TESE
1052

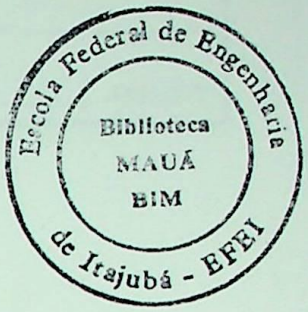
Federal de Engenharia de Itajubá
Progama de Pós Graduação em
Engenharia de Produção

Aplicação do desdobramento da função qualidade no gerenciamento da configuração de sistemas

José Eduardo Morello Lobo

ITAJUBÁ, JUNHO DE 2000

Escola Federal de Engenharia de Itajubá
Departamento de Produção



José Eduardo Morello Lobo

Aplicação do desdobramento da função qualidade no gerenciamento da configuração de sistemas

Dissertação submetida ao Programa de Pós Graduação em Engenharia de Produção como requisito parcial à obtenção do título de Mestre em Engenharia de Produção

Orientador: Prof. Dr. Luiz Gonzaga Mariano de Souza
Co-orientador: Prof. Dr. João Bosco Schumann Cunha

Itajubá, 27 de Junho de 2000

CLASS. 004.65:658.5610 43.2)
CUTTER. L799a
TOMBO. 1052

Lobo, José Eduardo Morello. Aplicação do desdobramento da função qualidade no gerenciamento da configuração de sistemas. Itajubá: EFEI, 2000. 132p. (Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Escola Federal de Engenharia de Itajubá).

Palavras-Chaves: QFD – CMM – SCM – desdobramento da função qualidade – Modelo de capacidade e maturidade de software – Gerenciamento da configuração de sistemas.

Escola Federal de Engenharia de Itajubá
Departamento de Produção

José Eduardo Morello Lobo

Aplicação do desdobramento da função qualidade no gerenciamento da configuração de sistemas

Dissertação submetida ao Programa de Pós Graduação em Engenharia de Produção como requisito parcial à obtenção do título de Mestre em Engenharia de Produção

Banca Examinadora:
Prof. Dr. Luiz Gonzaga Mariano de Souza (orientador)
Prof. Dr. João Bosco Schumann Cunha (co-orientador)
Prof. Dr. João Batista Turrioni
Prof. Dr. Tatuo Nakanishi

Itajubá, 27 de Junho de 2000

À minha esposa, ao meu filho
e aos meus pais.

Agradecimentos

Ao Deus Eterno, em nome de Jesus, acima de todas as coisas;

Aos amigos e colegas de Mogi e de Itajubá pelo incentivo;

À UMC-Universidade de Mogi das Cruzes por todo apoio;

Aos funcionários da EFEI e aos professores do programa por toda colaboração;

Aos professores João Batista e Tatuo pelas críticas e sugestões;

Ao Prof. João Bosco e ao Prof. Gonzaga pelo incentivo, apoio, colaboração, dedicação e empenho.

Resumo

Produtos de alta qualidade que satisfaçam as necessidades dos clientes tem sido um objetivo comum buscado por qualquer ramo de negócio. No caso da indústria de software apesar dos esforços e da tecnologia disponível, freqüentemente não se alcança esse objetivo.

Tal fato deve-se, entre outros motivos, à falta de gerenciamento do processo de desenvolvimento de software e à falta de conhecimento e/ou falta de vontade das pessoas e de suas organizações em aplicar a tecnologia disponível.

Esta pesquisa mostra algumas tecnologias propostas pela engenharia de software e ressalta a importância de um modelo de capacitação - das pessoas e da organização - e do controle de configuração no processo de gerenciamento do desenvolvimento.

Apresenta como resultado um esquema para esse controle obtido através da aplicação de uma ferramenta de Desenvolvimento de Produto, o QFD – Desdobramento da Função Qualidade.

Abstract

Products of high quality that satisfy the customers' needs have been a common objective looked for by any business branch. In the case of the software industry in spite of the efforts and of the available technology, frequently it is not reached this goal.

Such fact is due, among other reasons, to the lack of software development process management and the lack of knowledge and/or lack of the people's will and theirs organizations in applying the available technology.

This research exhibition some technologies proposals for the software engineering and it stands out the importance of a model of capability and maturity - for the people and theirs organization - and of the configuration control in software development process management.

It presents as result an schema for that control obtained through the application of one Product's Development tool, the QFD – Quality Function Deployment.

Sumário

Capítulo 1 – INTRODUÇÃO

1.1 - DESCRIÇÃO DO PROBLEMA 01
1.2 - OBJETIVO 06
1.3 - RELEVÂNCIA 08
1.4 - ESTRUTURA DO TRABALHO 09

Capítulo 2 – GESTÃO DO DESENVOLVIMENTO DE SISTEMAS

2.1 - INTRODUÇÃO 10
2.2 - ENGENHARIA DE SOFTWARE - DEFINIÇÃO 12
2.3 - METODOLOGIA, TÉCNICA E FERRAMENTAS 13
2.4 - QUALIDADE DO SOFTWARE 14
2.5 - SISTEMA DE SOFTWARE E DOMÍNIO DO PROBLEMA 19
2.6 - CICLO DE VIDA DO SOFTWARE 20
2.7 - PARADIGMAS DE ENGENHARIA DE SOFTWARE 23
2.8 - A UML – UNIFIED MODELING LANGUAGE 33

Capítulo 3 – CMM - MODELO DE CAPACIDADE E MATURIDADE

3.1 - INTRODUÇÃO 45
3.2 - ESTRUTURA 49
3.3 - OS NÍVEIS DE MATURIDADE 51
3.4 - ÁREAS-CHAVE DE PROCESSO 56
3.5 - OBJETIVOS DAS ÁREAS-CHAVE 58
3.6 - CARACTERISTICAS COMUNS E PRÁTICAS-CHAVE 62

Capítulo 4 – SCM - GERENCIAMENTO DA CONFIGURAÇÃO DE SISTEMAS

4.1 - INTRODUÇÃO 63
4.2 - DEFINIÇÃO 66
4.3 - ASPECTOS OPERACIONAIS 69
4.4 - ABORDAGENS DE CONFIGURAÇÃO 73
4.5 - USUÁRIOS DO SCM 76
4.6 - REQUISITOS DE FUNCIONALIDADE 78
4.7 - DIFICULDADES DE IMPLEMENTAÇÃO 81

Capítulo 5 – QFD - DESDOBRAMENTO DA FUNÇÃO DA QUALIDADE

5.1 - INTRODUÇÃO 84
5.2 - OBJETIVOS 90
5.3 - AS ETAPAS OU MATRIZES DO QFD 94
5.4 - OUTRAS ABORDAGENS DE QFD 106

Capítulo 6 – A APLICAÇÃO DO QFD NA DETERMINAÇÃO DE UM ESQUEMA PARA O SCM

INTRODUÇÃO 110
OBTENÇÃO DAS MATRIZES
MATRIZ 1 - MATRIZ DA QUALIDADE 112
MATRIZ 2 - MATRIZ DOS PROCESSOS 115
MATRIZ 3 - MATRIZ DOS RECURSOS 122

Capítulo 7 – CONCLUSÕES E RECOMENDAÇÕES

CONCLUSÕES 126
RECOMENDAÇÕES 128

Capítulo 8 – REFERÊNCIAS BIBLIOGRÁFICAS 129

Lista de Figuras

Figura 1.1 - Comparativo entre avanço tecnológico e capacitação e maturidade..... 04
Figura 1.2 - Visão geral do objetivo deste trabalho de pesquisa..... 07

Figura 2.1 – Sistema de Software e Domínio do Problema..... 19
Figura 2.2 – Visão genérica do Ciclo de Vida de um Software..... 20
Figura 2.3 - Técnicas Estruturadas e o Princípio de Dividir para Conquistar..... 24
Figura 2.4 – DFD - Diagrama de Fluxo de Dados..... 25
Figura 2.5 – DFD com expansão..... 26
Figura 2.6 - Diagrama de Estrutura dos Módulos..... 27
Figura 2.7 - Diagrama de Entidade-Relacionamento..... 28
Figura 2.8 – Um objeto encapsula inteligência..... 31
Figura 2.9 - Notação para o Diagrama de Casos de Uso..... 34
Figura 2.10 - Notação simplificada para o Diagrama de Classes..... 36
Figura 2.11 - Notação para o Diagrama de Estados..... 37
Figura 2.12 - Notação para o Diagrama de Atividades..... 39
Figura 2.13 – Notação para o Diagrama de Seqüência..... 40
Figura 2.14 – Notação para o Diagrama de Colaboração..... 41
Figura 2.15 - Notação para o Diagrama de Componentes..... 43
Figura 2.16 - Notação para o Diagrama de Execução..... 43

Figura 3.1 – Estrutura dos componentes do CMM..... 49
Figura 3.2 – Estrutura dos níveis de Maturidade do CMM..... 51
Figura 3.3 – Áreas-chaves dos níveis de Maturidade do CMM..... 57

Figura 4.1 – Processo da Configuração Estática..... 73
Figura 4.2 – Processo da Configuração Dinâmica..... 74

Figura 5.1 - As matrizes do QFD..... 85
Figura 5.2 – Conceito da Engenharia Simultânea..... 85
Figura 5.3 - O desdobramento da qualidade e o desdobramento da função qualidade..... 87
Figura 5.4 - O desdobramento das necessidades dos clientes em operações de manufatura..... 88
Figura 5.5 - O formato das matrizes do QFD..... 88
Figura 5.6 - Redução do número de reclamações utilizando QFD..... 90
Figura 5.7 - Formato da primeira matriz do QFD..... 95
Figura 5.8 - Legenda dos símbolos utilizados na matriz de relação..... 96
Figura 5.9 - Legenda dos símbolos utilizados na matriz de correlação..... 97
Figura 5.10 - Formato da segunda matriz do QFD..... 100
Figura 5.11 - Formato da terceira matriz do QFD..... 102
Figura 5.12 - Formato alternativo da terceira matriz do QFD..... 103
Figura 5.13 - Formato da quarta matriz do QFD..... 104
Figura 5.14 – Formato alternativo da quarta matriz do QFD..... 105
Figura 5.15 - As matrizes do QFD de Cheng..... 108

Figura 6.1 - Matriz 1 - A Matriz da Qualidade..... 114
Figura 6.2 - Análise de Pareto da primeira matriz do QFD..... 113
Figura 6.3 - Matriz 2 - A Matriz de Processos..... 116
Figura 6.4 - Análise de Pareto da segunda matriz do QFD..... 121
Figura 6.5 - Matriz 3 - A Matriz de Recursos..... 124
Figura 6.6- Análise de Pareto da terceira matriz do QFD..... 125

Lista das Tabelas

Tabela 3.1 – Comparativo de Maturidade das Organizações.....46
Tabela 3.2 – Níveis do CMM.....56

Tabela 6.1 - Exemplos de itens de configuração para a análise orientada.....118
Tabela 6.2 - Exemplos de itens de configuração para a análise estruturada.....119

1. INTRODUÇÃO

1.1 Descrição do problema

O produto software é caracterizado por um conjunto de idéias, conceitos, fórmulas e outros componentes abstratos enquanto o produto hardware pode ser conceituado como algo formado por componentes concretos, palpáveis ou físicos. A produção ou desenvolvimento destes produtos possuem aspectos semelhantes, mas também muitas diferenças devido a própria natureza de cada tipo.

Nota-se que com muita freqüência o software como produto final não atende totalmente as solicitações do cliente, possui uma documentação insuficiente tanto em relação aos documentos de projeto quanto aos manuais, apresenta custo final maior do que aquele previsto inicialmente, falhas de operação na sua utilização e, principalmente, é entregue bem mais tarde do que a data originalmente prevista (Rook, 1988) e (Cunha e Nakanishi, 1993).

Certamente problemas como estes sejam comuns em todas as áreas tecnológicas, porém na área do software, principalmente no caso de grandes sistemas, estes problemas surgem com maior freqüência e em grandes proporções.

Uma pesquisa feita pelo instituto norte-americano *Standish Group* com 365 companhias espalhadas pelo mundo constatou que 31% dos projetos de informática são cancelados antes de chegar ao fim, e que 53% superam a estimativa inicial dos gastos e de prazo de implantação. Somente 16% chegam ao final no tempo estipulado e sem estourar o orçamento (Militello, 1997).

As principais causas dos problemas encontrados no desenvolvimento de software estão relacionadas a própria complexidade do desenvolvimento e ao seu gerenciamento inadequado, pois trata-se de uma tarefa intrinsecamente complexa devido à sua natureza ampla e multidisciplinar envolvendo vários tópicos, tais como: métodos, ferramentas e técnicas de desenvolvimento, equilíbrio entre custo-qualidade-benefício, características das aplicações, processo iterativo de desenvolvimento, preparação da equipe, etc...

Recorrendo-se novamente a comparação com o desenvolvimento de hardware, que também possui sistemas complexos, a área do software apresenta algumas características peculiares que tornam o seu desenvolvimento ainda mais complexo (Rook, 1988) e (Cunha e

Nakanishi, 1993), pois o software que não tem uma aparência física e não é um elemento palpável só terá sua existência percebida através da sua execução em um computador, acarretando em uma maior dificuldade em integrar suas várias partes, acompanhar o seu progresso e fazer as correções necessárias durante seu desenvolvimento.

Constata-se a facilidade de alteração do produto software, que a princípio parece ser uma propriedade vantajosa mas, no entanto, é um fator preocupante durante o seu desenvolvimento. Devido a facilidade de se alterar um código fonte, há uma grande tendência em se fazer modificações a toda hora e de qualquer forma, até que o código fonte execute a função para o qual foi projetado. Como conseqüência há necessidade de maior controle, pois caso o desenvolvimento de software não seja realizado dentro da melhor linha de programação e controle poderá ocorrer uma dependência muito grande de seus elementos, uns com outros, e isto fará com que a propagação dos efeitos de uma alteração seja explosiva, ou seja, ao se alterar um único elemento de programa, pode haver a necessidade de se alterar outras dezenas de elementos. Se não existir um controle adequado de dependência entre esses elementos, levar-se-á mais tempo para a correção dos mesmos, provocando átrasos na conclusão do software.

Deve-se considerar ainda que uma simples alteração de uma linha de código poderá alterar totalmente a lógica de um programa, gerando resultados inesperados e que a identificação deste problema, além de não poder ser feita automaticamente, poderá tomar muito tempo dos profissionais envolvidos.

Do início das atividades até a obtenção do produto final o desenvolvimento de software passa por várias etapas que geram diferentes representações dos itens ou elementos gerados. Essas representações podem ser documentos descritivos, gráficos, esquemas, etc..., que demonstram as características dos itens do software e como eles se relacionam. Cada representação precisa manter uma consistência com as demais e a criação ou alteração de qualquer elemento do software gera uma alteração em uma ou mais representações. É muito comum os sistemas apresentarem documentação incompleta e inconsistente.

Portanto, é necessário um maior controle do software devido suas características peculiares que fazem com que seu desenvolvimento seja mais complexo que o de hardware.

Outra causa dos problemas no desenvolvimento de software está relacionada com o gerenciamento inadequado do desenvolvimento do software, pois este gerenciamento geralmente

é executado por pessoas inexperientes que desconhecem ou não aplicam adequadamente as metodologias e as técnicas. Os analistas de sistemas muitas vezes assumem a função de gerentes, mas não possuem qualidades gerenciais tais como liderança, capacidade de resolução de conflitos, visão sistêmica do projeto, etc...; e em outras ocasiões as pessoas que assumem a função de gerente possuem algumas qualidades gerenciais mas não têm a experiência necessária na área de software para conduzir corretamente um projeto.

Existe ainda a dificuldade no gerenciamento de software que está relacionada com a abordagem de desenvolvimento utilizada nas primeiras produções de software e que ainda é utilizada no desenvolvimento de sistemas de pequeno porte. Trata-se da abordagem baseada no conceito de que a programação é uma arte e não uma ciência (Cunha e Nakanishi, 1993). Neste tipo de gerenciamento é permitido aos analistas e programadores a total liberdade para criar e alterar seus programas, quando e como quiserem de acordo com a sua criatividade. Ao se tentar restringir essa liberdade com a introdução de um gerenciamento ou de uma padronização de projeto e de programação, alega-se que a criatividade será inibida e haverá prejuízo na produtividade pois os analistas e programadores consideram o controle e a padronização como perda de tempo ou como uma burocracia que nada tem a acrescentar.

Logo, além do problema relativo ao gerenciamento propriamente dito, existe a questão da cultura arraigada na maioria dos analistas e programadores, que rejeitam, ou aceitam a contragosto, qualquer tipo de controle nos seus trabalhos. Neste caso o gerente deverá ter a habilidade adicional para conduzir uma equipe de artistas criativos, procurando guiar o individualismo de cada um na direção de um objetivo comum, procurando fazer com que haja um bom relacionamento entre eles, e convencê-los a aceitar um mínimo de controle para que se possa concluir o software com sucesso.

Nota-se que tanto a tecnologia para desenvolvimento de hardware como a tecnologia para o desenvolvimento de software tem tido significativos avanços. No entanto, no caso do software os desenvolvedores de sistemas e demais usuários não tem acompanhado esse avanço tecnológico, o que acarreta no mau emprego das técnicas, ferramentas e métodos, gerando um produto sujeito a falhas, pouco confiável e de alto custo.

A maioria das pessoas envolvidas no processo de desenvolvimento de software ainda são imaturas ou não estão devidamente capacitadas, ou seja, possuem pouca disciplina, falta de

conhecimento e de compromisso e ainda empregam pouca atenção aos detalhes de continuidade do processo de desenvolvimento. Além disso o seu ambiente de atuação ou a organização onde desempenham suas atividades também ainda não atingiu um nível de maturidade ideal, impondo rotinas de trabalho repletas de falhas e inconsistências *(figura 1.1)*.

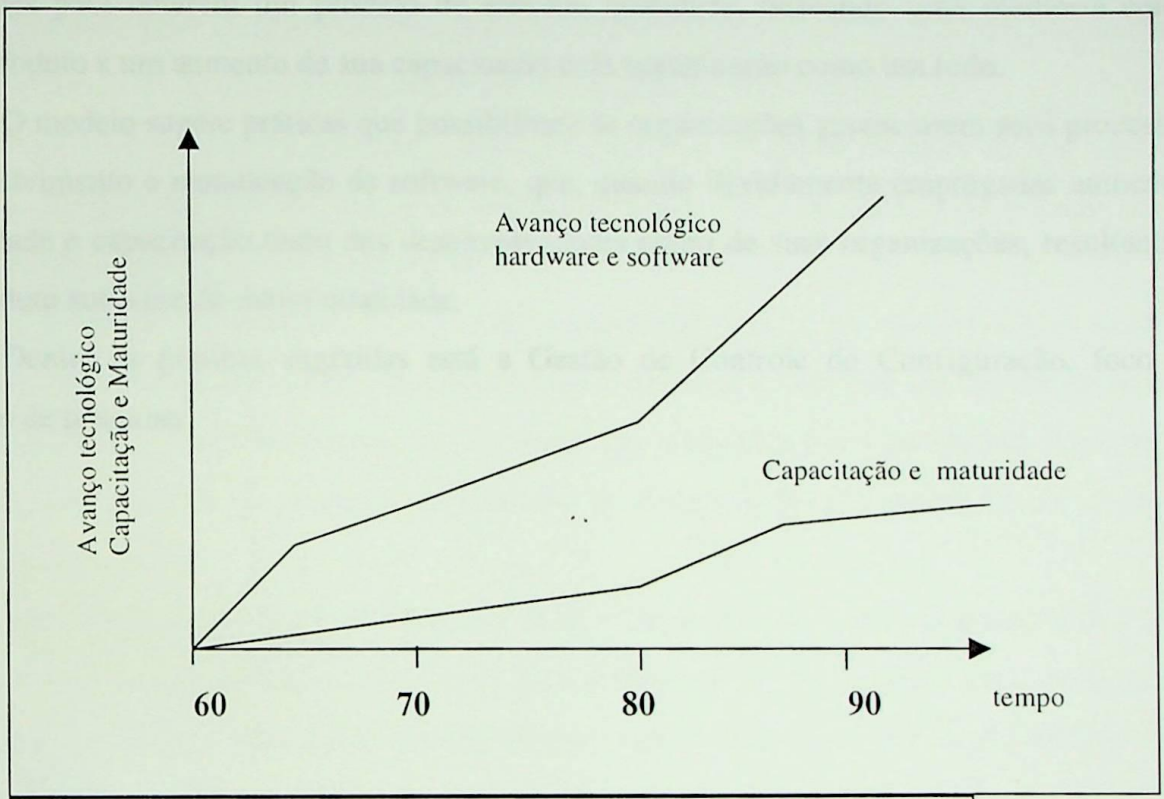


Figura 1.1 – Comparativo entre avanço tecnológico e capacitação e maturidade

Fabricantes e pesquisadores de software têm procurado incutir ou até mesmo impor uma maior disciplina nas atividades técnicas de desenvolvimento, introduzindo formalismos na sua execução, com a finalidade de facilitar o controle de produção de software, de forma a se obter um produto final de maior qualidade (Bersoff, 1985).

As abordagens modernas dentro da Engenharia de Software dão maior ênfase a estruturação do processo de desenvolvimento procurando facilitar o seu gerenciamento e diminuir a complexidade do desenvolvimento de software. Procura-se também incentivar o desenvolvimento e aperfeiçoamento na utilização de ferramentas CASE (Computer-Aided System Engineering – software que auxilia na modelagem de dados e documentação de sistema) e de ambientes de desenvolvimento automatizados. O objetivo principal é compatibilizar a

disciplina com a criatividade, de forma a aumentar a produtividade e a qualidade do software (Rook,1988) e (Cunha e Nakanishi, 1993).

Diante dessa necessidade, pesquisadores da SEI – Instituto de Engenharia de Software da Carnegie Mellon University desenvolveram o CMM – Capability Maturity Model ou Modelo de Capacidade e Maturidade, cujo objetivo é de orientar os desenvolvedores de sistemas em suas atividades para obterem um produto de elevada qualidade, buscando uma melhoria contínua desse produto e um aumento de sua capacitação e da organização como um todo.

O modelo sugere práticas que possibilitam às organizações gerenciarem seus processos de desenvolvimento e manutenção de software, que, quando devidamente empregadas aumentam a maturidade e capacitação tanto dos desenvolvedores como de suas organizações, resultando em um produto software de maior qualidade.

Dentre as práticas sugeridas está a Gestão de Controle de Configuração, foco deste trabalho de pesquisa.

1.2 Objetivo

Como visto, um dos pontos críticos no desenvolvimento de sistemas de software é o de gerenciar e controlar os elementos de software – programas, relatórios, gráficos, modelos, relacionamentos, etc - gerados durante o processo de desenvolvimento bem como suas alterações. Este é o objetivo do SCM – Gerenciamento da Configuração de Software.

O SCM é uma tecnologia que objetiva identificar e definir os itens de um sistema de software, gerenciando as criações e as alterações desses itens durante todo o ciclo de vida, fortemente recomendada pelo modelo CMM.

O CMM é estruturado em níveis de maturidade onde uma organização é avaliada e enquadrada em um desses níveis. Ao cumprir com todas as atividades de um determinado nível a organização muda para o nível subsequente, sendo que a maioria delas encontram-se no primeiro nível ou nível inicial pois possuem um processo de desenvolvimento desorganizado e sujeito à muitas falhas.

Logo no segundo nível encontra-se o SCM - Gerenciamento da Configuração de Software como um conjunto de atividades essenciais para a obtenção de um produto com qualidade. Isto significa que qualquer organização que não esteja no nível inicial deve ter implementado ou estar implementando o SCM.

A dificuldade de se empregar as recomendações do CMM é que o modelo não aborda como implementar o SCM e os outros subsistemas, descrevendo de forma sistemática quais são os objetivos a serem alcançados e seus respectivos desdobramentos, ou seja, sinalizando o quê deve ser feito.

No entanto, ferramentas da gestão da qualidade total podem efetivamente auxiliar na elaboração de um esquema que oriente a implementação do SCM. Nesse esquema seriam descritos os processos e atividades - como deve ser feito.

O QFD – Desdobramento da Função Qualidade, é a ferramenta identificada para descrever esses processos e atividades, pois trata-se de uma metodologia que objetiva principalmente a obtenção de um produto ou serviço de alta qualidade e que possua os requisitos do cliente. O QFD, através de suas matrizes, pode auxiliar na descrição de um esquema para realizar com sucesso o gerenciamento da configuração de software.

Portanto, o objetivo deste trabalho é (figura 1.2):

a) aplicar o QFD em problemas relacionados ao desenvolvimento de software;

b) propor, utilizando QFD, um esquema de implementação do SCM baseado nas recomendações do CMM.

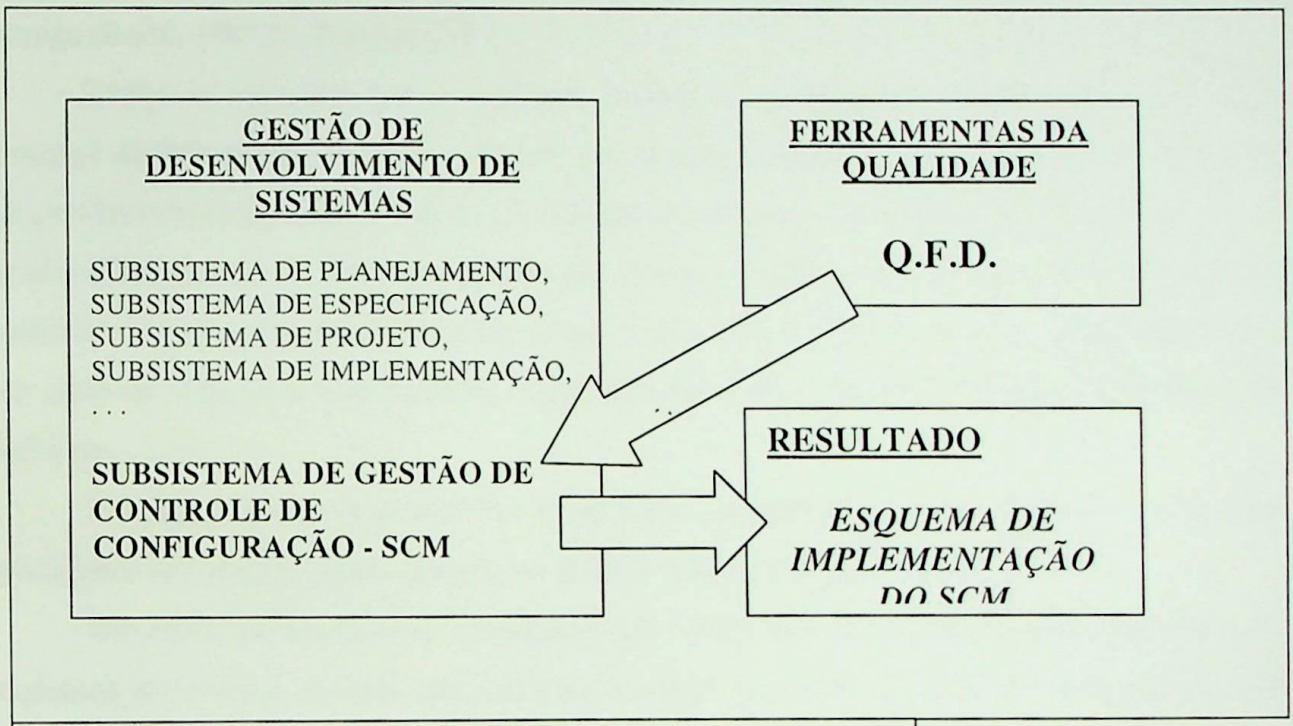


Figura 1.2 - Visão geral do objetivo deste trabalho de pesquisa

1.3 Relevância

Devido a grande competitividade na indústria de software, a qualidade e a produtividade passaram a ser as maiores preocupações no seu desenvolvimento, principalmente, no caso de grandes sistemas. Atualmente os usuários não estão somente levando em consideração a correção e a eficiência dos *softwares*, mas também estão preocupados com a facilidade de uso, a portabilidade, a facilidade de implementar alterações, a reutilização e o cumprimento do prazo de entrega (Rook, 1988) e (Boehm, 1991).

