número de fases : 5

Nome das fases : Análise
Projeto
Implementação
Teste do Software
Manutenção

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Análise

Objetivos :

- Permitir um perfeito entendimento do problema do usuário;

- Elaborar um estudo de viabilidade;

- Produzir uma solução estratégica recomendada;

- Determinar os critérios de aceitação do produto;

- Planejar o processo de desenvolvimento;

- Identificar as funções básicas do software; enfatizando o que o software irá fazer e quais restrições impostas pelo ambiente no qual ele irá operar.

2) Fase de Projeto

Objetivos :

- Especificar os componentes do software (processos, estruturas de dados e arquivos);

- Estabelecer os relacionamentos existentes entre esses componentes;

- Definir a estrutura do software;

- Elaborar um planejamento da fase de implementação.

3) Fase de Implementação

Objetivos :

- Traduzir em código fonte as especificações do projeto;

- Incluir atividades de depuração, teste e documentação dos programas produzidos;

4) Fase de Teste do Software

Objetivos :

- Testar a integração entre os componentes do software;

- Demonstrar que o produto implementado atende aos requisitos estabelecidos.

5) Fase de Manutenção

Objetivos :

- Corrigir erros que venham a ser detectados;

- Promover adaptações motivadas por novos ambientes de processamento;

- Aperfeiçoar o produto de modo a aumentar suas capacidades.

Esse modelo de ciclo de vida preconiza o retorno a fases anteriores do ciclo, sempre que correções ou modificações forem notadas como necessárias em fases posteriores.

O modelo básico está representado na figura 3.

II.2.2 - Modelo Semi Estruturado [YOURDON89]

O aparecimento e difusão dos métodos estruturados motivou o aparecimento do que se convencionou chamar de modelo semi estruturado de ciclo de vida. Esse modelo difere do modelo clássico em dois pontos básicos:

- Ênfase na implementação "top-down";

- Desenvolvimento utilizando os métodos e técnicas estruturados.

Este modelo foi proposto por Yourdon [YOURDON89] e pode ser caracterizado da seguinte forma:

Número de fases : 5
Nome das fases : Análise Preliminar dos Requisitos
Análise
Projeto Estruturado
Estudo do Hardware
Implementação Top-Down

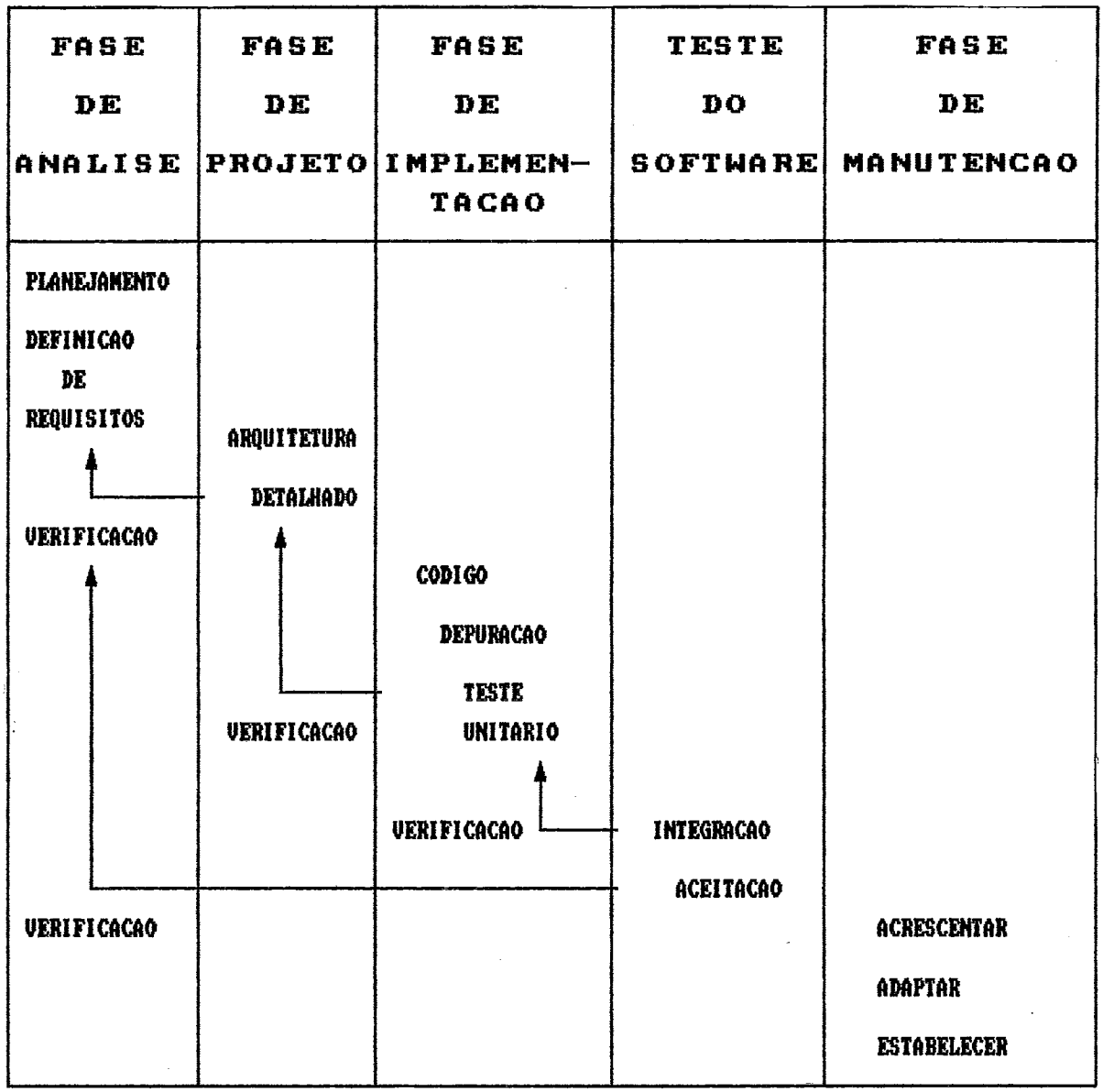


FIGURA 3 - Modelo das Fases de Ciclo de Vida [FAIRLEY85]

A fase de Projeto Estruturado é constituída dos seguintes processos :

Codificar a Especificação Funcional

Derivar Diagrama de Estrutura

Módulo do Projeto

Empacotar o Projeto

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Análise Preliminar dos Requisitos

Objetivos :

- Conduzir estudo de viabilidade;

- Definir requisitos do usuário.

2) Fase de Análise

Objetivos :

- Transformar os requisitos do usuário numa especificação de projeto estruturado;

- Refinar a estrutura lógica do sistema.

3) Fase de Projeto Estruturado

Objetivos :

- Identificar a hierarquia dos módulos e as interfaces entre os módulos, gerando a especificação do projeto físico;

- Preparar plano de teste.

4) Fase de Estudo do Hardware

Objetivos :

- Identificar as necessidades do usuário;

- Especificar a configuração do hardware adequado ao desenvolvimento do produto.

5) Fase de Implementação Top-Down

Objetivos :

- Codificar módulos;

- Integrar módulos.

As figuras 4 e 5 dão uma visão geral desse modelo.

II.2.3 - Modelo Estruturado

Como evolução do modelo semi estruturado surgiu o modelo estruturado de ciclo de vida, que permite que uma ou mais atividades sejam realizada ao mesmo tempo.

A utilização deste modelo é possível com dois enfoques. No primeiro todas as atividades estão sendo realizadas ao mesmo tempo (enfoque radical) e no segundo (enfoque conservativo) uma atividade só começa quando a anterior termina.

A decisão sobre a escolha de um ou outro enfoque deve estar baseada nos seguintes fatores:

- Inconstância dos usuários;

- Pressão para obtenção de resultados tangíveis a curto prazo;

- Perigo de haver um grande erro no sistema.

O enfoque radical é melhor num ambiente onde algo deve estar pronto, pontualmente, numa data específica e onde a opinião do usuário no que diz respeito a que o sistema fará

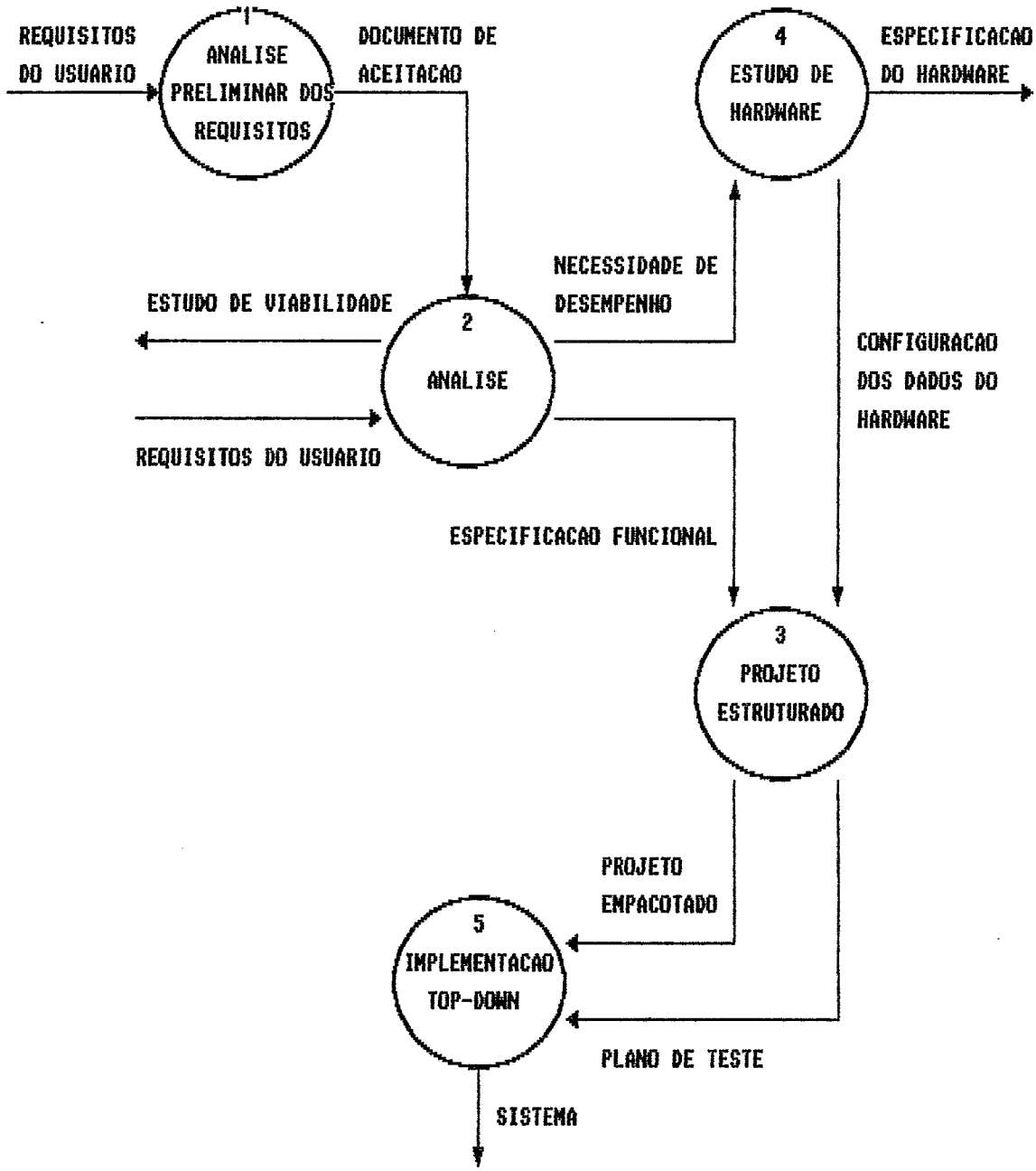


FIGURA 4 - Modelo Semi Estruturado de Ciclo de Vida [YOURDON89]

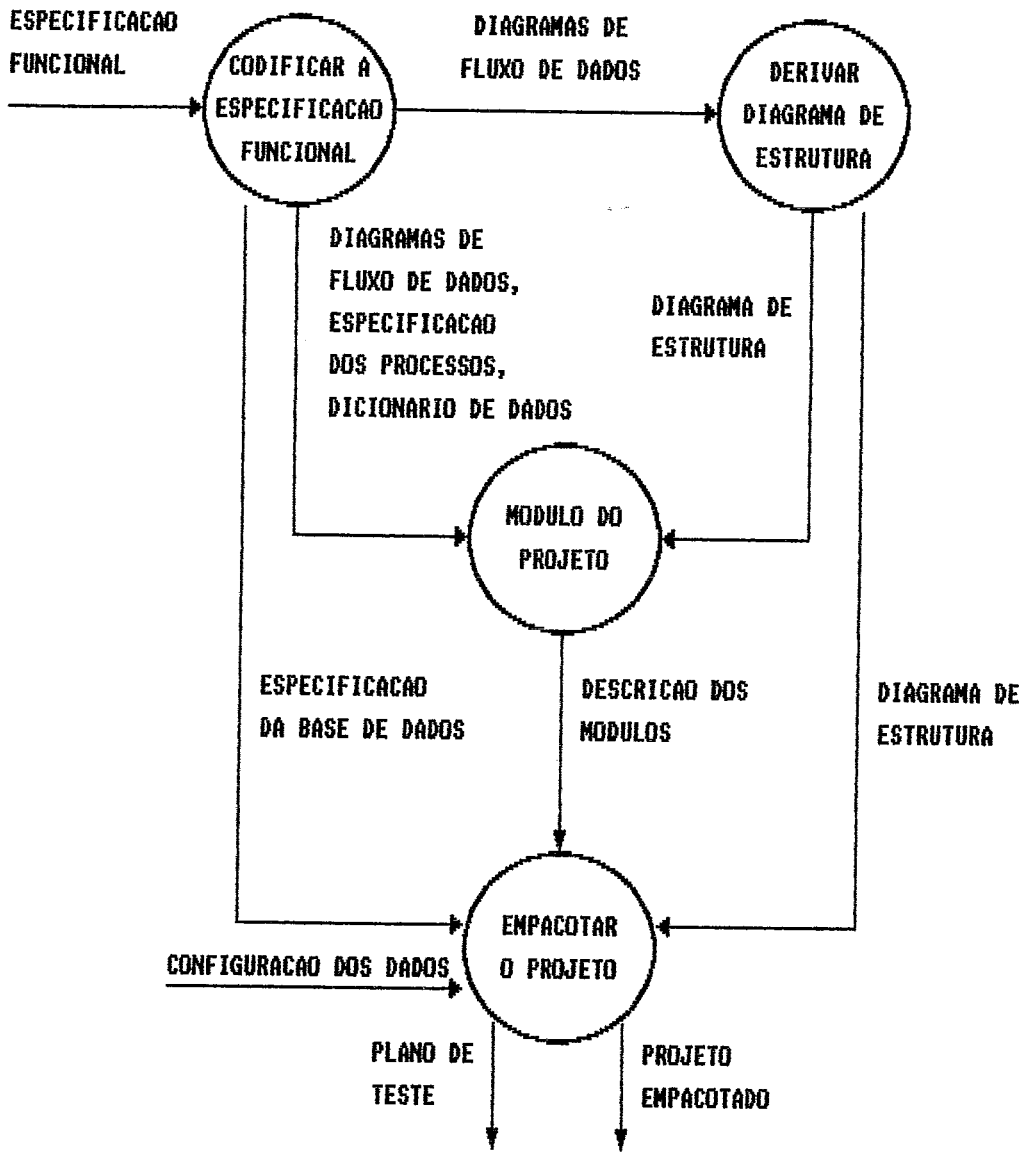


FIGURA 5 - Detalhes da Atividade de Projeto do Modelo Semi Estruturado [YOURDON89]

está sujeito a mudanças. O enfoque conservativo é usado em projetos grandes, de alto custo, e onde é necessária uma grande atenção à análise e projeto para prevenir desastres posteriores.

A seguir apresentamos três propostas de modelo estruturado.

II.2.3.1 - Modelo Proposto por Enger [ENGER80]

Número de fases : 7

Nome das fases : Análise dos Requisitos
Projeto Lógico (Análise)
Projeto Físico Estruturado
Implementação Top-Down
Geração de Teste de Aceitação
Garantia da Qualidade
Operação do Sistema

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Análise dos Requisitos

Objetivos :
- Avaliar o pedido do usuário;
- Conduzir estudo de viabilidade;
- Definir requisitos do usuário;
- Preparar plano do projeto.

2) Fase de Projeto Lógico

Objetivos :

- Transformar os requisitos do usuário numa especificação de projeto;

- Refinar a estrutura lógica do sistema.

3) Fase de Projeto Físico Estruturado

Objetivos :

- Identificar a hierarquia dos módulos e as interfaces entre os módulos, gerando a especificação de projeto físico;

- Finalizar equipamentos de configuração de dados;

- Preparar plano de teste.

4) Fase de Implementação Top-Down

Objetivos :

- Codificar módulos;

- Integrar módulos.

5) Fase de Geração de Teste de Aceitação

Objetivos :

- Definir a aceitação do sistema;

- Gerar casos de teste de aceitação.

6) Fase de Garantia da Qualidade

Objetivos :

- Conduzir o teste do sistema final;

- Produzir o manual do usuário;

- Realizar treinamento do usuário;

- Estabelecer controle de conversão;

- Executar conversão dos dados.

7) Fase de Operação do Sistema

Objetivos :

- Gerar os manuais de operação e de manutenção;
- Operar o sistema;
- Manter o novo sistema;
- Avaliar o sistema.

Vários métodos, ferramentas e técnicas com abordagem estruturada podem ser usados para apoio às fases deste modelo, conforme descritos a seguir:

METODOLOGIAS/FERRAMENTAS/TÉCNICAS	FASES
Diagrama de Fluxo de Dados	1,2,3 e 4
Dicionário de Dados	1,2,3,4 e 7
Diagrama de Estrutura de Dados	1,2,3,e 4
Descrição das Transformações	1,2,3 e 4
Projeto Top-Down	1,2,3 e 4
HIPO (Hierarquical Input Process Output)	1,2,3 e 4
Portugues Estruturado	2,3 e 4
Tabela de Decisão	2,3 e 4
Árvore de Decisão	2,3 e 4
Diagrama de Warnier	2,3 e 4
Diagrama de Nassi-Schneiderman	2,3 e 4
Diagrama de Chapin	2,3 e 4
Diagrama Estruturado	3,4 e 5
Implementação Top-Down	3,4,5,6 e 7
Equipe de Chefe de Programação	3,4,5 e 6
Programação Estruturada	2,3 e 4
Pseudocódigo	3 e 4

Biblioteca de Suporte de Desenvolvimento	4,5 e 6
Walkthroughs Estruturado	2,3,4,5 e 6
Testes Estruturados	4,5,6 e 7
Manutenção Estruturada	7

II.2.3.2 - Modelo Proposto por Rocha [ROCHA87]

Número de fases : 5

Nome das fases : Definição
Projeto
Implementação
Avaliação
Operação

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Definição

Objetivos :

- Identificar o problema a ser resolvido;

- Estabelecer os objetivos, requisitos, hipóteses e restrições do problema;

- Realizar um planejamento inicial do desenvolvimento.

2) Fase de Projeto

Objetivo :

- Determinar uma solução viável para o sistema definido na fase anterior . Consta da proposição de diversas alternativas de solução e da seleção da alternativa mais

conveniente; maximizando o objetivo de qualidade do sistema dentro das restrições econômicas e técnicas impostas.

3) Fase de Implementação

Objetivo :

- Produzir código em conformidade com os documentos gerados nas fases anteriores.

4) Fase de Avaliação

Objetivo :

- Verificar e validar o software produzido. A correção do código será verificada confrontando-o com as especificações e, além disso, examinando se está de acordo com as normas e padrões estabelecidos. A adequação ao usuário será validada por intermédio de experimentos (testes) especificamente projetados para este fim.

5) Fase de Operação

Objetivos :

- Utilizar o software produzido;

- Corrigir, aprimorar, adaptar e expandir o software, se necessário.

II.2.3.3 Modelo Proposto por Yourdon [YOURDON89]

Número de atividades : 9

Nome das atividades : Análise Preliminar dos Requisitos

Análise

Projeto

Implementação

Geração de Testes de Aceitação

Garantia de Qualidade

Descrição dos Procedimentos

Conversão da Base de Dados

Instalação

Os objetivos de cada uma das atividades são respectivamente:

1) Análise Preliminar dos Requisitos

Objetivos :

- Identificar deficiências no ambiente do usuário;

- Estabelecer metas para o novo sistema (pode, também, consistir de uma lista de funções existentes que necessitam ser implementadas ou novas funções que devem ser acrescidas);

- Determinar a possibilidade de automatizar o sistema, sugerindo novas alternativas;

- Preparar um documento de planejamento do projeto que será utilizado no decorrer do desenvolvimento.

2) Análise

Objetivo :

- Transformar os requisitos do usuário e o documento do produto numa especificação estruturada apresentada através de diagrama de fluxo de dados, diagrama de entidade e relacionamento, diagrama de transição de estados, etc.

3) Projeto

Objetivos :

- Identificar a hierarquia apropriada para a programação dos módulos;

- Identificar as interfaces dos módulos para implementar a especificação criada;

- Transformar o modelo de dados (apresentados da forma entidade e relacionamento) num modelo de base de dados.

4) Implementação

Objetivo :

- Codificar e integrar os módulos de maneira progressiva para a formação completa do esqueleto do sistema, utilizando técnicas de programação estruturada e implementação top-down.

5) Geração de Testes de Aceitação

Objetivos :

- Definir a aceitação do sistema;

- Gerar casos de teste de aceitação.

6) Garantia de Qualidade

Objetivos :

- Conduzir o teste do sistema final;

- Produzir o manual do usuário;

- Realizar treinamento do usuário.

7) Descrição dos Procedimentos

Objetivos :

- Descrever formalmente as partes do novo sistema;

- Descrever a interação do usuário com as partes automatizadas do novo sistema.

8) Conversão da Base de Dados

Objetivos :

- Estabelecer controle de conversão;
- Executar conversão dos dados.

9) Instalação

Objetivos :

- Treinar os usuários na utilização do novo sistema;
- Orientar os usuários no que diz respeito a hardware.

Na figura 6 apresentamos este modelo sob a forma de diagrama de fluxo de dados. Como se pode observar não há nenhuma regra com relação a se terminar a atividade N antes de começar a N+1. As atividades podem ser realizadas em paralelo, por isso utiliza-se a palavra atividade em vez de fase.

II.2.4 - Modelos Baseados em Protótipos

Segundo Sommerville [SOMMERVILLE82] é desejável, ao se desenvolver um produto, ter-se uma avaliação do mesmo através da construção de um protótipo antes de construir o produto final. Este protótipo serve para fins de avaliação e não necessita satisfazer todos os requisitos do produto final.