Todos os recursos como técnicas, métodos, ferramentas e ambientes aplicados no processo de desenvolvimento de software têm evoluído muito nestes últimos anos e contribuído para o aumento da qualidade e da produtividade de software. Entretanto, um dos problemas mais desafiadores durante o desenvolvimento de grandes sistemas de software é o de gerenciar e controlar as alterações de todos os elementos e itens de software gerados e mantidos durante o seu ciclo de vida. Este é justamente o objetivo do SCM - Gerenciamento de Configuração de Software.

A documentação dos sistemas e de seus procedimentos deve retratar a realidade da aplicação e sua configuração, além de ser de fácil manutenção (Presmann, 1995).

Isto significa desenvolver aplicativos que atendam o usuário final com total controle dos requisitos do próprio usuário considerando também os requisitos dos desenvolvedores desses aplicativos.

O QFD, conhecido como a voz do cliente, foi escolhido justamente por se tratar de uma técnica que identifica os desejos dos clientes, traduzindo-os em requisitos para o produto e atividades de produção. Proporciona uma visão completa de todo o processo e destaca-se pela interação cliente e produto.

A contribuição é o emprego prático do QFD no desenvolvimento de sistemas, mais especificamente na obtenção de um esquema para implementação do SCM, de forma que permita aos desenvolvedores alcançarem produzirem sistemas de software com a qualidade desejada.

1.4 Estrutura do trabalho

Neste capítulo foram expostos os principais motivos que levaram ao desenvolvimento desta pesquisa e propostos seus objetivos.

Os capítulos 2, 3, 4 e 5 referem-se à revisão bibliográfica, ou seja, a fundamentação teórica que serviu como base para o desenvolvimento deste trabalho.

No capítulo 2 é apresentada a abordagem da gestão de desenvolvimento de sistemas, enfocando-se os paradigmas de desenvolvimento de software clássico, orientado à objeto e ainda a UML – linguagem unificada para representação dos elementos de software.

O CMM – modelo de capacidade e maturidade, seus elementos e conceitos relacionados são exibidos no capítulo 3.

Ainda referente ao desenvolvimento de sistemas é apresentado no capítulo 4 o SCM – Gerenciamento de Configuração de Software.

No capítulo 5 são apresentadas as abordagens da tecnologia do QFD – desdobramento da função qualidade, como a básica de Akao, a de King, Macabe, Carpinetti e a de Cheng.

O capítulo 6 descreve a obtenção de um esquema para o desenvolvimento do SCM elaborado à partir da utilização do QFD.

No capítulo 7 encontram-se as conclusões do trabalho, sua contribuição científica e o destaque de alguns aspectos que não puderam ser avaliados nesta pesquisa e que devem ser levados em consideração, como também, recomendações para trabalhos futuros.

2. GESTÃO DO DESENVOLVIMENTO DE SISTEMAS

2.1 Introdução

Neste capítulo é apresentada a abordagem da gestão de desenvolvimento de sistemas com seus conceitos e fatores de qualidade, enfocando-se os paradigmas de desenvolvimento de software clássico e orientado à objeto. Ainda com relação ao paradigma orientado à objeto é apresentada a UML – Unified Modeling Language ou Linguagem de Modelagem Unificada que é utilizada para visualizar, especificar, construir e documentar os elementos de um sistema complexo de software.

Software é um produto que pode ser entendido como sendo um conjunto de instruções ou programa de computador que quando executadas produzem uma função e um desempenho desejados. Compõem também o software as estruturas de dados que possibilitam que os programas manipulem adequadamente a informação e ainda documentos que descrevem a operação e o uso dos programas (Pressman, 1995).

O software é formado, portanto, pelo conjunto de programas, bases de dados e documentos, tais como manual do usuário, documentação interna e externa dos programas, modelos, diagramas etc... gerados durante o processo de desenvolvimento e manutenção deste software.

Logo, pode-se concluir que a geração de documentos é uma atividade que deve fazer parte do processo de desenvolvimento do software, pois os documentos possibilitam que se tenha informações sobre o sistema e sobre o software disponíveis para uso posterior.

Para uma melhor compreensão dos programas e bases de dados recorre-se aos documentos que são, portanto, muito importantes para a compreensão e entendimento de todas as informações que estão armazenadas e que são manipuladas pelo sistema. Eles possibilitam melhor realização das alterações, sejam correção de defeitos, adaptações ou acréscimos que podem ocorrer após o início de utilização do software.

Um dos grandes desafios para os desenvolvedores de sistemas é justamente elaborar e manter uma documentação do software que seja capaz de atender essas necessidades e que

possua custos compatíveis com a organização, uma vez que as tarefas relacionadas a documentação consomem recursos humanos e físicos, além de demandar em um tempo considerado muito longo pelo usuário do sistema.

Uma prática bastante comum nos ambientes de desenvolvimento de software é a dar pouca importância a documentação, concentrando-se os esforços nos resultados finais com a implementação. Isso ocorre, entre outros fatores, pela pressão que a organização e seus usuários exercem sobre os desenvolvedores para apresentarem rapidamente o produto.

Escola Federal de Engenharia de Itajubá - EFEI
Biblioteca MAUÁ BIM

2.2 Engenharia de Software - Definição

Durante o desenvolvimento de software surgem muitos problemas, de ordem técnica, política e financeira, entre outras. Com uma metodologia pré-definida torna-se mais fácil para a organização encontrar as soluções. Sabe-se que os problemas não desaparecerão tão rapidamente, mas, com técnicas, ferramentas e estratégias adequadas aumentam-se as possibilidades de sucesso.

No entanto, nota-se que cada organização acaba por conceber uma metodologia própria, conforme suas necessidades e recursos disponíveis. Não existe na verdade uma única abordagem que seja a melhor para a solução dos problemas que abalam o desenvolvimento de software. No entanto, ao se combinar metodologias abrangentes para todas as fases de desenvolvimento do software, melhores ferramentas para automatizar essas metodologias, blocos de construção mais poderosos para a implementação do software, melhores técnicas para a garantia da qualidade do software, e uma filosofia de coordenação predominante de controle e administração pode-se conseguir uma disciplina para o desenvolvimento do software. Essa disciplina chama-se Engenharia de Software.

Diferentes meios para a abordagem do problema são propostos pela Engenharia de Software, cabendo aos desenvolvedores escolher aqueles que serão utilizados para o desenvolvimento do seu software, conforme sua situação particular. Essa escolha é até um tanto quanto difícil, pois envolve um conjunto de aspectos tais como o perfil da equipe de desenvolvimento, o tipo de sistema a ser desenvolvido, o rigor do cumprimento do cronograma, o nível de qualidade desejado, as ferramentas disponíveis e as formas de utilização, entre outros.

A Engenharia de software trata, portanto, a construção e a manutenção dos sistemas de software propondo técnicas, metodologias e ferramentas para abordar o processo de desenvolvimento, bem como suas utilizações.

Ela abrange também a organização do ambiente de trabalho, e o controle e gerenciamento de todas as atividades relacionadas ao processo de desenvolvimento, visando sempre a produtividade das tarefas e a qualidade dos produtos (Rook, 1988).

2.3 Metodologias, Técnicas e Ferramentas

Na Engenharia de Software, uma metodologia de desenvolvimento de software consiste basicamente de um conjunto integrado de métodos e procedimentos que devem ser aplicados ao longo do ciclo de vida de desenvolvimento de software, desde a concepção do sistema até a execução final do desenvolvimento (Rook, 1988), (Booch, 1991) e (Jacobson, 1992).

A Engenharia de Software abrange um conjunto de três elementos fundamentais - metodologias, técnicas e ferramentas - que possibilita ao gerente o controle do processo de desenvolvimento do software e oferece ao profissional uma base para a construção de software de qualidade.

As metodologias da Engenharia de Software constituem o elo de ligação que mantém juntas as técnicas e as ferramentas. Elas definem a seqüência em que as técnicas serão aplicadas, os produtos que serão entregues, tais como documentos, relatórios, formulários, etc..., os controles que ajudam a assegurar a qualidade e a coordenar as mudanças, e os marcos de referência que possibilitam aos gerentes de software avaliar o progresso do desenvolvimento (Pressman, 1995).

As técnicas da Engenharia de Software proporcionam os detalhes de *como fazer* para construir o software. Elas envolvem um amplo conjunto de tarefas que incluem planejamento e estimativa do projeto, análise de requisitos de software e de sistemas, projeto da estrutura de dados, arquitetura de programa e algoritmo de processamento, codificação, teste e manutenção.

As técnicas da Engenharia de Software muitas vezes introduzem uma notação gráfica ou orientada à linguagem, além de introduzirem um conjunto de critérios para a qualidade do software.

As ferramentas da Engenharia de Software proporcionam técnicas de apoio automatizado ou semi-automatizado para o processo de desenvolvimento do sistema. Quando as ferramentas são integradas de forma que a informação criada por uma ferramenta possa ser usada por outra, é estabelecido um sistema de suporte ao desenvolvimento de software chamado Engenharia de Software Auxiliada por Computador ou *CASE - Computer-Aided Software Engineering*.

As ferramentas *CASE* colaboram com os desenvolvedores em muitas atividades, e o objetivo principal é compatibilizar a disciplina com a criatividade de forma a aumentar a qualidade e a produtividade do software. (Cunha, 1993) e (Rook, 1988).

2.4 Qualidade do Software

A qualidade do produto é um objetivo de projeto em praticamente todos os ramos de atividades, sendo que, em raras oportunidades a qualidade é obtida de forma inconsciente ou, ainda, após o produto ter sido fabricado.

O software, como qualquer outro produto, deve ser desenvolvido de modo que satisfaça seu usuário, de acordo com um padrão de qualidade previamente estabelecido. A qualidade do software deve ser assegurada durante sua construção, tornando-se quase que impossível obter-se qualidade após o término do desenvolvimento. Nota-se que a qualidade de um software depende muito do *bom usuário* e do *bom desenvolvedor.*

Um software de boa qualidade, de uma forma genérica, pode ser entendido como aquele que produz resultados úteis e confiáveis no momento certo; é de fácil uso, mensurável, possível de ser corrigido, modificável e em constante evolução. Além disso, espera-se que o software opere em máquinas e ambientes reais com economia de recursos e que seja desenvolvido de forma econômica e no prazo estipulado.

Qualidade de software é, pois, um conceito muito mais amplo do que software correto e bem documentado, requerendo para ser conseguida, metodologias e técnicas de desenvolvimento específicas (Staa, 1987).

Fatores de Qualidade do Software

São atributos, condições ou características da construção do software que são determinantes de sua qualidade. Eles avaliam o software a partir de três pontos distintos:

a) operação do produto, através de sua utilização;

b) revisão do produto, quando ocorrem as alterações e

c) transição do produto, quando o software é modificado para funcionar num ambiente diferente.

São eles: (Staa,1987)

a) Utilidade: visa assegurar a confiabilidade e utilidade dos resultados; isto é, verifica se o software está realmente atendendo aos requisitos do usuário, se os resultados que estão sendo gerados estão corretos, e se o programa protege-se contra condições ambientais e de uso adversas.

b) Utilizabilidade: visa assegurar a facilidade de uso do software nas diversas condições de contorno, bem como, segundo os pontos de vista de seus diferentes tipos de usuários. Isto é, mede quanto o software é fácil de operar, instalar, fornecer dados, interpretar resultados, verificar a ocorrência de falhas, treinar novos usuários, etc...

c) Economicidade: visa assegurar a economia no consumo dos diversos recursos, como pessoal, energia elétrica, papel, CPU, memória, etc..., necessários para obter o serviço do produto, bem como para adquirir o software em si.

d) Alterabilidade: visa assegurar a capacidade de adaptação e alteração do software, durante e depois do desenvolvimento.

e) Validabilidade: um produto possui validabilidade na medida em que facilita a certificação, testes, verificação e aceitação completa das alterações incorporadas.

f) Portabilidade: um produto possui portabilidade na medida inversa do esforço necessário para transferi-lo de uma situação ambiental para outra, isto é, mede o quanto o software está preparado para uma mudança de ambiente de execução.

g) Flexibilidade: quando o software está preparado para diferentes formas de uso.

h) Reusabilidade: mede quanto fácil ou difícil é o reuso desse software.

Ainda em termos de qualidade de software, o padrão que se está buscando tem como principais atributos segundo Meyer (1988) e Arthur (1985):

a) Correteza: habilidade de um software executar suas tarefas exatamente como definidas nas especificações dos requisitos.

b) Eficiência: melhor uso da capacidade dos recursos de hardware e de software.

c) Flexibilidade: quantidade mínima de esforço necessário para melhorar o software.

d) Integridade: habilidade do software de proteger seus dados internos.

e) Interoperabilidade: facilidade do software se acoplar em outros softwares.

f) Manutenibilidade: facilidade para localizar e reparar erros no software.

g) Portabilidade: facilidade do software para ser transferido a vários ambientes de hardware ou de software.

h) Confiabilidade: grau de execução sem falhas em condições anormais.

i) Reusabilidade: facilidade de reusar partes do software.

j) Testabilidade: quantidade mínima de esforço para testar a estrutura e a correteza do código.

k) Usabilidade: facilidade de operação do software por parte do usuário.

Garantia de Qualidade do Software

Atualmente, a única forma de se ter uma certa garantia de qualidade do software é através do processo de desenvolvimento, o que significa que é difícil melhorar a qualidade de um software após sua construção.

A garantia da qualidade de software compreende uma variedade de tarefas associadas às atividades de desenvolvimento e manutenção, como:

a) aplicação de métodos técnicos;

b) realização de revisões técnicas formais;

c) atividades de teste de software;

d) controle de mudanças;

e) manutenção de registros e documentos técnicos;

f) aplicação de padrões; e

g) gerenciamento.

Portanto, a qualidade de software inicia-se, na verdade, com o conjunto de metodologias, técnicas e ferramentas que colaboram na obtenção de uma especificação de elevada qualidade.

Após a criação de uma especificação ou de um protótipo de um projeto, devesse avaliá-los quanto à qualidade, onde a *Revisão Técnica Formal* é a atividade central, ou seja, um esforço respaldado por um modelo realizado pelo pessoal técnico com o propósito único de descobrir problemas de qualidade.

Para a realização do *Teste de Software* utilizasse a combinação de uma estratégia de múltiplos passos com uma série de métodos de projeto de casos de testes que ajudam a garantir uma detecção efetiva de erros. Muitos desenvolvedores de software acreditam que a atividade de teste de software pode garantir a qualidade, ou seja, os desenvolvedores pressupõem que uma cuidadosa atividade de teste revela a maioria dos erros, diminuindo a necessidade de outras atividades de controle de qualidade. Porém, o que se tem verificado na prática é que, mesmo que a atividade de testes seja bem realizada não se consegue, na maioria dos casos, abranger todas as classes de erros.

Uma grande fonte de erros e problemas ocorridos no software são as mudanças ou alterações que surgem após o produto ter sido finalizado ou ainda durante o seu desenvolvimento.

A facilidade de se alterar um software parece inicialmente uma vantagem, mas constitui-se na verdade em uma atividade bastante comprometedora dos resultados. Torna-se então extremamente necessária a implantação de um processo de Controle de Mudanças que contribuirá diretamente para a qualidade do software ao se formalizar pedidos de mudança, avaliar a natureza da mudança e controlar o impacto da mudança. Esse controle deve ser aplicado durante a fase de desenvolvimento do software e, posteriormente, durante a fase de manutenção.

A Manutenção dos registros e documentos técnicos para a garantia da qualidade de software deve oferecer procedimentos para a coleta e disseminação de informações de controle de qualidade. Os resultados de revisões, auditorias, controle de mudanças, testes e outras atividades de controle de qualidade devem tornar-se parte do registro histórico de um projeto e devem ser levados ao conhecimento do pessoal de desenvolvimento (Pressman, 1995).

Logo, para obter-se a qualidade de um software é necessário:

a) seguir os fatores de qualidade descritos anteriormente;

b) definir e seguir um processo de desenvolvimento e um processo de manutenção para o software, usando para isso metodologias, técnicas e ferramentas.

c) controlar o processo de desenvolvimento e o processo de manutenção do software, com um gerenciamento que coordene todas as atividades e que permita que elas se interajam da melhor maneira possível.

Para se alcançar uma qualidade muito elevada do software é necessário adicionar a atividade de medição dos diferentes fatores de qualidade. Algumas medidas diretas podem ser utilizadas, tais como o número de erros detectados em um certo período de tempo, a velocidade de execução, o número de linhas de código produzidas, a quantidade de memória utilizada, etc...

Fatores tais como a facilidade de uso, a alterabilidade, a flexibilidade, entre outros só podem ser avaliados indiretamente, sendo que essa avaliação é difícil de ser realizada por envolver aspectos subjetivos.

2.5 Sistema de Software e Domínio do Problema

O domínio do problema consiste em um ambiente onde o sistema de software está relacionado com outros elementos que, de alguma maneira o afeta. Esses elementos podem ser pessoas (usuários, operadores), equipamentos ou *hardware* (*CPU*, memória, equipamentos eletromecânicos), procedimentos (etapas que definem o comportamento e/ou uso dos elementos do domínio), e outros sistemas. Além disso, o domínio do problema poderá conter ou estar contido em outros domínios do problema *(figura 2.1)* (Nakanishi, 1995).

No desenvolvimento de um software é necessário a investigação do domínio do problema desse software e suas *responsabilidades* dentro dele. Essa investigação para identificar as características do domínio do problema nem sempre é trivial, consistindo em um dos maiores problemas encontrados pelos analistas (Coad,1992).

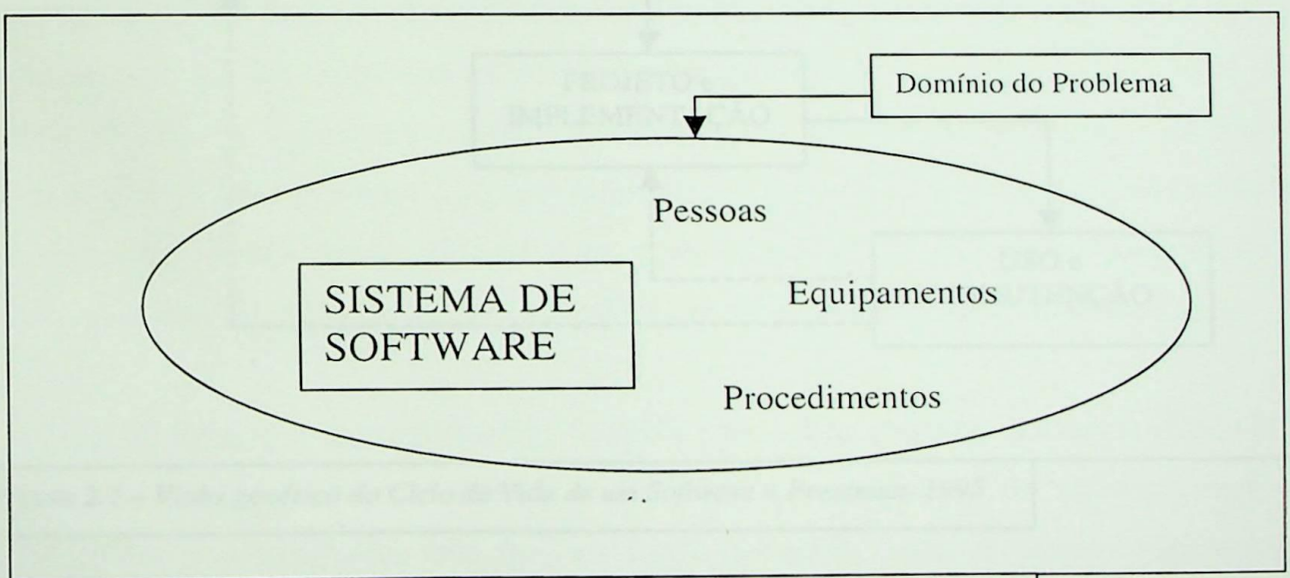


Figura 2.1 – Sistema de Software e Domínio do Problema – Nakanishi, 1995

O conhecimento dos elementos que compõem o domínio do problema colaborará na modelagem do sistema de software, permitindo o melhor relacionamento entre o software e esses elementos.

2.6 Ciclo de Vida do Software

Introdução

O ciclo de vida do software é uma seqüência de fases ou etapas que se iniciam ao conceber-se o software, e terminam ao descontinuá-lo.

Segundo Pressman (1995), de uma maneira genérica, independentemente do paradigma, da área de aplicação, do tamanho e da complexidade do software, o seu ciclo de vida contém três grandes fases: Análise, Projeto e Implementação, e Uso e Manutenção *(figura 2.2)*.

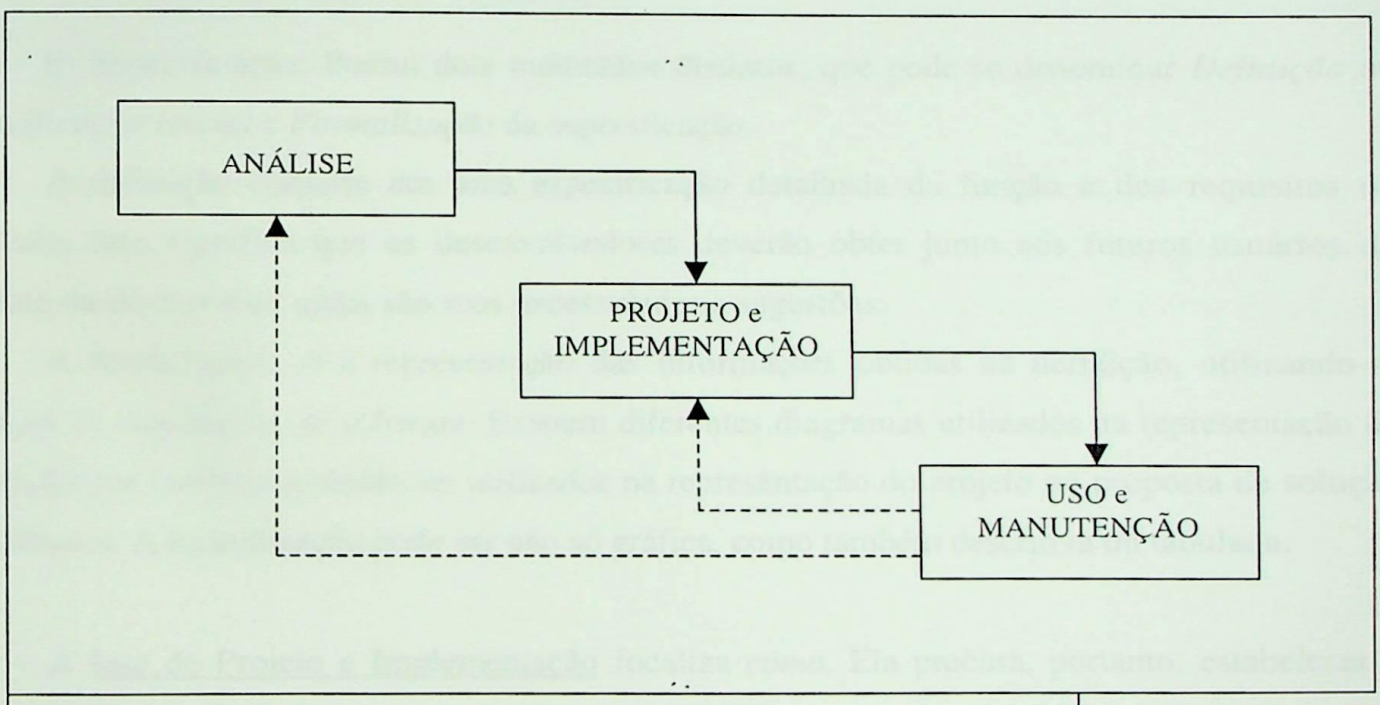


Figura 2.2 – Visão genérica do Ciclo de Vida de um Software – Pressman, 1995

A fase de Análise focaliza *o quê*, procurando identificar os requisitos do sistema e do software, identificando quais informações tem que ser processadas, qual função e desempenho são desejados, quais interfaces devem ser estabelecidas, quais são as restrições de projeto, e quais são os critérios de validação que são exigidos para se definir um sistema bem sucedido. Não importando as técnicas aplicadas durante a fase de especificação que variam conforme a metodologia de engenharia de software adotada, pode-se afirmar que a fase de análise é subdividida da seguinte forma:

a) Estudo de Viabilidade: Consiste na elaboração de um relatório indicador dos possíveis meios de realizar o projeto, definindo custos e benefícios de cada alternativa (Bingham e Garth, 1996). O estudo é feito levando-se em consideração aspectos econômicos, técnicos e legais.

O principal objetivo de um estudo de viabilidade é de demonstrar o retorno do investimento de uma determinada solução proposta. Considera-se diferentes meios para obtenção da solução de um problema, com seus custos e benefícios, observando-se o impacto técnico, político, legal e operacional. Certamente cada alternativa apresenta necessidades distintas e também oferece seus resultados específicos. Esta relação de dispêndios e resultados é comparada com as relações das outras alternativas. A melhor relação, não necessariamente a mais econômica, é a que deve ser adotada.

b) Especificação: Possui dois momentos distintos, que pode-se denominar *Definição* ou *Especificação Inicial* e *Formalização* da especificação.

A definição consiste em uma especificação detalhada da função e dos requisitos do software. Isto significa que os desenvolvedores deverão obter junto aos futuros usuários do produto ou do software quais são suas necessidades e sugestões.

A formalização é a representação das informações obtidas na definição, utilizando-se técnicas de engenharia de software. Existem diferentes diagramas utilizados na representação da definição que também poderão ser utilizados na representação do projeto ou proposta de solução do software. A formalização pode ser não só gráfica, como também descritiva ou tabulada.

A fase de Projeto e Implementação focaliza *como*. Ela procura, portanto, estabelecer a estrutura do software, definindo as estruturas de dados e a arquitetura do software, definindo como os detalhes procedimentais tem que ser implementados, como o projeto será traduzido numa linguagem de programação, e como os testes deverão ser realizados. As técnicas aplicadas durante a fase de projeto podem variar, mas em geral, ela é composta das seguintes partes :

a) Projeto do Software: apresenta os requisitos do software, obtidos na definição e complementados com aspectos técnicos, num conjunto de representações que descrevem a estrutura de dados, a arquitetura, o procedimento algorítmico e as características de interface ou comunicação entre o sistema e os usuários.

b) Codificação: O computador não entende a linguagem humana ou sua representação. Existe a necessidade do estabelecimento de uma linguagem comum, uma vez que o computador entende somente sua linguagem, ou seja, a linguagem de máquina. Podem ser encontradas no mercado diversas linguagens de codificação ou programação de computadores, como Basic, Cobol, C, Fortran, etc..., que, na verdade, são linguagens intermediárias entre o homem e o computador. A codificação consiste portanto, na conversão das representações do projeto em uma linguagem artificial que resulte em instruções que possam ser executadas pelo computador.

c) Testes: Após a codificação, o produto software já pode ser utilizado. Porém é necessário que este software seja testado, pois poderão existir erros ou defeitos de função, de lógica ou de implementação. Os testes devem ser feitos, no primeiro momento, pela equipe de desenvolvedores e depois pelos futuros usuários. São recomendados testes com dados reais.

Finalmente, a fase de Uso e Manutenção refere-se as modificações que podem ocorrer no uso do software após ele ter entrado em funcionamento. Essas alterações podem podem ser para a correção de defeitos, acréscimos exigidos pelo cliente ou pela legislação e ainda adaptações exigidas à medida que o ambiente de software evolui.

2.7 Paradigmas da Engenharia de Software

Dividir para conquistar. As técnicas para desenvolvimento de software baseiam-se neste princípio que nada mais é que dividir problemas complexos em problemas menores para resolvê-los por partes.

Dependendo do tipo de software que se deseja desenvolver pode-se adotar uma das inúmeras técnicas existentes. Essas técnicas podem compor diversas metodologias de desenvolvimento de software, e a maneira como serão empregadas dependem da metodologia escolhida, podendo ser enquadradas dentro do Paradigma Clássico ou dentro do Paradigma da Orientação a Objetos.

Paradigma Clássico

As técnicas enquadradas no Paradigma Clássico podem ser ainda subdivididas em Técnicas Estruturadas e Técnicas não Estruturadas:

a) Técnicas Estruturadas

Nas técnicas estruturadas pode-se aplicar o princípio de dividir para conquistar de forma recursiva, quebrando partes em partes ainda menores *(figura 2.3)* (Nakanishi, 1995).