Entretanto, prototipação não tem sido muito usada no desenvolvimento e avaliação de software. O principal

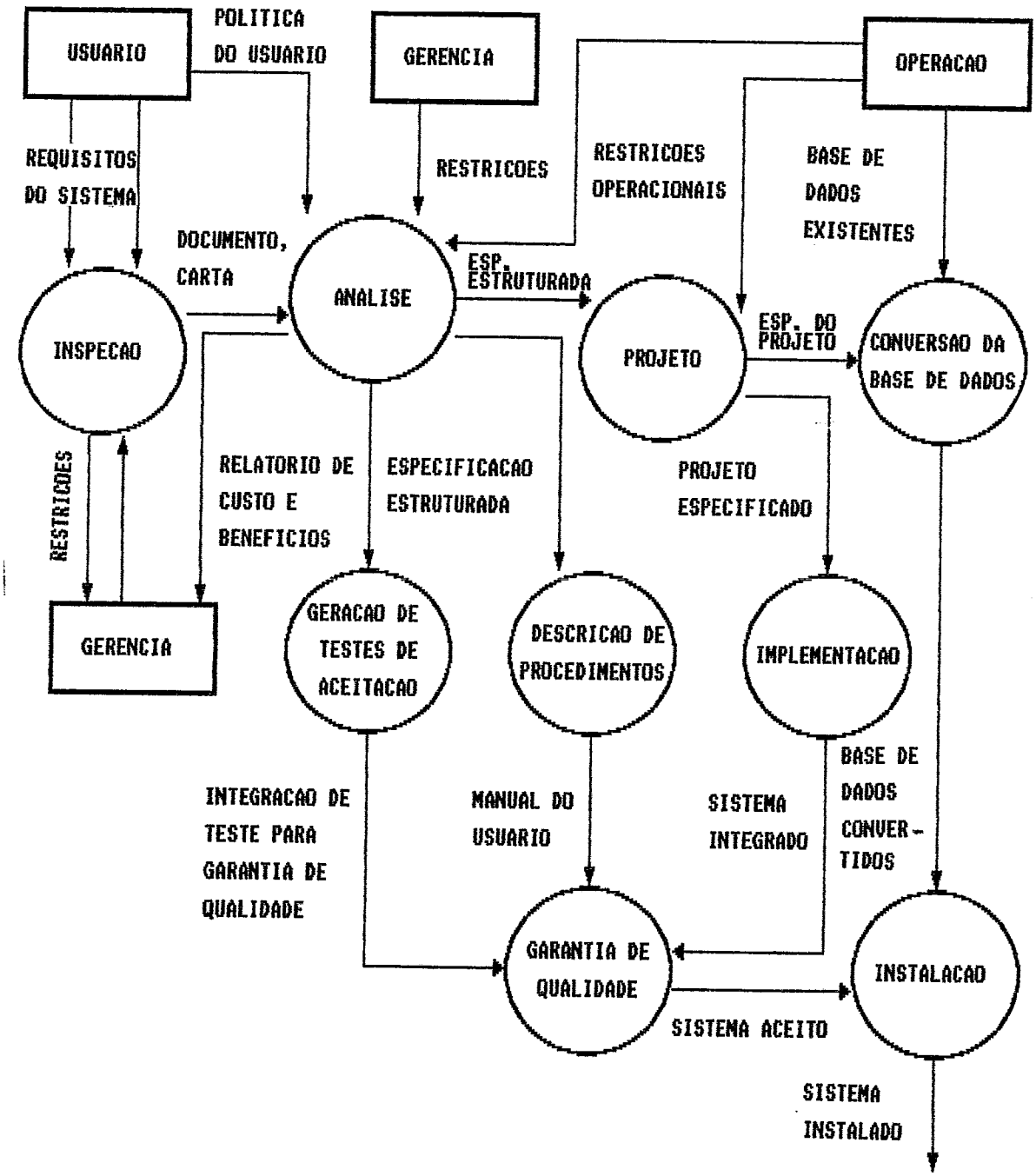


FIGURA 6 - Modelo Estruturado de Ciclo de Vida [YOURDON89]

argumento contra, é que o custo do desenvolvimento de um protótipo representa uma grande e inaceitável fração do custo total do produto. Considera-se, no caso de software, mais econômico modificar um sistema já pronto do que prover a oportunidade para o usuário compreender e aperfeiçoar sua necessidade antes da construção. Em contraste, o custo do protótipo de um sistema mecânico ou eletrônico no qual há produção em massa, representa um pequeno incremento no custo unitário final do sistema sendo que o custo de uma modificação mesmo que insignificante, depois de pronto é elevado e algumas vezes inviável.

Prototipação de software é cara se o protótipo é implementado usando as mesmas ferramentas e o mesmo critério que para o produto final. Entretanto se o protótipo tem como finalidade demonstrar algum aspecto fundamental do produto, ele pode ser desenvolvido com um custo significativamente mais baixo do que o produto final.

Segundo Boar [BOAR84] [YOURDON89], um enfoque alternativo para a definição dos requisitos é absorver inicialmente um conjunto reduzido de necessidades para implementá-las rapidamente com intenção de expandí-las iterativamente, refinando-as com o usuário e o desenvolvedor do sistema. Assim sendo, a definição do sistema ocorre gradualmente de forma evolutiva, o que caracteriza o enfoque de prototipação.

Segundo Mendes [MENDES87] a construção de um protótipo é considerada de grande utilidade, não só na fase de especificação do software, mas em todas as suas fases de

desenvolvimento. É claro que a construção de protótipos não é necessária para o desenvolvimento de qualquer tipo de software. Na realidade, protótipos são extremamente úteis para construção de produtos de software de comportamento dinâmico, onde o fator tempo é crucial.

Há várias razões para o desenvolvimento de um protótipo [FAIRLEY85] :

- Exemplificar ao usuário os dados de entrada do software, as mensagens e relatórios existentes no produto;

- Explorar novas técnicas e aperfeiçoar as técnicas já existentes para o produto;

- Realizar o desenvolvimento exploratório para produtos onde exista desconhecimento de como resolver o problema.

Existem diferentes abordagens para prototipação [DAVIS88], [HIRSCH85], [BOEHM86], [McCRACKEN82]:

a) Prototipação rápida descartável;

b) Desenvolvimento incremental;

c) Prototipação evolutiva.

O enfoque de prototipação rápida que se tornou popular através de Gomma [GOMMA81] busca assegurar que o produto de software a ser desenvolvido realmente atende à necessidade do usuário. Segundo este enfoque constrói-se uma implementação parcial rápida. O usuário utiliza esse protótipo por um determinado período de tempo e, assim fornece dados para o

desenvolvimento definitivo com relação aos pontos fracos e positivos encontrados no mesmo. Essa realimentação é utilizada para modificar as especificações dos requisitos do sistema refletindo a real necessidade do usuário. A partir deste ponto, é feito o projeto e a implementação do sistema real com maior segurança de que se está construindo o sistema correto. Uma possível extensão para esse enfoque é a construção de uma série de protótipos rápidos, que são descartados até se chegar ao final.

Desenvolvimento incremental é o processo de se construir, inicialmente, uma implementação parcial do produto e aos poucos ir adicionando ao mesmo funcionalidade ou desempenho.

Esse enfoque reduz o custo antes que uma implementação inicial seja concluída com êxito. Também gera um sistema, em forma operacional, mais rápido, reduzindo a possibilidade de que ocorram mudanças nas necessidades do usuário, durante o processo de desenvolvimento.

Na abordagem de prototipação evolutiva é construída uma implementação parcial do produto composta dos requisitos já conhecidos. O protótipo passa a ser utilizado por seus usuários para com o uso aumentar a compreensão de seus requisitos.

Considerando que a abordagem de desenvolvimento incremental embute a idéia de que conhecemos a maioria dos requisitos do produto e, simplesmente, escolhemos prioridades de implementação aumentando, pouco a pouco, sua capacidade, a prototipação evolutiva embute a idéia de que não conhecemos

todos os seus requisitos e que temos necessidade de experimentar o que conhecemos, em um sistema operacional, de forma a poder aprender sobre as necessidades do produto.

No caso de protótipos descartáveis são implementados, primeiro, os aspectos do sistema menos conhecidos. No caso dos protótipos evolutivos começa-se, no entanto, com os aspectos do sistema que mais se conhece.

A seguir detalhamos alguns modelos que consideram o enfoque de prototipação:

II.2.4.1 - Modelo Proposto por Connor [CONNOR80]

Connor considera que existem pelo menos três níveis relacionados ao ciclo de vida de software e que devem ser considerados por sua grande importância:

1) Organização do Ciclo de Vida de Software, que corresponde a sociedade, público ou entidade administrativa, onde outros produtos de software residem (onde há a definição e estudo de outros produtos).

2) Informação do Ciclo de Vida de Software, que corresponde a existência de manual e de requisitos de softwares automatizados.

3) Ciclo de Vida de Software de Aplicação, que está diretamente relacionado à parte automatizada de um software particular.

Antes do primeiro nível pode-se considerar o ambiente da

organização como um todo e após o terceiro nível há os ciclos de vida associados aos diversos produtos de software e hardware.

Atividades distintas para esses três níveis do ciclo de vida podem ser definidas como por exemplo: gerência, análise, projeto, implementação e operação.

Isto pode ser visto na figura 7 onde Connor evidência sua visão tridimensional do ciclo de vida de software.

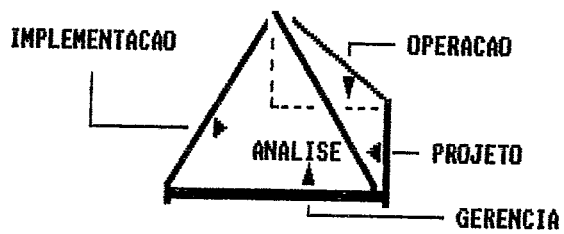


Figura 7 - Visão Tridimensional do Ciclo de Vida de Software [CONNOR80]

Connor descreve três modelos de ciclo de vida: um modelo para o desenvolvimento de um novo produto de software, um modelo para a modificação de um software já existente e um modelo baseado na construção de protótipo.

a) Modelo para desenvolvimento de um sistema

Nesse modelo uma decisão administrativa é o ponto de partida (1), o plano do projeto no nível de informação é desenvolvido a seguir (2), causando a análise no mesmo nível (3).

b) Modelo para modificação de um sistema já existente

Nesse modelo a decisão administrativa para modificar o software, também, é o ponto de partida (1). Uma análise (2) e posterior modificação no projeto já existente (3) é realizada antes que haja uma modificação no código (4). Finalmente há uma alteração na operação a nível de software (5).

c) Modelo envolvendo a construção de um protótipo

Nesse modelo uma decisão administrativa nos dois primeiros níveis (1) e (2) causa a implementação de um protótipo (3) ao nível de software.

O resultado dessa atividade é a análise (4) e posterior construção de um projeto (5), resultando numa implementação em dois níveis distintos (6) e (7). Uma análise (8) e posterior operação em dois níveis (9) e (10) encerram as atividades previstas neste modelo.

A figura 8 dá uma visão geral da proposta de Connor para os três modelos de ciclo de vida.

II.2.4.2 - Modelo Proposto por Fairley [FAIRLEY85]

O modelo proposto por Fairley enfatiza:

- As fontes que estabelecem requisitos para o software;
- Os principais pontos de decisão;
- Construção de um protótipo.

MODELOS / NIVEIS / ATIVIDADES	DESENVOLVENDO UM SOFTWARE			MODIFICANDO UM SOFTWARE			CONSTRUINDO UM PROTOTIPO		
	ORGANIZACAO	INFORMACAO	SOFTWARE DE APLICACAO	ORGANIZACAO	INFORMACAO	SOFTWARE DE APLICACAO	ORGANIZACAO	INFORMACAO	SOFTWARE DE APLICACAO
GERENCIA	1	2			1		1	2	
ANALISE	4	3	5		2			8	4
PROJETO		5	6			3		5	
IMPLEMENTACAO		8	7			4		7	3/6
OPERACAO		10	9			5			9

FIGURA 8 - Sequência de Atividades nos Modelos de Connor [CONNOR80]

De acordo com este enfoque, primeiramente, identificam-se os planos, demandas e requisitos dos usuários do produto, e solicita-se uma autorização para que os estudos de viabilidade sejam executados.

A seguir efetua-se uma análise preliminar dos requisitos, após a qual solicita-se autorização para construção do protótipo.

O protótipo, geralmente, possui capacidade funcional limitada, baixo desempenho e pouca confiabilidade.

Assim sendo estão previstas três fases: análise, construção do protótipo e construção do produto final.

A fase de análise identifica os planos, demandas e requisitos dos usuários do produto, após a qual é solicitada uma alteração para que os estudos de viabilidade sejam executados.

A fase de construção do protótipo é composta pelas atividades de projeto, geração de código, teste e demonstração ao usuário. Estas atividades são revestidas de pouco formalismo e são de rápida execução. Após a demonstração ao usuário, obtém-se a autorização para implementação do produto final.

A fase de construção do produto final é composta pelas atividades de projeto detalhado, codificação, verificação, integração dos componentes do software, documentação e demonstração.

Após a implementação do produto final é realizada uma validação e o software entra em produção.

A figura 9 representa esta proposta.

II.2.4.3 - Modelo Proposto por Gladden [GLADDEN82]

O modelo não cíclico é uma proposta alternativa para o processo de desenvolvimento de software onde se procura superar os seguintes problemas:

- Grande volume de horas gastas na alteração dos requisitos causando um impacto nas tarefas subsequentes gerando um ciclo de vida vicioso;

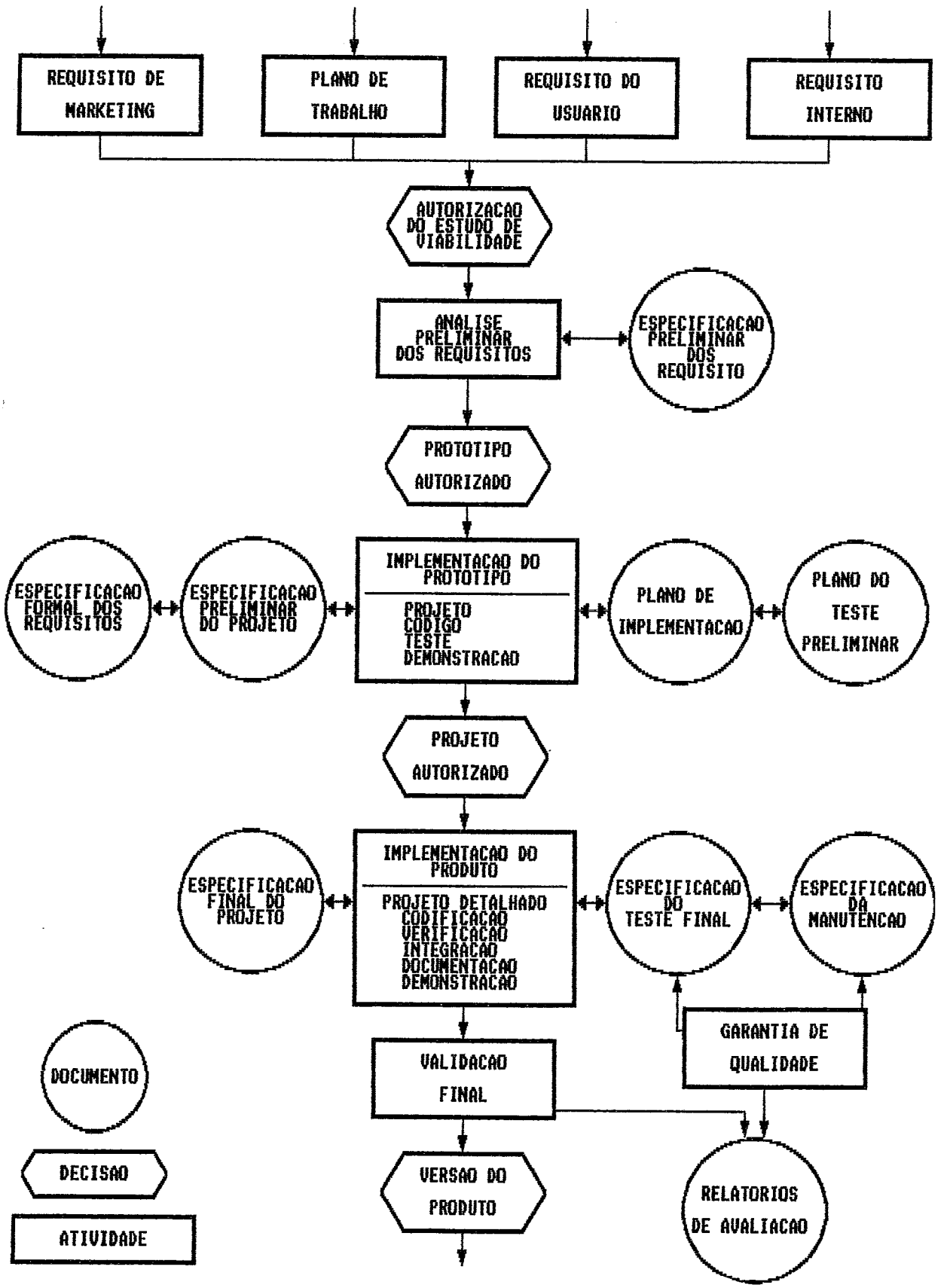


FIGURA 9 - Modelo Alterno de Ciclo de Vida [FAIRLEY85]

- Desgaste na relação com o usuário provocada pelo grande espaço de tempo entre a proposta de requisitos e o produto final, o que se manifesta em novas alterações, mais exigências ou mesmo na modificação dos elemento da tarefa.

Estas considerações formam uma cadeia de eventos que fazem com que o software se torne incompleto, pois durante o decorrer do desenvolvimento de software, novos requisitos aparecem e vão sendo incorporados. Assim quando o desenvolvimento do software finalmente se dá por encerrado (normalmente com atrasos), já não mais satisfaz ou pelo menos, lembra a primeira proposta.

A proposta de Gladden de um modelo de ciclo de vida não cíclico evita o retorno a fases anteriores do ciclo de vida, por considerar que na maioria das vezes, este é o principal causador de insucessos em projetos.

Este modelo pode ser descrito através das seguintes fases:

1) Primeira Fase: Os objetivos do software são mais importantes que os requisitos do software.

Nesta fase são estabelecidos os objetivos do software ao invés de seus requisitos, isto porque os objetivos, definidos em um nível mais elevado da organização, são mais estáveis e demandam um espaço de tempo mais curto para serem estabelecidos. Se modificados, o propósito do projeto certamente é alterado e provavelmente um novo software será necessário.

2) Segunda Fase: Um objetivo concreto contém mais informações que uma especificação escrita.

Aqui parte-se do princípio de que nada exprime melhor o conceito do software do que o próprio software. Num projeto em que devem atuar vários tipos de pessoas, nada é mais positivo e influente no seu sucesso do que vê-lo em operação. Assim sendo produz-se uma simulação do hardware hospedeiro e um protótipo rápido do software. Com isso demonstra-se aos usuários o produto a ser produzido e seus objetivos.

3) Terceira Fase: Objetivos do software mais demostrações concretas resultam em um produto de sucesso.

O software é desenvolvido e finalmente entra em operação, após uma nova demonstração ao usuário, que já possuirá uma noção do que receberá.

Este modelo apresenta as seguintes vantagens:

- Objetivos são mais estáveis e menos resistentes a mudanças do que os requisitos;

- Todos os objetivos podem ser definidos antes que o projeto comece;

- Ansiedade do usuário e posterior tendência a colocar mais exigências é atenuada pela versão inicial demonstrada;

- A flexibilidade na implantação do software é aumentada na medida em que o usuário está convencido do que o software fará quando for acionado;

- Há redução no volume de erros porque todos os participantes entendem os objetivos do software e as mudanças nessa compreensão são reduzidas.

A figura 10 dá uma visão geral desse modelo

II.2.4.4 - Modelo Proposto por Boar [BOAR84] [YOURDON89]

Este enfoque parte de uma consideração inicial, após a atividade de Análise Preliminar dos Requisitos, onde se determina se o projeto é ou não um candidato a desenvolvimento por prototipação. Essa determinação baseia-se nas seguintes características:

- O usuário é incapaz, ou sem vontade de examinar modelos abstratos do produto;

- O usuário tem dificuldade de expressar os requisitos do produto;

- O produto a ser desenvolvido é "on-line";

- O produto não requer uma especificação significativa.

Um perigo significativo, tanto para o usuário quanto para o grupo desenvolvedor, é fazer a transformação direta do protótipo num produto tornando-o irreal devido ao grande volume de transações necessárias para a eficiência do produto final.

A figura 11 representa esta proposta.

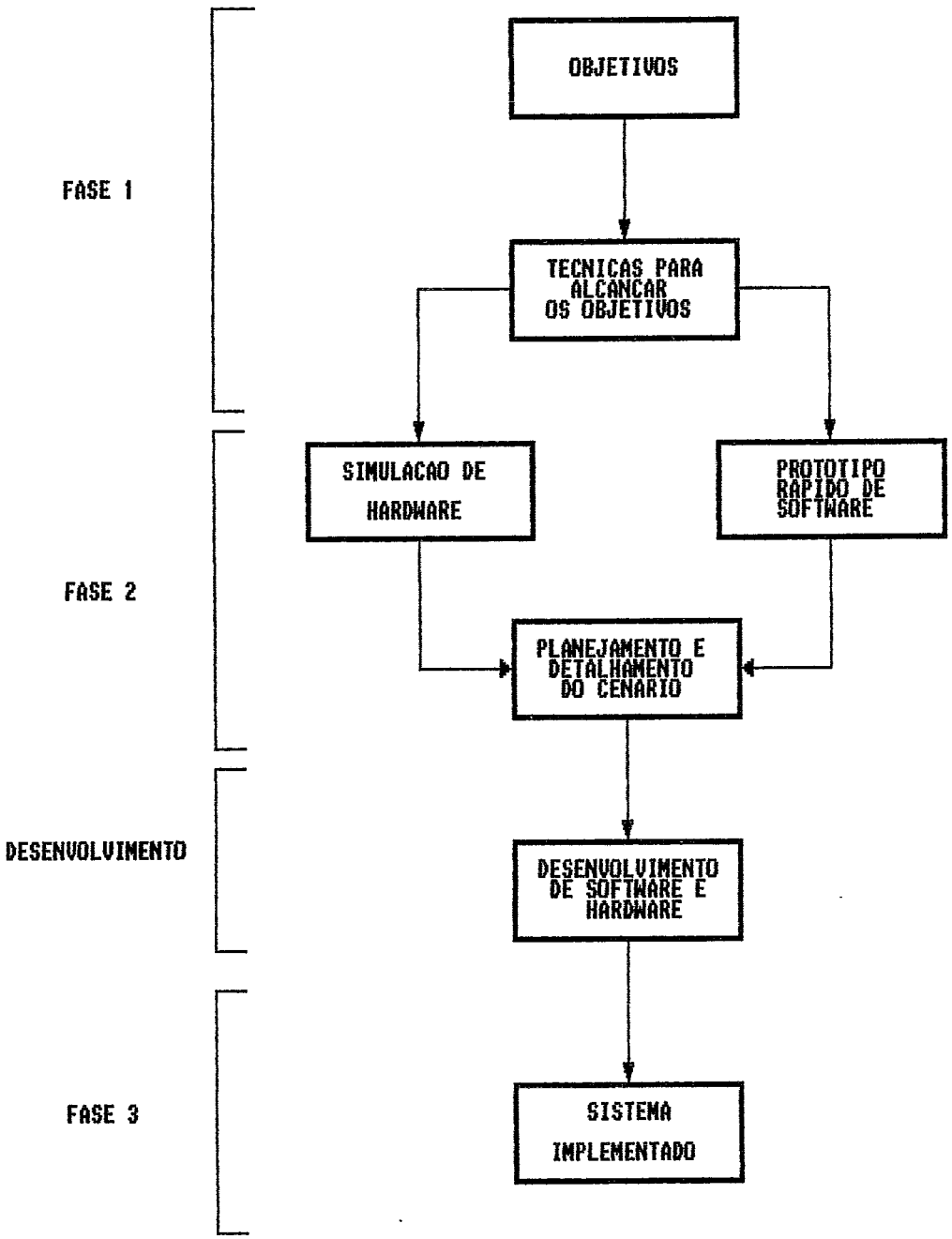


FIGURA 10 - Modelo Não Cíclico (Hollywood) [GLADDEN82]

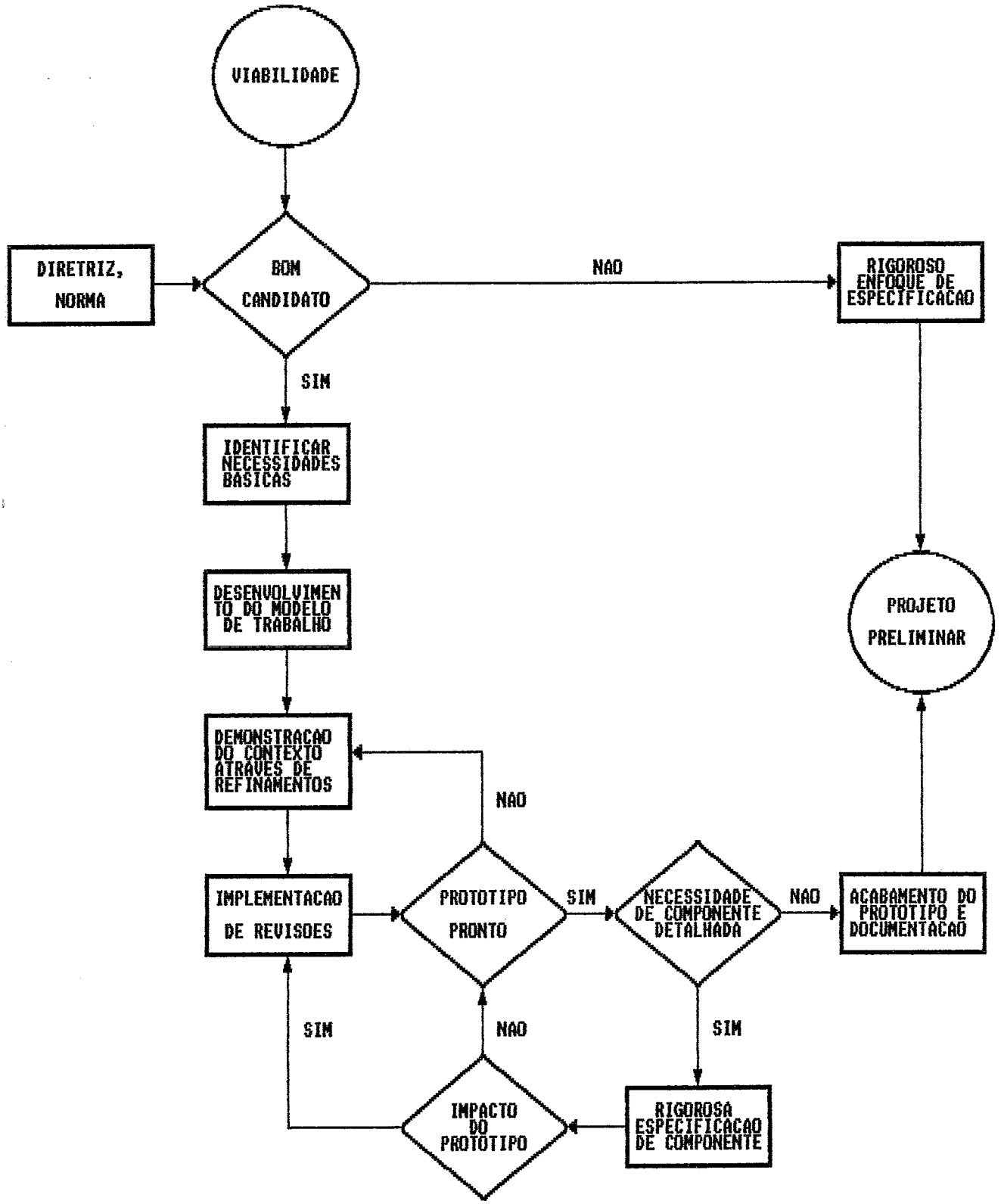


FIGURA 11 - Modelo Baseado em Protótipo [BOAR84] [YOURDON89]

II.2.4.5 - Modelo Baseado em Versões Sucessivas [FAIRLEY85]

O desenvolvimento de um produto baseado no modelo de versões sucessivas proposto por Fairley é um exemplo da abordagem de desenvolvimento incremental.

Nessa abordagem cada versão sucessiva do produto é um sistema funcional capaz de executar o trabalho de maneira proveitosa pelo usuário. Cada versão gerada pode ser vista como um protótipo onde um esqueleto inicial do produto é refinado em níveis crescentes de funcionalidade.

A construção da primeira versão no modelo segundo Fairley é constituída das fases de Análise, Projeto, Implementação e Uso do Produto.

A elaboração das versões sucessivas é vista através dos dois enfoques mencionados a seguir:

1) Projeto e Implementação das Versões Sucessivas

A fase de análise seguida por projetos iterativos, implementação e avaliação ou não da nova versão conforme pode ser visualizado na figura 12. Nesta figura, as linhas pontilhadas indicam que a aprovação da versão i pode preescrever a necessidade de uma análise adicional antes do projeto da versão i+1.

2) Análise e Projeto Sucedido Pela Implementação das Versões Sucessivas

Nesse enfoque, todas as versões do produto (1 a n) são

projetadas antes da implementação de qualquer atividade. Nesse caso, as características de cada projeto sucessivo devem ser planejadas durante a fase de análise.

As linhas pontilhadas na figura 13 indicam que a implementação da versão i pode manifestar a necessidade de uma análise e projeto adicional antes do procedimento da implementação da versão i+1.

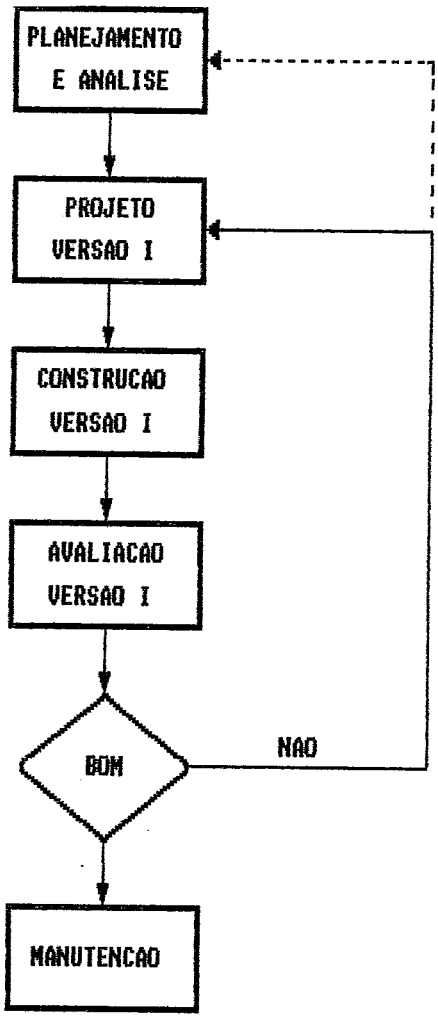


FIGURA 12 - Projeto e Implementação das Versões Sucessivas [FAIRLEY85]

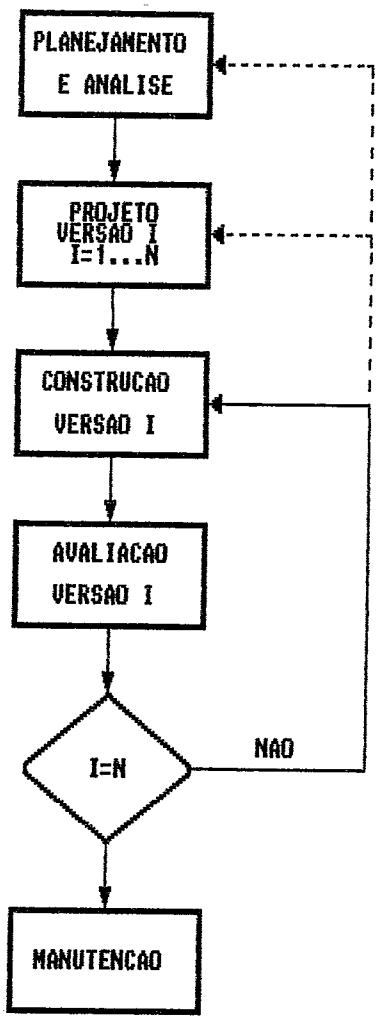


FIGURA 13 - Análise e Projeto Seguido da Implementação das Versões Sucessivas [FAIRLEY85]

II.2.5 - Modelo Baseado em Reutilização de Software [DAVIS88]

Considerando que a prototipação esforça-se para reduzir o custo de desenvolvimento através de implementação parcial, o enfoque baseado em reutilização de software busca reduzir o custo do desenvolvimento incorporando projetos e codificações anteriores em novos produtos de software.

Claramente, o que é necessário são técnicas para criar

componentes, técnicas de software e ferramentas que uma vez já usadas e testadas possam ser reutilizadas. A preocupação principal é ter este conjunto de insumos devidamente catalogados para que possam ser rapidamente resgatados. A vantagem obtida está em menos tempo de desenvolvimento e na geração de software mais confiável visto que os componentes já foram previamente avaliados.

II.2.6 - Modelo de Síntese Automática de Programas [DAVIS88]

Esse termo é usado para descrever a transformação da especificação de requisitos ou de projetos de alto nível em código de máquina, através do surgimento das linguagens de muito alto nível (VHLL), e de técnicas algoritmicas baseadas no conhecimento. Com essa abordagem, o tempo de desenvolvimento é bastante reduzido.

II.2.6.1 - Modelo de Transformação de Especificação em Código [BALZER83] [BOEHM86] [BOEHM88]

A transformação direta da especificação para o código executável encurta a solução do problema possibilitando assim uma ajuda substancial no entendimento tanto da especificação básica como no enfoque de prototipação evolutiva.

O modelo de transformação admite a existência da aptidão de converter automaticamente uma especificação formal de um produto de software em programas que satisfaça a especificação. Os passos determinados por esse modelo são :

- Uma especificação formal da melhor compreensão inicial do projeto desejado;

- Transformação automática da especificação em código;

- Um laço iterativo, se necessário, para aperfeiçoar a execução do código resultante, através de orientação ao sistema a ser transformado;

- Treinar o produto resultante e

- Um iterativo laço externo para ajustar a especificação baseada na experiência operacional resultante; e deduzir, aperfeiçoar e exercitar novamente o produto de software ajustado.

O modelo de transformação supera a dificuldade de modificar o código que foi pouco estruturado através de repetidos aperfeiçoamentos, desde as modificações feitas até as especificações. Evita também o tempo extra e gasto envolvido no projeto, código e atividades de teste intermediário.

Entretanto o modelo de transformação possui várias dificuldades. A competência da transformação automática é válida somente para pequenas aplicações em linguagem de quarta geração e nos domínios da ciência da computação.

Esse modelo tem em comum algumas dificuldades do modelo de prototipação evolutiva, como a suposição que o sistema do usuário será sempre flexível o suficiente para suportar passos evolutivos não planejados.

II.2.7 - Modelos Propostos para Desenvolvimento Orientado a Objetos

Alguns autores, [SEIDEWITZ87] e [MATTOS090], propuseram novos modelos ou variações de modelos já existentes, para serem adotados durante o processo de desenvolvimento de software com orientação a objetos. A seguir veremos as propostas destes autores.

II.2.7.1 - Modelo Proposto por Seidewitz [SEIDEWITZ87]

O método GOOD (General Object-Oriented software Development), proposto por Seidewitz, tem como objetivo desenvolver software para ser implementado na linguagem ADA. Para tal, ele se baseia na Análise Estruturada e em alterações do Projeto Estruturado, de modo a gerar as estruturas de ADA.

O modelo de ciclo de vida proposto, baseia-se no modelo cascata, tendo como principais fases a Análise, o Projeto, Implementação e Reutilização. As alterações ocorrem principalmente na fase de Projeto, onde é sugerida uma série de revisões incrementais em diferentes pontos do projeto, como revisão do projeto preliminar e revisão de projeto detalhado. A fase de reutilização tenta aproveitar as construções já existentes em ADA.

II.2.7.2 - Modelo Proposto por Mattoso [MATTOS090]

O modelo de ciclo de vida do Ambiente de desenvolvimento

de Software Orientado a Objetos TABA-OBJ é uma variação do modelo de desenvolvimento incremental ou por versões sucessivas. A variação no modelo de desenvolvimento incremental ou por versões sucessivas, se refere ao procedimento durante a elaboração de cada versão. No desenvolvimento por versões sucessivas, especifica-se inicialmente uma versão com as funções básicas do sistema, projeta-se e finalmente se constrói esta versão. A partir daí, novas versões vão sendo construídas introduzindo-se novas funções. Todavia, no desenvolvimento de software com orientação a objetos, mesmo durante o desenvolvimento de cada versão, as atividades de cada fase, por serem muito iterativas, acabam se misturando.

É comum, no desenvolvimento de software com orientação a objetos, durante a fase de projeto, identificarem-se novos objetos, necessitando assim especificá-los. Por outro lado, como o grau de reusabilidade é grande, muitos objetos já existentes poderão ser reaproveitados, não sendo necessário especificá-los e projetá-los. Pode-se também, aproveitar classes já existentes, introduzindo-se um método a mais sendo necessário somente a especificação e projeto desta extensão da classe.

Como não há uma sequência rígida entre as fases do desenvolvimento, as atividades de cada fase acabam se misturando, pode-se, simultaneamente, estar fazendo o detalhamento do projeto de uma classe e se especificando outra. Uma vez que o desenvolvimento por versões sucessivas parte do desenvolvimento de uma primeira versão bem

simplificada e aos poucos vão se construindo novas versões com um grau de complexidade maior, fica mais fácil de se fazer esse tipo de desenvolvimento não sequencial. Para suportar tal estratégia, o ciclo de vida proposto prevê atividades de integração das partes que forem sendo desenvolvidas e a consequente avaliação da versão resultante desta integração.

Caso fosse ser desenvolvido um sistema muito grande em uma única versão, seria muito difícil gerenciar os diversos objetos, cada um em uma fase diferente de desenvolvimento. Por outro lado, como os métodos de desenvolvimento de software com orientação a objetos ainda foram pouco utilizados, torna-se difícil sua utilização para especificar e projetar um sistema todo de uma única vez.

O modelo proposto por Mattoso [MATTOS090] é apresentado a seguir:

Número de fases : 4
Nome das fases : Definição
Projeto
Construção
Integração e Avaliação

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Definição

Objetivos :

- Entender o problema e definir os objetivos do produto;

- Definir e planejar os ciclos de versões;
- Analisar e especificar os requisitos.

2) Fase de Projeto

Objetivos :

- Elaborar o projeto da arquitetura baseado na especificação dos requisitos;
- Escolher o ambiente de programação;
- Detalhar o projeto.

3) Fase de Construção

Objetivos :

- Implementar os componentes do produto;
- Depurar cada classe de objetos codificada;
- Planejar a avaliação do produto.

4) Fase de Integração e Avaliação

Objetivo :

- Integrar e avaliar o produto.

II.2.8 - Modelo Espiral [BOEHM86]

O modelo espiral foi introduzido com o objetivo de resolver alguns problemas que tiveram origem na aplicação dos modelos já mencionados anteriormente e acrescentar vantagens através de uma nova estrutura de desenvolvimento de software. Para alcançar essa meta, o modelo espiral utiliza condutas bem diferentes dos outros modelos.

Este modelo adapta vários dos modelos anteriores e ajuda a prover orientação para combiná-los da melhor maneira em diversas situações, na busca de adaptar o processo de desenvolvimento para a especificação da fonte de risco do projeto em andamento.

Um aspecto importante neste modelo de ciclo de vida é que as considerações sobre os riscos têm um grande impacto nas atividades existentes durante o ciclo. Isto quer dizer que certas atividades podem ser omitidas se não houver nenhum risco envolvido nesta ação e em compensação outras atividades podem passar a ter uma grande atenção caso o risco nela envolvido seja alto.

Devido a não ordenação das atividades, o modelo espiral é um modelo genérico que engloba dois extremos : o modelo cascata e a prototipação evolutiva.

O ciclo de vida em espiral modela o processo de desenvolvimento de software como uma sequência de ciclos progressivos. Cada ciclo consiste de quatro passos:

- Determinação dos objetivos das partes do produto que será elaborado (execução, funcionalidade, capacidade de acomodar mudanças, etc), das alternativas de implementação das partes do produto (reutilização, compra, etc) e restrições impostas na aplicação das alternativas (custo, cronograma, interface, etc);

- Avaliação de alternativas, identificação e solução de riscos;

- Desenvolvimento e verificação do produto do próximo nível;

- Planejamento da próxima fase.

A partir da identificação do primeiro passo deve-se avaliar as alternativas relativas aos objetivos e restrições. Frequentemente este processo identificará áreas inseguras que são fontes significativas de risco de projeto. Caso existam essas áreas, o próximo passo envolve a formulação de estratégia de custo efetivo para resolver as fontes de risco; podendo envolver prototipação, simulação, banco de teste, questionários ao usuário ou mesmo combinações dessas e outras técnicas de resolução de risco. Uma vez que os riscos forem avaliados e estimados, o próximo passo é determinado pelos riscos relativos existentes.

Caso o risco da execução ou da iteração com o usuário domine fortemente o desenvolvimento do produto, o próximo passo pode ser uma prototipação evolutiva. O esforço mínimo para especificar a natureza global do produto, o plano para o próximo nível de prototipação e o desenvolvimento de um protótipo mais detalhado devem ter continuidade para resolver os riscos.

Se esse protótipo é operacionalmente útil e robusto o suficiente para servir como base de baixo risco para evoluções de produtos futuros, o próximo passo envolve uma série de protótipos evolutivos.

Caso o empenho de uma prototipação prévia resolva os

riscos de execução ou interface com o usuário e o risco de desenvolvimento de programas ou controle de interface, o próximo passo segue o enfoque do modelo cascata modificado adequadamente para incorporar o desenvolvimento incremental.

Uma característica do modelo espiral é que cada ciclo é completado por uma revisão envolvendo as principais pessoas interessadas no produto. A revisão cobre todo o produto desenvolvido no ciclo e inclui também o plano para o próximo ciclo. O maior objetivo dessa revisão é assegurar que todas as partes interessadas estão de pleno acordo ao enfoque dado para a próxima fase.

Em último caso deve-se visualizar uma série de ciclos paralelos da espiral, cada um com seus componentes, isto é, espirais distintas podem ser desenvolvidas por diferentes componentes de software.

Quatro questões fundamentais surgem ao ser considerado este modelo de desenvolvimento:

- Como começar uma espiral ?

- Como sair da espiral quando for conveniente terminar o projeto mais cedo ?

- Porque a espiral termina tão bruscamente ?

- O que acontece com a manutenção do software ?

A espiral começa através da hipótese que uma missão particular deve ser melhorada ou aperfeiçoada através de um software. O processo espiral envolve um teste dessa hipótese.

A qualquer momento se a hipótese for insuficiente para o teste a espiral é terminada. Por outro lado a espiral termina com a instalação de um software novo ou devidamente modificado, onde a hipótese é testada observando o efeito operacional.

Usualmente experiências com o software operacional conduzem a hipóteses adicionais no que diz respeito a melhorias e espirais para novas manutenções são iniciadas com a finalidade de testar as hipóteses. Iniciação, término e iteração das tarefas e produtos dos ciclos anteriores estão implicitamente definidos nesse modelo.

A primeira vantagem do modelo espiral é que ele contém uma série de opções que podem satisfazer os aspectos dos vários modelos de processo de software, enquanto o enfoque de direcionamento de risco evita a maioria das dificuldades.

A condição inicial no qual esse modelo torna-se equivalente a outros modelos pode ser resumida da seguinte maneira :

a) Caso o projeto possua um baixo risco em determinados aspectos como uma iteração errônea com o usuário ou não haver um ponto de encontro com os restritivos requisitos de execução e se este possui um alto risco no orçamento e em ítens como previsibilidade de controle, então estas considerações de risco levam ao modelo espiral adotar o equivalente ao modelo cascata.

b) Se o projeto tiver um baixo risco em certos aspectos

como orçamentos baratos e lista de previsibilidade e controle como também no encontro com o problema de software com informações "esclerosadas" e se possuir alto risco em aspectos como levar a uma iteração errônea com o usuário, então estas considerações de risco levam ao modelo espiral adotar o equivalente ao prototipação evolutiva.

c) Se as capacidades de geração de software automatizado são possíveis então o modelo espiral reúne tanto o modelo de prototipação rápida descartável ou a aplicação do modelo de transformação dependendo das considerações de riscos envolvidas.

d) Se os elementos de alto risco de um projeto envolverem um conjunto dos ítens de risco que já foram mencionados, então o enfoque do modelo espiral irá refletir um conjunto próprio desses modelos.

Assim sendo, os aspectos que evitam a realização dos riscos também evitarão as dificuldades dos outros modelos. O modelo espiral possui uma série de vantagens adicionais como:

- Focaliza opções envolvendo a reutilização de produtos de software já existentes;

- Reúne a preparação para a evolução do ciclo de vida, crescimento e mudanças no produto de software;

- Provê um mecanismo de incorporação dos objetivos de qualidade do software no desenvolvimento do próprio produto;

- Focaliza a eliminação de erros e alternativas

dispensáveis de modo rápido;

- Indaga até onde deve-se planejar, o que depende do grau de risco em não se executar todas as tarefas inerentes ao planejamento em grau suficiente;

- Não envolve a utilização de enfoques distintos para o desenvolvimento do software e futuros aumentos durante sua existência;

- Provê uma estrutura sólida para integração do desenvolvimento do sistema hardware e software.

Entretanto, este modelo, também apresenta dificuldades. As três principais dificuldades para a aplicação com sucesso do modelo espiral são : a negociação para o contrato do software, confiança no grau de experiência do engenheiro de software e a necessidade de se ter passos mais elaborados na execução do modelo espiral.

Uma solução para superar essas dificuldades é estabelecer planos de gerência de riscos.

A figura 14 dá uma visão geral desse modelo.

II.2.9 - Modelo Proposto para Desenvolvimento de Sistemas Especialistas [WATERMAN86] [RIBEIRO89]

O processo de construção de um Sistema Especialista é realizado em profunda interação entre o construtor do SE, denominado engenheiro do conhecimento, e um ou mais especialistas na área em que se está construindo o sistema.