O desenvolvedor do software ou analista de sistemas deve observar se as partes juntas realmente compõe o problema inicial e se as interfaces entre as partes são simples, não aumentando a complexidade do problema ao invés de reduzi-la.

As técnicas estruturadas visam justamente garantir que as partes decompostas correspondam ao todo e que as interfaces entre as essas partes sejam simples. Dentre as técnicas estruturadas pode-se destacar:

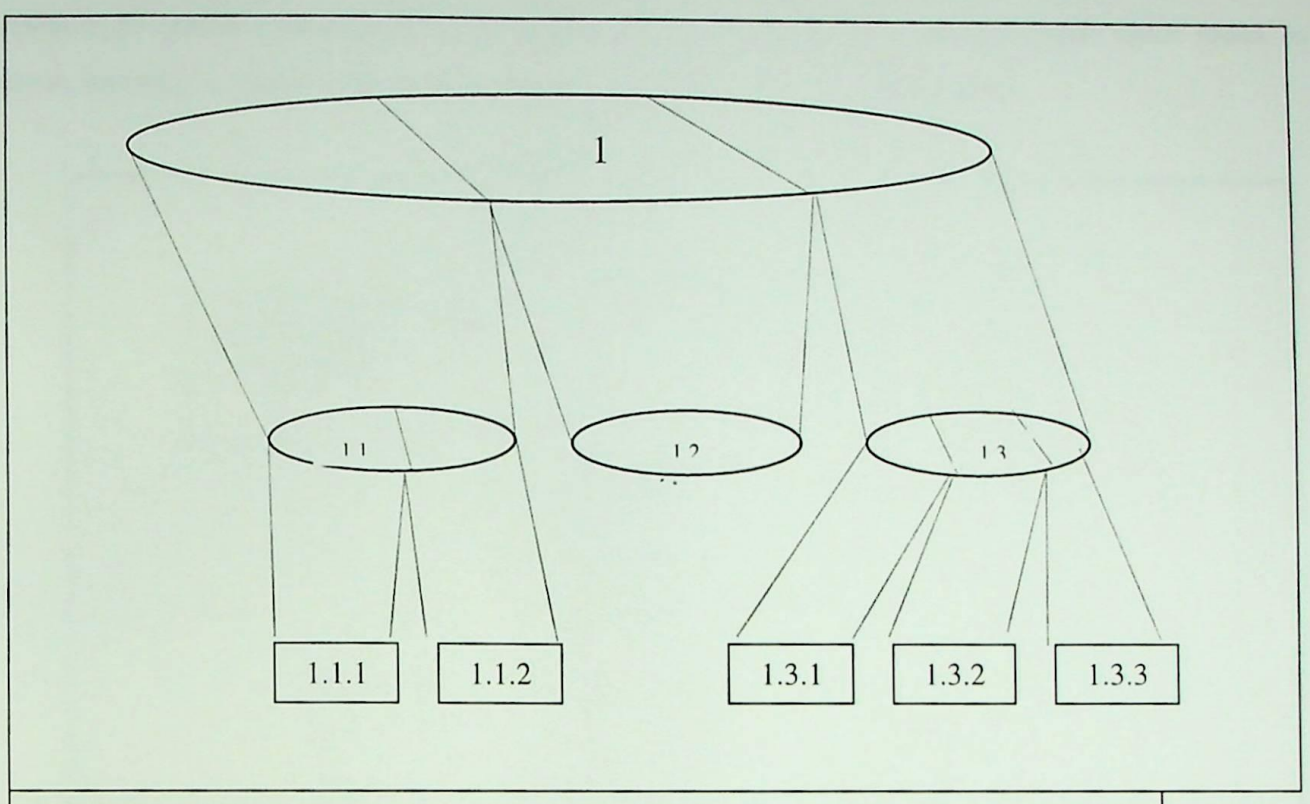


Figura 2.3 - Técnicas Estruturadas e o Princípio de Dividir para Conquistar - Nakanishi, 1995

• Diagrama de Fluxo de Dados (DFD): consiste na principal técnica da análise estruturada que representa as transformações da informação a medida que se movimenta pelo software, descrevendo portanto o fluxo de informação e as transformações que são aplicadas à medida que os dados se movimentam da entrada para a saída. Segundo Gane (1983), a forma básica de um diagrama de fluxo de dados é composta de entidades externas, processos, fluxos de dados e depósito de dados *(figura 2.4).*

A entidade externa corresponde a um produtor ou consumidor de informações que reside fora dos limites do sistema a ser modelado, podendo ser uma pessoa, um grupo de pessoas ou até mesmo uma organização, como por exemplo, o *Cliente* em um sistema de vendas. O processo é um transformador de informações, residente nos limites do sistema a ser modelado, como por exemplo, o processo *Atender Cliente* no mesmo sistema de vendas. O depósitos de dados é uma espécie de repositório de dados que são armazenados para serem usados oportunamente em um ou mais processos, ou seja, as informações são guardadas para serem utilizadas futuramente; exemplo: cadastro de clientes, com nome, endereço, telefone, etc.... e finalmente, o fluxo de

dados que refere-se as informações de entrada e saída das entidades externas, processos ou depósito de dados, consistindo-se no elemento que estabelece a comunicação entre todos os outros; exemplos: numero do pedido, data de emissão, valor do pedido, etc...

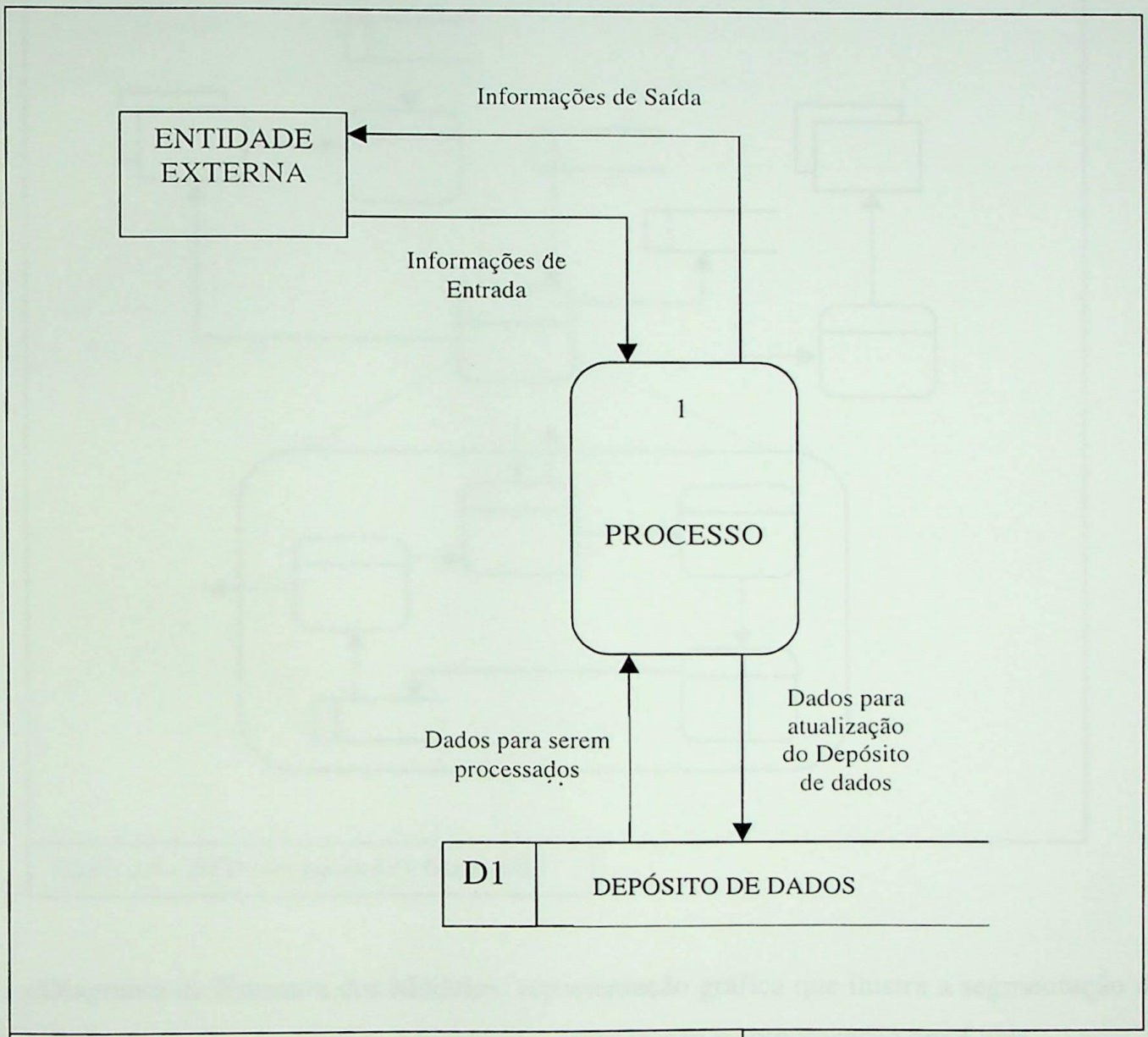


Figura 2.4 – DFD - Diagrama de Fluxo de Dados – Gane, 1983

O DFD pode ser usado para representar um sistema em qualquer nível de abstração, podendo ser dividido em partições de acordo com níveis que representem um crescente detalhamento funcional e do fluxo de informação, ou seja, pode-se efetuar uma expansão. Isto significa que cada *processo* de um DFD denominado DFD pai pode ser expandido para tornar-se

um novo DFD que representa um nível de abstração com maiores detalhes, conhecido como DFD filho *(figura 2.5)*:

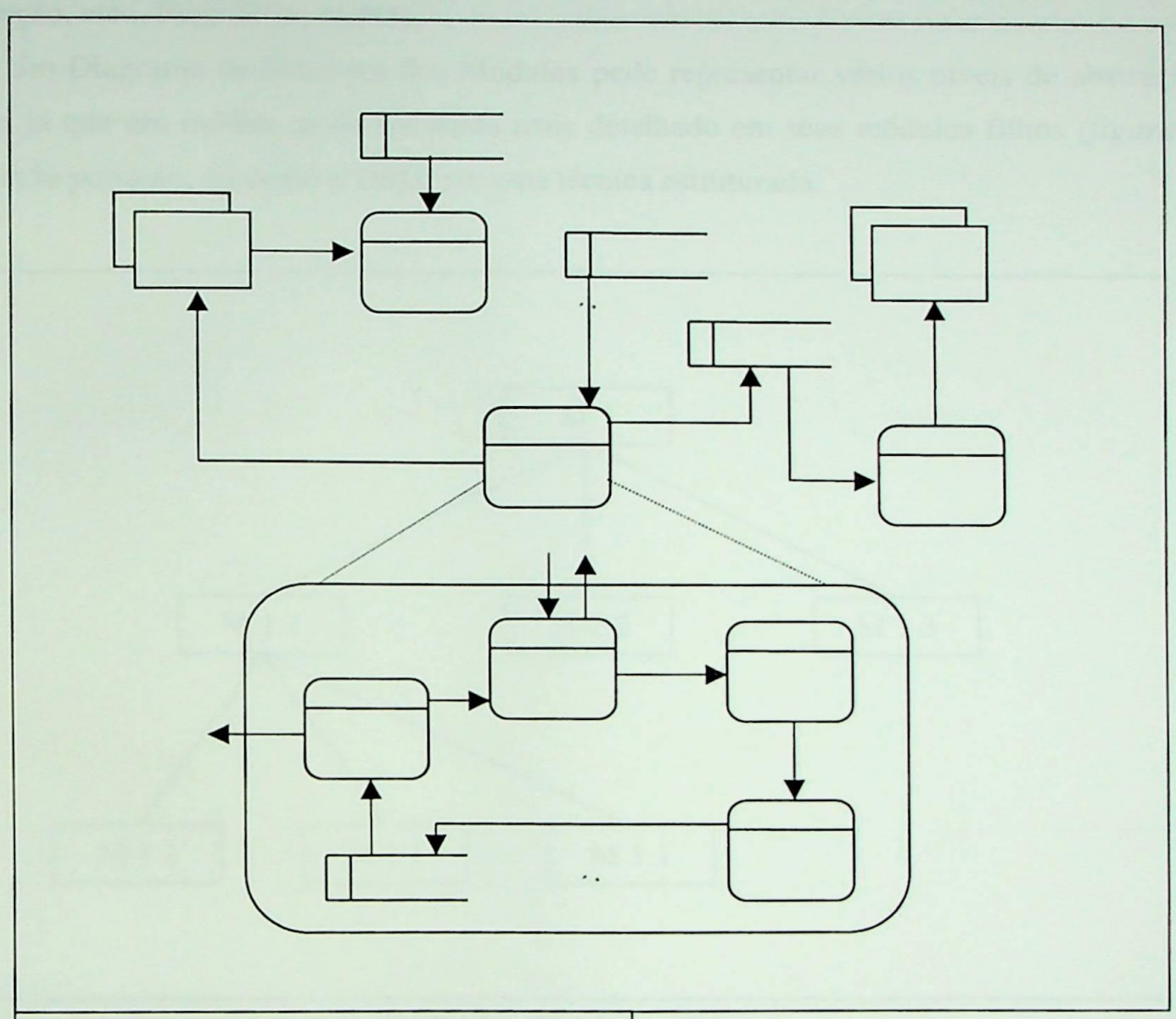


Figura 2.5 – DFD com expansão – Gane, 1983

• Diagrama de Estrutura dos Módulos: representação gráfica que ilustra a segmentação de um sistema em módulos, mostrando a hierarquia, organização e comunicação entre eles.

Entende-se por módulo uma parte do sistema que será transformada em um conjunto de instruções de programa, podendo ser caracterizado por quatro atributos básicos: entrada e saída ou informações que o módulo necessita e fornece; função ou o que módulo faz para a entrada produzir a saída; lógica que são os procedimentos ou algoritmos que executam a função e; dados internos que são dados que somente o módulo referencia na sua área de trabalho.

Um módulo deve ter um nome pelo qual ele é referenciado e pode ser visto de duas ópticas: a visão externa que é composta dos atributos entrada/saída, função e nome e a interna

que é formada pelos atributos lógica e dados internos. Cada módulo será posteriormente transformado em uma unidade de programa, podendo ser um programa principal, uma sub-rotina, uma função, etc... Page-Jones (1988).

Um Diagrama de Estrutura dos Módulos pode representar vários níveis de abstração do sistema, já que um módulo pode ser ainda mais detalhado em seus módulos filhos *(figura 2.6)*, consistindo portanto, tal como o DFD, em uma técnica estruturada.

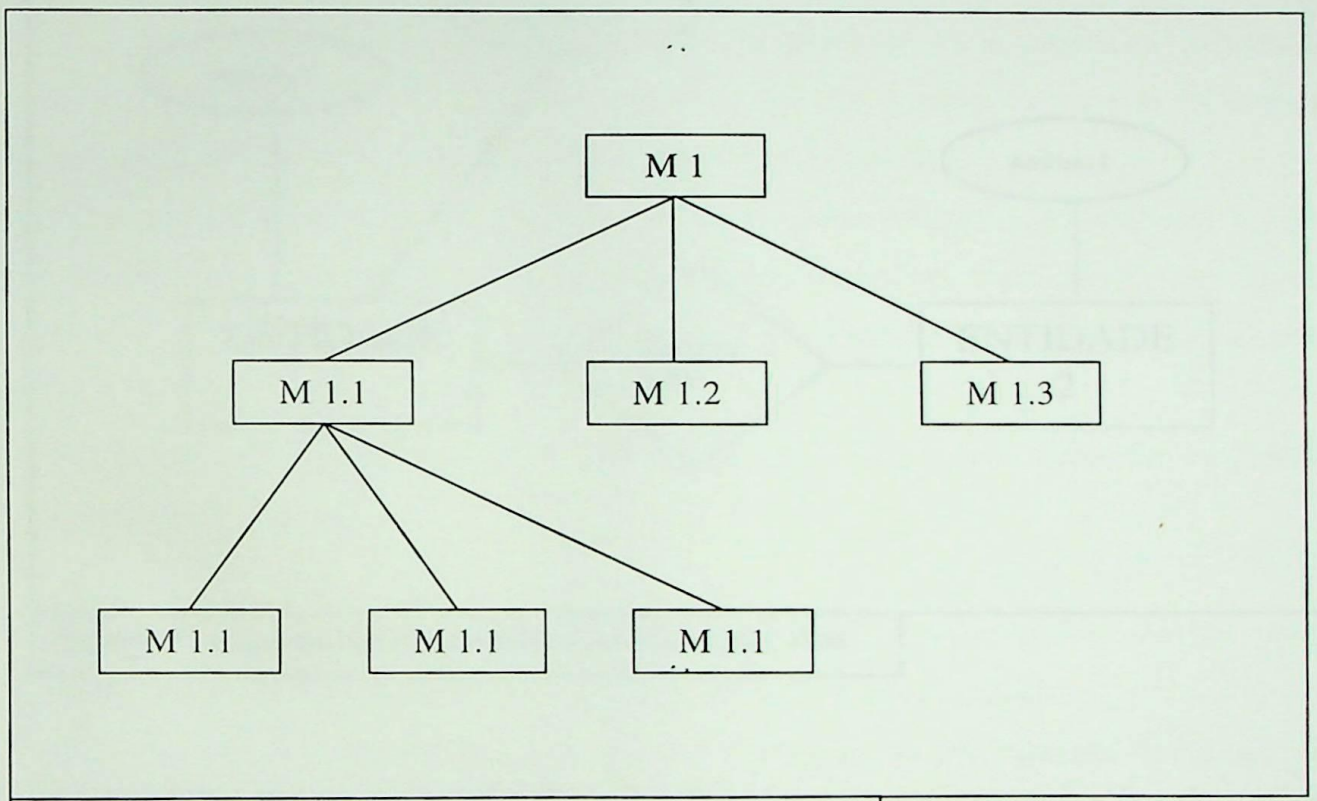


Figura 2.6 - Diagrama de Estrutura dos Módulos - Page-Jones, 1988

b) Técnicas não Estruturadas

As técnicas não estruturadas não utilizam o princípio de dividir para conquistar de maneira recursiva e, portanto não abordam o problema em diferentes níveis de abstração. Elas propõem apenas uma única quebra do problema em partes, o que significa que todos os detalhes são abordados de uma só vez. Dentre as técnicas não estruturadas pode-se citar :

• Diagrama Entidade-Relacionamento (E-R) que consiste em uma técnica de modelagem de dados que considera os dados independentemente dos processos que os transformam.

Consequentemente o Diagrama de Entidade-Relacionamento *(figura 2.7)* concentra-se apenas nos dados, representando uma *rede de dados* de um determinado sistema. O E-R é uma ferramenta gráfica que representa os relacionamentos entre os conjuntos de entidades do sistema (Setzer, 1986).

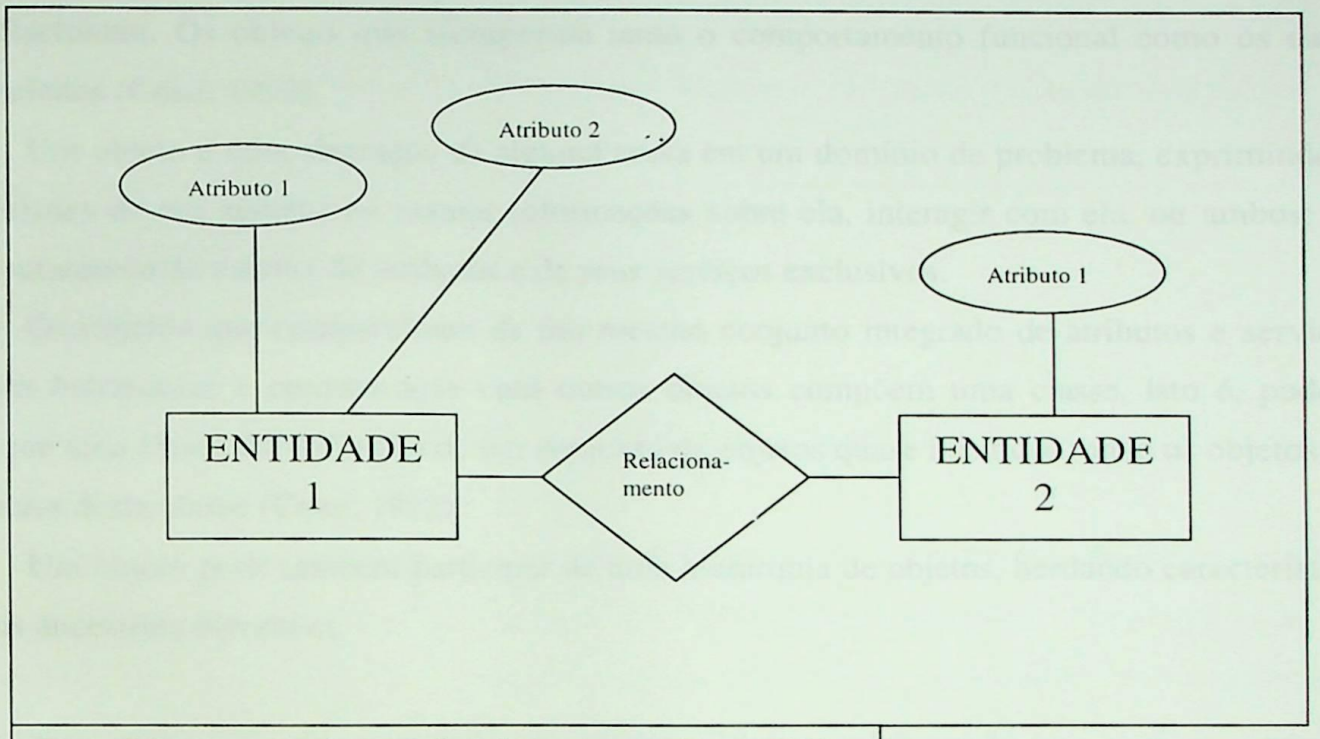


Figura 2.7 - Diagrama de Entidade-Relacionamento - Setzer, 1986

As entidades são representações abstratas de objetos, seres, e organizações do mundo real que precisam ter dados a seu respeito armazenados para que o sistema consiga cumprir seus objetivos. As entidades podem representar também coisas não concretas, como por exemplo *eventos*, desde que para satisfazer os objetivos do sistema deva-se armazenar dados sobre eles.

Os dados armazenados sobre uma entidade são chamados de *atributos* dessa entidade. Como exemplo o *Cliente* em um sistema de vendas e os atributos dessa entidade podem ser nome do cliente, endereço, cep, telefone, etc...

O relacionamento indica as associações entre os elementos de um conjunto de entidades, com elementos de um outro conjunto de entidades. Por exemplo, o cliente emite um pedido de venda; o que relaciona a entidade cliente com a entidade pedido de venda é o fato de cliente emitir o pedido.

Paradigma da Orientação a Objetos

No Paradigma Clássico o software é dividido em funções e dados enquanto que no paradigma da Orientação a Objetos o software constituído de uma coleção de objetos que se interrelacionam. Os objetos que incorporam tanto o comportamento funcional como os dados manipulados (Coad, 1992).

Um objeto é uma abstração de alguma coisa em um domínio de problema, exprimindo as capacidades de um sistema de manter informações sobre ela, interagir com ela, ou ambos; um encapsulamento de valores de atributos e de seus serviços exclusivos.

Os objetos que compartilham de um mesmo conjunto integrado de atributos e serviços, mesmas hierarquias e comunicação com outros objetos compõem uma classe, isto é, pode-se dizer que uma classe é a definição de um conjunto de objetos quase idênticos, onde os objetos são instâncias desta classe (Coad, 1992).

Um objeto pode também participar de uma hierarquia de objetos, herdando características de seus ancestrais (herança).

Um objeto vive em sociedade com outros objetos, colaborando uns com os outros na execução de operações mais complexas, trocando mensagens (comunicação).

A comunicação entre os diversos objetos e classes de um sistema surge da própria necessidade de execução de uma tarefa quando um objeto sozinho não consegue alcançar o objetivo, ou seja, seus serviços são insuficientes para a realização da tarefa. Os objetos armazenam informações sobre si mesmos (seus atributos e serviços), e quando não podem executar um determinado serviço ou necessita de uma outra informação, solicita a outro objeto: *um objeto não pode ou não sabe fazer, mas sabe quem pode ou quem sabe.*

A abordagem orientada para objetos está baseada numa visão do mundo como um sistema de agentes cooperadores. As tarefas em um sistema orientado para objeto são iniciadas por um objeto enviando uma requisição para outro objeto, pedindo a ele que realize uma de suas operações, ou que ele revele algumas de suas informações. A primeira requisição encadeia uma longa e complexa cadeia de requisições, fazendo com que os objetos sejam agentes de seus próprios sistemas (Wirfs-Brock, 1990).

Pode-se portanto, definir um software desenvolvido pelo paradigma *Orientado a Objeto* como um conjunto dos elementos classe, objeto, herança e comunicação (Coad, 1992).

Desta forma:

Orientado para Objetos = Classes e Objetos + Herança + Comunicação

Metodologias e técnicas orientado a objetos

No final dos anos 80 e meados dos anos 90 surgiram conjuntos de metodologias de desenvolvimento de software orientado a objetos.

Cada metodologia propunha sua própria linguagem de representação e processo de desenvolvimento. Em 1995 existiam cerca de 50 metodologias propostas. Apesar de toda esta diversidade, todas representavam princípios e conceitos comuns que caracterizam a tecnologia orientada a objetos.

A tecnologia orientada a objetos está fundamentada em características como :

a) Encapsulamento: é uma maneira de ocultar as informações. O conhecimento ou inteligência encapsulado em um objeto possibilita a separação dos seus aspectos externos dos detalhes internos da implementação daquele objeto. Os aspectos externos são acessíveis por outros objetos, enquanto os detalhes internos ficam então ocultos dos demais objetos.

Pode-se afirmar que o objeto têm um lado particular, ou seja, somente ele é quem sabe como realizar operações, sendo que *como* ele realiza tais operações não diz respeito a outras partes do sistema. Logo, os objetos são livres para mudar seus lados particulares sem afetar o resto do sistema (Wirfs-Brock, 1990). Um objeto encapsula inteligência, que é representada no software por dados e processos (Cohen, 1993).

O encapsulamento permite então que o modelo seja constituído de objetos que são vistos de fora e de maneira que cada um seja entendido em termos de seu procedimento intrínseco, representado pelos serviços que pode executar *(figura 2.8)* (Cohen, 1993).

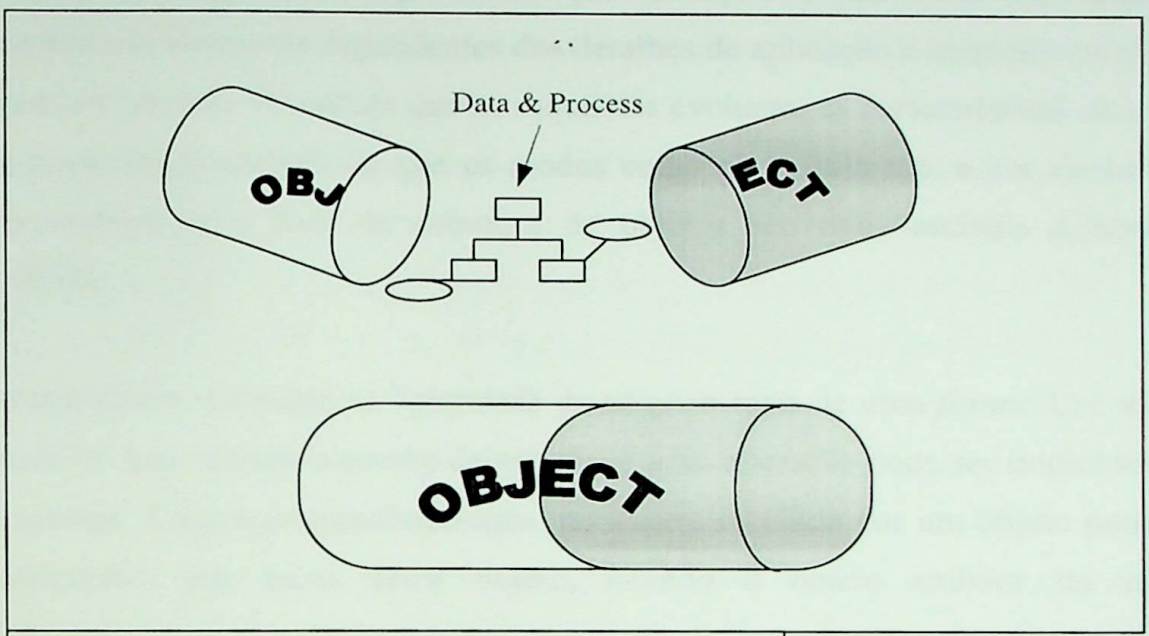


Figura 2.8 – Um objeto encapsula inteligência - Cohen, 1993

b) Abstração: é o fato de se concentrar no que um objeto é e faz, antes de decidir como ele é implementado. O uso da abstração preserva a liberdade de se tomar decisões, evitando tanto quanto possível, comprometimentos prematuros com detalhes (Rumbaugh, 1994).

c) Compartilhamento: dirige a criação de estruturas de herança onde dados e serviços que são compartilhados por um grupo de classes são descritos apenas uma vez, numa classe que é um antecessor comum para cada classe no grupo.