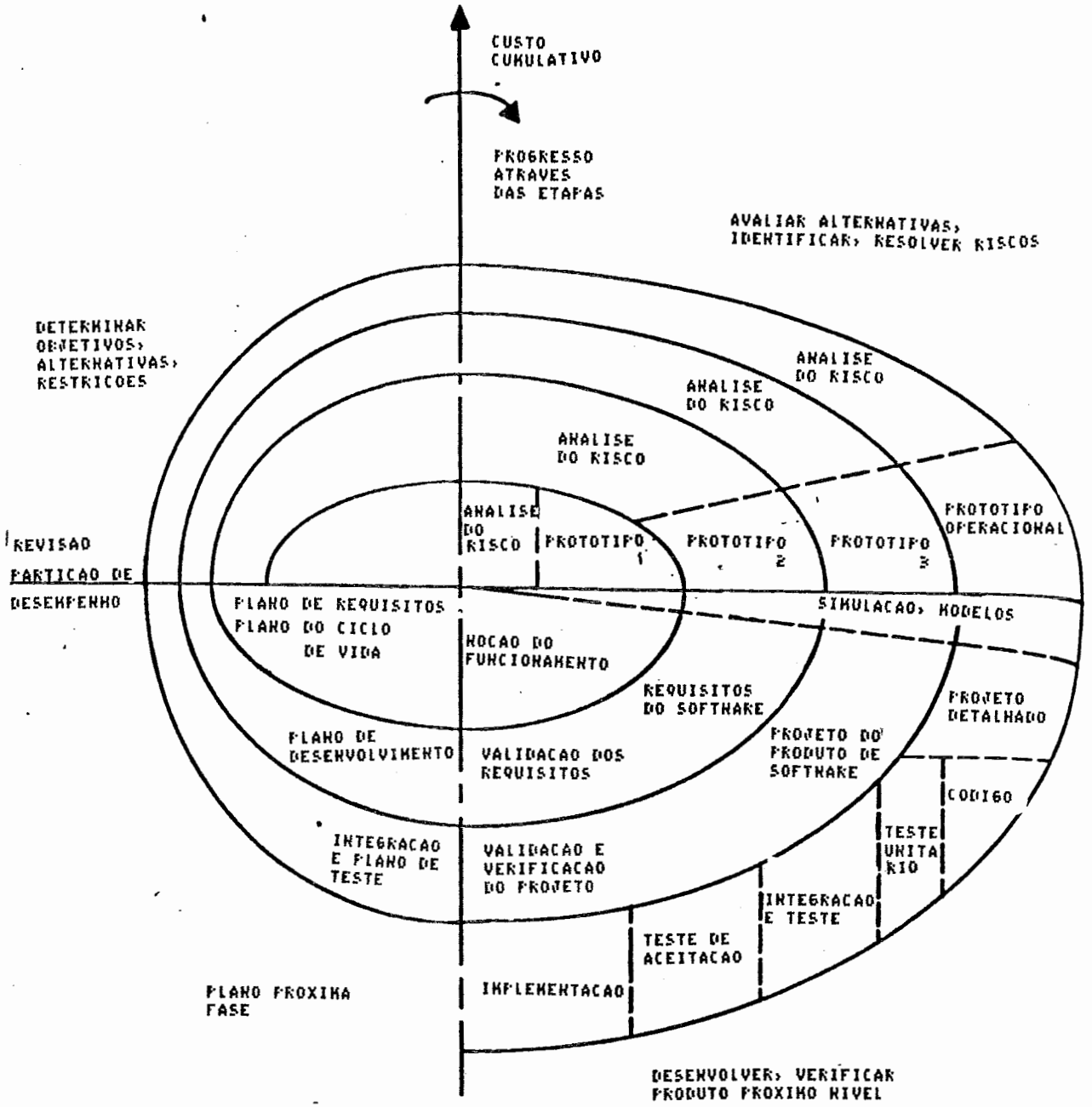


FIGURA 14 - Modelo Espiral [BOEHM86]

Durante o processo de construção do SE o engenheiro do conhecimento extrai do especialista seus procedimentos, estratégias e regras, para a solução do problema e incorpora esse conhecimento ao SE. O resultado será um sistema de computação capaz de resolver problemas de forma semelhante aos especialistas da área em questão.

O fator determinante do sucesso ou não do desenvolvimento de sistemas baseados no conhecimento está na aquisição do conhecimento.

O termo aquisição do conhecimento compreende a obtenção do conhecimento especializado necessário à solução de problemas em certo domínio e sua transferência para um programa de computador. São muitas as fontes de conhecimento úteis ao desenvolvimento de SE, como os especialistas no domínio do problema a ser tratado, os livros, os artigos publicados, os estudos de casos e a experiência do engenheiro do conhecimento.

Os princípios gerais que devem nortear os trabalhos de aquisição do conhecimento são:

- Nos primeiros estágios de desenvolvimento dos SE, o engenheiro do conhecimento deve procurar ser específico, incentivando o especialista a descrever problemas de particular interesse;

- É importante que o especialista represente o seu conhecimento da maneira que julgar mais natural. O engenheiro do conhecimento não deve impor métodos de representação

estranhos ao especialista;

- O engenheiro do conhecimento deve evitar interromper o especialista. É preciso ser paciente e considerar a possibilidade de que surjam contradições e inconsistências. No decorrer dos trabalhos as idéias serão esclarecidas sem a necessidade de interrupções que podem cortar a linha de raciocínio do especialista, fazendo com que detalhes importantes sejam esquecidos;

- A gravação das entrevistas com os especialistas e sua posterior transcrição é aconselhável, já que o construtor dificilmente será capaz de identificar, de pronto, que informações são relevantes para o sistema. A gravação metódica de todas as entrevistas e o estabelecimento de referências cruzadas do material com as regras levantadas pode ser um fator fundamental para o sucesso do sistema;

- O engenheiro do conhecimento deve observar, além dos fatos, teorias e heurísticas, a forma pela qual o especialista usa seu conhecimento, isto é, a ordem em que os problemas são levantados, a importância relativa de diferentes ítens ou a forma de atribuir pesos e evidências.

O conhecimento necessário ao desenvolvimento de um SE pode se originar de diversas fontes, sendo a principal os especialistas da área em questão. Através de sistemáticas e prolongadas entrevistas o construtor obtém dos especialistas o conhecimento necessário ao desenvolvimento do sistema. Com o objetivo de facilitar a comunicação especialista - construtor algumas técnicas para a extração do conhecimento

foram desenvolvidas. O engenheiro do conhecimento deve estar ciente da existência dessas técnicas e deve usá-las sempre que julgar conveniente. Algumas dessas técnicas são: estudos de casos interessantes, características e decisões, objetivos diferenciados, pensar em voz alta, construção por refinamentos e validação.

A medida que o conhecimento é levantado é preciso analisá-lo e refiná-lo. Essa tarefa, no entanto, não é nada fácil, pois é necessário garantir um resultado consistente, sem redundâncias. Algumas ferramentas utilizadas na representação do conhecimento são:

- Diagramas

O produto de uma entrevista, normalmente, está gravado ou escrito. Como as linguagens falada e escrita são ambíguas é preciso um grande esforço do engenheiro do conhecimento e do especialista no sentido de levantar os conceitos e relações tratados na entrevista. Os diagramas são muito úteis na representação desse conhecimento, pois facilitam sua análise e refinamento. Exemplos de diagramas são rede de inferências e árvore de decisão;

- Dicionário do conhecimento

É o dicionário de dados dos SE. Objetiva catalogar informações relevantes ao sistema em desenvolvimento;

- Diagrama de fluxo de Dados.

Algumas razões que justificam o desenvolvimento de SE são:

- A importância da disseminação do conhecimento especialista escasso: a transferência desse conhecimento de uma pessoa para outra é cara e demorada. O conhecimento de um especialista não é o tipo de conhecimento denominado público (obtido em conferências, livros, etc) mas sim privado, obtido através da experimentação em diferentes situações no decorrer de vários anos. Esse tipo de conhecimento raramente é escrito, e, por isso mesmo, deve ser salvo antes que não mais exista;

- A capacidade do SE aplicar o conhecimento de forma sistemática e consistente: um especialista pode ser levado a tomar decisões diferentes em situações idênticas, devido a fatores emocionais. O mesmo não ocorre com um SE;

- O fato do conhecimento armazenado em um SE ser permanente: um especialista ao deixar de utilizar seus conhecimentos por determinado tempo corre o risco de ter seu desempenho seriamente afetado;

- A capacidade dos SE de manipular informações contraditórias e incompletas: o uso de fatores de certeza e medidas de probabilidade têm sido incorporado a vários SE;

- O fato dos SE terem mostrado uma relação custo / benefício aceitável: fica assim demonstrada sua viabilidade econômica pois o baixo custo de operação desses sistemas tem compensado o alto custo de desenvolvimento;

- Sistemas especialistas são capazes de explicar suas decisões: ao contrário dos sistemas de software convencionais, os SE são capazes de explicar seus resultados e comportamentos. Tal habilidade é fundamental para o desenvolvimento e manutenção do software assim como para a manutenção da credibilidade do sistema.

Entretanto os SE possuem certas limitações:

- A criatividade: nem mesmo o programa mais inteligente é capaz de se aproximar do homem nesta área. Um especialista é capaz de reorganizar informações e usá-las de forma a sintetizar um novo conhecimento, enquanto um SE se comporta sempre da forma como foi programado;

- A capacidade de aprender: embora já existam programas com essa capacidade, tais programas só têm funcionado para domínios muito restritos;

- O raciocínio dos sistemas especialistas através da manipulação de símbolos que representam idéias e conceitos: para que um SE seja capaz de fazer uso de informações e transformá-las em símbolos. Nessa transformação, muitas informações relevantes são perdidas;

- O fato dos sistemas especialistas não possuirem o que se poderia chamar de "senso comum do conhecimento", isto é, um largo conhecimento do mundo que todos os seres humanos possuem e usam: sem esse tipo de conhecimento, um SE, caso inquirido, pode tentar responder às perguntas mais descabidas, cujas respostas nem mesmo existem.

Para que o desenvolvimento de um SE seja possível é necessário que o domínio do problema possua todas as seguintes características:

- A área do problema considerado deve possuir especialistas pois para que o engenheiro do conhecimento seja capaz de produzir programas verdadeiramente habilidosos é necessária uma poderosa fonte de conhecimentos;

- Os especialistas devem, de forma geral, concordar na solução dos problemas. Caso contrário será impossível avaliar o desempenho do sistema;

- Os especialistas devem ser capazes de explicar os métodos que utilizam na resolução de problemas, uma vez que tais métodos são parte do conhecimento que deve ser inserido no sistema;

- A solução dos problemas deve requerer habilidades cognitivas e não físicas, embora não seja impossível implementar um SE que possua um pouco de cada. Nesse caso, a parte física seria tratada por outros métodos;

- Assuntos extremamente difíceis não são adequados à construção de SE pois o processo de extração do conhecimento fica prejudicado;

- Assuntos cujos conhecimentos não estão suficientemente estruturados, por serem muito novos ou por não estarem ainda bem entendidos, também não devem ser tratados pela engenharia do conhecimento;

- Para que seja possível a implementação de um SE, a tarefa não pode requerer significativa quantidade de senso comum.

O simples fato do desenvolvimento de um SE ser possível não significa que ele deva ser implementado. É preciso que os motivos justifiquem tal esforço. Alguns desses motivos são apresentados a seguir:

- Quando a relação custo / benefício é altamente vantajosa;

- Quando há falta de especialistas no mercado de trabalho;

- Quando, por fatores diversos, o conhecimento especialista de uma empresa esteja sendo perdido;

- Quando o conhecimento especialista é requerido em diversos lugares, simultaneamente, ou quando requerido em locais hostis ao ser humano.

Por fim, os fatores relativos à determinação de quando um SE é apropriado. Tais fatores estão relacionados à natureza, complexidade e escopo do problema a ser tratado.

- Quanto à natureza, para que um problema possa ser resolvido por um SE é necessário que ele possa ser naturalmente resolvido via manipulação de símbolos;

- A complexidade deve ser tal que um ser humano leve anos de estudo ou prática para se tornar um especialista na área. Entretanto, como já foi dito, o problema não deve ser

extremamente difícil;

— Quanto ao escopo, não deve ser amplo demais, para que o problema seja manipulável, nem restrito demais, para garantir o interesse prático no assunto.

Não existe uma sequência bem definida de passos que possa ser seguida no desenvolvimento de um SE. A complexidade inerente ao processo de desenvolvimento desses sistemas impede que essa sequência exista, fazendo com que seus construtores desenvolvam os sistemas de forma evolutiva.

O primeiro objetivo no método de desenvolvimento de um SE deve ser a implementação de um protótipo do sistema desejado que propiciará ao construtor um melhor entendimento do problema. Uma vez atingido esse objetivo, o engenheiro do conhecimento passa à fase seguinte que consiste aperfeiçoar o protótipo através de refinamentos sucessivos.

Apesar de não se conhecer uma sequência exata de passos a seguir, sabe-se que o processo de desenvolvimento é composto das seguintes fases [RIBEIRO89]:

Número de fases : 5
Nome das fases : Identificação
Conceituação
Formalização
Implementação
Testes

Os objetivos de cada uma destas fases são respectivamente:

1) Fase de Identificação

Objetivo :

- Identificar alguns importantes aspectos do problema como: quem são os participantes, quais as características do problema, quais são os recursos disponíveis e que objetivos pretendemos atingir.

2) Fase de Conceituação

Objetivos :

- Detalhar as relações e os conceitos definidos na fase anterior pelo engenheiro do conhecimento e o especialista no assunto;

- Verificar a conveniência da diagramação dessas idéias de forma a facilitar a prototipação do sistema propiciando um melhor entendimento do problema, que pode ser adquirido através de leitura a respeito do problema apresentado.

3) Fase de Formalização

Objetivos :

- Mapear as relações e os conceitos definidos na fase anterior de forma que essas idéias sejam mais formalmente representadas;

- Planejar o método de representação do conhecimento

apropriado e definir que ferramentas serão utilizadas na fase de implementação;

- Implementar um protótipo, caso necessário, para ajudar na validação dos trabalhos até então realizados.

4) Fase de Implementação

Objetivo :

- Codificar nos termos da ferramenta de implementação escolhida o conhecimento formalizado na etapa anterior (estruturas de dados, regras de inferência e estratégias de controle).

5) Fase de Testes

Objetivos :

- Avaliar o protótipo e a técnica de representação do conhecimento utilizada na implementação do sistema. O especialista deve avaliar o comportamento do sistema e ajudar o engenheiro do conhecimento na sua revisão;

- Testar o protótipo exaustivamente de forma a se poder avaliar seu desempenho e utilidade;

- Identificar problemas como: esquema de representação do conhecimento inadequado, falta de conceitos e relações, nível errado de detalhamento na representação do conhecimento inadequado, falta de conceitos e relações, nível errado de detalhamento na representação do conhecimento ou ineficiência com relação aos tempos de execução devido a pesados

mecanismos de controle;

- Aperfeiçoar o protótipo através de refinamentos sucessivos, de forma que ele evolua tanto em largura quanto em profundidade.

II.3 - Conclusão

Neste capítulo foi feito um estudo na literatura sobre as principais abordagens para modelos de ciclo de vida de software que possibilitará a elaboração de comparações entre os modelos de ciclo de vida e uma posterior classificação dos mesmos.

CAPITULO III

ANÁLISE E COMPARAÇÃO ENTRE OS MODELOS DE CICLO DE VIDA

III.1 - Comparações entre os Modelos de Ciclo de Vida

Nesta seção apresentamos algumas comparações encontradas na literatura entre os diversos modelos de ciclo de vida de software descritos no capítulo anterior, apresentando uma classificação dos modelos e uma posterior seleção.

III.1.1 - Modelo Tradicional x Modelos Alternativos

Segundo Davis [DAVIS88] o modelo tradicional pode ser utilizado do começo ao fim no desenvolvimento de software. Entretanto com o aumento constante do custo associado ao seu desenvolvimento bem como falta de confiança e funcionalidade em seu resultado, houve uma motivação para desenvolver modelos alternativos para o desenvolvimento de software.

Nesta seção serão comparados com o modelo tradicional os seguintes modelos alternativos: prototipação rápida descartável, desenvolvimento incremental, prototipação evolutiva, desenvolvimento baseado em reutilização de software e síntese automática de programas.

Para testar a funcionalidade do sistema de acordo com o tempo, isto é, a relação entre as necessidades dos usuários e as possibilidades do sistema em satisfazê-las, Davis propõe um modelo de comparação e recorre a solução gráfica para quantificar as características deste descompasso (figura 15).

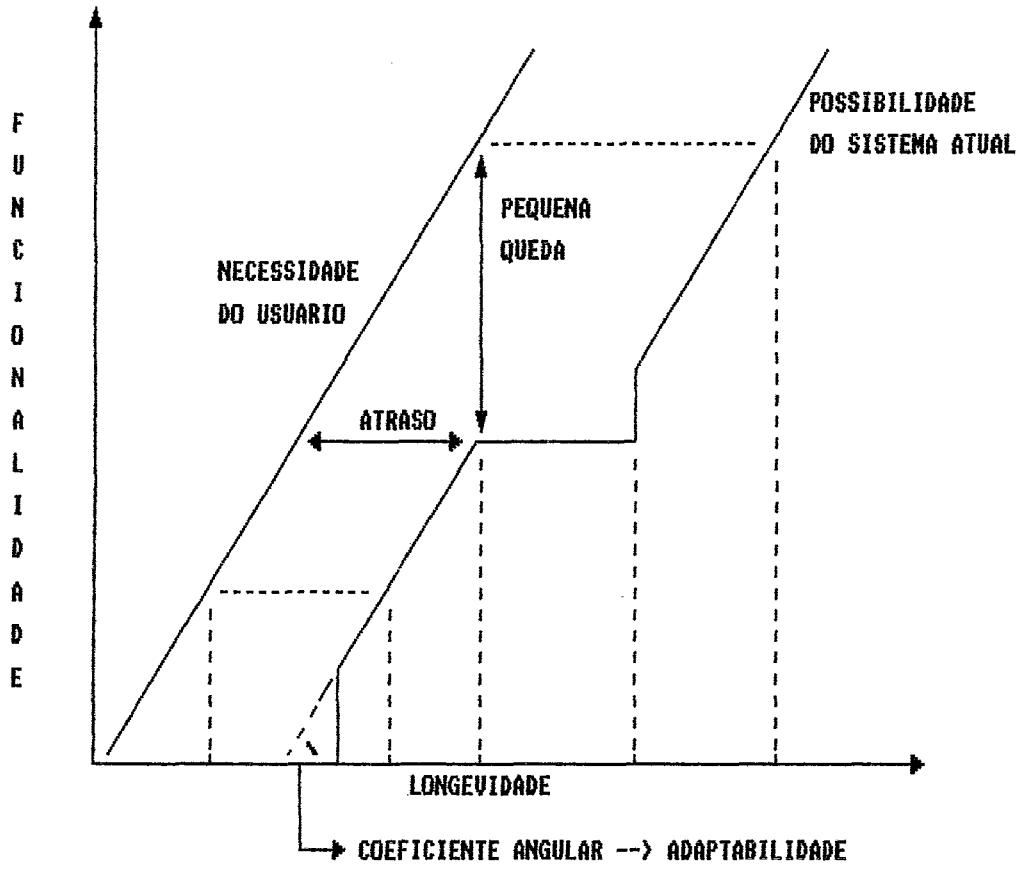


FIGURA 15 - Métrica de Produtividade de Software [DAVIS88]

Neste modelo de comparação consideram-se os seguintes critérios:

1) Lacuna - é a medida da distância existente entre o produto operacional e os requisitos solicitados.

2) Atraso - é a medida do tempo entre o aparecimeto de uma nova necessidade e sua satisfação.

3) Adaptabilidade - é a taxa com que a solução do sistema pode mudar para se adaptar a novos requisitos e é medida pelo coeficiente angular da curva de solução.

4) Longevidade - é o tempo desde a criação do sistema.

5) Impropriedade - é a área compreendida entre a linha das necessidades dos usuários e a curva solução e representa o comportamento da lacuna durante o tempo. Quando o modelo for apropriado a área será zero, significando que novos requisitos podem ser imediatamente satisfeitos.

Considerando-se estes cinco critérios listados tem-se a seguinte situação para os modelos de ciclo de vida:

- Prototipação rápida descartável

Num tempo zero os usuários e os analistas terão um entendimento melhor das suas reais necessidades. Seu uso no entanto não afeta o ciclo de vida como um todo, porém aumenta o impacto no sistema resultante (figura 16).

- Desenvolvimento incremental

O desenvolvimento do sistema no início só atende a poucos requisitos mas ele é construído para captar novos e assim aumentar a adaptabilidade. Este enfoque tem duas características: Primeiramente o desenvolvimento inicial é reduzido por conta do baixo nível de funcionalidade. Segundo, a adaptabilidade ao longo do tempo de vida do sistema é maior do que o enfoque tradicional (figura 17).

- Prototipação evolutiva

Este enfoque é uma extensão do desenvolvimento incremental. A ênfase é a evolução para uma solução de uma maneira contínua. O protótipo inicial aparece rápido e demonstra funcionalidade quando os requisitos estão bem

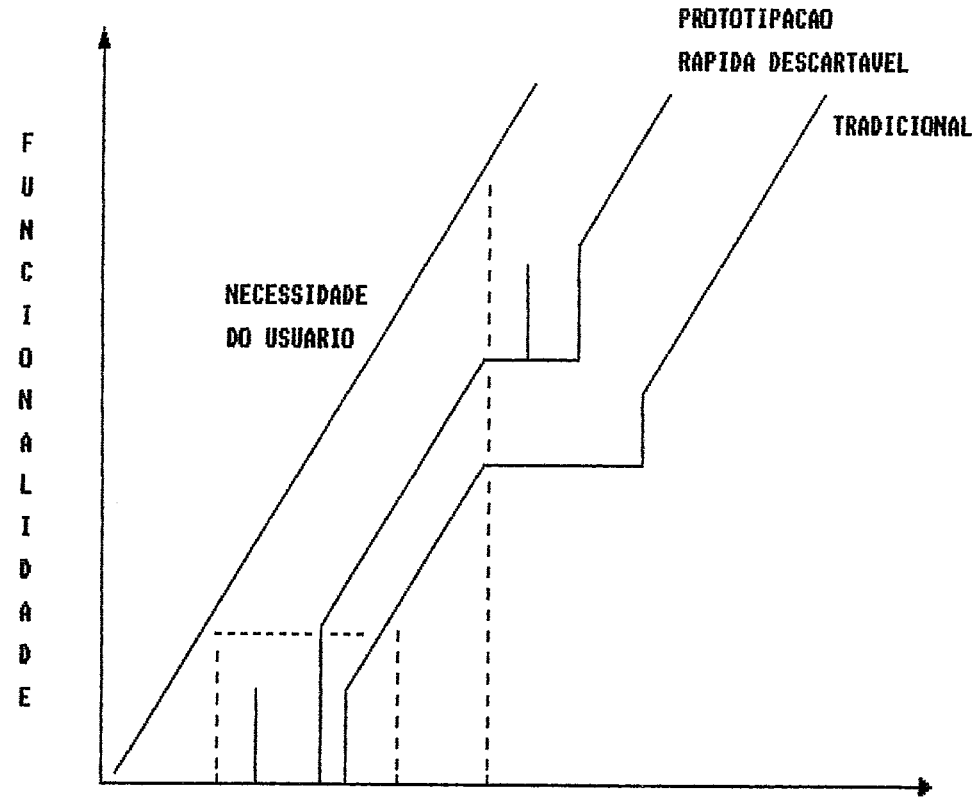


FIGURA 16 - Prototipação Rápida Descartável x Tradicional

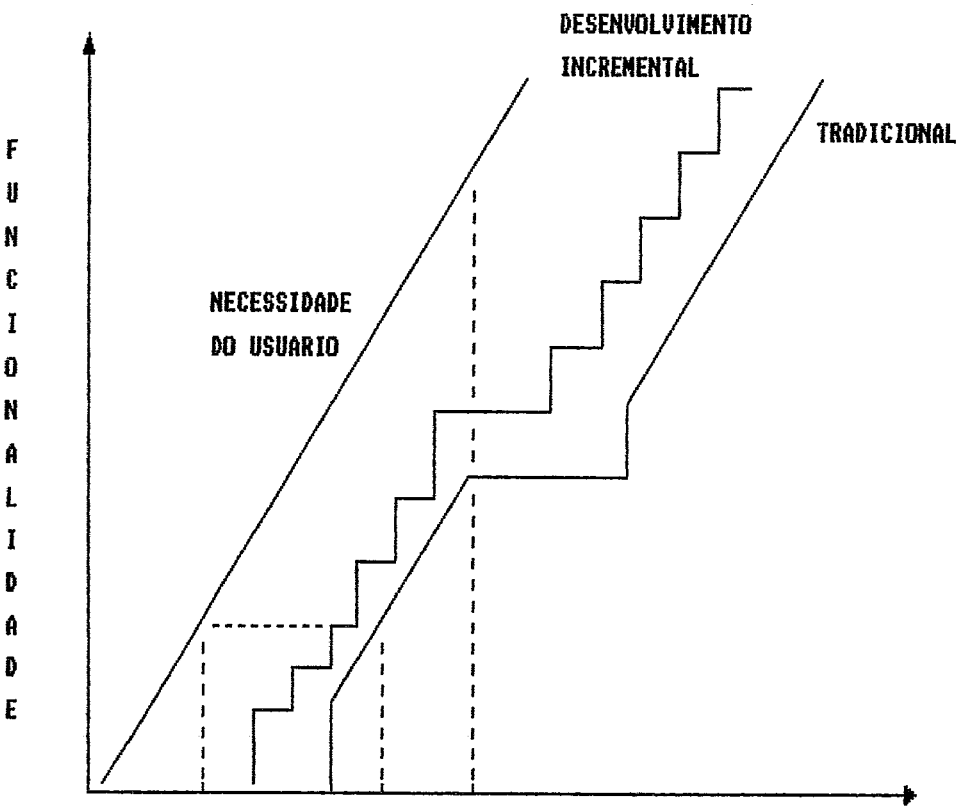


FIGURA 17 - Desenvolvimento Incremental x Tradicional

compreendidos ao contrário da prototipação rápida, pois estes são mal conhecidos.