As técnicas orientada a objetos promovem o compartilhamento em diversos níveis. A herança da estrutura de dados e do seu comportamento permite que a estrutura comum seja compartilhada por diversas subclasses semelhantes sem redundâncias. O compartilhamento de código com utilização de herança também é possível, constituindo-se em uma das principais vantagens das linguagens baseadas em objetos. Além disso, o desenvolvimento baseado em

objetos oferece a possibilidade de reutilização de modelos e códigos em projetos futuros (Rumbaugh, 1994).

d) Ênfase na Estrutura de Objetos e Não na Estrutura de Procedimentos. A tecnologia orientada a objetos preocupa-se em especificar o que um objeto é, e não como ele é utilizado. Os usos de um objeto são altamente dependentes dos detalhes de aplicação e freqüentemente mudam durante o desenvolvimento. À medida que os requisitos evoluem, as características de um objeto permanecem muito mais estáveis do que os modos como ele é utilizado, e por causa disso, os softwares construídos com base na estrutura de objetos são mais estáveis a longo prazo (Rumbaugh,1994).

e) Polimorfismo: Consiste na habilidade de adquirir mais de uma *forma*. Um atributo de um objeto pode ter mais de um conjunto de valores, e uma operação pode ser implementada por mais de um serviço. Com o polimorfismo uma mensagem recebida por um objeto pode resultar em ações diferentes por parte desse objeto, ficando o objeto emissor da mensagem despreocupado com detalhes de implementação do receptor.

Um exemplo comum de polimorfismo vem da computação gráfica, onde operações com desenhos diferentes podem ser implementados por um serviço denominado *Desenhar*. Esse serviço poderá desenhar por exemplo um arco (Desenhar_Arco) ou um retângulo (Desenhar_Retângulo) de acordo com a necessidade, sendo acionado em tempo de execução do programa (Kung, 1995).

f) Incorporação Dinâmica: O serviço que implementa uma operação só é conhecido em tempo de execução do programa (Run Time). Esse mecanismo possibilita a implementação do polimorfismo.

Portanto, como o conceito de objeto engloba modularidade e encapsulamento, e as relações de herança facilitam as alterações e reutilizações, pode-se gerar softwares mais flexíveis para se alterar, valorizando a reutilização, a modularidade e o encapsulamento e fazendo com que o degrau entre análise, projeto e implementação seja bem mais suave do que nas metodologias do Paradigma Clássico (Coad, 1992).

2.8 A UML – Unified Modeling Language
Linguagem de Modelagem Unificada

Consiste em uma linguagem para especificar, construir, visualizar e documentar Sistemas de Software (Rational, 1997). A UML é uma fusão dos conceitos de Booch (Booch, 1994); da OMT - Object Modeling Technique (Rumbaugh, 1991); da OOSE - Object Oriented Software Engineering (Jacobson, 1992), e de outros métodos e combina ainda algumas das melhores idéias da Engenharia de Software, teoria de bancos de dados e projeto de Sistemas em uma linguagem de modelagem independente de qualquer linguagem de programação (Whitlock, 1999).

A UML é uma linguagem padrão, adotada pelo OMG – Object Management Group e está tendo uma rápida aceitação na indústria de software. Essencialmente, ela define um vocabulário para representar todos os artefatos produzidos no desenvolvimento de software em todo o seu ciclo de vida, desde a análise até a implantação.

Este vocabulário consiste de itens estruturais, comportamentais, de agrupamentos e anotacionais, relacionamentos entre esses itens e diagramas.

Desta forma a OMG padronizou a linguagem de representação a ser usada no desenvolvimento de software, e deixou livre a adoção do processo que deve ser seguido para se desenvolver Sistemas de Software (Fowler, 1999).

Não importa como o software é desenvolvido; o que deve ser produzido e a representação do produto está padronizada.

Os diagramas utilizados pela UML são compostos de 9 tipos:

a) Diagrama de Casos de Uso
b) Diagrama de Classes
c) Diagrama de Objetos
d) Diagrama de Estado
e) Diagrama de Atividade
f) Diagramas de Interação:
    Diagrama de Seqüência
    Diagrama de Colaboração

g) Diagramas de Implementação:

Diagrama de Componentes

Diagrama de Execução.

Cada um desses diagramas é descrito adiante.

Sendo uma linguagem, a UML tem uma sintaxe e uma semântica bem definidas. A parte mais visível da sintaxe da UML é sua notação gráfica.

Diagrama de Casos de Uso

Um caso de uso *(figura 2.9)* representa uma maneira específica em que um ator – pessoa, organização, equipamento, dispositivo ou outro sistema – interage com o sistema, fornecendo ou consumindo informações (agentes externos) ou operando o sistema (usuários do sistema). Por exemplo, para um sistema de Vendas, os seguintes casos de uso, entre outros, podem ser considerados:

Cadastrar cliente,

Cadastrar proposta e

Registrar pedido.

Este diagrama ajuda a definir o que existe externamente ao sistema (atores) e o que deve ser executado pelo sistema (casos de uso). O conjunto de casos de uso determina a completa funcionalidade do sistema. Desta forma o diagrama de caso de uso captura o comportamento pretendido do sistema do ponto de vista de quem interage com ele.

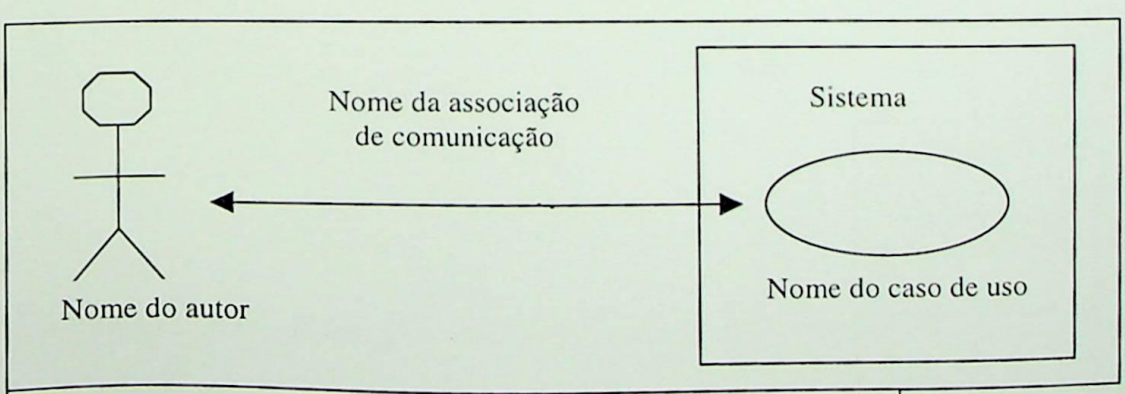


Figura 2.9 - Notação para o Diagrama de Casos de Uso - Whitlock, 1999

Conforme demonstrado na *figura 2.9,* no Diagrama de Casos de Uso o sistema é representado como uma caixa e os atores são representados por uma pessoa fora da caixa. Os casos de uso são representados como elipses dentro da caixa, pois fazem parte do sistema.

Cada caso de uso deve ser descrito detalhadamente. Os casos de uso dirigem todo o restante do desenvolvimento e são a base para a modelagem orientada a objetos.

Diagrama de Classes

O Diagrama de Classes *(figura 2.10)* representa a estrutura estática do sistema, exibindo as suas classes, suas estruturas internas, correspondentes aos atributos e serviços, seus relacionamentos, correspondentes às conexões todo-parte e associações, e suas estruturas de generalização-especialização.

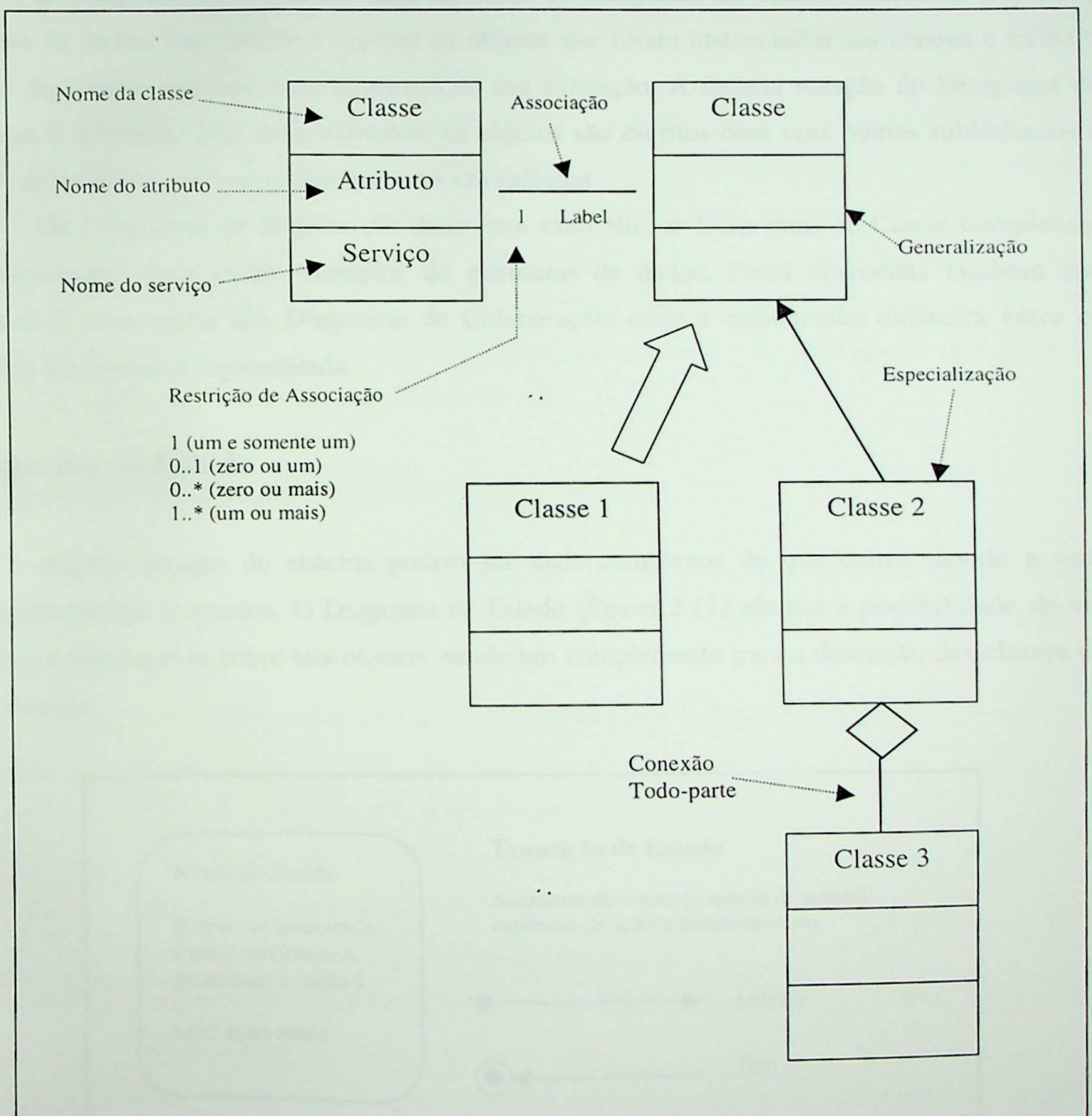


Figura 2.10 - Notação simplificada para o Diagrama de Classes - Whitlock, 1999

Geralmente um sistema possui alguns Diagramas de Classes, que representam a sociedade de objetos que participam da realização dos casos de uso.

Diagrama de Objetos

O Diagrama de Objetos é uma instância do Diagrama de Classes, incluindo objetos e valores de dados. Sua função é mostrar os objetos que foram instanciados das classes e exibe o perfil do sistema em um certo momento de sua execução. A mesma notação do Diagrama de Classes é utilizada, com duas exceções: os objetos são escritos com seus nomes sublinhados e todas as instâncias em um relacionamento são exibidas.

Os Diagramas de Objetos são úteis para exemplificar Diagramas de Classe complexos, principalmente para exibir exemplos de estruturas de dados. Esses diagramas também são utilizados como parte dos Diagramas de Colaboração, onde a colaboração dinâmica entre os objetos do sistema é representada.

Diagrama de Estado

Alguns objetos do sistema podem ser mais complexos do que outros devido a seus comportamentos e estados. O Diagrama de Estado *(figura 2.11)* oferece a possibilidade de um melhor entendimento sobre tais objetos, sendo um complemento para a descrição das classes de um sistema.

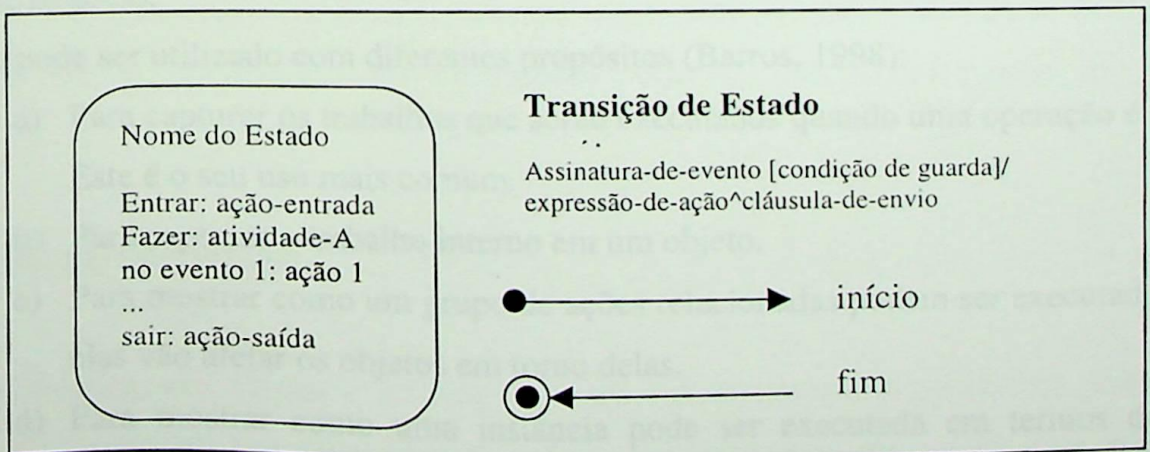


Figura 2.11 - Notação para o Diagrama de Estados - Whitlock, 1999

Ele exibe todos os estados possíveis - representados através dos valores de atributo do objeto - que um objeto de uma certa classe pode se encontrar, e quais os eventos do sistema provocam tais mudanças.

Um estado corresponde a uma certa condição durante a vida de um objeto, durante a qual ele satisfaz alguma condição, realiza alguma ação ou espera por algum evento. Um evento consiste em alguma mensagem recebida, ou condições a serem satisfeitas.

Este diagrama não é descrito para todas as classes de um Sistema, mas apenas para aquelas que possuem um número definido de estados conhecidos, e onde o comportamento das classes é afetado e modificado pelos diferentes estados.

Os Diagramas de Estado possuem um ponto de início e vários pontos de finalização. Um ponto de início (estado inicial) é representado por um círculo todo preenchido, e um ponto de finalização (estado final) é representado por um círculo em volta de um outro círculo menor preenchido. Os estados são representados por retângulos com cantos arredondados. As setas representam as transições, progressões que ocorrem de um estado para outro, que são representadas pela chamada de um serviço de uma classe. As setas são rotuladas com o nome do evento que causa a transição de um estado para outro.

Diagrama de Atividade

O Diagrama de Atividade é uma variação do Diagrama de Estados, no qual os estados são atividades representando a execução de operações, e as transições são disparadas após a conclusão das operações.

O Diagrama de Atividade é utilizado para documentar a lógica de uma operação ou de um processo de negócio.

Ele pode ser utilizado com diferentes propósitos (Barros, 1998):

a) Para capturar os trabalhos que serão executados quando uma operação é disparada. Este é o seu uso mais comum.
b) Para capturar o trabalho interno em um objeto.
c) Para mostrar como um grupo de ações relacionadas podem ser executadas, e como elas vão afetar os objetos em torno delas.
d) Para mostrar como uma instância pode ser executada em termos de ações e objetos.

e) Para mostrar como um negócio funciona em termos de trabalhadores (atores), fluxos de trabalho, organização e objetos (fatores físicos e intelectuais usados no negócio).

O Diagrama de Atividade *(figura 2.12)* utiliza *swimlanes* para organizar responsabilidades para uma atividade pertencente a uma classe. *Swimlanes* geralmente correspondem a unidades organizacionais em um modelo de negócios, como por exemplo, seções ou departamentos.

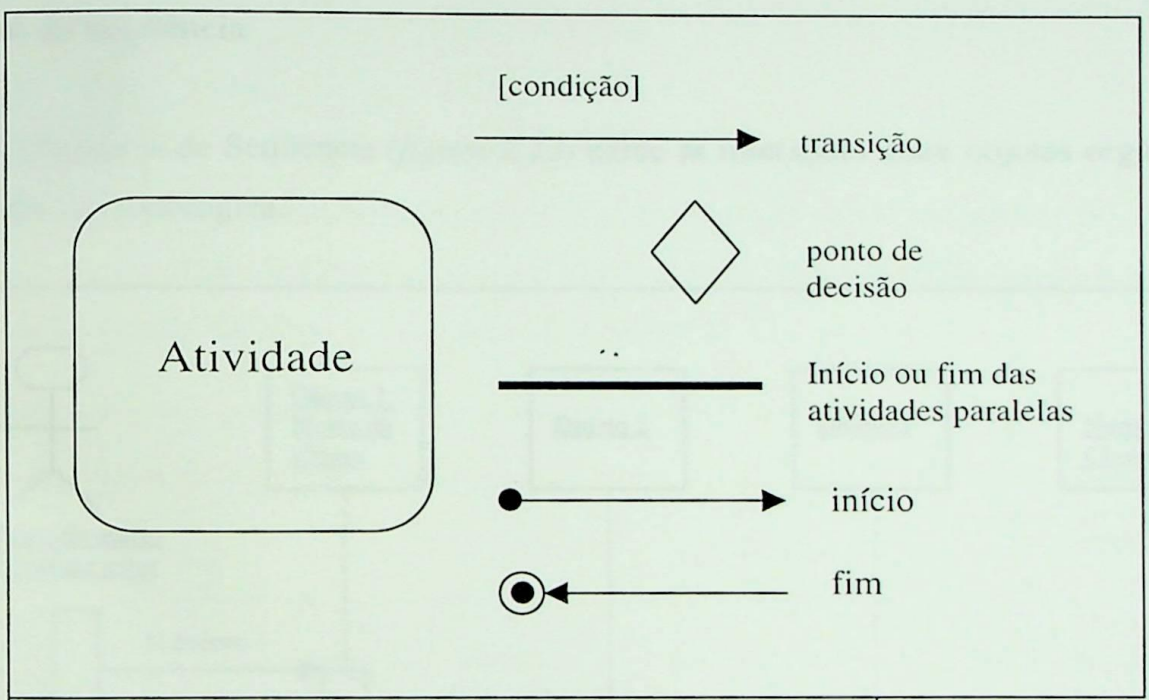


Figura 2.12 - Notação para o Diagrama de Atividades - Whitlock, 1999

Este diagrama consiste em uma maneira alternativa de se mostrar interações, com a possibilidade de expressar como as ações são executadas, o que elas fazem (mudanças dos estados dos objetos), quando elas são executadas (seqüência das ações), e onde elas acontecem (*swimlanes*).

Diagramas de Interação

Os Diagramas de Interação exibem os padrões de interações entre objetos de um sistema. Uma interação consiste em uma especificação comportamental que inclui uma seqüência de

objetos dentro de um contexto para realizar um propósito específico, tal como a realização de um caso de uso (Furlan, 1998).

A UML propõe dois tipos de Diagramas de Interação:

Diagrama de Seqüência e

Diagrama de Colaboração.

Estes diagramas representam as mesmas informações; porém, cada um enfatiza um aspecto particular da informação.

Diagrama de Seqüência

O Diagrama de Seqüência *(figura 2.13)* exibe as interações entre objetos organizadas em uma seqüência cronológica.

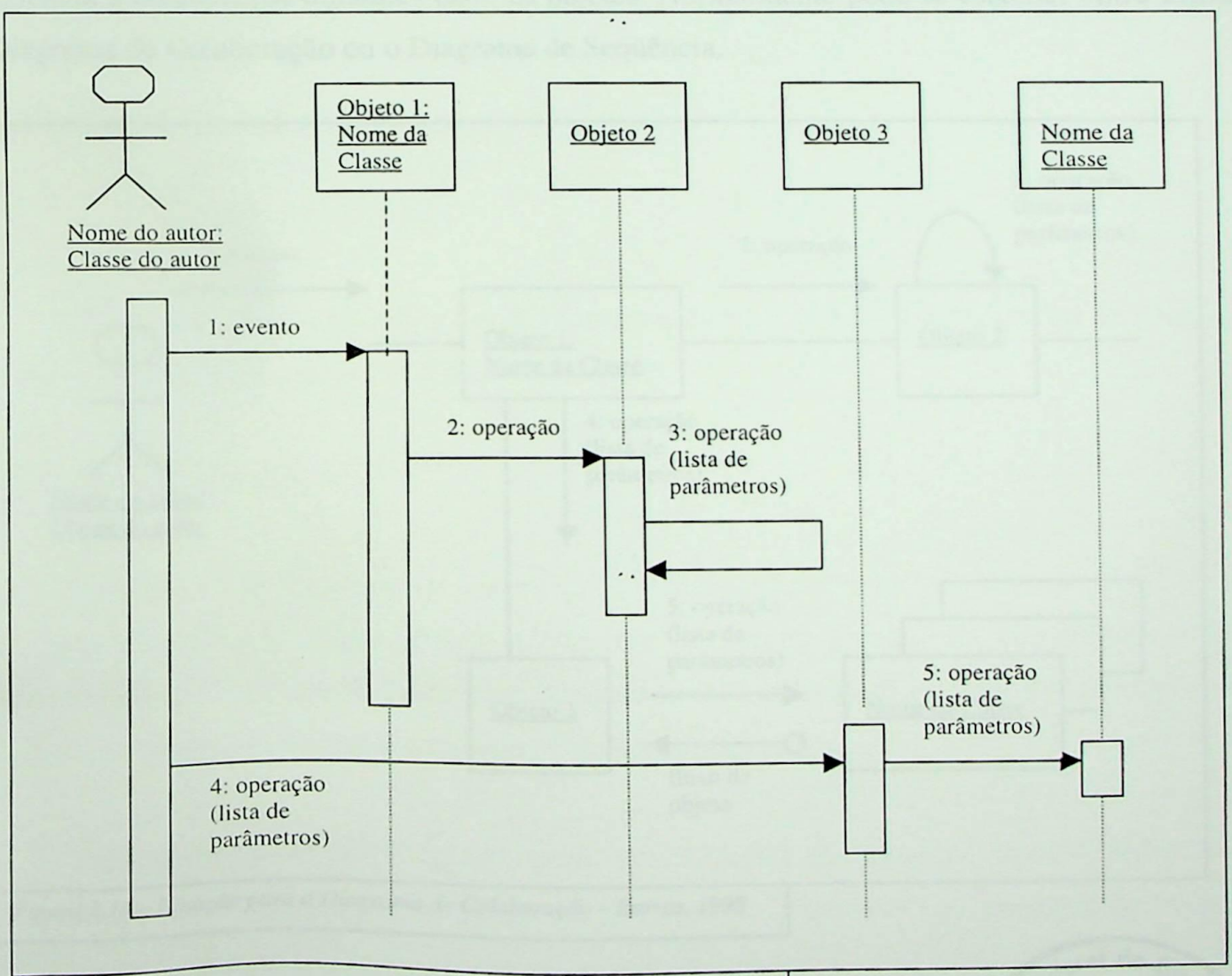


Figura 2.13 – Notação para o Diagrama de Seqüência - Furlan, 1998

Este diagrama é utilizado para documentar e verificar a lógica contida em casos de uso, mostrando a colaboração dinâmica entre os vários objetos envolvidos em um caso de uso, as mensagens que eles enviam entre si, e qualquer valor de retorno associado à mensagem. Permite que se perceba a seqüência de mensagens enviadas entre os objetos.

O Diagrama de Seqüência consiste em um número de objetos mostrados em linhas verticais. O decorrer do tempo é visualizado observando-se o diagrama no sentido vertical, de cima para baixo. As mensagens enviadas por cada objeto são simbolizadas por setas entre os objetos que se relacionam (Barros, 1998).

Diagrama de Colaboração

O Diagrama de Colaboração *(figura 2.14)* mostra de maneira semelhante ao Diagrama de Seqüência a colaboração dinâmica entre os objetos. Normalmente pode-se escolher entre utilizar o Diagrama de Colaboração ou o Diagrama de Seqüência.

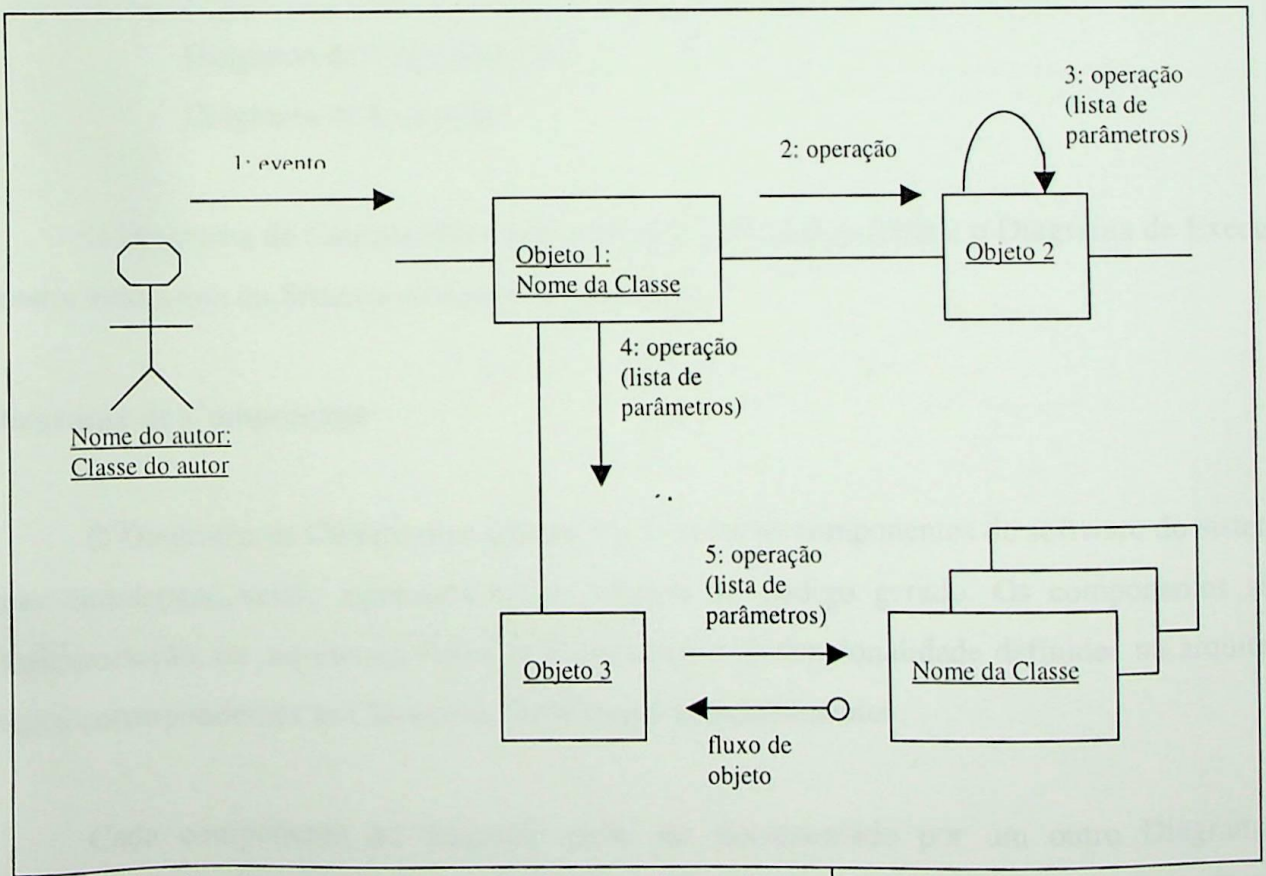


Figura 2.14 – Notação para o Diagrama de Colaboração – Barros, 1998

Escola Federal de Engenharia de Itajubá - EFEI
Biblioteca MAUÁ BIM

Diferentemente do Diagrama de Seqüência, o Diagrama de Colaboração exibe as interações entre os objetos, mostrando a troca de mensagens entre eles, bem como seus relacionamentos. Por outro lado, o Diagrama de Colaboração não exibe o tempo como uma dimensão separada; a seqüência das mensagens e processos (*threads*) concorrentes devem ser determinados utilizando números em seqüência.