O coeficiente angular que representa adaptabilidade é maior do que no enfoque tradicional porque o protótipo é evolutivo e foi projetado para ter adaptabilidade (figura 18).

- Desenvolvimento de software baseado em reutilização

O tempo inicial de desenvolvimento diminui sensivelmente pois já se começa utilizando algo previamente testado. O restante dos parâmetros é o mesmo (figura 19).

- Síntese automática de programas

Os requisitos são fornecidos por analistas e estão especificados em alguma linguagem de alto nível, assim o sistema é automaticamente sintetizado. Esse enfoque tem dois efeitos :

- O tempo de desenvolvimento é amplamente reduzido;

- Os custos de desenvolvimento são reduzidos de tal forma que adaptar o sistema antigo é raramente mais complexo que recompor o sistema inteiro. Por isso a longevidade de cada nova versão é baixa (figura 20).

III.1.2 - Modelo Cascata x Desenvolvimento por Evolução x Modelo de Transformação x Modelo Espiral [BOEHM88]

Alguns dos novos modelos de processos de software

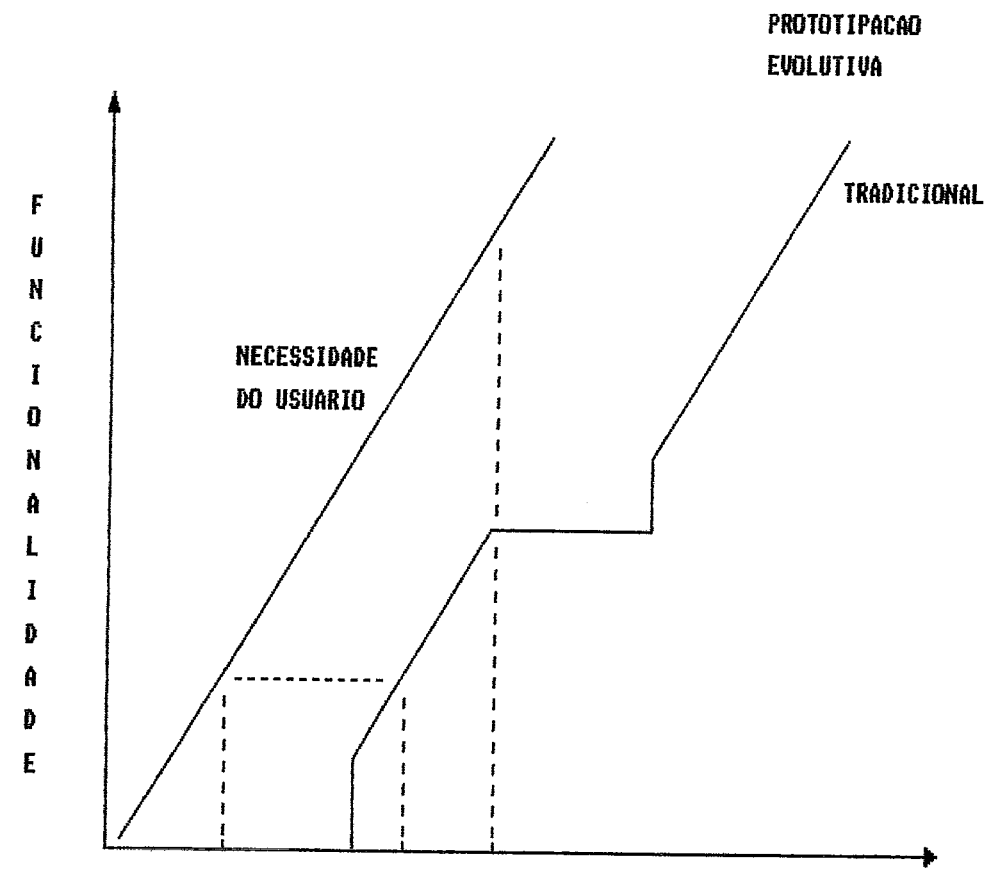


FIGURA 18 - Prototipação Evolutiva x Tradicional

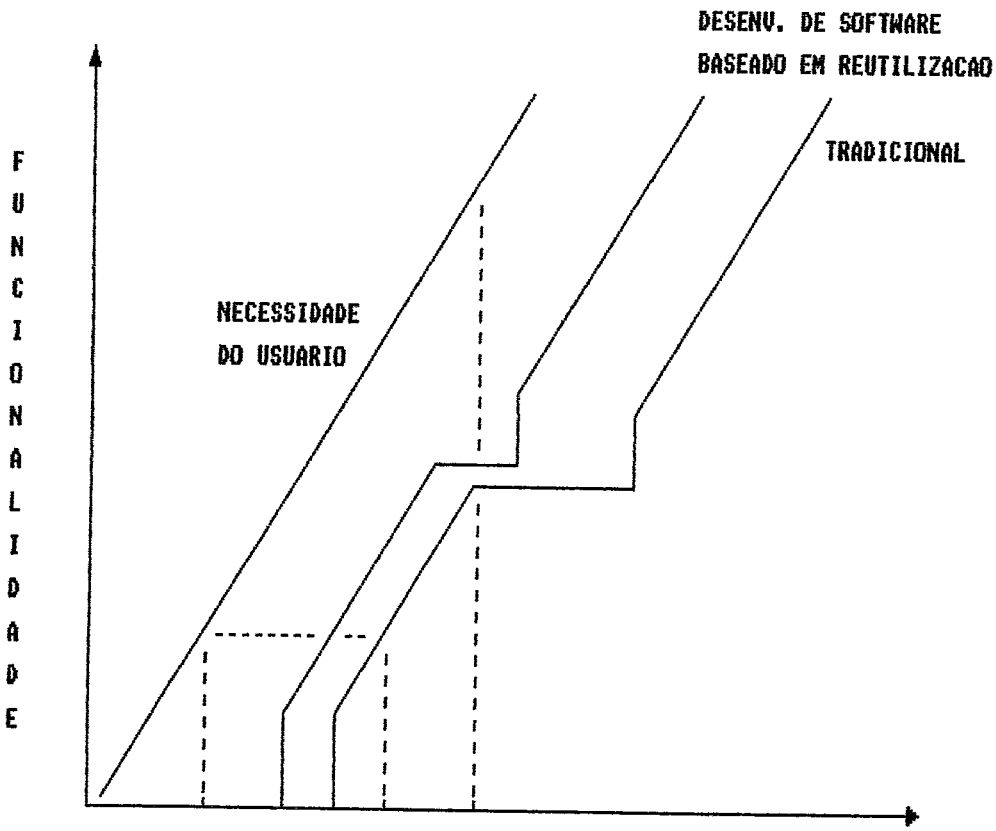


FIGURA 19 - Desenvolvimento de Software baseado em Reutilização x Tradicional

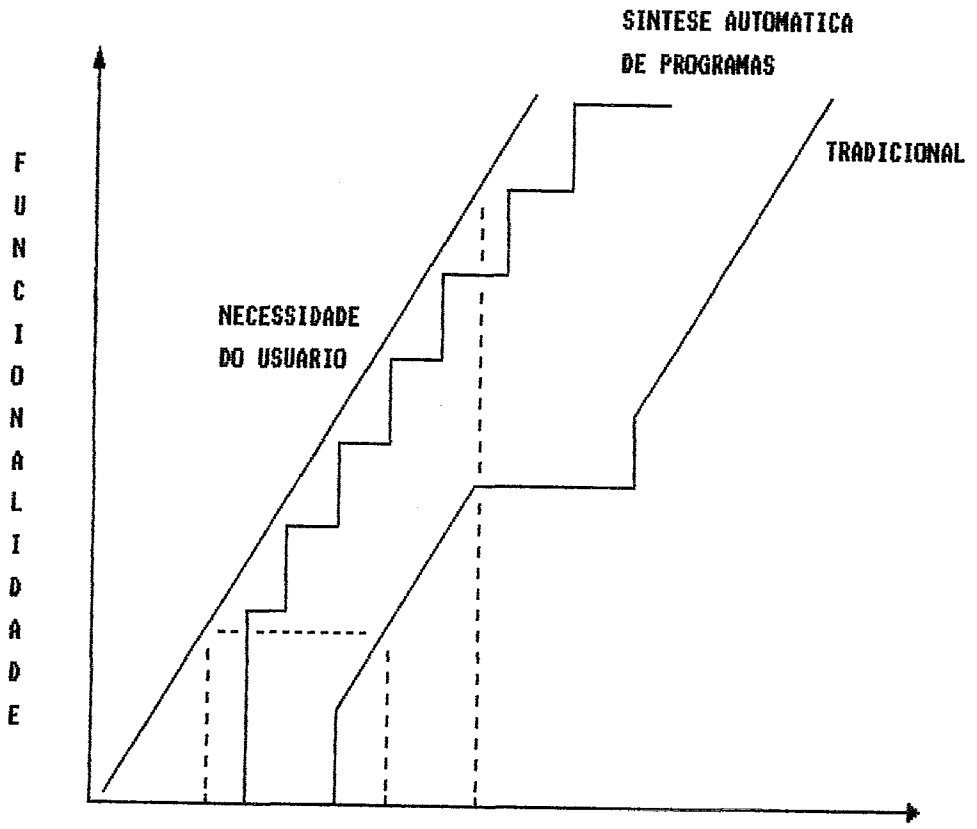


FIGURA 20 - Síntese Automática de Programas x Tradicional

estimulam o desenvolvimento de produtos mais simples. A dificuldade principal do modelo cascata é que seu enfoque de direção da especificação leva para a idéia que as palavras e os documentos são abundantes. Esta condição acarreta a criação de produtos caríssimos pois desenvolvem-se farto material de documentação com alguns ítens desnecessários.

O modelo de desenvolvimento por evolução realça o uso de capacidades de prototipação que convergem para um produto de software de alto nível cujos aspectos são essenciais para os objetivos do usuário.

O modelo de transformação encurta este problema fornecendo diretamente a transformação da especificação para

o código de execução. Assim auxilia a especificação básica e o enfoque do desenvolvimento por evolução.

O modelo espiral focaliza os objetivos determinados pelo usuário e a análise contínua de custo x benefício nos termos de contribuição para o objetivo final.

III.1.3 - Modelo Tradicional x Modelo Orientado a Objetos [YAU86]

O modelo de desenvolvimento de software no qual utiliza-se técnicas orientadas a objeto é caracterizado principalmente pela abstração de dados, abstração de programas e domínio de proteção.

Essas técnicas possuem vantagens sobre o modelo tradicional no que diz respeito a melhor produtividade, manutenibilidade, integridade dos dados e segurança de programa.

Produtividade é direcionada para o mecanismo de abstração que permite a construção de componentes reusáveis.

Manutenibilidade é alcançada por não deixar claro ao usuário tudo o que o sistema é capaz de transformar.

Integridade dos dados e segurança de programa são alcançadas pelo domínio de proteção no qual define os corretos acessos e operações disponíveis a um usuário específico. Esse enfoque é mais adequado para aplicações do domínio de controle ou tempo real que não são bem direcionadas para as técnicas do modelo tradicional.

III.1.4 - Relação entre os Modelos de Ciclo de Vida e o Controle de Qualidade do Produto [WESSELIUS90]

Nesta seção são analisados, no que se refere a controle da qualidade, os seguintes modelos de ciclo de vida de software: modelo cascata, prototipação rápida descartável, prototipação evolutiva, desenvolvimento incremental e modelo espiral.

Para essa análise Wesselius considera os conceitos de verificação e validação assim definidos:

- A verificação julga o produto, isto é, se as características requeridas estão completas e corretas;

- A validação julga o produto inserido no contexto ou no problema a ser resolvido e só é importante nos primeiros estágios do processo de desenvolvimento pois a plena ralização da completeza e correção não é possível em todos os estágios do processo.

No que se refere ao modelo cascata, a especificação completa dos requisitos antes de cada atividade de desenvolvimento é necessária pois como esse modelo trabalha por encadeamento sucessivo de vários estágios o reencadeamento de trás para frente a fim de detectar erros é muito problemático.

Só quando a validação acontece, os problemas dos usuários podem ser vistos. A determinação da qualidade acontece somente no fim do projeto e sendo assim, neste modelo, não

ocorre um real controle de qualidade.

A prototipação rápida descartável difere do modelo cascata somente no primeiro estágio onde se analisam os requisitos, constrói-se o protótipo e este é experimentado. Caso o protótipo seja satisfatório desenvolve-se o produto, caso contrário, retorna-se a análise de requisitos.

Por essa razão não se pode considerar um modelo de processo diferente. Isto quer dizer que encurta o laço do retorno vital para um dos estágios do processo, mas não permite o controle da qualidade durante todo o processo de desenvolvimento.

O modelo de prototipação evolutiva é calcado na idéia que qualquer degrau no processo de desenvolvimento só pode ser desenvolvido a partir de um sistema já existente que se modifica lentamente para acompanhar as necessidades do usuário.

Este modelo provê um aperto no controle da qualidade no tocante a subjetividade pois tem-se um frequente retorno por parte do usuário. A principal diferença entre a prototipação rápida descartável e a prototipação evolutiva é o fato de que esta última permite o retorno em todos os estágios do processo de desenvolvimento e a anterior só oferece retorno no primeiro estágio deste processo.

O desenvolvimento incremental oferece condições de validação fornecendo retorno apoiados na experiência prática com uma parte do sistema, no entanto a sua dimensão é menor

que a da prototipação evolutiva.

No modelo em espiral, a completeza do ciclo só deve ser perseguida no nível onde os riscos ocasionados por incompleteza não sejam mais ponderáveis. Isto não se refere só para a documentação mas igualmente para o controle da qualidade, verificação e validação do planejamento, análise e configuração.

Como todo ciclo envolve determinação dos riscos da execução do projeto, objetivos de qualidade podem ser integradas no processo de desenvolvimento.

Uma vez que objetivos de qualidade forem mencionadas, o risco de não encontrar os objetivos é determinado no final de cada ciclo. Se o risco de não encontrar os objetivos de qualidade mencionadas tornar-se um perigo para o sucesso do projeto esses objetivos serão indicados para que não se perca o controle da qualidade.

III.2 - Classificação dos Modelos de Ciclo de Vida

Nesta seção apresentamos uma classificação dos modelos de ciclo de vida descritos no capítulo II e sobre os quais apresentamos algumas comparações presentes na literatura.

A classificação que apresentamos tem como objetivo fornecer ao Projeto TABA, subsídios para a base de conhecimentos do sistema especialista de apoio a decisão [AGUIAR89] cujo principal objetivo é indicar o Ambiente de Desenvolvimento de Software mais adequado a uma determinada

aplicação.

Para determinação do ambiente de desenvolvimento mais adequado, serão levados em consideração uma série de aspectos relativos às características do domínio de aplicação [WERNECK90].

Determinar este ambiente, implica também, conforme a definição de ADS válida no contexto do Projeto TABA [ROCHA87a], identificar o modelo de ciclo de vida mais adequado ao desenvolvimento.

Os modelos de ciclo de vida estudados podem ser agrupados em oito tipos:

1) Modelo Tradicional ou Cascata [BOEHM81], [CAPRON86], [ENGER80], [HOUGHTON87] e [FAIRLEY85], que tem como características:

- Ênfase na divisão em fases;
- Ênfase na documentação;
- Desenvolvimento "bottom-up";
- Desenvolvimento sequencial.

2) Modelos Semi Estruturado e Estruturado [YOURDON89], [ENGER80], [ROCHA87], que tem como características:

- Uso de métodos e técnicas estruturadas;
- Desenvolvimento "top-down";
- Divisão em fases com atividades paralelas.

Os modelos estruturados diferenciam-se dos semi estruturados pela introdução da preocupaçao com garantia da qualidade.

3) Modelos Baseados em Protótipos [CONNOR80], [FAIRLEY85], [GLADDEN82], [BOAR84], [DAVIS88], [HIRSCH85], [McCRACKEN82], [BOEHM86], [GOMMA81], que tem como características:

- Fornecer rapidamente uma versão operacional;
- A não necessidade de satisfazer todos os requisitos do produto final desde o início;

- Facilitar o desenvolvimento de produtos onde não se conhece totalmente o problema.

4) Modelo Baseado em Reutilização de Software [DAVIS88], cujo objetivo é realizar o desenvolvimento de um novo produto utilizando, ao máximo, componentes de produtos anteriores.

5) Modelo de Síntese Automática de Programas [DAVIS88], [BALZER83], [BOEHM86], [BOEHM88], cujo objetivo é gerar o código a partir das especificações.

6) Modelo para Desenvolvimento Orientado a Objetos [SEIDEWITZ87], [MATTOS090], cujo objetivo é apoiar ao desenvolvimento de produtos que utilizam métodos de desenvolvimento orientado a objetos.

7) Modelo Espiral [BOEHM86], que tem como características:

- Adaptar e combinar vários dos modelos anteriores

(cascata, prototipação, transformação);

- Grande importância da consideração sobre riscos nas decisões do processo de desenvolvimento;

- Modelar o processo de desenvolvimento de software como uma sequência de ciclos progressivos;

- Terminar cada ciclo com uma revisão.

8) Modelo para Desenvolvimento de Sistemas Especialistas [RIBEIRO89], [WATERMAN86], cujo objetivo é apoiar o desenvolvimento de um sistema capaz de resolver problemas de forma semelhante aos especialistas da área em questão.

Se considerarmos os aspectos relacionados a critérios de seleção destes modelos para o desenvolvimento de um produto, podemos agrupá-los em dois tipos:

1) Modelos orientados para um paradigma de desenvolvimento de software, que inclui:

- Modelo Semi Estruturado;
- Modelo Estruturado;
- Modelo para Desenvolvimento Orientado a Objetos;
- Modelo Baseado em Reutilização de Software;
- Modelo de Síntese Automática de Programas.

2) Modelos orientados para as características do produto a ser desenvolvido, que inclui:

- Modelo Tradicional;

- Modelo de Prototipação Rápida Descartável
- Modelo de Desenvolvimento Incremental / Versões Sucessivas;
- Modelo de Prototipação Evolutiva;
- Modelo para Desenvolvimento de Sistema Especialista;
- Modelo Espiral.

III.3 - Conclusão

Neste capítulo foi apresentada algumas comparações entre os modelos de ciclo de vida de software descritos no capítulo II e uma classificação dos modelos considerando-se suas características fundamentais. Para realizar esta classificação teve-se presente os objetivos do Projeto TABA.

CAPÍTULO IV

SUBSÍDIOS PARA A BASE DE CONHECIMENTO DA ESTAÇÃO TABA

Neste capítulo foram selecionados dentre os modelos apresentados no capítulo II alguns modelos de ciclo de vida de software, considerando-se sua consequente adequação às particularidades do desenvolvimento de diferentes produtos de software.

IV.1 - Modelos Selecionados: Justificativa

Os modelos selecionados, para inclusão neste capítulo, foram aqueles de interesse para inclusão na base de conhecimentos da Estação TABA, que são:

- Modelo Estruturado;
- Modelo de Prototipação Rápida Descartável;
- Modelo de Prototipação Evolutiva;
- Modelo de Desenvolvimento Incremental;
- Modelo para Desenvolvimento Orientado a Objetos;
- Modelo para Desenvolvimento de Sistemas Especialistas.

A escolha destes modelos e exclusão dos demais deve-se às seguintes razões:

- O modelo tradicional foi excluído por seu enfoque excessivamente conservador dando demasiada ênfase a sequencialidade das fases e não permitindo atividades em paralelo.

- Foi escolhido o modelo estruturado (excluindo o semi

estruturado) por este incluir atividades relacionadas à garantia da qualidade e permitir que atividades sejam realizadas em paralelo.

- Os modelos voltados para o enfoque de prototipação foram incluídos por sua importância no atendimento de características específicas dos produtos em desenvolvimento.

- O modelo de desenvolvimento orientado a objetos foi incluido por sua importância atual como paradigma de desenvolvimento de software e pelo fato do TABA-OBJ ser um dos ambientes específicos disponíveis na Estação TABA.

O modelo para desenvolvimento de sistema especialista foi incluído por ser um dos modelos, inicialmente, considerados pela Estação TABA e apresentar características próprias bastante diferente dos demais modelos.

Os modelos baseados em reutilização, síntese de programas e o modelo espiral foram excluídos porque, neste momento, não serão considerados na Estação TABA.

Cada um dos modelos selecionados foi organizado através de um diagrama de atividades [PRESSMAN87], conforme será exibido ao usuário da Estação TABA, pelo sistema especialista de apoio à decisão. Associado a este diagrama o sistema pode informar, quando solicitado, os produtos a serem gerados em cada atividade bem como os recursos (métodos, instrumentos e ferramentas possíveis de serem utilizadas para realização da atividade.

Para seleção do modelo mais adequado ao desenvolvimento

de um determinado produto, a base de conhecimentos deverá conter regras que orientem a escolha. Um primeiro conjunto de regras foi, então, incluído para cada modelo escolhido.