Portanto, se a ênfase da modelagem for o decorrer do tempo, é aconselhável utilizar o Diagrama de Seqüência; porém, se a ênfase for o contexto do Sistema, é melhor dar prioridade ao Diagrama de Colaboração.

Diagramas de Implementação

Os Diagramas de Implementação utilizados pela UML exibem os aspectos de implementação física do sistema, incluindo as estruturas do código fonte e de sua implementação em tempo de execução. Estes diagramas possuem duas formas:

Diagrama de Componente e
Diagrama de Execução.

O Diagrama de Componente exibe a estrutura do código fonte e o Diagrama de Execução mostra a estrutura do Sistema em tempo de execução.

Diagrama de Componente

O Diagrama de Componente *(figura 2.15)* exibe os componentes de software do sistema e suas interdependências, representando a estrutura do código gerado. Os componentes são a implementação, na arquitetura física, dos conceitos e da funcionalidade definidos na arquitetura lógica correspondentes às Classes&objetos e seus relacionamentos.

Cada componente do diagrama pode ser documentado por um outro Diagrama de componente mais detalhado, por um Diagrama de Casos de Uso ou por um Diagrama de Classes.

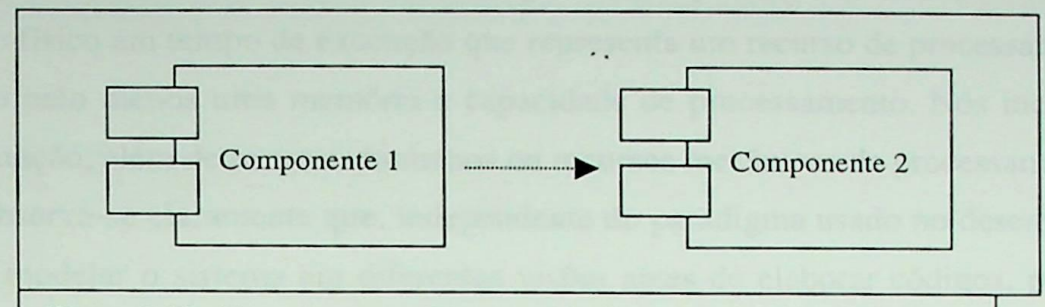


Figura 2.15 - Notação para o Diagrama de Componentes - Whitlock, 1999

Diagrama de Execução

O Diagrama de Execução *(figura 2.16)* representa a última descrição física da topologia do Sistema, mostrando a arquitetura física do hardware e do software que executam em cada unidade. Ele exibe a configuração dos elementos de processamento em tempo de execução e os componentes de software, processos e objetos que residem nestes processadores.

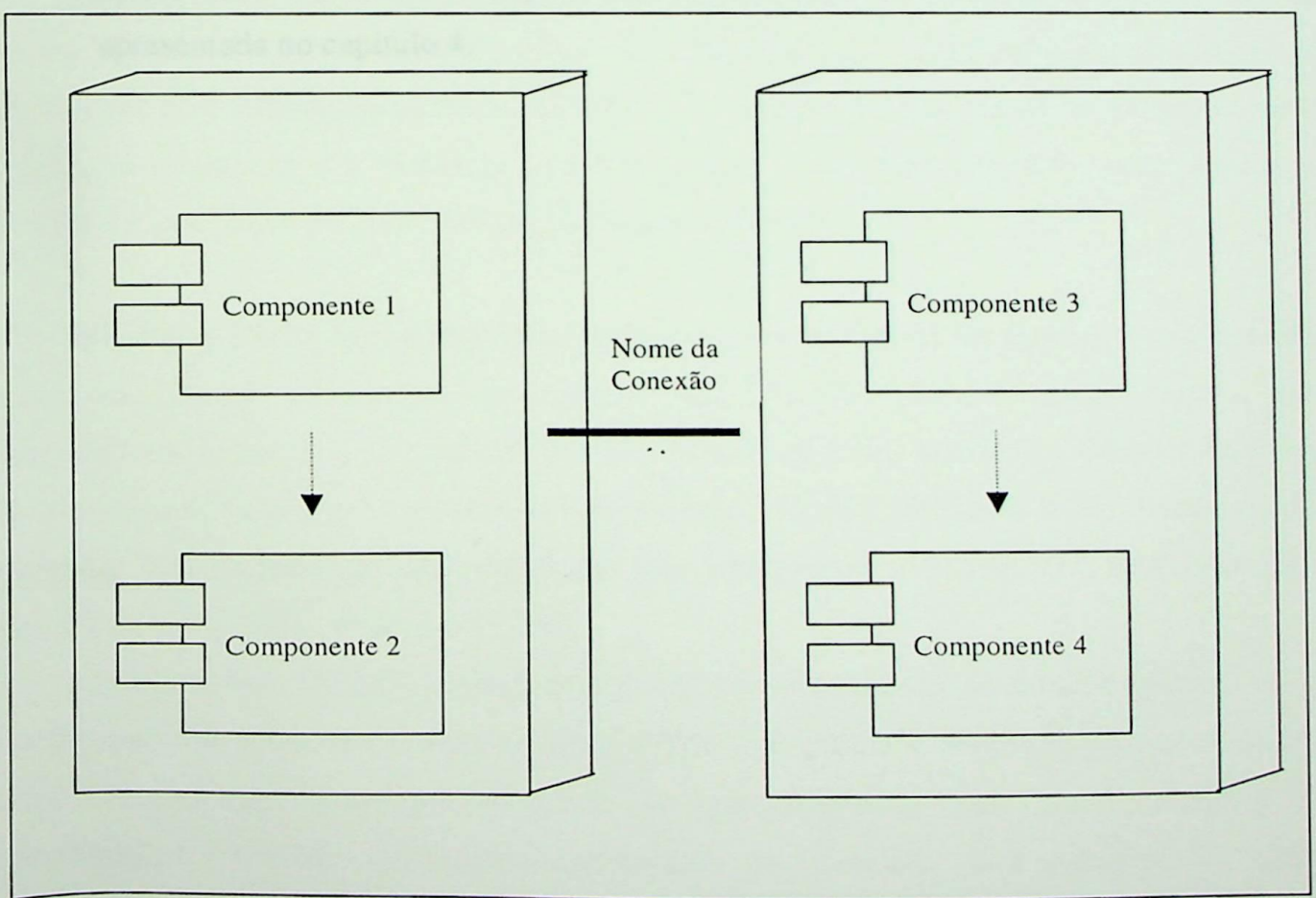


Figura 2.16 - Notação para o Diagrama de Execução - Whitlock, 1999

Este diagrama exibe nós conectados por associações de comunicação. Um nó consiste em um objeto físico em tempo de execução que representa um recurso de processamento, geralmente possuindo pelo menos uma memória e capacidade de processamento. Nós incluem dispositivos de computação, além de recursos humanos ou recursos mecânicos de processamento.

Observa-se claramente que, independente do paradigma usado no desenvolvimento, todos propõem modelar o sistema em diferentes visões antes de elaborar códigos, possibilitando uma melhor visualização do que será obtido bem como alterações com baixo custo. Desenvolver software desta forma requer:

a) maturidade das pessoas e das organizações. Sendo assim será discutido no próximo capítulo um modelo de capacidade e maturidade que visa o alcance de uma melhor disciplina no desenvolvimento de software e consequente melhoria da qualidade do software.
b) manutenção das visões obtidas e o gerenciamento da configuração de software – SCM que contribui fortemente para obtenção de um software gerenciado e de qualidade, sendo uma das recomendações do CMM. Uma discussão sobre o SCM será apresentada no capítulo 4.

3. CMM - Capability Maturity Model
Modelo de capacidade e maturidade

3.1 Introdução

Neste capítulo é apresentado um dos modelos existentes hoje no mercado de maturidade e capacidade para software. Este modelo orienta as organizações de software de como obter controle de suas atividades de desenvolvimento. Seus conceitos, estrutura e aplicação são descritos a seguir.

O SEI – Instituto de Engenharia de Software da *Carnigie Mellon University* em *Pittsburgh* iniciou o desenvolvimento do CMM - Modelo de Maturidade e Capacidade para Software em 1988, baseado no conhecimento adquirido de avaliações de processo e de capacidade de software, de informações das próprias indústrias desenvolvedoras e de informações do governo.

O modelo orienta as organizações de software como obter controle de seus processos para desenvolver e manter software e como obter uma cultura de engenharia e gerenciamento de software, pois orienta estes desenvolvedores na seleção de estratégias de processos que garantam a qualidade e a melhoria contínua., sendo então uma iniciativa para avaliar e melhorar a capacitação de empresas que produzem software.

O projeto CMM foi apoiado pelo Departamento de Defesa do Governo dos Estados Unidos, um grande consumidor de software que precisava de um modelo formal que permitisse selecionar os seus fornecedores de software de forma adequada. Embora não seja uma norma emitida por uma instituição internacional, como a ISO ou o IEEE, este modelo, até mesmo fora do mercado americano, tem tido uma grande aceitação. O CMM também é chamado de SW-CMM - Software CMM.

A estrutura do CMM é baseada nos princípios de qualidade do produto que existe há mais de sessenta anos. A estrutura de maturidade é inspirada e adaptada dos preceitos de Philip Crosby, que descreveu a grade de gerenciamento com cinco estágios de desenvolvimento no reconhecimento das práticas da qualidade. Sua estrutura foi então adaptada para o processo de software pela equipe coordenada por Watts Humphrey, que inclusive sugeriu ao SEI que o modelo fosse continuamente refinado e revisado, garantindo assim uma evolução das práticas recomendadas (Humphrey, 1989).

O Modelo é uma estrutura que descreve os elementos chaves de um processo de software indicando um caminho de melhoria contínua de processo imaturo, pouco confiável, para um processo maduro, disciplinado. O CMM abrange práticas de planejamento, criação e administração do desenvolvimento de software e manutenção que, quando seguidas, aumentam a habilidade das organizações para alcançarem suas metas com relação a custos, prazos, funcionalidade e qualidade do produto, estabelecendo uma estratégia que permite julgar a maturidade do processo de software de uma organização e a comparação do estado do sistema com o da prática da indústria de software, podendo ainda ser usado para o planejamento da melhoria do processo de software (Paulk, 1993).

O SEI desenvolveu uma versão inicial do modelo a pedido do governo americano e deu especial atenção aos conselhos de *experts* envolvidos no desenvolvimento, melhorando gradativamente os processos de software. O objetivo foi criar um modelo que estivesse baseado em práticas atuais, refletisse o melhor do estado de uma prática e as necessidades de indivíduos que executam a melhoria de processo de software, permitisse a avaliação do processo de software e a avaliação de capacidade do software, além de ser documentado e publicamente disponível.

A definição do que é Maturidade pode ser melhor compreendida através da análise da seguinte tabela:

Organizações maturas	Organizações imaturas
Papéis e responsabilidades bem definidos	Processo improvisado
Existe base histórica	Não existe base histórica
É possível julgar a qualidade do produto	Não há maneira objetiva de julgar a qualidade do produto
A qualidade dos produtos e processos é monitorada	Qualidade e funcionalidade do produto sacrificadas
O processo pode ser atualizado	Não há rigor no processo a ser seguido
Existe comunicação entre o gerente e seu grupo	Resolução de crises imediatas

Tabela 3.1 – Comparativo de Maturidade das Organizações

Um conhecimento adicional e uma perspicácia em maturidade de processo de software foram obtidos desde as primeiras versões do modelo. Isso ocorreu porque os especialistas

estudaram também organizações que não utilizavam software, executando e observando os processos de software e avaliações de capacidade de software, além de analisarem sugestões de mudanças do modelo, participarem de reuniões e seminários com indústrias desenvolvedoras de software e representantes do governo e, todo o tempo solicitarem informações de realimentação desses desenvolvedores e de outros interessados (Paulk, 1993).

Uma preocupação do SEI foi desenvolver um modelo que pudesse ser facilmente utilizado. Quase sempre ferramentas poderosas são sub-utilizadas. O CMM constitui-se em uma grande ferramenta, contanto que os desenvolvedores o compreendam e o utilizem corretamente.

Um problema que surge com qualquer estrutura é que as pessoas tentam reduzi-la à uma lista de conferência e no caso do CMM a interpretação literal na avaliação de um item pode não trazer os resultados esperados, ou seja, com a precisão desejada. Este pode ser um modo racional de uso do CMM quando se busca uma avaliação rápida do processo de desenvolvimento de uma organização, quando na realidade uma verdadeira compreensão de uma organização requer muito mais que uma conferência superficial dos elementos de software (Humphrey, 1995).

Processos de software podem ser extremamente complexos, pois, entre outras características, eles necessitam de uma grande interação das pessoas envolvidas na sua complicada produção e quando se reduz a avaliação à uma lista de conferência, muitas tarefas críticas podem ser negligenciadas, isto é, o avaliador poderá obter facilmente uma visão distorcida do processo da organização.

Para entender um processo complexo é necessário que se identifique e ordene todos os detalhes e itens críticos que representam a essência da capacidade da organização. Foi para isso também que o CMM foi projetado, ou seja, para ajudar as pessoas a identificarem essas atividades críticas que indicam a habilidade da organização. A hierarquia de níveis do CMM ajuda os avaliadores ou desenvolvedores a identificarem as atividades que devem ser corretamente gerenciadas para que a organização execute suas tarefas de forma rápida e precisa. Colocar estas atividades em ordem de prioridade auxilia os desenvolvedores a gerenciarem seus esforços nas áreas que precisam de mais ou menos atenção, facilitando a alocação de recursos.

Um outro problema surge quando os desenvolvedores perdem visão das metas do modelo e enfocam somente os detalhes de implementação, pois diferentes interpretações

dessas metas, dependendo de cada organização, levam consequentemente à diferentes soluções. Isto dependerá, entre outros fatores, das tecnologias utilizadas e da cultura de desenvolvimento da organização. A preocupação básica não deve ser como a atividade é implementada mas sim, se esta atividade ajuda a organização alcançar seus objetivos, ou seja, se tal atividade é importante para o processo. A implementação dependerá então dos recursos empregados, variando de organização para organização (Kenneth, 1996).

Portanto, o CMM consiste em um processo de software que se utiliza de conceitos de gerenciamento de processos para a melhoria da qualidade do desenvolvimento e manutenção de software (Humphrey, 1989). O modelo está estruturado em níveis que atualmente apenas recomendam o que deve ser feito, e não como deve ser feito. Isto acontece devido a dependência do paradigma de concepção de software que será seguido pelos desenvolvedores.

3.2 Estrutura do CMM

O CMM é formado por cinco níveis de maturidade. Com a exceção de Nível 1, cada nível está composto de várias áreas-chave de processo. Cada área-chave de processo é organizada em cinco seções chamadas características comuns que especificam as práticas-chaves. Estas práticas-chaves, quando coletivamente endereçadas, concluem os objetivos da sua respectiva área-chave de processo. A *figura 3.1* ilustra essa estrutura:

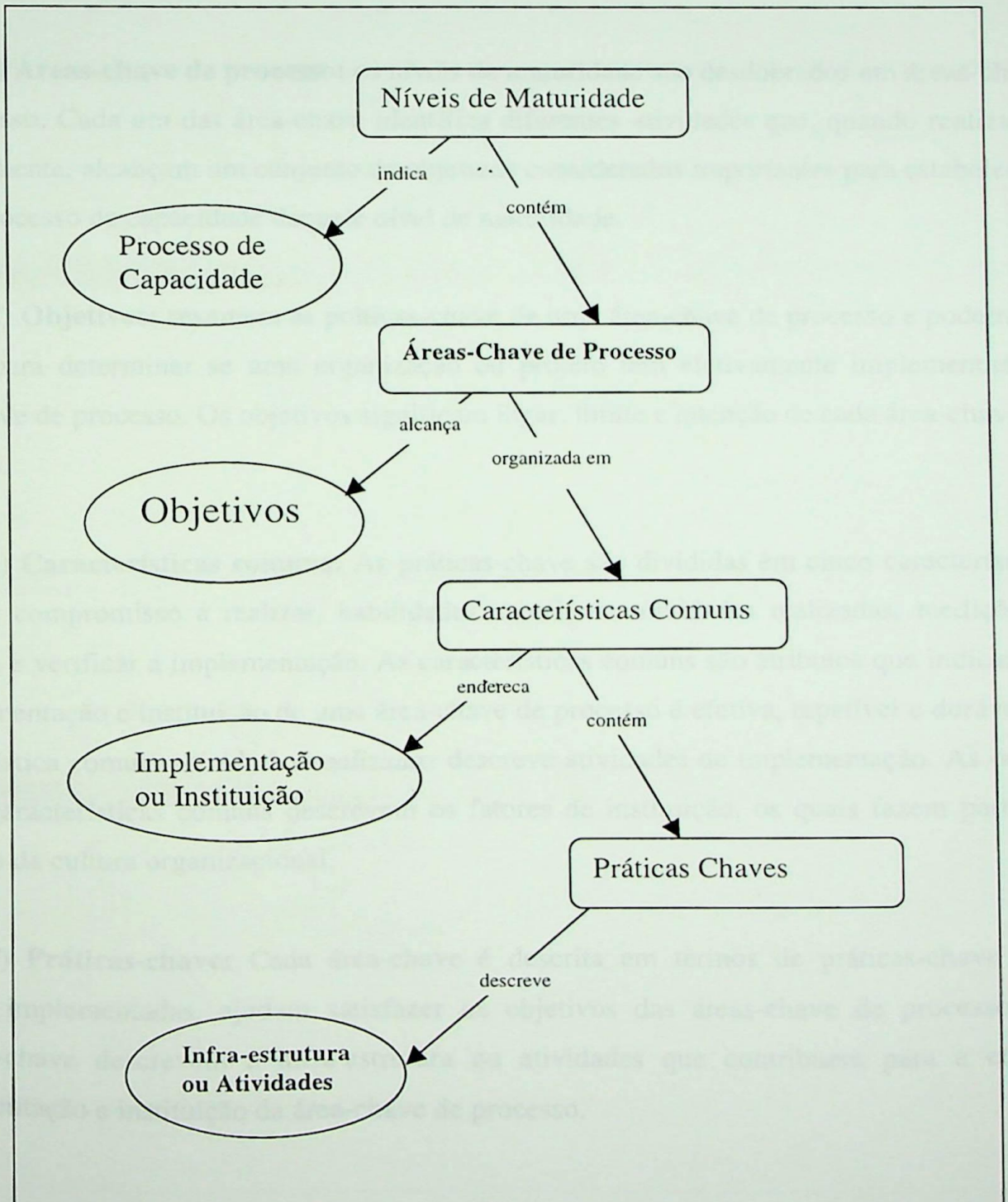


Figura 3.1 – Estrutura dos componentes do CMM – Paulk, 1993

a) Níveis de Maturidade: um nível de maturidade pode ser perfeitamente definido como um topo no desenvolvimento de software, ou seja, todas as atividades executadas dentro dos padrões de cada organização alcançam um determinado nível e, através de processos de melhoria contínua, estas atividades podem fazer com que o software alcance um nível de maturidade mais alto.

b) Processo de Capacidade: descreve uma faixa de resultados esperados que podem ser alcançados seguindo-se um processo de software.

c) Áreas-chave de processo: os níveis de maturidade são desdobrados em áreas-chave de processo. Cada um das área-chave identifica diferentes atividades que, quando realizadas coletivamente, alcançam um conjunto de objetivos considerados importantes para estabelecer-se um processo de capacidade daquele nível de maturidade.

d) Objetivos: resumem as práticas-chave de uma área-chave de processo e podem ser usados para determinar se uma organização ou projeto tem efetivamente implementado a área-chave de processo. Os objetivos significam lugar, limite e intenção de cada área-chave de processo.

e) Características comuns: As práticas-chave são divididas em cinco características comuns: compromisso a realizar, habilidades a realizar, atividades realizadas, medições e análises, e verificar a implementação. As características comuns são atributos que indicam se a implementação e instituição de uma área-chave de processo é efetiva, repetível e durável. A característica comum *atividades realizadas* descreve atividades de implementação. As outras quatro características comuns descrevem os fatores de instituição, os quais fazem parte do processo da cultura organizacional.

f) Práticas-chave: Cada área-chave é descrita em termos de práticas-chave que, quando implementadas, ajudam satisfazer os objetivos das áreas-chave de processo. As práticas-chave descrevem a infra-estrutura ou atividades que contribuem para a efetiva implementação e instituição da área-chave de processo.

3.3 Os níveis de Maturidade

A forma como as organizações estabelecem e produzem seus processos de software, através dos quais desenvolvem e mantém seus produtos de software, permite que elas progridam através dos níveis de maturidade.

A *figura 3.2* apresenta os cinco níveis de maturidade onde as setas indicam o tipo de processo de capacidade que está sendo instituído pela organização e cada passo do modelo de maturidade:

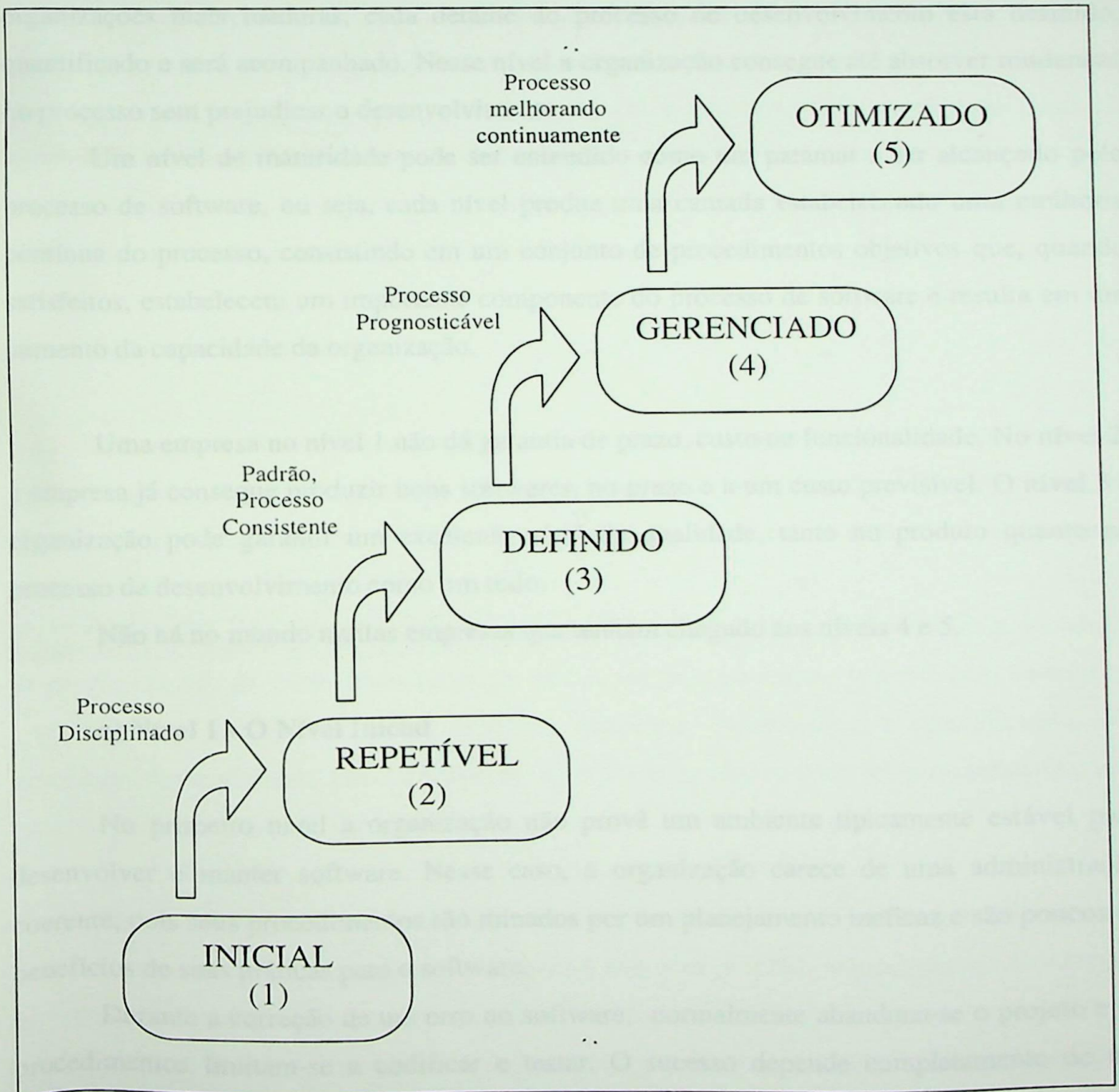


Figura 3.2 – Estrutura dos níveis de Maturidade do CMM – Paulk, 1993

A melhoria contínua do processo é baseada em vários pequenos passos no desenvolvimento ao invés de inovações revolucionárias. O CMM é portanto, uma ferramenta que objetiva organizar esses passos juntamente com os cinco níveis de maturidade, definindo uma escala ordinal para medir a maturidade de um processo de software e para estimar capacidade da organização no processo de software.

O modelo classifica as organizações em cinco níveis distintos, cada um com suas características próprias. No nível 1, o das organizações mais imaturas, não há nenhuma metodologia implementada e tudo ocorre de forma desorganizada. No nível 5, o das organizações mais maduras, cada detalhe do processo de desenvolvimento está definido, quantificado e será acompanhado. Nesse nível a organização consegue até absorver mudanças no processo sem prejudicar o desenvolvimento.

Um nível de maturidade pode ser entendido como um patamar a ser alcançado pelo processo de software, ou seja, cada nível produz uma camada estabelecendo uma melhoria contínua do processo, consistindo em um conjunto de procedimentos objetivos que, quando satisfeitos, estabelecem um importante componente do processo de software e resulta em um aumento da capacidade da organização.

Uma empresa no nível 1 não dá garantia de prazo, custo ou funcionalidade. No nível 2, a empresa já consegue produzir bons softwares, no prazo e a um custo previsível. O nível 3 a organização pode garantir um excelente nível de qualidade, tanto no produto quanto no processo de desenvolvimento como um todo.

Não há no mundo muitas empresas que tenham chegado aos níveis 4 e 5.

a) Nível 1 - O Nível Inicial

No primeiro nível a organização não provê um ambiente tipicamente estável para desenvolver e manter software. Nesse caso, a organização carece de uma administração coerente, pois seus procedimentos são minados por um planejamento ineficaz e são poucos os benefícios de suas práticas para o software.

Durante a correção de um erro no software, normalmente abandona-se o projeto e os procedimentos limitam-se a codificar e testar. O sucesso depende completamente de um gerente excepcional e do esforço individual de cada profissional do time de desenvolvimento do software. Ocasionalmente, um gerente de software capaz e dedicado pode resistir as

pressões de utilizar atalhos no processo de software, ou seja, não permite que os procedimentos sejam executados de forma não planejada, mesmo que mais rápida. No entanto, quando eles deixam o projeto, sua influência de estabilização e gerenciamento parte com eles. Nem mesmo um forte processo de engenharia pode superar a instabilidade criada pela ausência de práticas coerentes de administração.

A capacidade de processo das organizações de software no Nível 1 é impossível de ser prevista porque o processo de software é constantemente mudado ou modificado. Prazos, orçamentos, funcionalidade e qualidade de produto são geralmente impossíveis de se prever. O desempenho depende da capacidade individual de cada membro da equipe e varia de acordo com as suas habilidades inatas, conhecimento e motivações. Nas organizações do nível *inicial*, poucos são os processos de software estáveis, e o desempenho só pode ser avaliado individualmente ao invés da avaliação da capacidade organizacional como um todo.

O processo de desenvolvimento é desorganizado e até caótico. Poucos processos são definidos e o sucesso depende de esforços individuais e heróicos.

b) Nível 2 – Repetível

As organizações desse nível possuem uma política de gerenciamento de projetos e os procedimentos para implementação dessa política estão estabilizados. As atividades de planejamento e gerenciamento de um novo projeto são baseadas na experiência de projetos similares.

Para alcançar o nível 2 a organização deve institucionalizar efetivamente os processos de gerenciamento de software, os quais permitem que as organizações repitam com sucesso os procedimentos desenvolvidos em projetos anteriores, mesmo que a especificação dos processos implementados sejam diferentes. Um processo efetivo pode ser caracterizado como prático, documentado, testado e medido, além de favorecer a melhoria contínua.

No nível *repetível* a organização tem implementado o gerenciamento do software, onde os novos projetos são elaborados baseados nos seus próprios requisitos e na observação de resultados anteriores. Esses processos básicos de gerenciamento de projeto estão estabelecidos e permitem acompanhar custo, cronograma e funcionalidade. É possível repetir o sucesso de um processo utilizado anteriormente em outros projetos similares.

c) Nível 3 – Definido

Tanto as atividades de gerenciamento quanto de engenharia do processo de desenvolvimento de software estão documentadas, padronizadas e integradas em um padrão de desenvolvimento da organização. Todos os projetos utilizam uma versão aprovada e adaptada do processo padrão de desenvolvimento de software da organização.

No nível *definido* um processo padrão para desenvolvimento e manutenção de software da organização é documentado, incluindo procedimentos de engenharia de software de forma integrada com uma coerente administração e gerenciamento. Esses processos são usados e modificados de forma apropriada para ajudarem efetivamente toda a equipe de desenvolvimento, tanto gerentes de projetos como analistas e programadores. Uma parte da equipe fica responsável pelas atividades relacionadas aos processos padrão da organização, conhecida como SEPG - software engineering process group, ou grupo de engenharia de software (Paulk, 1993).

Os gerentes são devidamente treinados para obterem confiança, conhecimento e técnica para implementarem essa política de desenvolvimento e enfrentar os desafios que surgirão no novo ambiente com novas regras.

As organizações do nível 3 do CMM possuem um processo padrão de desenvolvimento e manutenção de software suportado por uma engenharia de software planejada e gerenciada com atividades estáveis e repetitíveis. Esse processo de capacidade é baseado no conhecimento comum de toda a organização das atividades, regras e responsabilidades, permitindo a previsão e cumprimento de prazos, custos e padrões de qualidade.

d) Nível 4 – Gerenciado

Neste nível a organização determina os objetivos quantitativos da qualidade tanto para produtos como para os processos. A produtividade e a qualidade das atividades dos processos de software são medidas por todo o projeto como parte de um programa de avaliação organizacional. Este programa é usado para coletar e analisar os dados disponíveis à partir de processos de software do nível definido. Os processos de software no nível 4 são suportados por uma bem definida e consistente metodologia de avaliação.

Os produtos e processos são controlados por uma metodologia que detecta qualquer que seja a variação na execução dos processos ou resultados, representados pelos produtos.

As organizações do nível *gerenciado* tem todos os seus processos medidos e toda e qualquer operação é rigidamente controlada afim de obter-se alta qualidade e produtividade em níveis previamente prognosticados.

São coletadas medidas detalhadas da qualidade do produto e processo de desenvolvimento de software. Tanto o produto quanto o processo de desenvolvimento de software são entendidos e controlados quantitativamente.

e) Nível 5 – Otimizado

No nível *otimizado* a organização coloca todo o foco na sua melhoria contínua, identificando a essência de cada processo de forma a prevenir a ocorrência de defeitos. Os dados dos processos são utilizados na análise custo-benefício e novas propostas de mudanças nos processos estão sempre sendo avaliadas. As inovações nos processos constituem-se em um alvo, mesmo que esses processos já apresentem resultados satisfatórios. Essas medidas possibilitam o gerenciamento e a evolução contínua dos processos de software e dos produtos.

Quando algum defeito ou erro acontece, uma análise criteriosa é efetuada buscando corrigir os efeitos, mas principalmente, identificar as causas. Essa análise resulta em medidas preventivas que poderão ser disseminadas para outros projetos.

O processo de software das organizações do nível 5 são caracterizados por uma melhoria contínua obtida através das inovações de técnicas, linguagens, métodos e equipamentos.

O melhoramento contínuo do processo é conseguido através de um *feedback* quantitativo dos processos e pelo uso pioneiro de idéias e tecnologias inovadoras.

3.4 Áreas-chave de processo

As áreas-chave de processo ou KPAs - Key Process Areas foram definidas para residir em um único nível de maturidade e são constituídas de blocos de atividades. Com exceção do nível Inicial, todos os outros níveis são detalhados em áreas-chave de processo *(figura 3.3)*.

Cada área-chave de processo é composta por uma determinada quantidade de atividades que deverão ser realizadas para que a organização alcance um conjunto de objetivos específicos.

Isso estabelecerá um processo de capacidade daquele nível de maturidade.

Estas áreas compreendem exatamente o que a organização deve focar para melhorar o seu processo de desenvolvimento de software.

Cada nível possui um foco específico, demonstrado na tabela seguinte, ou seja, as diversas áreas-chaves que o compõem possuem um relacionamento direto com o objetivo do nível, de forma que se todas as diferentes atividades forem realizadas coletivamente, a organização alcança os objetivos relativos ao processo de capacidade daquele nível de maturidade, conforme demonstrado no quadro seguinte:

NÍVEL	FOCO
1) Inicial	Pessoas competentes e heróis
2) Repetível	Processos de gerenciamento de projetos
3) Definido	Processos de engenharia e apoio
4) Gerenciado	Qualidade do produto e do processo
5) Otimizado	Melhoramento contínuo do processo

Tabela 3.2 – Níveis do CMM

A qualificação ou classificação de uma organização em um determinado nível de maturidade CMM está diretamente relacionada ao cumprimento ou realização dos processos relacionados às áreas-chave daquele determinado nível.

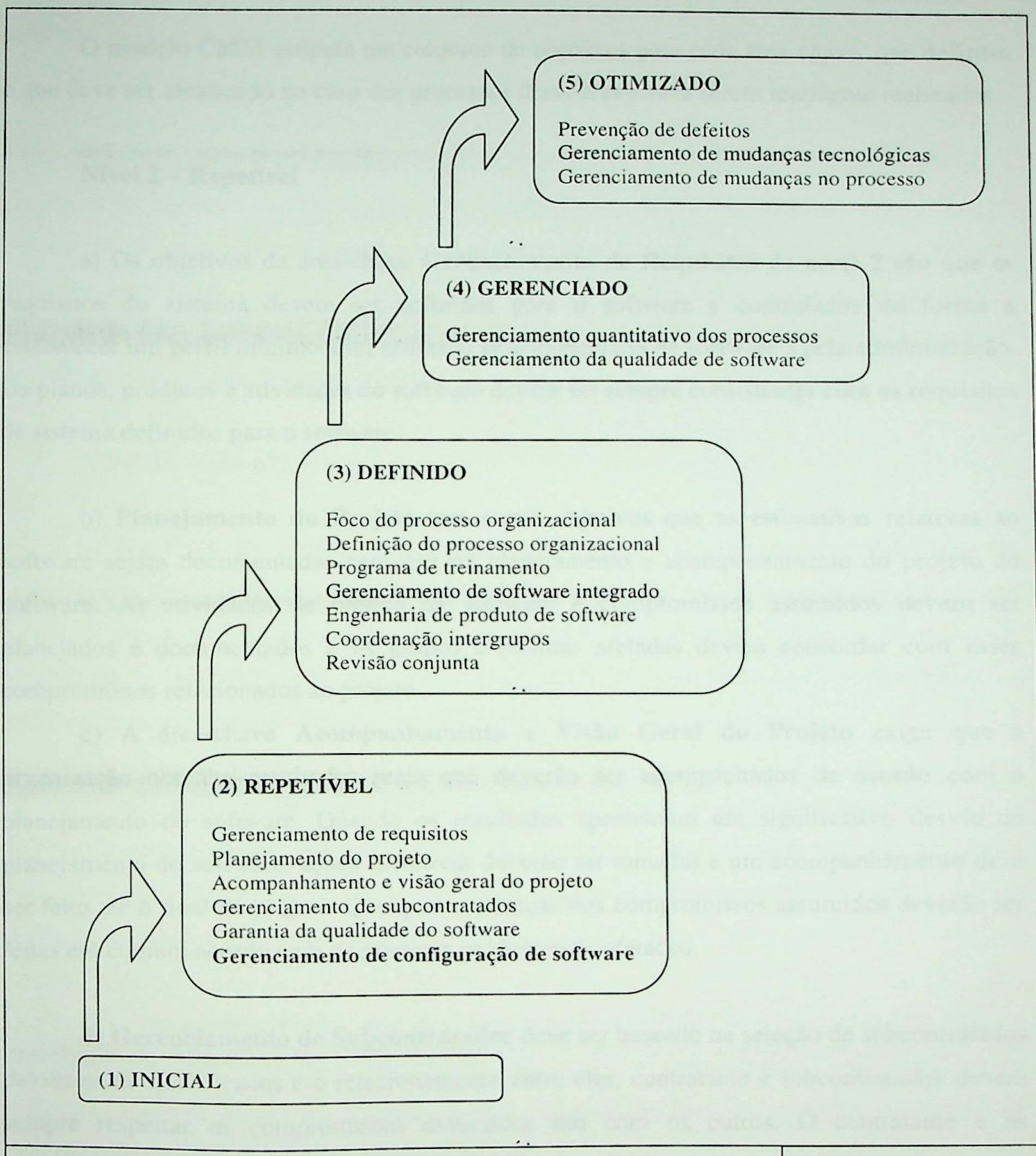


Figura 3.3 – Áreas-chaves dos níveis de Maturidade do CMM – Paulk, 1993

3.5 Objetivos das áreas-chave de processo

O modelo CMM estipula um conjunto de objetivos para cada área-chave que definem o que deve ser alcançado no caso dos processos desta área-chave serem realmente realizados.

Nível 2 – Repetível

a) Os objetivos da área-chave **Gerenciamento de Requisitos** do nível 2 são que os requisitos do sistema devem ser definidos para o software e controlados de forma a estabelecer um perfil mínimo a ser utilizado pela engenharia de software e pela administração. Os planos, produtos e atividades do software devem ser sempre consistentes com os requisitos de sistema definidos para o software.

b) **Planejamento do Projeto** tem como objetivos que as estimativas relativas ao software sejam documentadas para uso no planejamento e acompanhamento do projeto do software. As atividades de projeto de software e compromissos assumidos devem ser planejados e documentados e os grupos e pessoas afetadas devem concordar com esses compromissos relacionados ao projeto.

c) A área-chave **Acompanhamento e Visão Geral do Projeto** exige que a organização obtenha resultados reais que deverão ser acompanhados de acordo com o planejamento do software. Quando os resultados apresentam um significativo desvio do planejamento do software, ações corretivas deverão ser tomadas e um acompanhamento deve ser feito até o final do projeto. Qualquer mudanças nos compromissos assumidos deverão ser feitas em comum acordo com os grupos e profissionais afetados.

d) **Gerenciamento de Subcontratados** deve ser baseado na seleção de subcontratados devidamente qualificados e o relacionamento entre eles, contratante e subcontratados devem sempre respeitar os compromissos assumidos um com os outros. O contratante e os subcontratados necessitam manter uma comunicação constante e os resultados reais de cada subcontratado devem ser acompanhados pelo contratado, de acordo com os compromissos assumidos.

e) Na **Garantia da qualidade do software** as atividades de garantia de qualidade de software devem ser planejadas e a conformidade dos produtos de software e atividades com

os padrões, procedimentos e requisitos deve ser verificada objetivamente. As equipes e profissionais afetados devem ser informados das atividades de garantia de qualidade de software e de seus resultados e questões relacionadas à não conformidade que não sejam resolvidas dentro do projeto de software deverão ser encaminhadas à gerência geral.

f) Na área-chave **Gerenciamento de configuração** as atividades de gerenciamento de configuração deverão ser planejadas e os produtos de trabalho de software identificados, controlados e disponíveis. Quaisquer mudanças nos produtos de trabalho identificados deverão controladas e todos os envolvidos informados da situação atual e projetada dos produtos de trabalho de software. Maiores detalhes sobre o Gerenciamento de Configuração de Software está apresentado no capítulo 4.

Nível 3 – Definido

a) No **Foco do processo organizacional** deverão ser coordenadas atividades de desenvolvimento e melhoramento do processo de software de toda a organização. Os pontos fortes e fracos do processo de desenvolvimento de software utilizado deverão ser identificados, de acordo com um padrão de processo. As atividades de desenvolvimento e melhoramento do processo deverão ser planejadas, a nível de organização.

b) Os objetivos da área-chave **Definição do processo organizacional** são que o processo padrão de desenvolvimento de software da organização seja desenvolvido e mantido e a informação relacionada ao uso do processo padrão de desenvolvimento de software seja coletada, revisada e disponibilizada.

c) O **Programa de treinamento** deverá possuir atividades de treinamento planejadas e ser fornecido para o desenvolvimento de habilidades e conhecimentos necessários para realizar o gerenciamento do software e das funções técnicas. As pessoas no grupo de engenharia de software e outros grupos relacionados a software deverão receber o treinamento necessário para realizar as suas funções.

d) No **Gerenciamento de software** integrado o processo de software definido para o projeto deve ser uma versão adaptada do processo padrão de desenvolvimento de software da

organização. O projeto deverá ser planejado e gerenciado de acordo com o processo de desenvolvimento de software definido para o projeto.

e) Na área-chave **Engenharia de produto de software** as atividades de engenharia de software deverão ser definidas, integradas e consistentemente realizadas para produzir o software. Os produtos de trabalho do software deverão ser mantidos consistentes entre si.

f) A **Coordenação intergrupos** tem como objetivo que todos os grupos de trabalho afetados concordem com os requisitos dos cliente. Todos os grupos de trabalho afetados deverão concordar e estarem cientes dos acordos entre os grupos de engenharia. Os grupos de engenharia deverão identificar, acompanhar e resolver todas as questões intergrupos.

g) A **Revisão conjunta** das atividades deverão ser planejadas e os defeitos nos produtos do trabalho identificados e removidos.

Nível 4 – Gerenciado

a) Os objetivos do **Gerenciamento quantitativo dos processos** são que as atividades de gerenciamento quantitativo dos processos sejam planejadas. A performance do processo de desenvolvimento de software definido para o projeto deverá ser controlada quantitativamente e a capacidade do processo desenvolvimento de software padrão da organização dever conhecida em termos quantitativos.

b) No **Gerenciamento da qualidade de software** as atividades de gerenciamento da qualidade de software do projeto deverão ser planejadas. Deverão ser definidos objetivos mensuráveis da qualidade do produto de software e suas prioridades e o progresso real em direção à realização dos objetivos de qualidade para os produtos de software deverá ser quantificado e gerenciado.

Nível 5 – Otimizado

a) Na área-chave **Prevenção de defeitos** as atividades de prevenção de defeitos deverão ser planejadas e as causas comuns de defeitos deverão ser procuradas e identificadas, bem como priorizadas e sistematicamente eliminadas.

b) No **Gerenciamento de mudanças tecnológicas** a incorporação de mudanças tecnológicas deverá ser planejada e novas tecnologias serem avaliadas para determinar seu efeito na qualidade e na produtividade. Essas novas tecnologias deverão ser adequadas e incorporadas na prática normal de toda a organização.

c) Os objetivos da área-chave **Gerenciamento de mudanças no processo** são o planejamento do melhoramento contínuo do processo e que toda a organização participe das atividades de melhorias do processo de software. O padrão de processo de software da organização e os processos de software de cada projeto definido deverão ser melhorados continuamente.

3.6 Características comuns e práticas-chave

As características comuns são itens a serem observados para que se possa verificar a implementação e institucionalização de cada área-chave de processo. Elas podem indicar se a área-chave de processo é eficiente, repetível e duradoura. São cinco as características comuns no modelo CMM e cada uma possui suas práticas-chave a serem realizadas:

a) **Compromisso de realizar** – esta característica comum refere-se à todas as atitudes a serem tomadas pela organização com o intuito de garantir que o processo se estabeleça e seja duradouro. Suas práticas-chave são o estabelecimento de políticas e a indicação de um gerente experiente.

b) A característica comum **Capacidade de realizar** constitui-se de pré-requisitos que devem existir no projeto ou na organização para implementar o processo de forma competente. Suas práticas-chave são alocação de recursos, definição da estrutura organizacional e de treinamento.

c) Atividades realizadas é a característica comum referente aos papéis e os procedimentos necessários para implementar uma área-chave de processo. Suas práticas-chave são estabelecimento de planos e procedimentos, realização do trabalho, acompanhamento do trabalho e tomada de ações corretivas, se necessário.

d) **Medições e análise** é a necessidade de medir o processo e analisar as medições. Realização de medições para determinar o estado e a efetividade das atividades realizadas constitui sua prática-chave.

e) **Implementação com Verificação** é a característica comum que refere-se aos passos para garantir que as atividades serão realizadas de acordo com o processo estabelecido. Suas práticas-chave são revisão, auditoria e garantia de qualidade.

As práticas-chave descrevem as atividades que contribuem para atingir os objetivos de cada área-chave do processo. Em geral são descritas com frases simples, seguidas de descrições detalhadas, chamadas de subpráticas. As práticas-chave descrevem o que deve ser feito e não como os objetivos devem ser atingidos.

4. O SCM – Software Configuration Management Gerenciamento da Configuração de Software

4.1 Introdução

Neste capítulo é apresentado o SCM – subsistema do processo de desenvolvimento de sistemas de software, altamente recomendado pelo CMM – para o qual será elaborado um esquema de implementação utilizando a tecnologia do QFD.

As primeiras aplicações de gerenciamento de configuração ocorreram na área militar dos Estados Unidos da América, mais especificamente na fabricação de seus equipamentos. Dentre estes primeiros princípios, procedimentos e métodos se destacavam a documentação de todas as informações do produto e do seu processo de fabricação, a utilização de identificação única dos componentes do sistema através de *part-numbers* e o controle do desenvolvimento através do mapeamento das alterações e da verificação da qualidade de cada um de seus componentes (Dart, 1992).

A necessidade de se padronizar tais princípios, procedimentos e métodos do gerenciamento surgiu à medida que foi crescendo a complexidade dos sistemas a serem desenvolvidos, com o aumento do número de equipes de trabalho envolvidas no projeto e também devido a contratação de empresas especializadas externas à organização. Esta padronização permitia o estabelecendo das responsabilidades de cada participante a partir da clara definição da interface de trabalho entre as equipes de desenvolvimento e dos termos do contrato entre a organização, suas equipes e as empresas contratadas.

Surgiu então a disciplina CM - *Configuration Management* ou Gerenciamento da Configuração, que fundamentalmente agia com o objetivo de propiciar o controle e a visibilidade do projeto como um todo, aperfeiçoar a sua produtividade e o seu controle de qualidade e, principalmente, evitar a confusão e a redundância de trabalhos (Bersoff, 1985). Este gerenciamento era aplicado ao desenvolvimento de hardware e fornecia basicamente os procedimentos e métodos para construir o histórico do produto e de seus componentes, além de criar a identificação única de cada componente, inclusive de cada versão. O CM ainda era

utilizado para autorizar, avaliar e controlar as alterações do produto e de seus componentes ao longo do seu desenvolvimento e durante a sua manutenção.

Devido ao surgimento do computador e do *software*, que tornou os sistemas ainda mais complexos, percebeu-se a necessidade de também se aplicar os princípios, procedimentos e métodos do CM à produção de *software*, resultando então na nova disciplina conhecida como SCM - *Software* Configuration Management ou Gerenciamento de Configuração de *Software*.

O SCM foi se evoluindo e consolidando os seus conceitos básicos e suas aplicações ao ponto de adquirir uma caracterização distinta do CM, sua disciplina de origem. As três principais diferenças entre o CM e SCM são descritas a seguir (Feiler, 1991) e (Dart, 1992):

A primeira diferença refere-se aos sistemas que serão gerenciados e às suas formas de produção. No CM os procedimentos e métodos de gerenciamento podem ou não estar disponíveis em computadores e o sistema a ser desenvolvido junto com seus processos de produção estão relacionados com hardware, ou seja, equipamentos, máquinas ou instrumentos, enquanto que no SCM todos seus elementos, procedimentos e métodos de gerenciamento, o próprio sistema a ser desenvolvido e os seus processos de produção, estão disponíveis no computador na forma de *softwares*. Portanto, o ambiente de trabalho onde se localiza o SCM além de fornecer os meios para se manter as informações do sistema em desenvolvimento, do seu *status* e do seu histórico, fornece também os próprios meios onde serão feitos o armazenamento, a recuperação e a produção do sistema.

A segunda diferença entre o CM e o SCM é que o Gerenciamento de Configuração de *Software* possui maior potencial para automação e para a integração direta entre o sistema e seu desenvolvimento, já que se encontram no mesmo ambiente de trabalho, pois além de fornecer a função primária de gerenciamento e controle, pode fornecer também uma função de suporte ao desenvolvedor.

A maior dificuldade de implementação do SCM devido à diferença propriamente dita entre *hardware* e *software* é a terceira e última diferença básica entre as disciplinas. Geralmente o *software* é visto como um elemento que pode ser facilmente alterado, seja através da inserção de um comando, de uma linha de programação, de uma função ou procedimento. Isto faz com que haja uma maior freqüência de alterações no *software* que está sendo desenvolvido exigindo que o SCM implemente diferentes graus de controle, muitas vezes tendo que chegar ao nível de controlar o conteúdo dos programas para atender às suas funções de gerenciamento e de suporte ao desenvolvedor. A facilidade de alterar *software* implica não somente em diferenças de

procedimentos entre CM e SCM, mas também na diferença de implementação do SCM pois há a necessidade de se estabelecer um novo comportamento dos desenvolvedores de *software*, que na maioria das vezes procuram fugir das regras impostas de controle.

Apesar das dificuldades em se reconhecer a necessidade do SCM no desenvolvimento do *software*, esta disciplina vem evoluindo e ganhando seu espaço dentro da Engenharia de *Software*, principalmente, por causa da busca de *software* de qualidade.

4.2 Definição

Existem algumas definições de SCM onde os autores se expressam de acordo com suas área de atuação. Logo, fica difícil afirmar que uma definição é mais exata que a outra, mas certamente existe aquela mais ou menos concisa, com palavras mais apropriadas ou mais aceitas pela comunidade, dentre as quais podem ser destacadas as seguintes:

SCM, como também CM, pode ser definido como uma tecnologia que objetiva identificar a configuração de um sistema em diferentes pontos no tempo, além de sistematicamente controlar todas as mudanças e manter a integridade e rastreabilidade da configuração durante todo o ciclo de vida do sistema (Dart, 1992).

CM é o processo de identificar e definir os itens de um sistema, controlando suas mudanças durante o ciclo de vida do software, armazenando e fornecendo informações sobre as modificações ocorridas em cada item (IEEE, STD 729, 1983).

SCM é o gerenciamento do projeto e produção do software ou sistema. Isto significa a comunicação entre os desenvolvedores e projetistas na definição de detalhes técnicos e eventos relacionados a produção e entrega do produto final. SCM também pode ser descrito como uma disciplina que gerencia a identificação, controle, situação e auditoria de uma determinada entidade, como um software ou sistema e seus componentes, constituindo-se em um processo que acompanha o produto em todas as suas fases (Berlack, 1992).

Conforme o CMM - *Capability Maturity Model version 1.1*, SCM é a tecnologia que gerencia uma biblioteca dos elementos básicos do software, contendo suas descrições e como eles são implementados. Qualquer mudança em um desses elementos geram uma nova versão do produto, que é devidamente controlada e gerenciada.

Uma outra definição é que CM é a prática de controlar sistematicamente as mudanças em um sistema de forma a manter sua integridade durante todo o tempo de vida útil. Outro nome para isso seria *Controle de Mudanças*, que inclui técnicas para a avaliação das propostas de mudanças, informações das modificações implementadas e cópias das diversas versões (McConnell, 1993).

SCM é a combinação de técnicas e ferramentas para controle do processo de desenvolvimento de software implementada através da utilização de diferentes ferramentas de diferentes naturezas, que são usadas por profissionais de variadas posições e responsabilidades. A tarefa básica do gerenciamento é resolver o clássico problema de compartilhamento de

informações, múltiplas manutenções com simultâneas alterações, superando assim os obstáculos de comunicação e provendo os usuários com informações úteis. Um aspecto básico do paradigma do gerenciamento é que ele alimenta a perspectiva dinâmica do projeto, ou seja, todos os requisitos e especificações dos componentes, documentação e testes até o produto final são importantes para o controle e todas as mudanças que porventura ocorram em qualquer um desses elementos devem ser na direção do objetivo final do projeto.

Diferentes grupos de pessoas possuem diferentes necessidades do ponto de vista global do projeto. O desenvolvedor pode necessitar saber as diversas revisões de um código fonte enquanto o gerente do projeto necessita conhecer as diferentes versões do sistema. O mais alto nível do gerenciamento ou o departamento de *marketing*, por exemplo, precisa saber apenas qual é a versão atual do produto. Os analistas e programadores necessitam então de um conhecimento mais específico de cada parte do sistema e utilizam ferramentas apropriadas para tal fim, não atendendo muitas vezes as necessidades do gerente do projeto. Utilizando-se da tecnologia do SCM, uma simples ferramenta proporcionará que as diferentes equipes compartilhem todas as informações de acordo com suas necessidades.

A ferramenta a ser utilizada deve ser configurável e de fácil adaptação ao ambiente de desenvolvimento do software (Rawlings, 1994).

Um item de configuração é qualquer parte do ciclo de vida do desenvolvimento ou entrega de um sistema, seja de software, hardware ou firmware – software implementado diretamente em hardware, fluxos e documentos em geral, os quais necessitam ser independentemente identificados, arquivados, testados, revisados, usados, modificados, entregues e mantidos. Estes itens ou elementos de configuração podem ser muito complexos e possuírem inclusive uma hierarquia (Marion, 1996).

O CM tem como objetivo controlar estes itens ou elementos.

O SCM gerencia e desenvolve a evolução de um sistema identificando com exatidão seu estado e composição com suas respectivas configurações, em qualquer tempo e com efetivo controle das alterações, permitindo a rastreabilidade da evolução do sistema (Armstrong, 1996).

Uma outra definição, do Departamento de Defesa Norte-americano é a de que SCM consiste em uma disciplina que aplica técnicas e métodos administrativos durante o ciclo de vida dos itens do software para: identificar e documentar as funcionalidades e características físicas de configuração dos itens; controlar mudanças dos itens e suas respectivas documentações; armazenar e fornecer informações necessárias ao efetivo gerenciamento da configuração dos

itens, incluindo a situação da aprovação das mudanças, seu desenvolvimento e implementação; examinar ou verificar se os itens de configuração estão de acordo com as especificações, fluxos, controle de documentos e cláusulas contratuais (USA, Department of Defense, 1992).

SCM é o controle de mudanças que responde a todas as perguntas acerca dos elementos ou itens do software. Se, por outro lado, SCM for visto como uma simples ferramenta de controle ou uma obrigação contratual, será facilmente um problema burocrático que atrapalhará o desenvolvimento. No entanto, é mais que um requisito de contrato; é um gerenciamento do desenvolvimento onde nada é perdido ou destruído, consistindo em um efetivo apoio aos desenvolvedores (Humphrey, 1989).