IV.2 - Modelo Estruturado

Foram apresentadas no capítulo II várias propostas de modelo de ciclo de vida estruturado. Estas propostas são bastante semelhantes diferindo apenas no número de fases propostas. Dentre os modelos apresentados, selecionamos para a Estação TABA, o modelo proposto por Rocha [ROCHA87].

Este modelo consta de cinco fases: Definição, Projeto, Implementação, Avaliação e Operação.

Durante a fase de Definição é identificado o problema, são estabelecidos seus requisitos e feito um planejamento inicial. O final da fase é determinado por uma avaliação da Especificação e da Viabilidade do desenvolvimento.

Na fase de Projeto é determinado uma solução viável para o produto. O final desta fase é determinado pela finalização do Projeto da Arquitetura, do Projeto Detalhado e do Projeto de Arquivos.

Durante a fase de Implementação é produzido o código dos programas. No final da fase tem-se os programas, a documentação dos testes realizados e os manuais do usuário e de operação do produto.

Na fase de Avaliação realiza-se a verificação e validação

final do software produzido baseado nas especificações e normas e padrões estabelecidos. O final da fase é determinado pelos testes de aceitação documentados e pelo documento de aceitação formal do produto.

Durante a fase de Operação tem-se a utilização do produto produzido. Caso necessário poderão haver correções, aprimoramentos, adaptações e expansões no software.

O modelo estruturado é adequado para o desenvolvimento de produtos se:

a) O problema é bem conhecido;

O principal fator a ser considerado na seleção de um modelo de ciclo de vida é o grau de compreensão do problema. Conhecer bem o problema refere-se a completude da compreensão do problema apresentado [BLUM82].

b) O usuário consegue especificar com relativa clareza os requisitos do produto;

c) Deseja-se implementar todas as funções do produto desde o início;

d) Deseja-se ter uma documentação abundante;

É dada grande ênfase a uma completa elaboração de documentos como critério de conclusão das fases de especificação de requisitos e projeto [BOEHM81].

e) Não há pressa em oferecer uma versão operacional;

f) A abordagem de desenvolvimento a ser utilizada é

funcional ou de dados com uso de métodos e técnicas estruturadas [YOURDON89];

g) O tipo de processamento é "batch" ou "tempo real";

h) Deseja-se ter desenvolvimento "top-down" [YORDON89].

A codificação e a integração dos módulos devem ser implementadas de maneira "top-down".

IV.3 - Modelo de Prototipação Rápida Descartável

Prototipação rápida descartável tem como objetivo construir uma versão operacional de uma determinada função do produto para que o usuário possa utilizá-la e desta forma, fazer uma avaliação de sua adequação.

Constrói-se, então, um protótipo que após a avaliação é descartado [GOMMA81], para se proceder a implementação definitiva. Esta implementação deve ser feita segundo algum outro modelo de ciclo de vida.

O desenvolvimento através de prototipação rápida descartável é adequado se:

a) O problema não é bem conhecido;

É realizado o desenvolvimeto exploratório para produtos onde existe desconhecimento de como solucionar o problema [FAIRLEY85].

b) O usuário não consegue especificar, com clareza, os requisitos do produto;

c) Deseja-se oferecer rapidamente (antes ou durante o processo de desenvolvimento) uma implementação parcial rápida para que o usuário avalie [SOMMERVILLE82];

A implementação parcial permite exemplificar ao usuário os dados de entrada, as mensagens e relatórios existentes no produto [FAIRLEY85], fornecendo dados para o desenvolvimento definitivo com relação aos pontos críticos encontrados.

d) Deseja-se implementar primeiro os aspectos menos conhecidos;

Essa implementação é realizada através da construção de uma série de protótipos rápidos, que são descartados até se chegar ao final no momento em que o protótipo satisfaz ao usuário.

e) Se o produto tem muita interação com o usuário e se quer avaliar a interface.

IV.4 - Modelo de Prototipação Evolutiva

Na abordagem de prototipação evolutiva [BOAR84] é construída uma implementação parcial do produto composto dos requisitos já conhecidos. Através do uso do protótipo aumenta-se o conhecimento sobre os requisitos e pode-se partir para uma evolução do produto.

O desenvolvimento através de prototipação evolutiva é adequado se:

a) Não se conhecem todos os requisitos do produto e tem-

se necessidade de experimentar o que se conhece num sistema em operação para aprender sobre as necessidades do produto;

Deve-se absorver inicialmente um conjunto reduzido de necessidades para implementá-las rapidamente com intenção de expandí-las iterativamente [YOURDON89].

b) Se deseja começar o desenvolvimento pelos aspectos já conhecidos;

Esses aspectos conhecidos farão parte da primeira implementação parcial do produto que será evoluído para uma solução de maneira contínua.

c) O usuário não sabe especificar claramente seus requisitos, mas sabe identificá-los ao experimentar o protótipo;

A incapacidade de examinar modelos abstratos do produto e a dificuldade de expressar os requisitos do produto são eliminados quando o protótipo passa a ser utilizado por seus usuários para que com o uso aumente a compreensão de seus requisitos.

d) Se deseja mostrar a funcionalidade do produto;

O protótipo inicial aparece rápido e demonstra funcionalidade quando os requisitos estão bem compreendidos. A ansiedade do usuário e posterior tendência a colocar mais requisitos é atenuada pela versão inicial demonstrada.

e) Se deseja um alto grau de adaptabilidade no produto;

A medida em que o usuário está convencido do que o software fará quando este for acionado, o grau de adaptabilidade torna-se crescente.

f) Se deseja o retorno a fases anteriores em todos os estágios do processo de desenvolvimento;

Esse retorno deve ser bem controlado para não causar insucessos no desenvolvimento.

g) O produto não é especificável.

IV.5 - Modelo de Desenvolvimento Incremental

A abordagem de prototipação por desenvolvimento incremental [FAIRLEY85], caracteriza-se pela construção do produto através de N versões, onde em cada uma são incluídas novas funções de interesse do usuário.

a) Os requisitos são conhecidos;

Apesar de serem conhecidos a maioria dos requisitos do produto, escolhem-se prioridades de implementação.

b) Não se quer implementar todo o sistema de uma vez;

Através da escolha de prioridades de implementação, aumenta-se pouco a pouco sua capacidade.

c) Se quer ter rapidamente uma versão operacional mesmo que não tenha todas as funções.

As atividades de construção do protótipo são revestidas

de pouco formalismo e são de rápida execução.

IV.6 - Modelo para Desenvolvimento Orientado a Objetos

Esse modelo é utilizado durante o processo de desenvolvimento de software com orientação a objetos. Foi proposto para o ambiente TABA.OBJ [MATTOS090], mas é aplicável sempre que for utilizado o paradigma da orientação a objeto.

O desenvolvimento através de orientação a objetos é adequado se:

a) Se há decisão quanto ao método de desenvolvimento orientado a objetos e a linguagem orientada a objetos;

b) Se é recomendável o uso de método orientado a objetos para se desejar ter estabilidade no modelo.

IV.7 - Modelo para Desenvolvimento de Sistemas Especialistas

Este modelo de ciclo de vida, proposto por Waterman [WATERMAN86] [RIBEIRO89] é adequado para o desenvolvimento de sistemas especialistas.

O desenvolvimento através desse modelo é adequado se:

a) Se trata do desenvolvimento de um sistema especialista.

A construção de um sistema especialista, é adequada se este puder ser capaz de resolver problemas de forma

semelhante aos especialistas da área em questão.

b) A área do problema possuir especialistas com poderosa fonte de conhecimento.

Esses especialistas devem ser capazes de concordar na solução dos problemas e capazes de explicar os métodos que utilizam na resolução de problemas, pois o conhecimento adquirido serão inseridos no sistema.

c) Abordar assuntos compreensíveis;

Assuntos extremamente difíceis prejudicam o processo de extração do conhecimento, e assuntos muitos novos ou assuntos ainda não tão entendidos, não devem ser tratados pela engenharia do conhecimento.

d) O problema for naturalmente resolvido via manipulação de símbolos;

e) O problema for manipulável.

Não deve ser amplo demais, nem restrito demais, para garantir o interesse prático do assunto.

IV.8 - Conclusão

As regras indicativas da adequação dos modelos descritas neste trabalho serão ponto de partida para a construção da base de conhecimentos da Estação TABA.

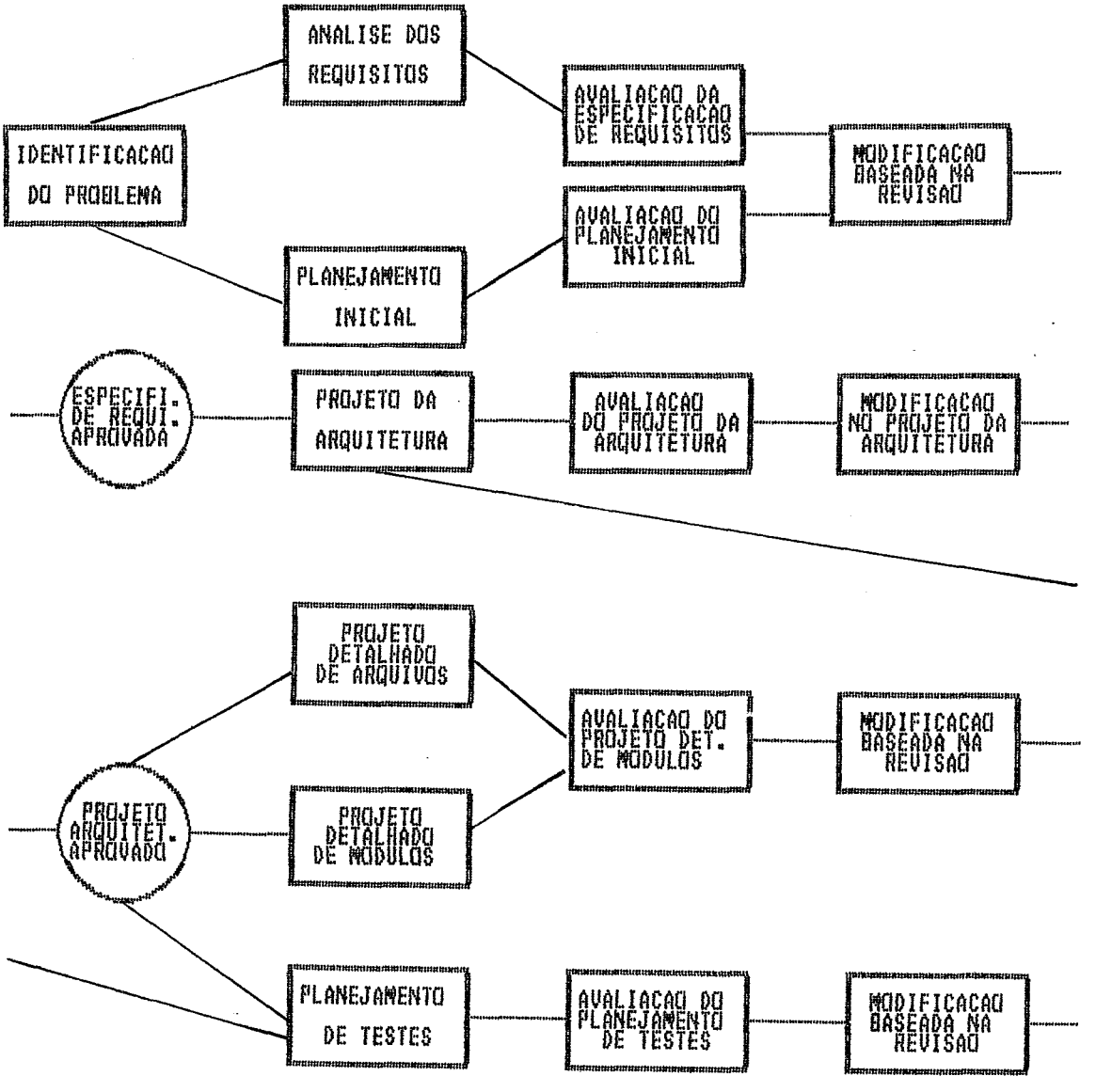
CAPÍTULO V

ORGANIZAÇÃO DOS MODELOS: ATIVIDADES E PRODUTOS GERADOS

Neste capítulo são apresentados os modelos de ciclo de vida de software selecionados organizados através de um diagrama de atividades, bem como os produtos a serem gerados em cada atividade e os recursos utilizados.

V.1 - Modelo Estruturado

V.1.1 - Diagrama de Atividades



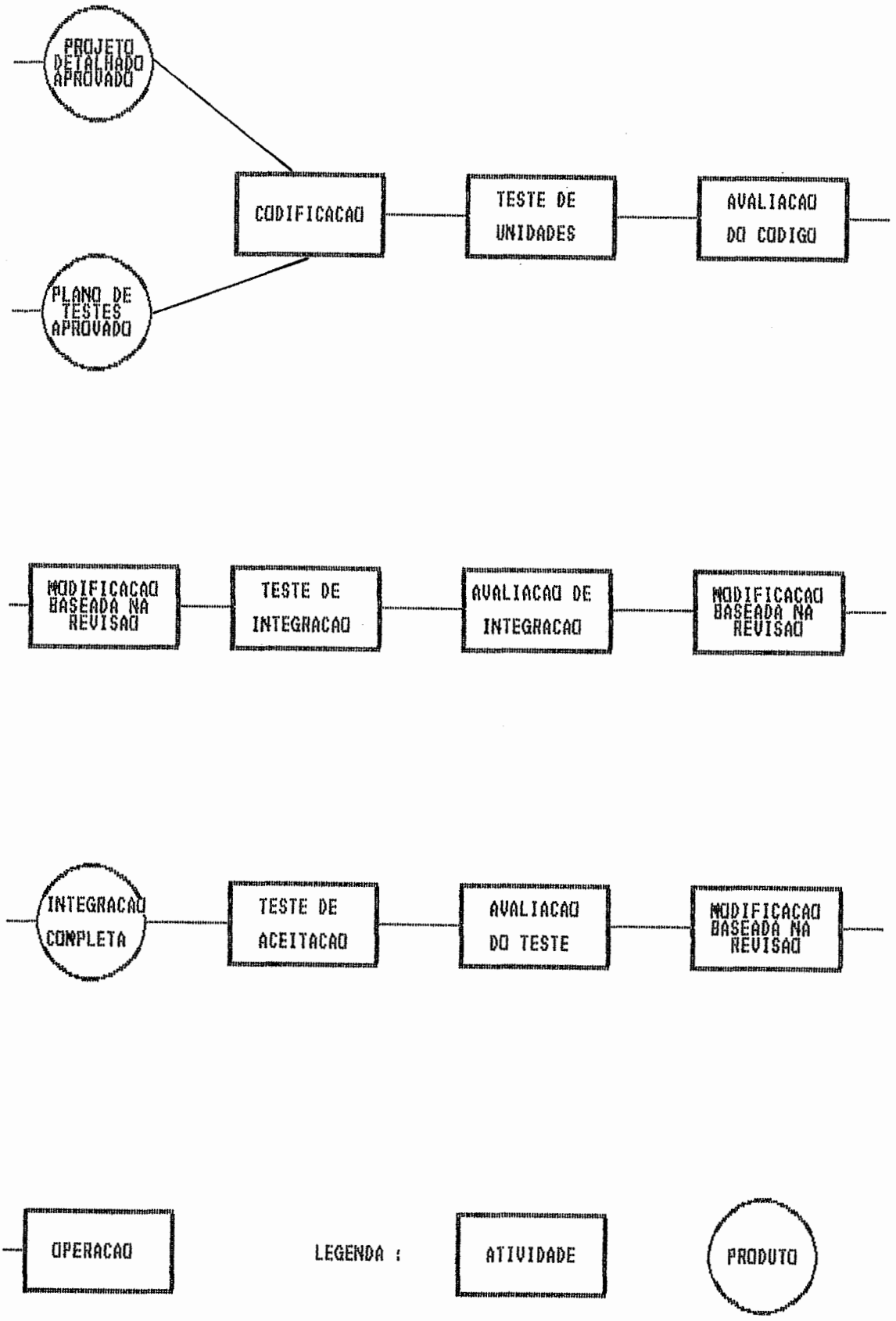


FIGURA 21 - Diagrama de Atividades do Modelo Estruturado

V.1.2 - Recursos Adequados e Produtos Gerados por Atividade

Identificação do Problema

Descrição: Obter e avaliar os dados sobre o sistema a ser construído através de informações que permitam decidir sobre a conveniência de realizar o desenvolvimento. Estas informações estabelecerão um contrato de desenvolvimento, a Proposta de Desenvolvimento.

Após a elaboração da Proposta de Desenvolvimento, deve-se conduzir uma avaliação da mesma, com o objetivo de conseguir um consenso sobre a adequação da Proposta de Desenvolvimento e sobre o interesse e viabilidade de se realizar o desenvolvimento do sistema proposto.

A avaliação pode ser feita através de uma inspeção que é uma técnica disciplinada usada para avaliar os produtos gerados.

Recursos Adequados: Editor de Documentos;
Instrumento para modelagem;
Método de Avaliação (ex : inspeção).

Produtos gerados:

Proposta de Desenvolvimento que conterá:

- Uma descrição em linhas gerais do sistema atual e suas deficiências;

- Uma descrição do sistema a ser desenvolvido, definindo seus objetivos, requisitos, as necessidades e expectativas do

usuário e porque é necessário e oportuno desenvolver o sistema proposto;

- Dados que permitam avaliar a viabilidade do desenvolvimento e da operação do sistema proposto.

Laudo de Avaliação.

Análise de Requisitos

Descrição: Analisar o software através de uma visão macroscópica (especificação de requisitos). As especificações devem ser base para o desenvolvimento, permitir o controle da qualidade do produto, estabelecer a comunicação entre o pessoal envolvido no projeto e auxiliar no entendimento do problema a ser resolvido.

As especificações são a primeira forma em que o software existe e, portanto, a base de todo o desenvolvimento futuro. Especificações são produzidas utilizando-se metodologias e linguagens de especificação.

Recursos Adequados: Editor de Documentos;
Métodos/Linguagem de Especificação (ex: Análise Estruturada de Sistemas, Darts, Entidade e Relacionamento).

Produtos gerados: Especificação de Requisitos.

Planejamento Inicial

Descrição: Desenvolver um plano, que especifique o que deve ser feito, quem deve realizar cada tarefa e qual é o custo

previsto. Ao planejar, deve-se ter em conta a ocorrência de alterações. O planejamento é benéfico porque reduz a probabilidade de imprevistos e consequentemente dos problemas daí decorrentes.

Recursos Adequados: Editor de Documentos;
Ferramenta para estimar custo de desenvolvimento de software.

Produtos gerados: Plano Inicial do Projeto.

Avaliação da Especificação de Requisitos

Descrição: Conseguir um consenso sobre a adequação da especificação de requisitos, isto é, verificar se é realmente o que o usuário deseja e se o método utilizado está correto.

Recursos Adequados: Editor de Documentos;
Método de Avaliação (inspeção, walkthroughs).

Produtos gerados: Documento com o laudo de avaliação baseado na especificação de requisitos.

Avaliação do Planejamento Inicial

Descrição: Conseguir um consenso sobre a adequação do plano inicial do projeto.

Recursos Adequados: Editor de Documentos;
Método de Avaliação.

Produtos gerados: Documento com o laudo de avaliação baseado no Plano Inicial do Projeto.

Modificação Baseada na Revisão

Descrição: Com o laudo de avaliação da especificação de requisitos e do plano inicial do projeto serão elaboradas as modificações necessárias.

Recursos Adequados: Os utilizados nas atividades de análise de requisitos e planejamento inicial.

Produtos gerados: Especificação de Requisitos e Plano Inicial do Projeto modificados.

Projeto da Arquitetura

Descrição: É a especificação de uma organização modular lógica, capaz de tornar real o programa em algum ambiente de programação. Nesta etapa é relevante apenas a função que o módulo venha a possuir.

Recursos Adequados: Editor de Gráficos de Estrutura.

Produtos gerados: Projeto da Arquitetura.

Avaliação do Projeto da Arquitetura

Descrição: Conseguir um consenso sobre a adequação do projeto da arquitetura, isto é, verificar se é realmente o que o usuário deseja e se o método utilizado está correto.

Recursos Adequados: Método de avaliação.

Produtos gerados: Laudo da avaliação baseado no Projeto da Arquitetura.

Modificação no Projeto da Arquitetura

Descrição: Com o laudo de avaliação do projeto da arquitetura serão elaboradas as modificações necessárias.

Recursos Adequados: Editor de Gráficos de Estrutura.

Produtos gerados: Projeto da Arquitetura modificado.

Projeto Detalhado de Arquivos

Descrição: Contém a definição do conteúdo dos arquivos bem como as características físicas de armazenamento de dados e do acesso aos mesmos.

Recursos Adequados: Linguagem de especificação de projeto de arquivos.

Produtos gerados: Projeto Detalhado de Arquivos

Projeto Detalhado de Módulos

Descrição: Elaborar o algoritmo detalhado de cada módulo a ser construído.

Recursos Adequados: Pseudo-código (baseado na linguagem utilizada).

Produtos gerados: Projeto Detalhado de Módulos.

Planejamento de Testes

Descrição: Planejar quais métodos de teste poderiam ser utilizados para a avaliação. Recursos Adequados: Editor de Documentos.

Produtos gerados: Plano de Teste.

Avaliação do Projeto Detalhado de Módulos

Descrição: Analisar o pseudo-código gerado.

Recursos Adequados: Reuniões;
Métodos de avaliação.

Produtos gerados: Laudo de avaliação baseado no Projeto Detalhado de Módulos.

Modificação Baseada na Revisão

Descrição: Com o laudo de avaliação do projeto detalhado de módulos serão elaboradas as modificações necessárias.

Recursos Adequados: Pseudo-código.

Produtos gerados: Projeto Detalhado de Módulos modificado.

Avaliação do Planejamento de Testes

Descrição: Analisar o plano de teste gerado.

Recursos Adequados: Reuniões;
Editor de Documentos.

Produtos gerados: Laudo de avaliação baseado no Plano de Testes.

Modificação Baseada na Revisão

Descrição: Com o laudo de avaliação do plano de testes serão elaboradas as modificações necessárias.

Recursos Adequados: Editor de Documentos.

Produtos gerados: Plano de Teste modificado.

Codificação

Descrição: Construir o código do sistema.

Recursos Adequados: Linguagem de Programação;
Software Básico (compilador, sistema operacional, depuradores, etc).

Produtos gerados: Programas Codificados.

Teste de Unidades

Descrição: Testar cada módulo projetado e codificado separadamente.

Recursos Adequados: Método de teste de unidade.

Produtos gerados: Documentação do Teste de Unidade.

Avaliação do Código

Descrição: Analisar o teste de unidade.

Recursos Adequados: Reuniões;
Testes.

Produtos gerados: Laudo de avaliação baseado no documento do teste de unidade.

Modificação Baseada na Revisão

Descrição: Com o laudo de avaliação do teste de unidade serão

elaboradas as modificações necessárias.

Recursos Adequados: Linguagem de Programação;
Software Básico (compilador, sistema operacional, depuradores, etc).

Produtos gerados: Programa Codificado modificado.

Teste de Integração

Descrição: Integrar os módulos incrementalmente, para que a cada módulo projetado, codificado e testado separadamente seja adicionado um novo módulo da cada vez ao conjunto que é então testado.