De qualquer forma todas essas definições contem o núcleo principal da questão onde SCM é uma disciplina para controlar a evolução de sistemas de *software* (Dart, 1992). E, para uma definição formal podemos optar por aquela contida nos manuais do IEEE (STD 1045, 1987): SCM é o processo de identificar e definir componentes em um sistema, controlando a atualização e alteração através do seu ciclo de vida, registrando e relatando o *status* dos componentes, das solicitações de alteração, e verificando se os componentes do sistema estão completos e corretos.

Escola Federal de Engenharia de Itajubá - EFEI
Biblioteca MAUÁ BIM

4.3 Aspectos Operacionais

Clássicos

Os aspectos operacionais clássicos do SCM são identificar, controlar, acompanhar e auditar ou examinar (Bersoff, 1985). Pode-se resumidamente assim definir cada aspecto:

a) **Identificar:** consiste no aspecto de tornar identificáveis e acessíveis tanto a estrutura quanto os tipos de componentes do *software*, através de um nome e de uma versão, tornando-os únicos.

b) **Controlar:** é o aspecto relativo ao dinamismo do desenvolvimento, onde deve ser feito o controle da atualização e das alterações dos componentes ao longo do ciclo de vida do *software* de forma a garantir a sua consistência.

c) **Acompanhar:** este aspecto está voltado para a divulgação entre os desenvolvedores das alterações que estão sendo feitas nos componentes, registrando e relatando o seu *status*.

d) **Examinar:** é o aspecto da conferência, de forma a validar a completeza do *software* mantendo a consistência entre os componentes, assegurando que foram bem identificados e que estão sendo apropriadamente desenvolvidos.

Os Elementos de *Software* (ES), ou Itens de *Software* ou, ainda, Itens de Configuração são os componentes de um *software*. Pode-se incluir entre eles os documentos de especificações, documentos de projeto, planos de verificação e validação, código fonte, planos de teste, relatórios de teste, manuais do usuário e de manutenção, cronogramas, documentos de procedimentos, etc...

A integridade do *software* deve ser mantida durante seu desenvolvimento e há portanto a necessidade de se identificar e colocar sob controle de configuração todos os elementos relevantes que compõem o *software*. O nível de granularidade desses elementos, ou seja, até que nível serão controlados, dependerá do tamanho e complexidade do *software* a ser desenvolvido.

Esses elementos são essenciais na manutenção da integridade do *software*, pois fornecem o gerenciamento e o controle das alterações e permitem a visibilidade do *software* durante o seu desenvolvimento.

Todos ES's recebem uma identificação única e começam a ser controlados quanto às suas alterações, sendo estabelecidas linhas de demarcação ao final de cada uma das fases ao longo do ciclo de vida do *software*. Desta forma, em cada fase existirá uma coleção de ES's referenciado com uma única versão e sob estas condições eles somente poderão ser alterados a partir de uma solicitação formal. As linhas de demarcação representariam então os pontos de entrega de ES's para o prosseguimento do seu desenvolvimento, ou seja, os ES's são promovidos de uma fase para outra a medida que atendem às especificações.

Esses elementos e as suas alterações deverão ser gerenciadas e controladas a fim de manter a integridade do produto. Desta forma, as solicitações de alteração devem ser identificadas e classificadas. Há basicamente três categorias de alterações (Dart, 1992):

a) **Corretivas:** são aquelas realizadas por discrepância com as especificações, implementação incorreta ou desrespeito aos padrões.

b) **Adaptativas:** são aquelas realizadas pela introdução de novos requisitos do cliente, acréscimo ou troca do hardware.

c) **Perfectivas:** são aquelas que visam o aperfeiçoamento do *software*, principalmente da sua performance.

Todas as solicitações de alteração, independentemente da categoria, deverão ser analisadas para se determinar quem, quando e como será feita a alteração. A decisão em implementar ou não a alteração deverá levar em conta fatores técnicos e gerenciais, sendo os fatores técnicos relacionados com tamanho e complexidade da alteração, impacto na performance, nos testes, disponibilidade de hardware e, principalmente, custo. Os fatores gerenciais estão relacionados com a criticalidade do ES envolvido, a vontade do cliente ou do *marketing*, disponibilidade de recursos humanos ou de *hardware* e impacto no cronograma.

Também deverá ser exercido o controle da execução das alterações. É necessário impor que a alteração real implementada seja realmente a alteração solicitada, pois os desenvolvedores poderiam aproveitar a oportunidade de uma autorização de alteração para se implementar alterações não autorizadas. Outro aspecto que deve ser observado quanto ao controle de alteração é a coordenação de atualização para que todos os ES's dependentes dos ES's alterados também sejam atualizados. Sendo assim, o *status* das solicitações de alteração é registrado e mantido, relacionando às versões que refletem a alteração, possibilitando o fornecimento de tais informações ao gerenciamento e aos clientes.

O relato do *status* dos ES's e de suas alterações e revisão da correteza e a consistência do *software* e do seu desenvolvimento possibilita estabelecer o *status* de um projeto nas várias fases do ciclo de vida do projeto.

Complementares

Aos aspectos operacionais clássicos, alguns outros foram acrescentados, sendo então denominados aspectos operacionais complementares (Dart, 1992). Resumidamente, podem ser assim definidos:

a) **Produção:** este aspecto está relacionado com o gerenciamento da produção ou da construção do *software*; com a facilidade de se obter um congelamento do status do produto a qualquer hora; com os mecanismos para otimizar esforços, reduzindo a necessidade da recompilação de componentes que não foram alterados nem que foram afetados por alterações em outros componentes; com a facilidade para se fazer a análise do impacto de uma alteração de tal forma que forneça todas as ramificações possíveis ao se fazer determinada alteração; e com a facilidade de regerar qualquer fase ou parte do produto a partir de qualquer fase do desenvolvimento.

b) **Processo:** tem relação direta com a facilidade de fornecer suporte para a política e para o modelo do ciclo de vida adotado pela organização; com a facilidade de identificar tarefas a serem realizadas e como e quando elas serão completadas; com a facilidade para divulgar

informações para pessoas certas sobre eventos relevantes; e com a facilidade para documentar o conhecimento sobre o produto.

c) **Equipe:** este aspecto está relacionado com o espaço de trabalho individual ou em grupo que cada um terá para executar seu trabalho; com a solução de conflitos quando se estiver mesclando diferentes alterações; e com facilidade para suportar a criação e manutenção de uma família de produtos.

4.4 Abordagens de Configuração

No desenvolvimento de software podem ser aplicadas basicamente duas abordagens de configuração, a configuração estática e a configuração dinâmica.

Na abordagem da configuração estática para a construção de um sistema *(figura 4.1)*, o Processo Construtor é coordenado pela especificação de configuração. Ele busca os componentes especificados na Biblioteca de Componentes, entregando-os ao *Loader*, que montará o sistema. Isto significa que todos os componentes do sistema serão configurados ao mesmo tempo, antes de entrarem em operação (Kramer, 1985).

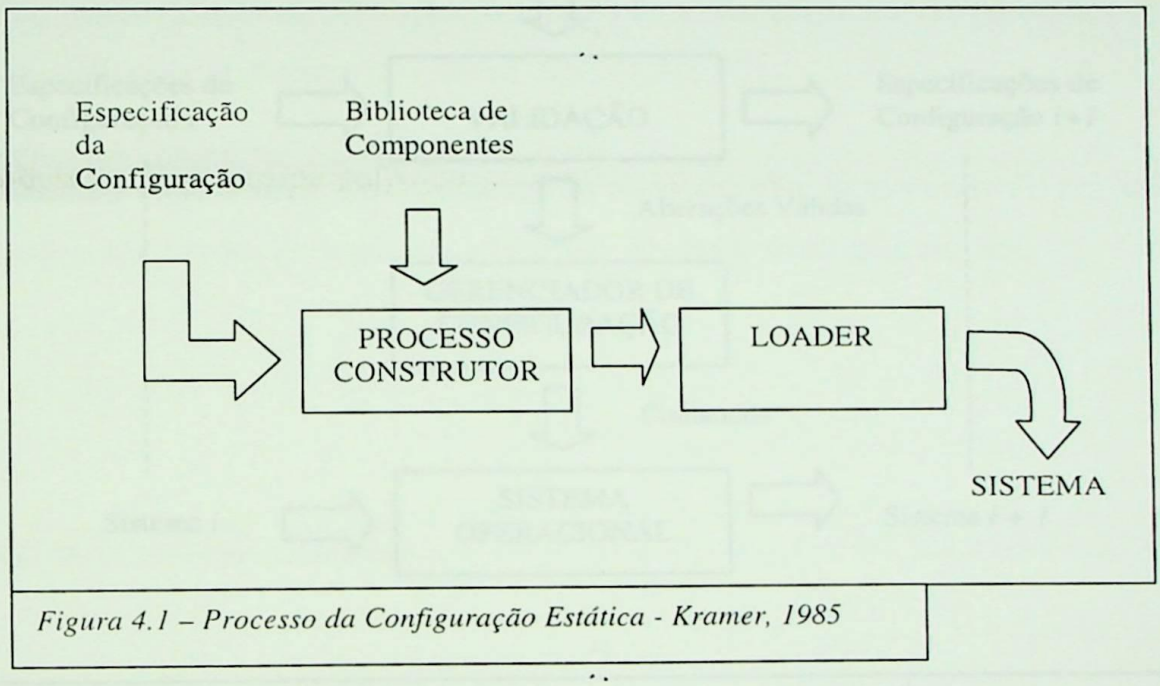


Figura 4.1 – Processo da Configuração Estática - Kramer, 1985

A configuração estática pode ser entendida como o processo editar/compilar/*link* usado na programação seqüencial, não oferecendo nenhuma possibilidade de se fazer alterações incrementais à configuração do sistema e consequentemente não possibilitando a alteração do sistema durante a sua execução.

Qualquer alteração exige que se pare a operação ou utilização do sistema. Isso é necessário para que ele seja reconstruído de acordo com a nova configuração e somente depois é que volta novamente à operação.

Na configuração dinâmica *(figura 4.2)*, a Especificação de Configuração descreve a estrutura lógica e física do sistema e toda e qualquer a alteração deve ser submetida na forma de especificações de alteração, como por exemplo, introduzir novos componentes, modificar componentes existentes, fornecer interconexões diferentes, etc. Essas modificações deverão ser validadas para assegurar sua compatibilidade com o sistema existente, resultando uma nova especificação (Kramer, 1985).

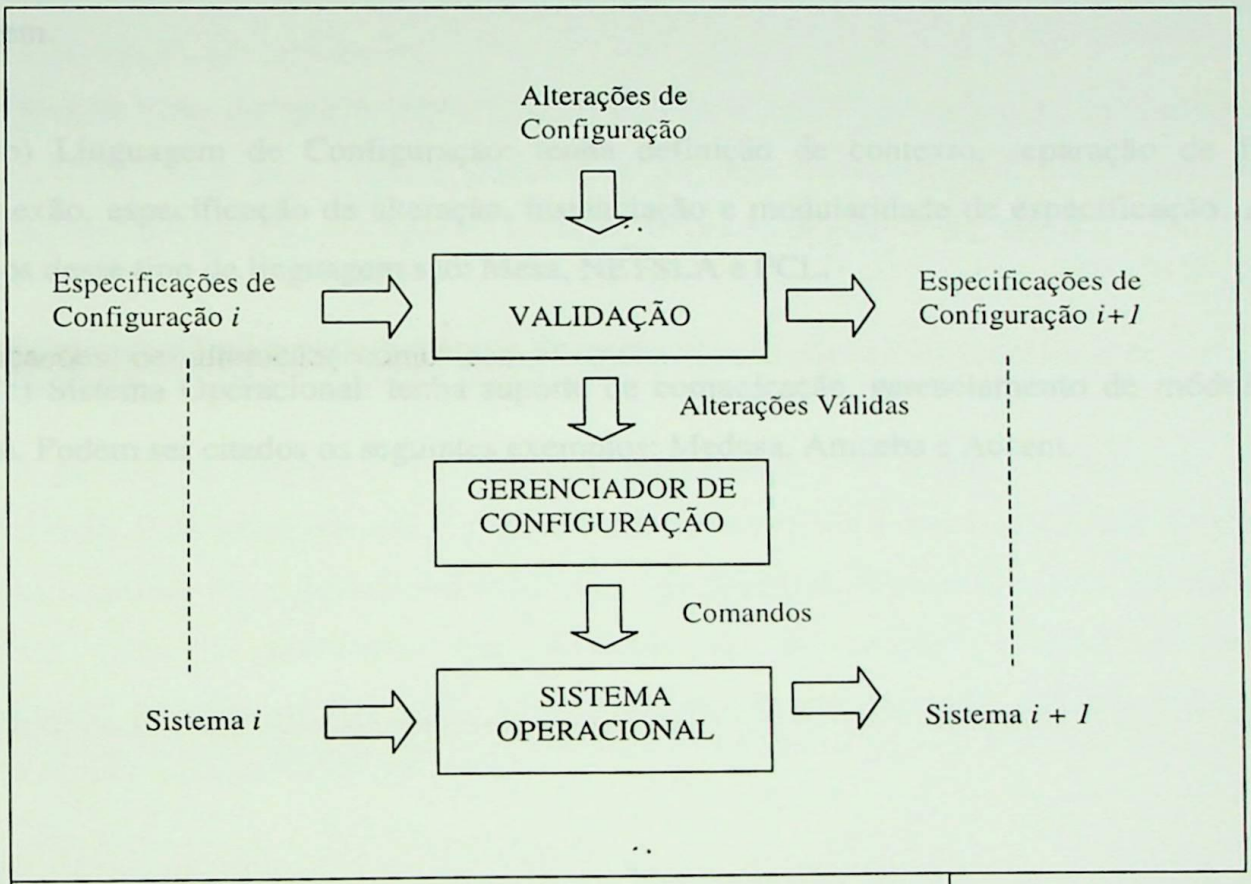


Figura 4.2 – Processo da Configuração Dinâmica - Kramer, 1985

No entanto, a abordagem de configuração dinâmica permite que as alterações incrementais sejam realizadas sem reconstruir o sistema inteiro. As alterações incrementais ao sistema podem ser feitas sem a necessidade de se parar a operação das partes não afetadas do sistema.

O Gerenciador de Configuração gera os comandos necessários ao sistema operacional tais como requerer, apagar ou criar componentes ou conexões. O resultado é um novo sistema descrito pela nova especificação de configuração.

A implementação de uma configuração dinâmica exige que tanto as linguagens de programação quanto os sistemas operacionais tenham determinadas características e que exista uma linguagem própria de configuração. Podemos ser destacar as seguintes características (Kramer, 1985):

a) Linguagem de programação: tenham independência de contexto e de interconexão, além de transparência de comunicação. PRONET e CONIC são exemplos desse tipo de linguagem.

b) Linguagem de Configuração: tenha definição de contexto, separação de função, interconexão, especificação de alteração, instanciação e modularidade de especificação. Alguns exemplos desse tipo de linguagem são: Mesa, NETSLA e PCL.

c) Sistema Operacional: tenha suporte de comunicação, gerenciamento de módulo e de conexão. Podem ser citados os seguintes exemplos: Medusa, Amoeba e Accent.

4.5 Usuários do SCM

Toda equipe de desenvolvimento de software possui vários integrantes, cada um com suas tarefas e responsabilidades, desempenhando uma determinada função dentro do projeto. A interação desses profissionais com o SCM dependerá de sua função, pois, conforme a necessidade, estarão fornecendo ou buscando informações, de diferentes maneiras. Esses profissionais podem ser denominados usuários, e no caso do SCM, terão diferentes privilégios, restrições de acesso, facilidades, etc.

Deve-se observar que o termo usuário não se refere à pessoa em si, mas ao papel ou função desempenhada pela mesma, podendo-se ter uma mesma pessoa desempenhando, ou fazendo o papel de dois ou mais tipos de usuários dentro do projeto (Dart, 1992).

Encontra-se basicamente seis tipos de usuários:

a) gerente de projeto: é o profissional responsável pelo desenvolvimento do software como um todo. Para assegurar que o produto será desenvolvido e atenderá a todos os requisitos do cliente, o gerente de projeto deverá monitorar o progresso do desenvolvimento, identificando e resolvendo todos os problemas que surgirem. As informações necessárias para esse gerenciamento poderão ser fornecidas pelo SCM através, por exemplo, dos relatórios sobre o *status* do projeto e das revisões realizadas.

b) gerente de configuração: responsável pela aplicação das técnicas e procedimentos do SCM. Seu objetivo é assegurar que todas as técnicas e os procedimentos do SCM estejam definidos e implementados de forma que o Gerenciamento da Configuração realmente controle o desenvolvimento de software, divulgando os mecanismos entre os demais usuários e constatando se estão sendo seguidos.

c) analistas e programadores: responsáveis pelo desenvolvimento e manutenção do software. Trabalham efetivamente na construção e na manutenção do software, executando tarefas como criar, alterar, testar e integrar, muitas vezes de forma independente um dos outros, cada qual na sua área de trabalho ou no seu equipamento. No entanto, há uma enorme necessidade de comunicação entre esses profissionais, pois o projeto é único e a troca de

informações a respeito da criação ou alteração de novos elementos de software é constante. O SCM permitirá à esses profissionais obter e fornecer tais informações.

d) aplicadores de testes: validam a correteza do software, ou seja, conferem se os elementos de software estão de acordo com as especificações. Executam também o controle e o registro dos testes e de seus resultados, verificando se um determinado elemento foi testado, quais testes foram aplicados e qual é a versão. Todas essas informações poderão estar disponíveis no SCM.

e) gerente do controle de qualidade: seu principal objetivo é assegurar a qualidade do software. Para tanto deverá estabelecer técnicas e procedimentos a serem respeitados quanto à qualidade do produto, verificando se todas as especificações do cliente estão sendo atendidas. O SCM poderá conter todas estas informações.

f) cliente: usa o software. Seu objetivo é obter o software de acordo com as suas especificações, obtendo informações sobre o andamento do projeto, testando uma versão do software ou de uma parte dele e até mesmo solicitando modificações às partes já desenvolvidas, seja por critério de alteração de especificação ou por não atendimento às especificações já definidas.

O SCM poderá também controlar a atuação do cliente no desenvolvimento do software.

4.6 Requisitos de Funcionalidade

Os requisitos do SCM podem ser estabelecidos de acordo com a visão de cada um dos diversos usuários e de acordo com suas necessidades operacionais (Dart, 1992).

Esses usuários necessitam, em relação aos Elementos de Software, de funções que lhes auxiliem:

a) No registro de versões dos ES;

b) na demonstração das linhas de demarcação do ciclo de vida do desenvolvimento para os ES;

c) na identificação de um grupo de ES que forma uma configuração, inclusive com as respectivas versões de cada ES;

d) no fornecimento de meios de manipular repositórios ou bibliotecas onde serão armazenados os ES e as informações do SCM. Deverá ser permitido trabalhar com todos os tipos de ES, desde o código fonte, objeto ou executável, até diagramas e documentos de projeto;

e) na identificação do contexto do projeto como uma coleção de ES.

Quanto ao aspecto de equipe, as funções devem permitir aos usuários a:

a) definição do espaço de trabalho individual ou de grupos de trabalho;

b) disponibilidade do suporte à criação, controle e manutenção de um família de produtos;

c) facilidade da resolução de conflitos quando mesclando versões.

Para o acompanhamento operacional, as funções devem permitir aos usuários:

a) a instalação do acesso controlado aos ES do produto para não permitir alteração não autorizada; detectar e tratar conflitos de alterações;

b) o rastreamento dos *bugs* identificando como, quando, e por quem eles foram tratados; acompanhar a propagação das alterações, de forma controlada;

c) a disponibilidade de suporte *on-line* à geração de formulários de solicitação de alteração e de relatórios de problemas;

d) o oferecimento de meios de particionar o produto para limitar os efeitos das alterações.

Os usuários necessitam ainda de funções que lhes permitam, em relação à Estrutura de Relacionamentos a:

a) seleção de ES compatíveis para se obter uma versão válida e consistente do produto;

b) identificação e manutenção dos relacionamentos entre ES;

c) especificação das interfaces entre ES, ou entre versões e configurações, procurando torná-los reusáveis;

d) modelagem da estrutura do produto através de um modelo de sistema contendo todos os ES daquele produto.

Os aspectos relacionados à auditoria também devem ser considerados, sendo que as funções devem permitir aos usuários:

a) a geração de arquivos *log* contendo todos os detalhes das operações executadas;

b) o rastreamento dos ES e suas evoluções;

c) o registro da história das alterações.

Em relação às facilidades de construção, os usuários necessitam de funções para:

a) otimização dos esforços na construção de sistemas diminuindo-se a necessidade de recompilar os ES;

b) facilidade para regerar qualquer fase ou parte do produto a qualquer momento;

c) viabilidade da análise do impacto das alterações previstas, inclusive em todas as ramificações de versões;

d) facilidade da construção do produto;

e) habilidade para, a qualquer momento, tirar um *flash* ou congelar o desenvolvimento do produto.

Finalmente, em relação ao processo as funções devem oferecer:

a) facilidades para transmitir informações para as pessoas envolvidas sobre os eventos relevantes;

b) facilidades para identificar as tarefas a serem realizadas e também como e quando elas serão completadas;

c) documentação gerada durante o desenvolvimento;

d) suporte ao modelo de ciclo de vida e à política de trabalho da organização.

4.7 Dificuldades de Implementação

Não existe disponível atualmente um sistema SCM que atenda todas as funcionalidades, tornando sua implementação uma tarefa um pouco complexa. Vários aspectos, desde o preço até o porte do sistema, devem ser considerados antes de se escolher qual sistema deverá ser implementado (Dart, 1992).

Pode-se definir um sistema completo de SCM como aquele que implementa todas as características citadas anteriormente. Em termos práticos sabe-se que nenhum sistema possui todas estas características. Logo, considera-se um sistema SCM, qualquer sistema que:

a) forneça alguma forma de controle de versão, identificação de configuração, estrutura ou modelagem do sistema a ser desenvolvido e

b) que tenha por objetivo realizar as funções de um SCM em algum nível.

Dentre as principais dificuldades na implementação de um sistema SCM, destaca-se os aspectos de porte, integração e amplitude do sistema SCM.

A metodologia de trabalho ou as ferramentas disponíveis devem ser consideradas, além do porte do sistema SCM. Logo, pode-se escolher um sistema de SCM que faça parte de um ambiente de desenvolvimento onde as funções do SCM são partes integrantes deste ambiente. Um exemplo deste tipo é o ambiente *Relational* (Feiler, 1991). Pode ainda o SCM não ser integrado ao ambiente de desenvolvimento, sendo as funções fornecidas através de uma ferramenta separada que é utilizada simultaneamente com outras ferramentas. Um exemplo deste tipo é *o RCS (Revision Control System)* (Tichy, 1985).

Esse nível de integração da ferramenta SCM ao ambiente de desenvolvimento determina a integração a nível de processo, onde o padrão de uso da ferramenta SCM será totalmente integrado ao padrão de uso do ambiente. Outra possibilidade é a integração da ferramenta a nível de sistema de forma que o sistema SCM seja utilizado com outras ferramentas do ambiente ou ainda a integração de banco de dados, pois a ferramenta SCM pode possuir o seu próprio banco de dados ao invés de acessar um banco de dados separado.

Outro problema é determinar qual a amplitude ou abrangência do SCM na organização. O sistema SCM pode ser implementado para a obtenção do controle a partir do final do ciclo de

desenvolvimento quando o produto já está pronto para ser entregue ao cliente e o controle será feito somente para a etapa de manutenção, desde o início do projeto até o início da fase de manutenção, ou ainda durante todo o ciclo de vida do produto.

Para a realização de qualquer alteração em um sistema existe a necessidade de número muito grande de procedimentos manuais, como preencher formulários de solicitação de alteração, esperar pela resposta do comitê e receber a aprovação, entre outros, adiando-se para o mais tarde possível o início da aplicação do SCM, fazendo com que cada membro da equipe de desenvolvimento se responsabilize pelo controle da sua própria produção (Dart, 1992).

Portanto, uma dificuldade na implementação do SCM é a mudança de cultura e pré-disposição dos usuários em seguirem as normas de utilização do sistema.

No entanto, a maioria dos procedimentos podem ser realizados de forma *on-line* e, sendo assim, podem ser rapidamente realizados. Se o sistema SCM possuir muitos desses procedimentos automatizados, o SCM poderá ser aplicado desde o início do ciclo de vida do desenvolvimento com menor grau de dificuldade (Feiler, 1991).

Há necessidade também de ficar clara a distinção entre processo e controle dentro do SCM. Um processo representa a seqüência de tarefas necessárias para executar o SCM. O processo é um plano que define o que precisa ser feito, quem vai fazer e como será feito. O suporte ao processo é uma função de gerenciamento.

Já o modelo de processo leva em consideração os procedimentos da organização e de seu modelo de ciclo de vida. O produto SCM é o resultado do processo. Um sistema SCM precisa fornecer funcionalidades para estes dois aspectos. Os sistemas, geralmente, fornecem suporte ao produto ou ao processo, mas geralmente não aos dois ao mesmo tempo (Dart, 1990).

Outro cenário de discussão está relacionado com a quantidade de automação que o SCM deverá fornecer. É possível realizar SCM sem qualquer tipo de facilidades *on-line*, mas, hoje em dia isto é ineficiente. O objetivo é automatizar tanto quanto possível as partes não-criativas do SCM, como por exemplo, formulários de solicitação de alteração por escrito, protocolo para a resposta e outros que podem ser implementados através de serviços *on-line*, onde o SCM fornece

os formulários automaticamente. Os sistemas SCM automatizam uma certa quantidade de funções e os usuários deverão implementar outros procedimentos de forma manual a fim de atender aos requisitos de se ter todas as funcionalidades de um SCM completo (Feiler, 1991).

5. QFD - Quality Function Deployment Desdobramento da função qualidade

Escola Federal de Engenharia de Itajubá - EFEI
Biblioteca MAUÁ BIM

5.1 Introdução

Neste capítulo é apresentada a ferramenta da qualidade QFD que orientará a elaboração do esquema para a implementação do SCM.

Enquanto as ferramentas disponíveis de engenharia medem a qualidade do projeto sob o ponto de vista do cliente interno das organizações, como pôr exemplo, a manufatura como cliente do projeto, o QFD mede a qualidade do projeto como função da satisfação dos clientes ou usuários do produto fornecido pela organização (Juran, 1988). Todas as operações da empresa em cada etapa do ciclo de vida do produto são avaliadas sob o ponto de vista do cliente.

O QFD consiste em uma forma sistemática de traduzir as necessidades do cliente em requisitos à serem providos pela organização que deverão estar presentes nos diversos estágios do ciclo de desenvolvimento do produto, desde a pesquisa e o desenvolvimento até a engenharia, suprimentos, produção, marketing, vendas e distribuição (Eureka, 1992).

No ambiente competitivo atual, os consumidores buscam produtos de qualidade que os satisfaçam em seu desempenho. Isto justifica a abordagem de Taguchi para o desenvolvimento de produtos e processos, que tenta minimizar perdas (alto valor agregado) melhorando consistência e desempenho (Bedworth, 1991).

O QFD pode ajudar a identificar as preocupações fundamentais com produtos e processos no que diz respeito às necessidades do cliente, enquanto os métodos de Taguchi, tais como projeto de experimentos, podem identificar que relações entre produtos e processos verdadeiramente existem, além de suas intensidades relativas e a natureza das relações. Métodos de Taguchi e QFD são ferramentas complementares.

Através da coleta de informações junto ao mercado ou consumidor final do produto estabelece-se uma série de matrizes que desdobram as necessidades e seus requisitos técnicos à elas relacionados, de forma a representar a especificação, desdobramento das partes e planejamentos dos processos e produção.

Na elaboração das matrizes *(figura 5.1)* envolvem-se todos os participantes de todas as etapas do ciclo de vida do produto, o que faz do QFD uma implementação da essência da Engenharia Simultânea (Boehm, 1991).