Recursos Adequados: Método de teste.

Produtos gerados: Documentação do Teste de Integração.

Avaliação da Integração

Descrição: Analisar o teste de integração.

Recursos Adequados: Reuniões;
Testes.

Produtos gerados: Laudo de avaliação baseado no Documento do Teste de Integração.

Modificação Baseada na Revisão

Descrição: Com o laudo de avaliação do teste de integração serão elaboradas as modificações necessárias.

Recursos Adequados: Linguagem de Programação;

Software Básico (compilador, sistema operacional, depuradores, etc).

Produtos gerados: Programas Integrados modificados.

Teste de Aceitação

Descrição: Realização de testes no ambiente onde o software foi desenvolvido com a finalidade de verificar se o produto satisfaz as expectativas do usuário.

Recursos Adequados: Editor de Documento.

Produtos gerados: Documento de aceitação formal do software.

Avaliação do Teste

Descrição: Analisar o software desenvolvido.

Recursos Adequados: Editor de Documentos;
Reuniões.

Produtos gerados: Laudo de avaliação baseado no teste de aceitação.

Modificação Baseada na Revisão

Descrição: Decidir soluções a serem adotadas dependendo do que foi observado e avaliado.

Recursos Adequados: Reuniões.

Produtos gerados: Proposta de soluções.

Operação

Descrição: Colocar em uso o software produzido e quando necessário realizar correções, aprimoramentos, adaptações e expansões.

Recursos Adequados: Software desenvolvido;
Software Básico.

Produtos gerados: Software em operação.

V.2 - Modelo de Prototipação Rápida Descartável

V.2.1 - Diagrama de Atividades

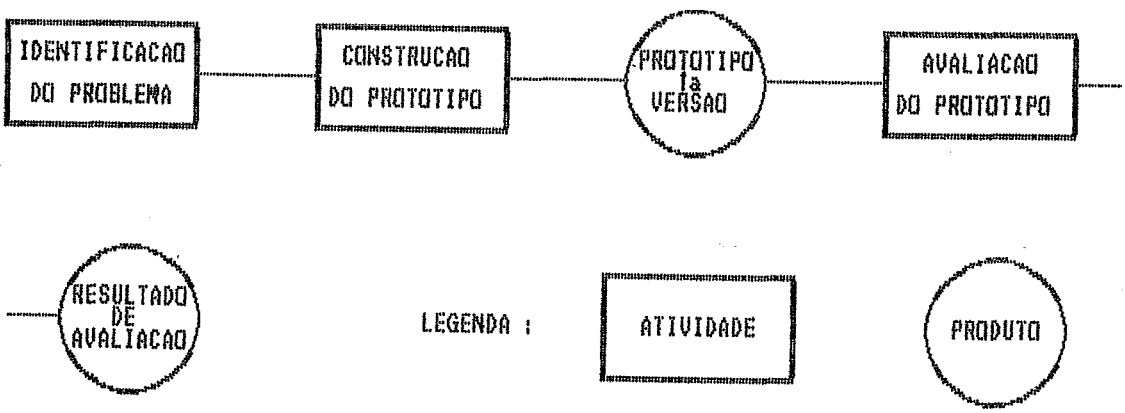


FIGURA 22 - Diagrama de Atividade do Modelo de Prototipação Rápida Descartável

V.2.2 - Recursos Adequados e Produtos Gerados por Atividade

Identificação do Problema

Descrição: Obter os dados do sistema a ser construído através

de informações fornecidas pelo usuário.

Recursos Adequados: Editor de Documentos;
Reuniões.

Produtos Gerados: Proposta de Construção do Protótipo Descartável.

Construção do Protótipo

Descrição: Construir uma versão operacional baseada na satisfação dos requisitos iniciais determinados pelo usuário.

Recursos Adequados: Linguagem de 4a geração.

Produtos Gerados: Protótipo.

Atividade: Avaliação do Protótipo

Descrição: Conseguir um consenso sobre a adequação do protótipo, isto é, verificar se é realmente o que o usuário deseja. Esse consenso é realizado através da experimentação do protótipo.

Esta atividade terá como produto um laudo de avaliação apontando o problemas e possíveis soluções.

Recursos Adequados: Reunião com grupo interessado.

Produtos Gerados: Protótipo Avaliado.

V.3 Modelo de Prototipação Evolutiva

V.3.1 - Diagrama de Atividades

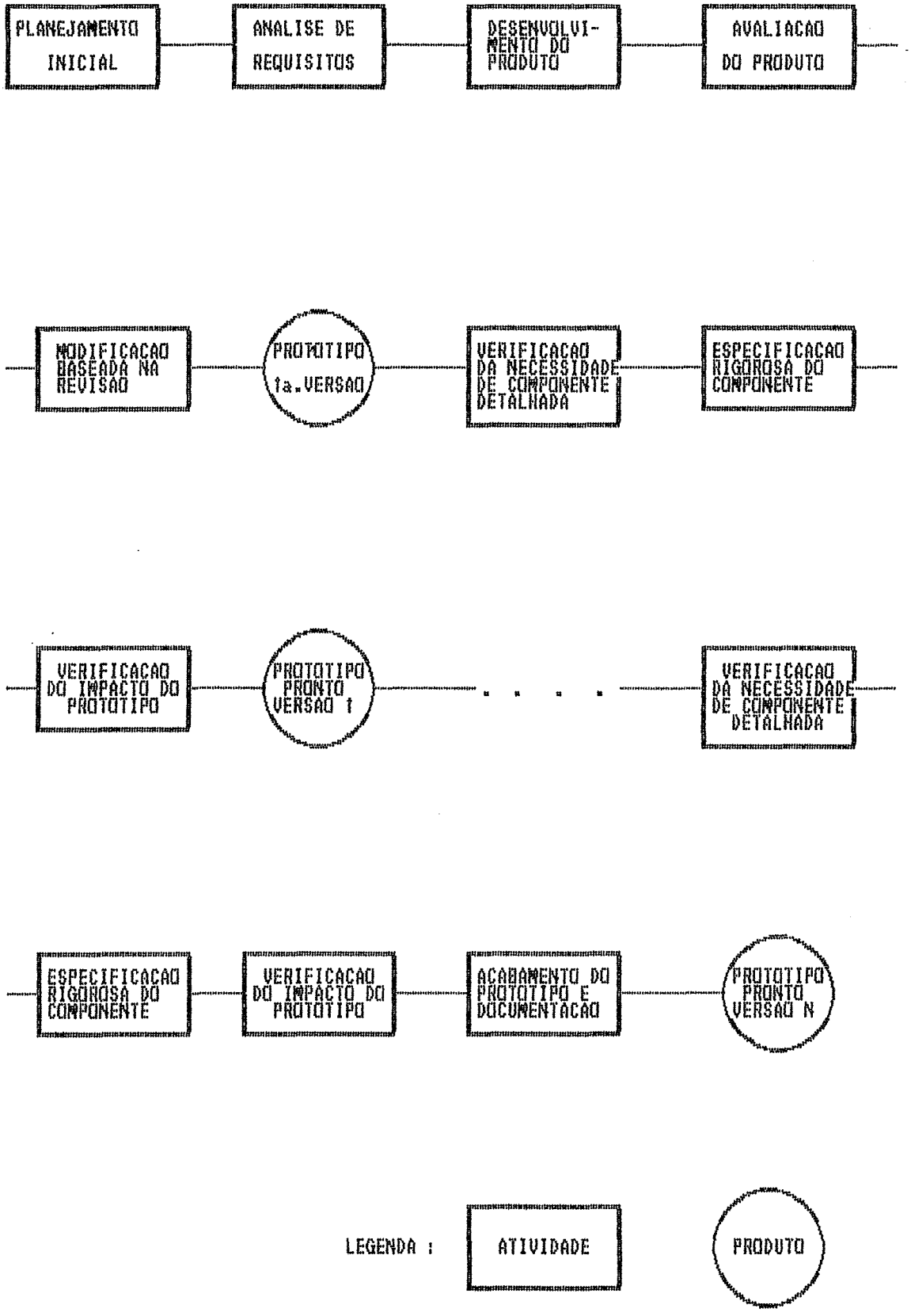


FIGURA 23 - Diagrama de Atividades do Modelo de Prototipação Evolutiva

V.3.2 - Recursos Adequados e Produtos Gerados por Atividade

Planejamento Inicial

Descrição: Desenvolver um plano, que especifique o que deve ser feito, quem deve realizar cada tarefa e qual é o custo previsto. Ao planejar, deve-se ter em conta a ocorrência de alterações. O planejamento é benéfico porque reduz a probababilidade de imprevistos e consequentemente dos problemas daí decorrentes.

Recursos Adequados: Editor de Documentos;
Ferramenta para estimar custo de desenvolvimento de software.

Produtos gerados: Plano Inicial do Projeto.

Análise de Requisitos

Descrição: Analisar o software através de uma visão macroscópica (especificação de requisitos). As especificações devem ser base para o desenvolvimento, permitir o controle da qualidade do produto, estabelecer a comunicação entre o pessoal envolvido no projeto e auxiliar no entendimento do problema a ser resolvido.

As especificações são a primeira forma em que o software existe e, portanto, a base de todo o desenvolvimento futuro. Especificações são produzidas utilizando-se metodologias e linguagens de especificação.

Recursos Adequados: Editor de Documentos;

Métodos/Linguagem de Especificação.

Produtos gerados: Especificação de Requisitos.

Desenvolvimento do Produto

Descrição: Desenvolver o software baseado na especificação dos requisitos e do plano inicial do projeto.

Recursos Adequados: Ferramentas para construção do protótipo.

Produtos Gerados: Protótipo.

Avaliação do Produto

Descrição: Conseguir um consenso sobre a adequação do protótipo, isto é, verificar se é realmente o que o usuário deseja através de experimentação.

Recursos Adequados: Reuniões.

Produtos Gerados: Laudo de avaliação baseado na demonstração do protótipo.

Modificação Baseada na Revisão

Descrição: Com o laudo de avaliação do protótipo serão elaboradas as modificações necessárias.

Recursos Adequados: Ferramentas para construção do protótipo.

Produtos Gerados: Protótipo 1a versão.

Verificação da Necessidade de Componente Detalhada

Descrição: Verificar a necessidade de acrescentar novas

funções e/ou melhorar as funções já existentes no protótipo produzido anteriormente.

Recursos Adequados: Método de Avaliação.

Produtos Gerados: Laudo de verificação baseado no protótipo apresentado.

Especificação Rigorosa do Componente

Descrição: Acrescentar à especificação de requisitos novas informações.

Recursos Adequados: Editor de Documentos;
Métodos/Linguagem de Especificação.

Produtos Gerados: Componente especificado.

Verificação do Impacto do Protótipo

Descrição: Verificar se o protótipo apresentado é realmente o que o usuário deseja através de experimentação.

Recursos Adequados: Reuniões.

Produtos Gerados: Laudo de verificação baseado no protótipo apresentado.

Acabamento do Protótipo e Documentação

Descrição: Terminar o protótipo para que satisfaça totalmente as necessidades do usuário e elaborar uma documentação que possa servir como base de seu desenvolvimento.

Recursos Adequados: Linguagem de 4a geração;

Editor de Documento.

Produtos Gerados: Protótipo pronto versão N.

V.4 - Modelo de Desenvolvimento Incremental

V.4.1 - Diagrama de Atividades

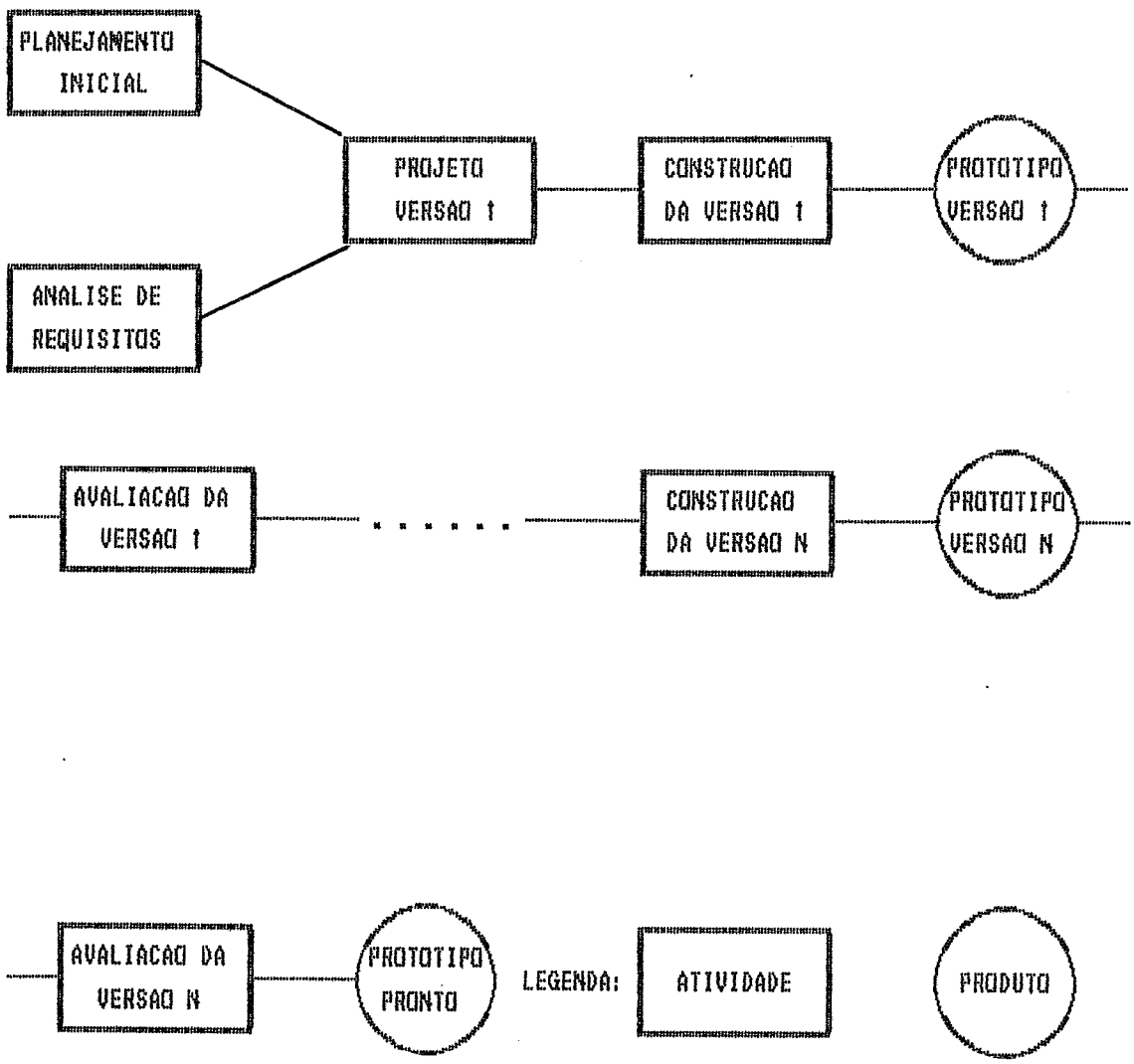


FIGURA 24 - Diagrama de Atividades do Modelo de Desenvolvimento Incremental

V.4.2 - Recursos Adequados e Produtos Gerados por Atividade

Planejamento Inicial

Descrição: Desenvolver um plano, que especifique o que deve ser feito, quem deve realizar cada tarefa e qual é o custo previsto. Ao planejar, deve-se ter em conta a ocorrência de alterações. O planejamento é benéfico porque reduz a probabilidade de imprevistos e consequentemente dos problemas daí decorrentes.

Recursos Adequados: Editor de Documentos;
Ferramenta para estimar custo de desenvolvimento de software.

Produtos gerados: Plano Inicial do Projeto.

Análise de Requisitos

Descrição: Analisar o software através de uma visão macroscópica (especificação de requisitos). As especificações devem ser base para o desenvolvimento, permitir o controle da qualidade do produto, estabelecer a comunicação entre o pessoal envolvido no projeto e auxiliar no entendimento do problema a ser resolvido.

As especificações são a primeira forma em que o software existe e, portanto, a base de todo o desenvolvimento futuro. Especificações são produzidas utilizando-se metodologias e linguagens de especificação.

Recursos Adequados: Editor de Documentos;

Métodos/Linguagem de Especificação.

Produtos gerados: Especificação de Requisitos.

Projeto da Versão I

Descrição: Baseado no plano inicial do projeto e na especificação de requisitos ou no laudo de avaliação baseado no protótipo produzido é elaborado a especificação capaz de tornar real o programa em algum ambiente de programação.

Recursos Adequados: Método de especificação de projeto.

Produtos Gerados: Projeto da Versão I

Construção da Versão I

Descrição: Codificar e integrar o projeto da versão I de maneira progressiva para a formação completa do esqueleto do protótipo.

Recursos Adequados: Ferramentas.

Produtos Gerados: Protótipo Versão I.

Avaliação da Versão I

Descrição: Conseguir um consenso sobre a adequação do protótipo versão I, isto é, verificar se é realmente o que o usuário deseja através de experimentação.

Recursos Adequados: Reuniões;

Editor de Documentos.

Produtos Gerados: Laudo de avaliação baseado no protótipo

versão I.

V.5 - Modelo para Desenvolvimento Orientado a Objetos

V.5.1 - Diagrama de Atividades

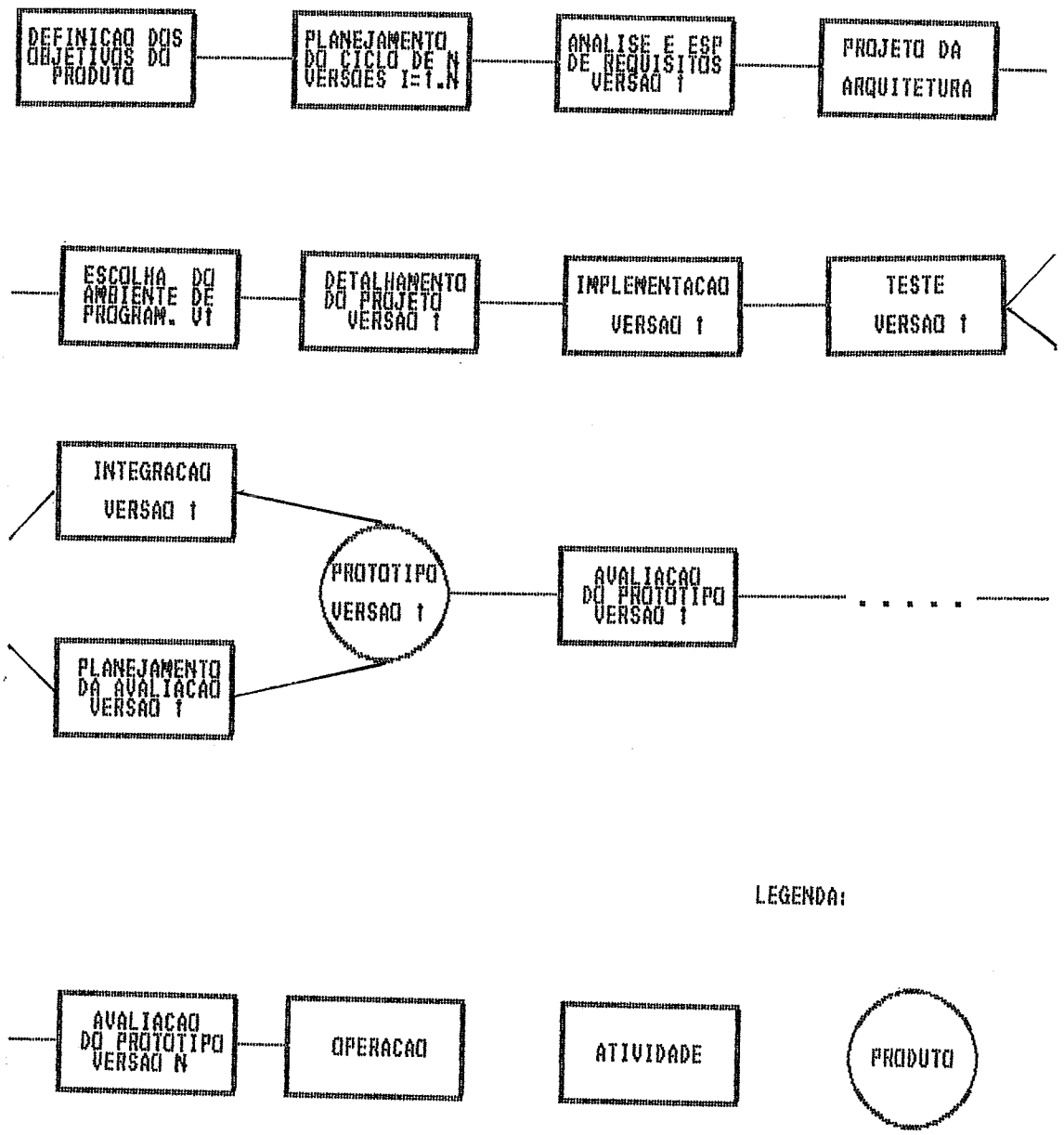


FIGURA 25 - Diagrama de Atividades do Modelo para Desenvolvimento Orientado a Objetos

V.5.2 - Recursos Adequados e Produtos Gerados por Atividade

Definição dos Objetivos do Produto

Descrição: Deve-se procurar entender todos os aspectos envolvidos no problema a ser resolvido, a fim de se definirem os objetivos do produto a ser desenvolvido. Para se compreender bem o espaço do problema, deve-se conversar com especialistas, ler sobre o assunto, fazer desenhos ou diagramas, etc.

Recursos Adequados: Editor de Documentos;
Reuniões.

Produtos Gerados: Definição do Sistema;
Planejamento Preliminar.

Planejamento do Ciclo da Versão I

Descrição: Definir e Planejar a versão a ser construída

Recursos Adequados: Editor de Documentos;
Reuniões.

Produtos Gerados: Plano da Versão I.

Análise e Especificação de Requisitos da Versão I

Descrição: A especificação dos requisitos deve ser feita tomando-se como base a análise cujo objetivo principal é a identificação dos objetos básicos e suas operações para a elaboração da versão inicial. Esta versão conterá somente um núcleo mínimo de funções, tornando mais fácil esta

identificação. A partir desta versão novos objetos poderão ser facilmente integrados no sistema.

Recursos Adequados: Editor de Documentos;
Editor de Diagrama de Classes.

Produtos Gerados: Especificação de Requisitos
Diagrama de Classes Preenchidos

Projeto da Arquitetura da Versão I

Descrição: Deve-se ter como base a especificação de requisitos elaborada anteriormente efetuando-se os seguintes passos:

- Definir os algoritmos das operações identificadas;
- Elaborar o dicionário de classes;
- Detalhar os requisitos de interface com o usuário.

Recursos Adequados: Editor de Documentos.

Produtos Gerados: Documento de especificação de projeto.

Escolha do Ambiente de Programação da Versão I

Descrição: Deve anteceder ao detalhamento do projeto uma vez que, dependendo do ambiente escolhido, é possível conhecer-se as classes já existentes.

Este conhecimento é importante para a análise de reutilização de classes de objetos feita durante o detalhamento do projeto, já que em função do ambiente de programação a ser adotado, pode existir ou não uma biblioteca

de classes disponíveis para serem reutilizadas.

Recursos Adequados: Reuniões.

Recursos Adequados: Reuniões.

Produtos Gerados: Ambiente de Programação Escolhido

Detalhamento do Projeto da Versão I

Descrição: Deve-se tomar como base o projeto da arquitetura. Para cada classe de objetos deve-se verificar hipóteses de reutilização.

Recursos Adequados: Método de projeto detalhado.

Produtos Gerados: Especificação de Projeto Detalhado.

Implementação da Versão I

Descrição: Codificar os componentes do produto levando em consideração a linguagem de programação utilizada.

Recursos Adequados: Linguagem de Programação;
Software Básico.

Produtos Gerados: Versão I codificada.

Teste da Versão I

Descrição: Para depurar cada classe de objetos codificada, deve ser simulada a sua execução.

Recursos Adequados: Ambiente de Programação.

Produtos Gerados: Plano de Teste.

Integração da Versão I

Descrição: Integrar o produto

Recursos Adequados: Método de Integração.

Produtos Gerados: Protótipo Versão I.

Planejamento da Avaliação da Versão I

Descrição: A avaliação do produto deve ser feita em função do processo de integração, isto é, o produto deve ser avaliado à medida que suas partes forem sendo integradas.

Recursos Adequados: Editor de Documentos;
Reuniões.

Produtos Gerados: Laudo de avaliação baseado no protótipo versão I.

Avaliação do Protótipo da Versão I

Descrição: Conseguir um consenso sobre a adequação do protótipo versão I, isto é, verificar se é realmente o que o usuário deseja através de experimentação.

Recursos Adequados: Reuniões.

Produtos Gerados: Protótipo avaliado.

Operação

Descrição: Colocar em uso o protótipo produzido e quando necessário realizar correções, aprimoramentos, adaptações e expansões.

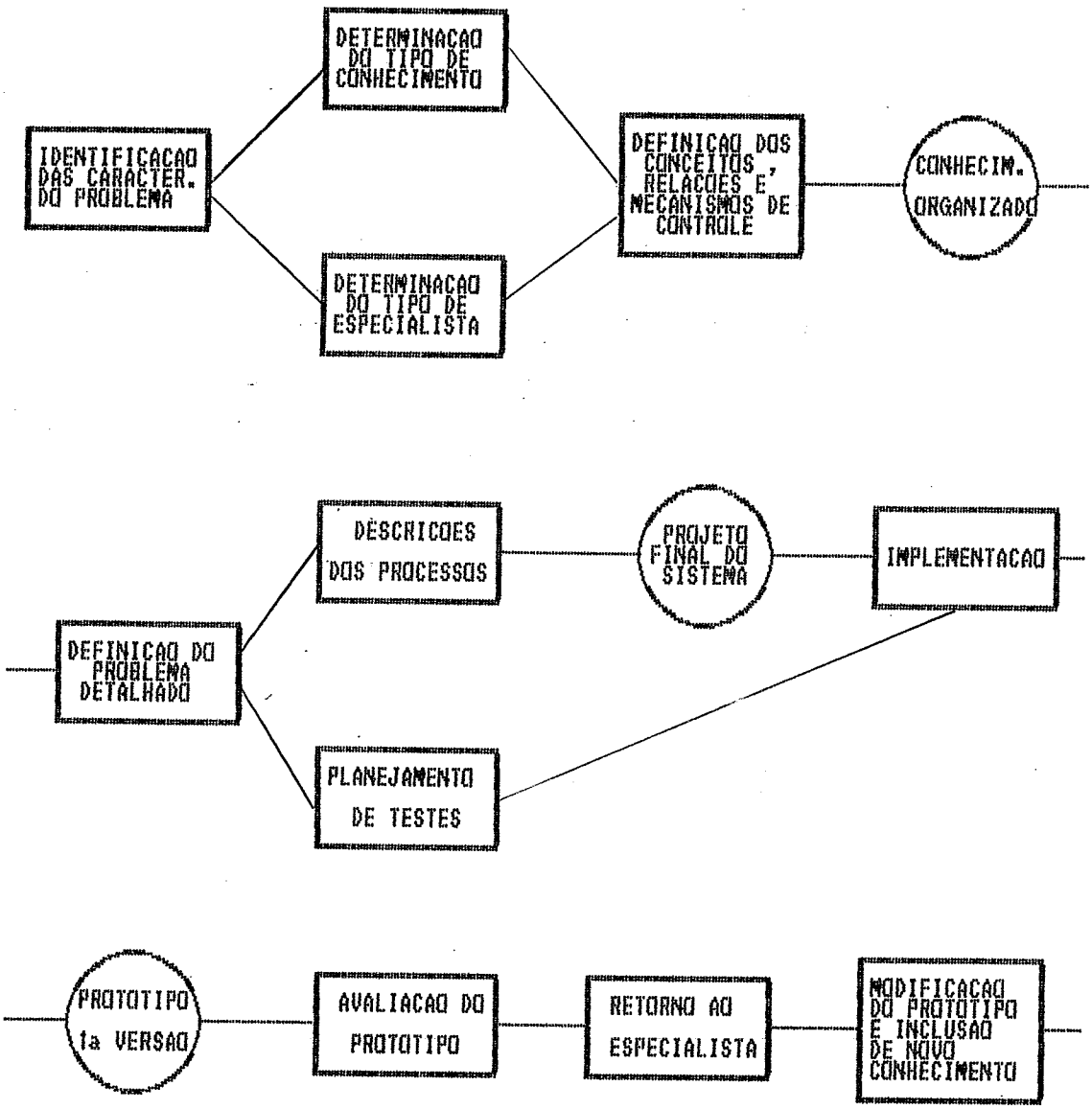
Recursos Adequados: Protótipo avaliado;

Ambiente de programação escolhido.

Produtos Gerados: Protótipo pronto.

V.6 - Modelo Proposto para Desenvolvimento de Sistemas Especialistas

V.6.1 - Diagrama de Atividades



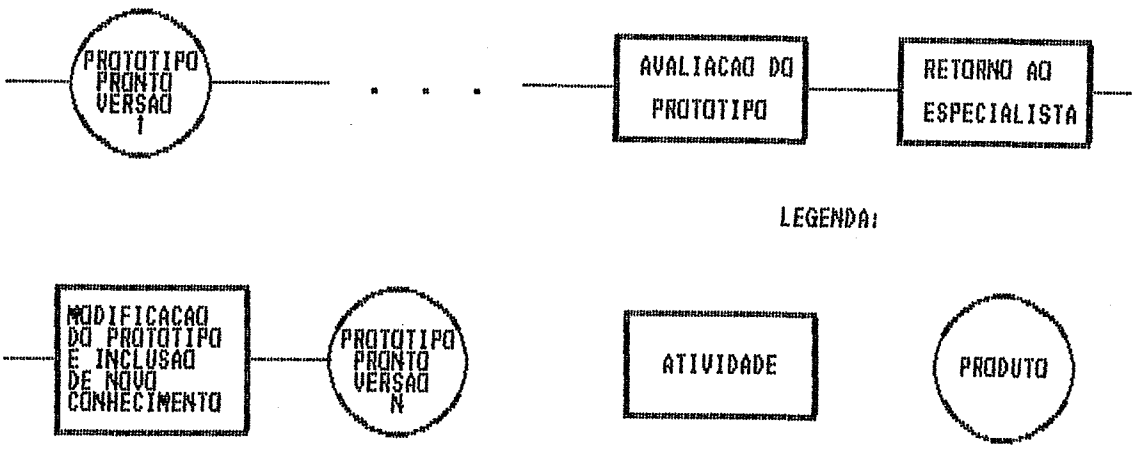


FIGURA 26 - Diagrama de Atividades do Modelo para Desenvolvimento de Sistemas Especialistas

V.6.3 - Recursos Adequados e Produtos Gerados por Atividade

Identificação das Características do Problema

Descrição: Identificar o problema, os participantes no processo de desenvolvimento e os objetivos da construção do sistema especialista.

Recursos Adequados: Editor de Documento;
Reuniões.

Produtos Gerados: Relatório de narrativa do sistema (contendo as entradas e saídas, o que contém os dados e que decisões devem ser feitas pelo sistema);
Diagrama de Fluxo de Dados (mostrando as interfaces do sistema).

Determinação do Tipo de Conhecimento

Descrição: Determinar a área de conhecimento a ser enfocada.

no desenvolvimento do projeto.

Recursos Adequados: Reuniões;
Editor de Documentos.

Produtos Gerados: Descrição do conhecimento.

Determinação do Tipo de Especialista

Descrição: Determinar o perfil do especialista que será necessário para o desenvolvimento do projeto.

Recursos Adequados: Reuniões;
Editor de Documentos.

Produtos Gerados: Descrição do especialista.

Definição dos Conceitos, Relações e Mecanismos de Controle

Descrição: O engenheiro de conhecimento e o especialista devem definir conceitos, relações e mecanismo de controle que serão necessários para descrever a resolução do problema no domínio. Subtarefas, estratégias e restrições relacionadas para a atividade de resolução do problema são também exploradas.

Recursos Adequados: Editor de Documento;
Reuniões;
Método de especificação do conhecimento.

Produtos Gerados: Especificação estruturada (o sistema pode ser dividido em tarefas ou mini-sistemas, mais fáceis de especificar, que serão descritos por uma mini-especificação).

Definição do Problema Detalhado

Descrição: Mapear as relações e os conceitos definidos anteriormente e planejar o método de representação do conhecimento apropriado de modo a esclarecer pontos que podem não ter ficado claros no início do projeto.

Recursos Adequados: Reuniões;
Editor de Documento.

Produtos Gerados: Descrição funcional (define o problema a ser solucionado, os objetivos a serem alcançados, restrições que servem para limitar o escopo do projeto e as expectativas do usuário).

Descrições dos Processos

Descrição: Descrever as funções do sistema de forma não ambígua e não redundante.

Recursos Adequados: Métodos.

Produtos Gerados: Projeto final do sistema.

Planejamento de Testes

Descrição: Definir os elementos necessários para verificar a validade do sistema depois de desenvolvido. Inclui a identificação de casos de testes e procedimentos para executá-los.

Recursos Adequados: Editor de Documento.

Produtos Gerados: Plano de teste.

Implementação

Descrição: Implementar um protótipo de modo que seja codificado nos termos da ferramenta de implementação escolhida o conhecimento formalizado inicialmente. É definido o modelo geral de consultas, paradigma de inferência e representação do conhecimento. O modelo geral de consultas descreve tipos de questões que podem ser perguntadas, tipos de usuários e formas nas quais as respostas devem ser respondidas.

Recursos Adequados: Linguagem de codificação.

Produtos Gerados: Protótipo 1a versão.

Avaliação do Protótipo

Descrição: Conseguir um consenso sobre a adequação do protótipo baseado na demonstração de sua funcionalidade. Testar o protótipo implementado com alguns problemas simples, que são supostamente possíveis de serem solucionados. Esses problemas são identificados pelo especialista.

Recursos Adequados: Método de Avaliação;
Método de Teste.

Produtos Gerados: Documento com laudo de avaliação baseado no protótipo apresentado anteriormente.

Retorno ao Especialista

Descrição: Caso seja produzido resultados errôneos, o especialista deve explicar porque o programa produziu

CAPÍTULO VI

CONCLUSÃO

Esta tese teve como objetivo realizar um estudo da literatura sobre os diferentes modelos de ciclo de vida de software propostos e uma posterior classificação dos mesmos tendo em vista sua adequação ao desenvolvimento de produtos com diferentes características.

Para atingir-se este objetivo foram descritos diversos modelos de ciclo de vida de software encontrados na literatura, onde se identificou suas principais propostas.

Algumas comparações entre os diferentes modelos de ciclo de vida de software mencionados foram mostradas.

Considerando os aspectos relacionados a critérios de seleção foi possível agrupar os diversos modelos em dois tipos: modelos orientados para um paradigma de desenvolvimento de software e modelos orientados para as características do produto a ser desenvolvido.

A partir de estudos anteriores e características de cada modelo foram selecionados os modelos de ciclo de vida de software de interesse para implementação na Estação TABA. Para os modelos selecionados, o conhecimento adquirido foi organizado conforme será exibido ao usuário do meta-ambiente. Um conjunto inicial de regras com relação à adequação do modelo, foram definidas. Estas regras farão parte da base de conhecimentos do sistema especialista de apoio a decisão que auxiliará o usuário da Estação TABA na especificação do

ambiente de desenvolvimento de software mais adequado a um projeto específico.

Esta tese está inserida no contexto do Projeto TABA [ROCHA90], e mais especificamente do meta-ambiente da Estação. Os resultados deste trabalho fornecem subsídios para uma tese de doutorado, em andamento, e completa-se com os trabalhos de Werneck [WERNECK90] e Crispim [CRISPIM91].

Esta tese pode ser base para futuros trabalhos através do estudo de novos modelos que forem surgindo na literatura e a organização das informações obtidas para inclusão na Estação TABA.

Para se obterem mais dados sobre a adequação dos modelos, pode-se, também, fazer uma pesquisa de campo e/ou adquirir novos conhecimentos a partir de entrevistas a especialistas em Engenharia de Software. Estes conhecimentos poderão, depois, ser incorporados à base de conhecimentos da Estação TABA.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIAR, T. C. (1989), **"Protótipo do Especificador de Ambientes da Estação Taba"**, Relatório Técnico ES-208/89 - COPPE/UFRJ, Julho de 1989.

BALZER, R. M., CHEATHAM, T. E., GREEN, C. (1983), **"Software Technology in the 1990's: Using a new Paradigm"**, Computer, Vol. 16, pp 39-45, Nov. 1983.

BLUM, B. I. (1982), **"The Life Cycle - A Debate Over Alternate Models"**, Acm Sigsoft Software Engineering Notes, Vol. 7, No. 4, Oct 1982.

BLUM, B. I. (1983), **"Still More About Rapid Prototyping"**, Acm Sigsoft Software Engineering Notes, Vol. 8, No. 3, Jul 1983.

BOAR, Bernard (1984), **"Application Prototyping"**, Reading, Mass .: Addison-Wesley, 1984

BOEHM, B. W. (1976), **"Software Engineering"**, IEEE Transactions on Computers, Vol. C-25, No. 12, pp 1226-1241, Dec 1976.

BOEHM, B. W., BROW, J. R., KASPAS, H., et alli. (1978), **"Characteristics of Software Quality"**, Nort Holland Publishing Co., 1978.

BOEHM, B. W. (1981), **"Software Engineering Economics"**, Englewood Clifs, Prentice-Hall,

Inc., 1981.

BOEHM, B. W. (1986), **"A Spiral Model of Software Development and Enhancement"**, Software Engineering Notes, Vol. 11, No 4, pp 22-42, 1986, reprinted in IEEE Computer, Vol 21, No 5, pp 61-72, May 1988.

BOEHM, B. W. (1988), **"Understanding and Controlling Software Costs"**, IEEE Transactions on Software Engineering, Vol. 14, No 10, October 1988.

CAPRON, H. L. (1986), **"System Analyses and Design"**, The Benjamin/Cummings Publishing Company, Inc., 1986.

CONNOR, M. F. (1980), **"An Integrated View of the Computer Software Application Development Life Cycle"**, in Cotterman, W. W., and Couger, J. D., System Analysis and Design: A Foundation for the 1980's, North Holland, 1980.

CRISPIM, E. M. H. (1991), **"Avaliação de métodos para o desenvolvimento de software"**, Tese Msc. COPPE/UFRJ, 1991.

DAVIS, Alan M., BERSOFF, E. H., COMER, E. R. (1988), **"A Strategy for Comparing Alterantive Software Development Life Cycle Models"**, IEEE Transactions on Software Engineering, Vol. 14, No. 10, October 1988.

ENGER, L. N. (1980), **"Classical and Structured Systems Life Cycle Phases and Documentation"**, in Cotterman, W. W., and Couger, J. D., System Analysis and Design: A Foudation for the 1980's, North Holland, 1980.

FAIRLEY, R. (1985), **"Software Engineering Concepts"**, McGraw Hill, 1985.

GILB, T. (1988), **"Principles of software engineering management"**, Addison - Wesley, 1988.

GLADDEN, G. R. (1982), **"Stop the Life Cycle, I Want to Get Off"**, Acm Sigsoft Software Engineering Notes, Vol. 7, No. 2, April 1982.

GOMMA, H., SCOTT, D. (1981), **"Prototyping as a tool in the specification of user requirements"**, in Proc. 5th IEEE Int. Conf. Software Eng., pp. 333-342, March 1981.

GUTTAG, J., HOROWITZ, E., MUSSER, D.R. (1978), **"Abstract data types and Software Validation"**, Commun ACM, Vol.21, No.12, pp.1048-1063, Dec 1978.

HALL, P. A. V. (1982), **"In Defence of Life Cycles"**, Scicon Consultancy International Ltd., 49 Berners Street, London W.L..

HAUSEN, Hans-Ludwing, MULLERBURG, M.(1981), **"Conspestus of Software Engineering Environments"**, in:

Proc. 5th International Conference on Software Engineering, San Diego, March 1981, IEEE.

HIRSCH, E. (1985), **"Evolutionary acquisition of command and control systems"**, Program Manager, pp. 18-22, Nov.- Dec. 1985.

HOUGHTON, R. C. Jr., WALLACE, D. R. (1987), **"Characteristics and Functions of Software Engineering Environments: An Overview"**, Acm Sigsoft Software Engineering Notes, Vol. 12, No. 1, Jan 1987.

HOWDEN, Wilian E. (1982), **"Life-Cycle Software Validation"**, IEEE Transactions on Computers, pp 71-78, February 1982.

JALOTE, Pankaj (1989), **"Functional Refinement and Nested Objects for Object-Oriented Design"**, IEEE Transactions on Software Engineering, Vol. 15, No 3, March 1989.

MATOS, J. P. (1988), **"Ambiente para desenvolvimento de sistemas especialistas"** Monografia COPPE/SISTEMAS, UFRJ, 1988.

MATTOSO, Adriana L. Q. (1990), **"TABA-OBJ: Um ambiente de desenvolvimento de software com orientação a objetos"**, Tese - UFRJ, COPPE - 1990.

MCCRACKEN, D. D., JACKSON, M. A. (1982), **"Life Cycle Concept Considered Harmful"**, Acm Sigsoft

Software Engineering Notes, Vol. 7, No. 2, April 1982.

MENDES, S., e ROCHA, A. R. C. da (1987), **"Paradigmas de Ambientes de Desenvolvimento de Software"**, Monografia COPPE/SISTEMAS, UFRJ, 1987.

MEYERS, G. J. (1978), **"Composite / Structure Design"**, New York, Van Nostrand, 1978.

O'NEILL, Donald (1980), **"Software Engineering Techniques Applied to the Systems Developments Process"**, in Cotterman, W. W., and Couger, J. D., System Analysis and Design: A Foudation for the 1980's, North Holland, 1980.

PARNAS, D. L. (1972), **"On the criteria to be used in decomposing systems into modules"**, Commun. ACM, Vol. 15, No. 12, pp. 1053-1058, Dec 1972.

PRESSMAN, R. (1987), **"Software Engineering : A Practioner's Approach"**, 1987.

RIBEIRO, Carlos A. S. (1989), **"Um Sistema de Apoio à Avaliação de Custos de Software"**, Tese Msc. COPPE/UFRJ, 1989.

ROCHA, A. R. C. da (1987), **"Análise e Projeto Estruturado de Sistemas"**, Editora Campus, 1987.

ROCHA, A. R. C. da, AGUIAR, T.C. de, e BLASCHEK, J.R. (1987a), **"Ambientes para Desenvolvimento de**

Software: Definição de Termos", Relatório Técnico do Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, ES-137/87.

ROCHA, A. R. C. da, e SOUZA, J. M. de (1988), **"Uma Estação de Trabalho Para o Engenheiro de Software"**, Relatório Técnico do Programa de Engenharia de Sistemas e Computação, 1988.

ROCHA, A. R. C. da, AGUIAR, T. C. de, SOUZA, J. M. de, D'IPOLITO, Claudio (1989), **"O Meta-Ambiente da Estação Taba"**, Relatório Técnico do Programa de Engenharia de Sistemas e Computação, Março 1989.

ROCHA, A. R. C. da, et alli (1989a), **"O Meta -Ambiente da Estação TABA"**, XV Conferência Latina Americana de Informática, Santiago, Chile, Julho de 1989.

ROCHA, A. R. C. da, SOUZA, J. M., AGUIAR, T.C. (1990), **"TABA: A Heuristic Workstation for Software Development"**, COMPEURO 90, Tel Aviv, Israel, Maio de 1990.

ROYCE, W. W. (1970), **"Managing the Development of Large Software Systems : Concepts and Techniques"**, Proceeding Wescon, August 1970.

SEIDEWITZ, E., STARK, M. (1987), **"Toward a general object oriented ADA lifecycle"**, Proceedings of the joint 4th Washington Area ADA Technology, ACM

NASA Goddard Space Center, et al., Washington D. C. , 1987.

SOMMERVILLE, I. (1982), **"Software Engineering"**, International Computer Science series, 1982.

STAA, A. V. (1979), **"Desenvolvimento Estruturado de Sistemas Automatizado"**, Série: Monografias em Ciência da Computação, No. 9/79, Depto Inf, PUC/RJ, 1979.

TRIPP, Leonard L. (1981), **"Evaluation of Software Development Life Cycle Methodology Implementation"**, ACM - Software Engineering Symposion, Colorado, 1981.

WATERMAN, D. A. (1986), **"A Guide to Expert Systems"**, Addison - Wesley Publishing Company, 1986.

WERNECK, V. M. B., ASSIS, S. G., MATOS, J. P., AGUIAR, T. C. de (1989), **"Especificador de Ambientes da Estação Taba : Fase de Identificação"**, Relatório Técnico do Programa de Engenharia de Sistemas e Computação, Julho de 1989.

WERNECK, V. M. B. (1990), **"Taxonomia de domínios de aplicação"**, Tese Msc. COPPE/UFRJ - 1990.

WESSELIUS, J. , VERVERS, F. (1990), **"Some elementary questions on software quality control"**, Software Engineering Journal, Vol. 5, No. 6, November 1990.

YAU, Stephen S. (1986), **"A Survey of Software Design Techniques,"** IEEE Transactions on Software Engineering, Vol. SE-12, No. 6, June 1986.

YOURDON, E. N., CONSTANTINE, L. L. (1979), **"Structured Design"**, Englewood Cliffs, N.J., Prentice-Hall, 1979.

YOURDON, E. N. (1989), **"Modern Structured Analysis"**, Prentice-Hall, 1989.

ZAVE, P. (1984), **"The Operational Versus the Conventional Approach to Software Development"**, Communications of the ACM, vol 27, no 2, pp 104-118, February 1984.

resultado errado e recomenda o que é que o sistema poderia saber ou fazer para não repetir o erro.

Recursos Adequados: Reuniões.

Produtos Gerados: Novo conhecimento.

Modificação do Protótipo e Inclusão de Novo Conhecimento

Descrição: Com o laudo de avaliação do protótipo serão elaboradas as modificações necessárias, de modo a introduzir o novo conhecimento na forma de novas regras ao sistema.

Recursos Adequados: Linguagem de codificação.

Produtos Gerados: Protótipo pronto versão N.

V.7 - Conclusão

Este capítulo descreveu cada modelo de ciclo de vida selecionado representando-os através de diagramas de atividades e definindo, detalhadamente, cada uma das atividades identificadas no mesmo.