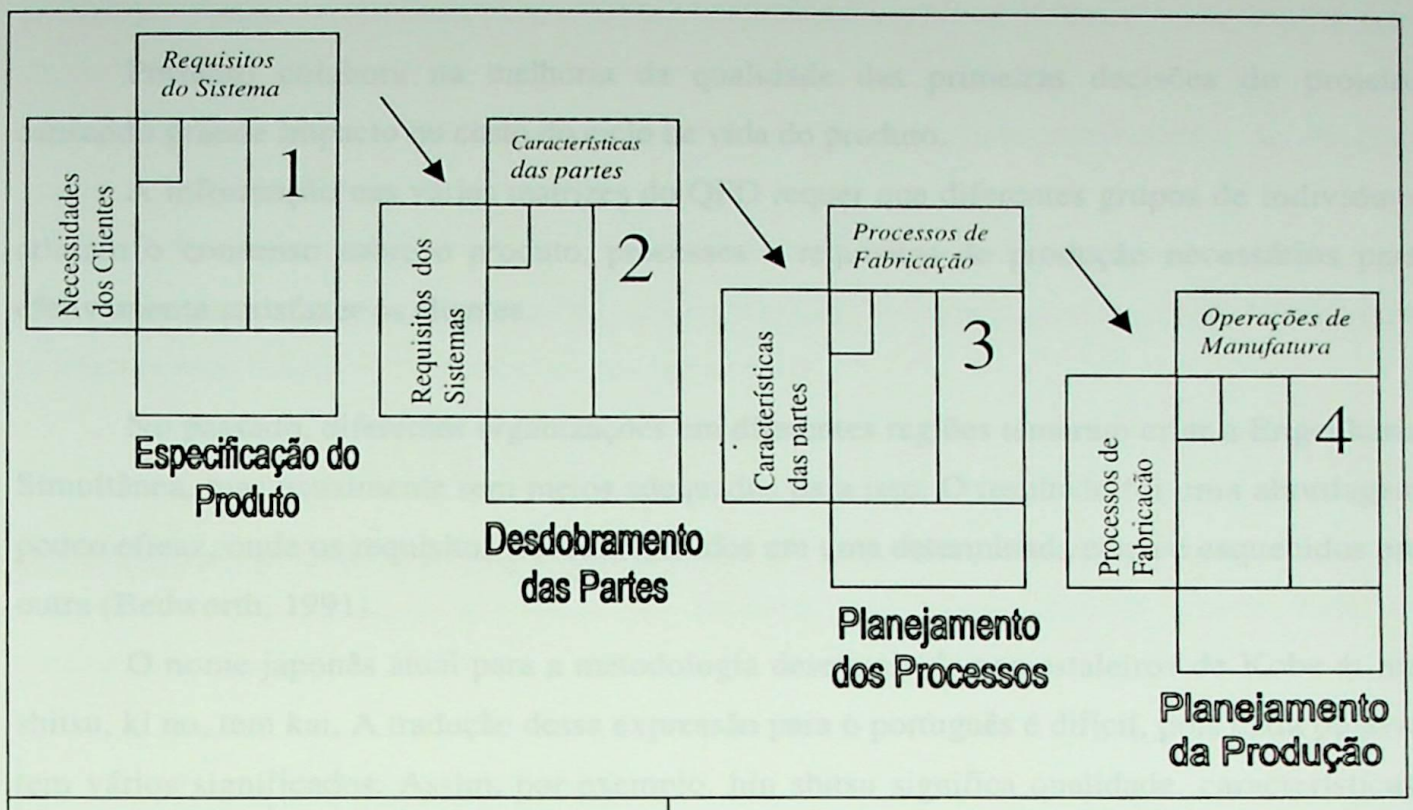


Figura 5.1 - As matrizes do QFD - Boehm, 1991

Entende-se pôr Engenharia Simultânea *(figura 5.2)* uma metodologia de detalhamento do projeto enquanto simultaneamente desenvolve a capacidade de produção, capacidade de suporte de campo e qualidade. A essência da Engenharia Simultânea é a integração do projeto do produto e planejamento de processo em uma atividade comum (Boehm, 1991).

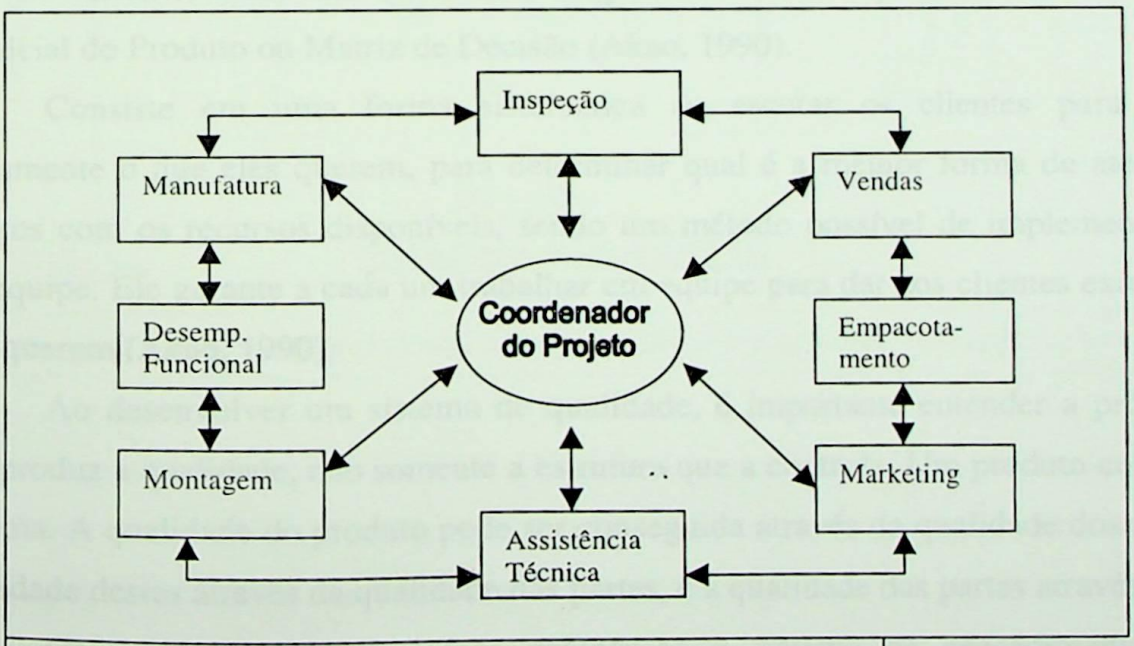


Figura 5.2 – Conceito da Engenharia Simultânea - Boehm, 1991

Portanto colabora na melhoria da qualidade das primeiras decisões do projeto, causando grande impacto no custo do ciclo de vida do produto.

A informação nas várias matrizes do QFD requer que diferentes grupos de indivíduos atinjam o consenso sobre o produto, processos e requisitos de produção necessários para efetivamente satisfazer os clientes.

No passado, diferentes organizações em diferentes regiões tentaram criar a Engenharia Simultânea, mas usualmente sem meios adequados para isso. O resultado foi uma abordagem pouco eficaz, onde os requisitos são considerados em uma determinada etapa e esquecidos em outra (Bedworth, 1991).

O nome japonês atual para a metodologia desenvolvida nos estaleiros de Kobe é: hin shitsu, ki no, tem kai. A tradução dessa expressão para o português é difícil, pois cada palavra tem vários significados. Assim, por exemplo, hin shitsu significa qualidade, características, qualidades ou atributos. Ki no significa função ou mecanização e as palavras mais próximas para tem kai são desenvolvimento, desdobramento, difusão ou evolução.

Desdobramento da função qualidade – QFD – é um nome bastante incompleto para uma poderosa ferramenta. Com ela pode-se resolver problemas, tomar decisões e fazer planejamentos. O QFD, durante muitos anos foi rotulado com uma série de outros nomes com Voz do Cliente, Casa da Qualidade, Engenharia voltada para o Cliente, Planejamento Matricial do Produto ou Matriz de Decisão (Akao, 1990).

Consiste em uma forma sistemática de escutar os clientes para compreender exatamente o que eles querem, para determinar qual é a melhor forma de atender aos seus desejos com os recursos disponíveis, sendo um método possível de implementar o trabalho em equipe. Ele garante a cada um trabalhar em equipe para dar aos clientes exatamente o que eles querem (Akao, 1990).

Ao desenvolver um sistema de qualidade, é importante entender a própria estrutura que produz a qualidade, não somente a estrutura que a controla. Um produto constitui-se num sistema. A qualidade do produto pode ser conseguida através da qualidade dos subsistemas, a qualidade destes através da qualidade das partes, e a qualidade das partes através do processo.

Como sistema de qualidade entende-se o sistema de procedimentos técnicos e administrativos para fornecer um produto com padrões de qualidade especificados (Feigenbaum, 1983). Juran define uma função qualidade como uma função que produz a

qualidade. Todas as atividades são funções que produzem ou contribuem para a qualidade (Juran, 1988).

Logo, o sistema qualidade pode ser definido como uma combinação de funções qualidade *(figura 5.3)*.

O desdobramento das funções qualidade é então, o desdobramento passo-a-passo mais detalhado das funções ou operações que produzem sistematicamente a qualidade com procedimentos objetivos preferenciais aos subjetivos.

Akao define o desdobramento da função qualidade como a conversão das necessidades dos clientes em requisitos do sistema (produto), desdobrando estes nas características de cada componente funcional do sistema, as características em processos e estes últimos nas suas respectivas operações. A qualidade do produto será obtida através dessa rede de desdobramentos (Akao, 1990).

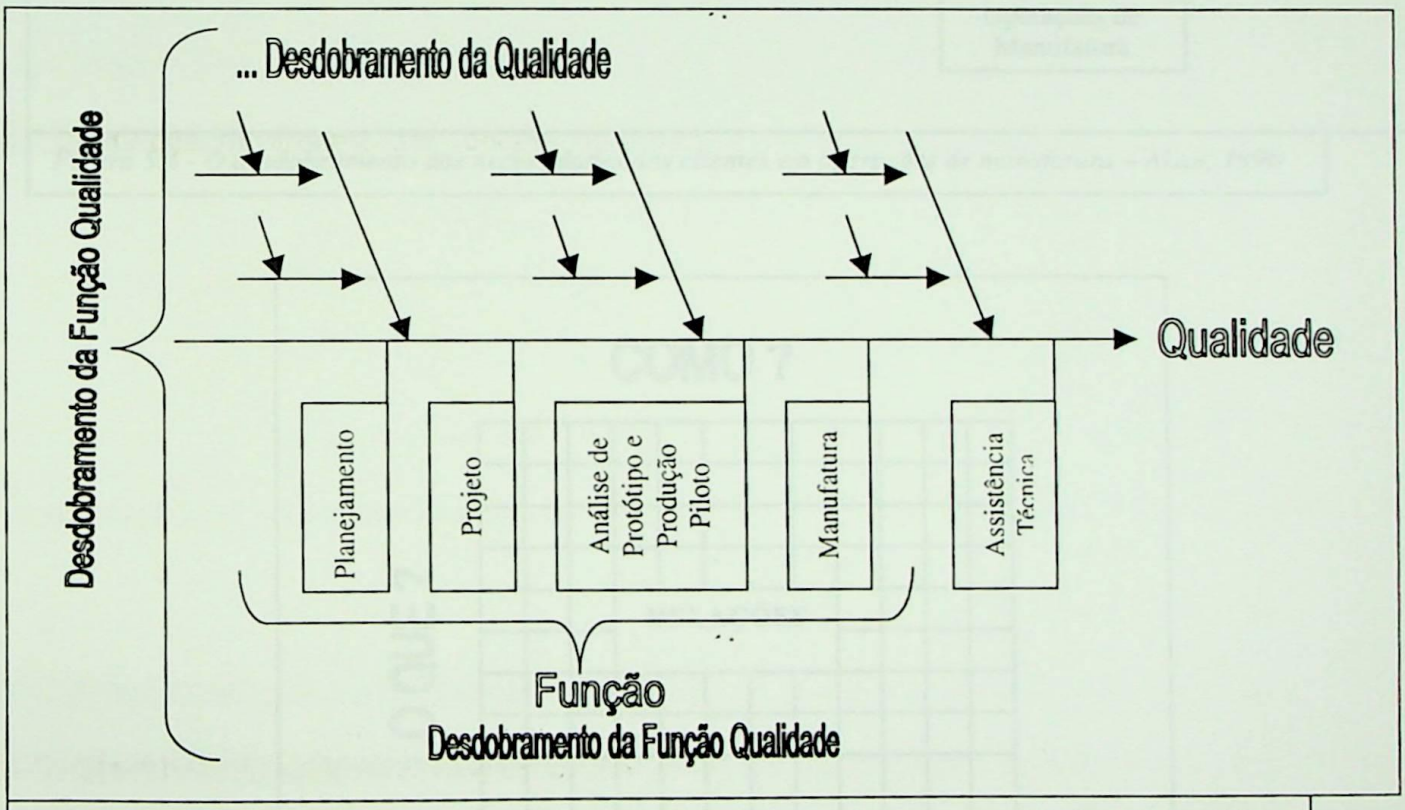


Figura 5.3 - O desdobramento da qualidade e o desdobramento da função qualidade – Akao, 1990

Segundo Eureka (1992) QFD é um sistema que traduz as necessidades do cliente em requisitos apropriados para a empresa, em cada estágio do ciclo de desenvolvimento do produto.

Cada uma das flechas na *figura 5.4* representa uma matriz que pode ser gerada, sendo um modo de conectar as necessidades dos clientes às operações de produção. Uma matriz do QFD *(figura 5.5)* pode ser usada para prover quatro itens de informação:

O que é importante para o cliente ?

Como pode ser provido ?

Quais são as relações existentes entre os o *que's* e os *como's* ?

Quanto tem de ser provido pelos *como's* para satisfazer os consumidores ?

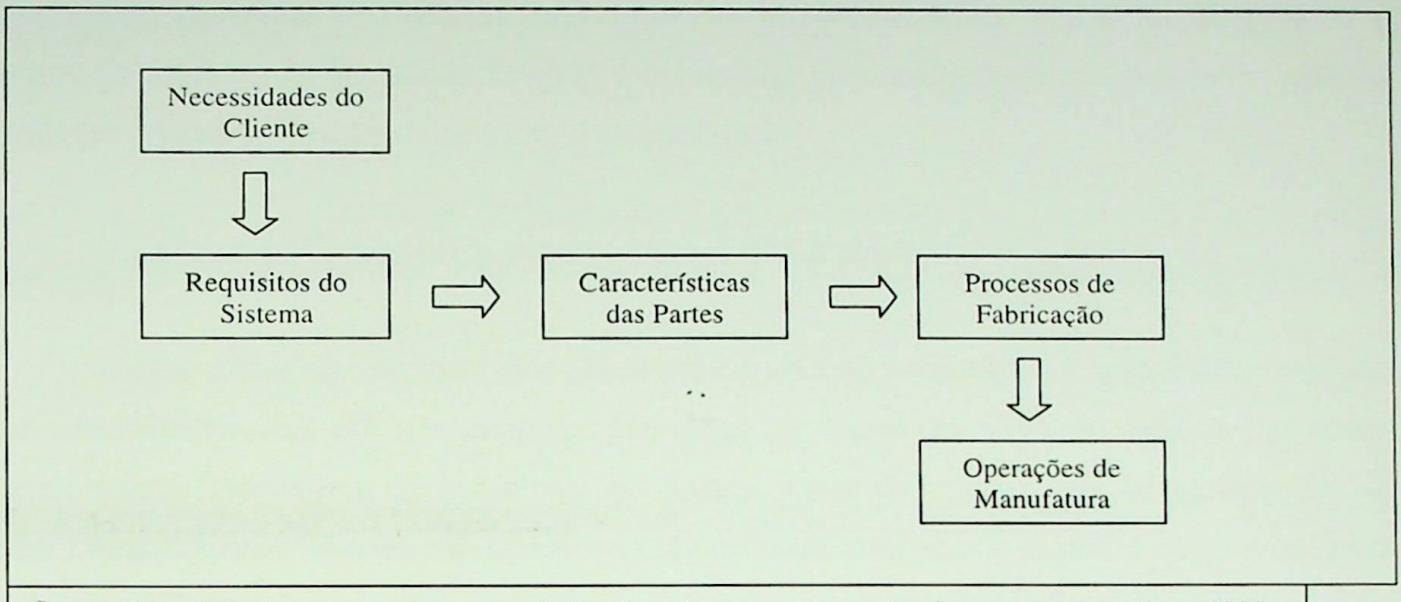


Figura 5.4 - O desdobramento das necessidades dos clientes em operações de manufatura – Akao, 1990

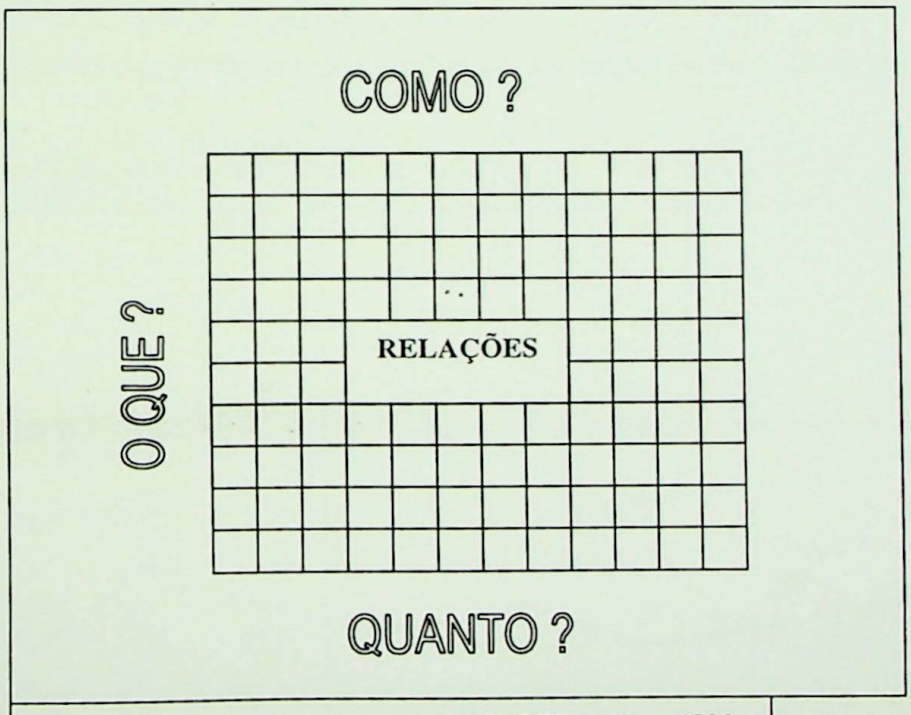


Figura 5.5 - O formato das matrizes do QFD – Akao, 1990

A abordagem de equipe é um ponto forte do QFD, onde os objetivos genéricos no desenvolvimento do produto são subdivididos em ações específicas através de um esforço global de todos os envolvidos. Isto é alcançado por uma série de matrizes e gráficos, que

desdobram as necessidades do cliente e os requisitos técnicos com elas relacionados, a partir do planejamento e do projeto do produto até o planejamento do processo e o chão de fábrica.

5.2 Os Objetivos do QFD

Trata-se de uma metodologia para o desenvolvimento da qualidade do projeto com o objetivo de satisfazer os clientes, traduzindo sua necessidades em requisitos de projeto e pontos de garantia da qualidade. O QFD é o caminho para assegurar a qualidade de projeto enquanto o produto está ainda no estágio de concepção.

O QFD possibilita principalmente (Akao, 1990) e (Mirshawka, 1990):

a) Que a boa interpretação dos objetivos em relação ao produto seja mantida, baseada nas necessidades dos clientes ao longo de todas as etapas do ciclo de vida do produto. Determinadas estratégias de marketing ou pontos fortes de vendas não se perdem ou se deterioram durante o processo de transformação do marketing para o projeto e do projeto para a produção. Pontos importantes no controle da qualidade na produção não são negligenciados; tudo o que foi considerado necessário para se obter o resultado final desejado está planejado nas matrizes do QFD.

Uma importante contribuição é a redução de reclamações sobre o produto, como ilustra a *figura 5.6.*

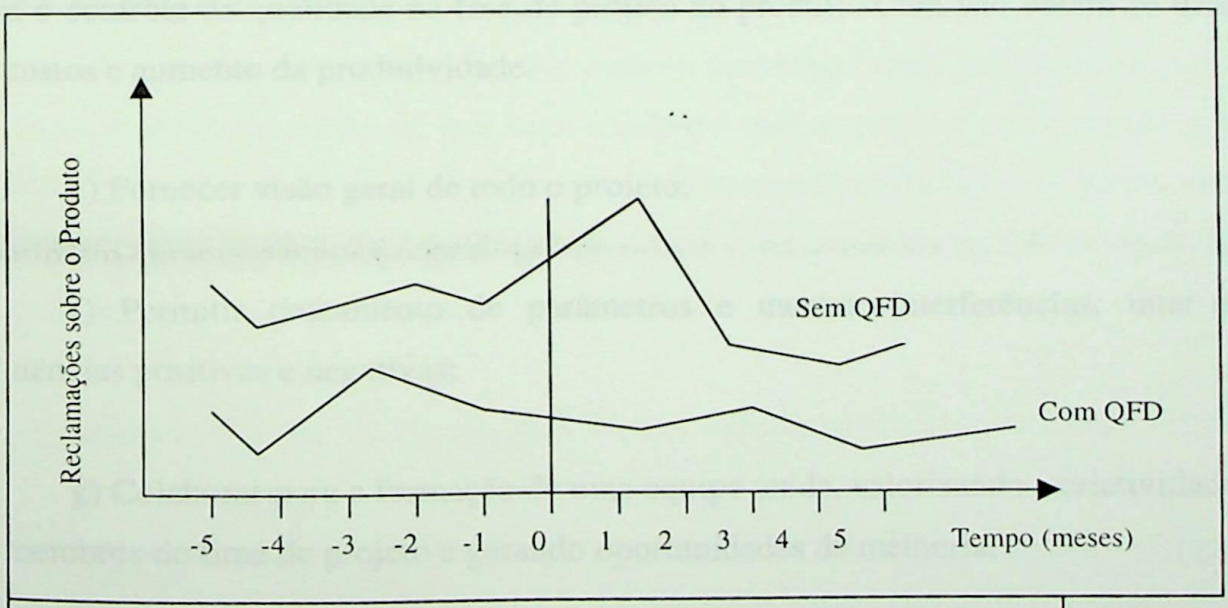


Figura 5.6 - Redução do número de reclamações utilizando QFD - Akao, 1990

b) Elaborar o desenvolvimento de produto em um ciclo menor, reduzindo também o seu tempo de lançamento. Ciclos de desenvolvimento do produto muito longos tornam

extremamente difícil a previsão das necessidades dos clientes. A redução destes ciclos ajuda as empresas a colocar produtos junto aos clientes de forma mais acurada e acabar com o empirismo na previsão de mercado. O QFD proporciona uma especificação integrada do produto, fazendo praticamente desaparecer a fase de reprojeto, reduzindo o ciclo de desenvolvimento do produto e consequentemente o tempo necessário para lançá-lo no mercado (Mirshawka, 1990).

c) Reduzir as mudanças de engenharia. Custo, prazo, e qualidade são vistos muitas vezes como elementos conflitantes. Entretanto, as melhores empresas japonesas aprenderam a otimizar com sucesso estes elementos aparentemente conflitantes, fazendo o mínimo de modificações. Isto possibilitou não só aumentar a participação no mercado e os lucros, mas também obter a lealdade dos clientes.

d) Obter custos menores e maior produtividade. Os japoneses concentram-se no planejamento e na prevenção de problemas e não na solução de problemas. Como 80% dos custos são conseqüência da fase de concepção do produto e os 20% restantes determinados pela etapa de manufatura, o maior esforço do time de projeto deve incidir nas primeiras etapas do ciclo de desenvolvimento do produto (Mirshawka, 1990).

A transição do enfoque reativo para o preventivo constitui um dos objetivos do QFD. O que era antes orientado para o controle da qualidade na fabricação passa a ser orientado para o controle de qualidade na fase de projeto do produto. Com isto obtém-se uma redução de custos e aumento da produtividade.

e) Fornecer visão geral de todo o projeto;

f) Permitir cruzamento de parâmetros e mostrar interferências, inter-relações e influências positivas e negativas;

g) Colaborar para a formação de uma equipe unida, valorizando a criatividade de todos os membros do time de projeto e gerando oportunidades de melhoria;

h) Facilitar o uso de maneira coerente e sistematizada de outras ferramentas, como Engenharia do Valor, Árvores de Objetivos, Diagramas de Causa e Efeito, entre outras.

i) Obrigar a participação disciplinada de todos os envolvidos no ciclo de desenvolvimento do produto nas diversas etapas.

Com o objetivo de garantir a qualidade de novos produtos torna-se necessário uma atenção especial não somente na qualidade negativa expressas pelas reclamações do consumidor, mas também às idéias não expressas de qualidade positiva nas necessidades do consumidor. Isto não acontece nos produtos existentes, pois as características da qualidade já são conhecidas. Entretanto, com novos produtos é necessário aprender o que garantir.

Entender as demandas de qualidades latentes e reais dos consumidores é o primeiro passo, estudando as reclamações dos clientes sobre os produtos existentes como uma abordagem analítica, desde o produto acabado até os processos de produção. Busca-se a identificação dos fatores que contribuem para os problemas. Nos novos produtos a abordagem deve ser a de buscar as qualidades que os consumidores querem no produto acabado, onde tais necessidades têm de ser incorporadas em planos da qualidade e qualidade de projeto que serão sistematicamente desdobradas até o fim do processo de produção, com atenção nas principais especificações de manufatura. Este método é conhecido como abordagem de projeto e o QFD foi concebido para implementá-lo tanto na melhoria dos produtos já existentes como na concepção de novos produtos (Akao, 1990).

Até poucas décadas atrás novos produtos e tecnologias eram levadas da Europa e dos Estados Unidos para o Japão, de modo que se pudesse ter produtos similares. Hoje a qualidade dos produtos japoneses é apreciada por todo o mundo. No entanto, quando ocorriam reclamações, os fatores que contribuíam para os problemas eram indicados em um diagrama de causa e efeito de modo de que uma completa análise pudesse ser efetuada e que ações pudessem ser adotadas para evitar recorrência. Esse exercício foi repetido várias vezes. Como resultado destas análises, a tecnologia começou a ser acumulada, melhor qualidade que a original foi obtida, e finalmente, a qualidade dos produtos japoneses excedeu aquelas da Europa e dos Estados Unidos.

Desdobrar a voz do cliente para melhor determinar as características importantes de um produto é pratica comum das melhores companhias japonesas. Seus engenheiros projetam e fabricam de acordo com tais objetivos, procurando na fase final, reduzir as variações desses objetivos. Os japoneses concentram-se na otimização do produto e do processo, não somente maximizando o desempenho, mas também reduzindo a variação. Como resultado têm-se um alto e consistente desempenho, de produto para produto e durante a sua vida útil. Isto é feito definindo *o que* fazer e *como* fazer, satisfazendo os clientes.

Portanto o QFD ajuda na obtenção do produto com a qualidade definida pelo cliente, pela clara definição dos objetivos das tarefas necessárias para alcançá-la. Embora o emprego do QFD não seja garantia de sucesso, sem dúvida é grande a probabilidade de atingí-lo, pois possibilita uma nova e melhor abordagem para o planejamento do produto.

A definição de qualidade adotada neste contexto é (Feigenbaum, 1983):

Satisfazer as necessidades dos clientes;

No momento certo e;

Com preço justo.

5.3 As etapas ou matrizes do QFD

Existem quatro etapas do QFD – conforme demonstrado *figura 5.1,* partindo das necessidades dos clientes até a determinação das operações de manufatura que produzem algo que satisfaça essas necessidades. As etapas, que geram as matrizes são:

1) Especificação do produto

O cliente auxilia a definir os requisitos do produto, conhecidos também como requisitos do sistema. A matriz QFD possibilita que o pessoal do projeto traduza as necessidades dos clientes em uma lista de itens *o que* (figura 2.5) (Akao, 1990). Com esta lista de itens finalizada, desenvolve-se outra parte da matriz. A equipe de projeto cria então diferentes formas para atender os requisitos e com isto surge a lista de itens *como*. Após uma avaliação, alguns desses itens vão para a próxima etapa. As necessidades dos clientes podem ser definidas como um conjunto de desejos expressos, latentes ou potenciais dos usuários finais do produto, sejam clientes externos à organização produtora, aqueles que comercializam o produto ou ainda os envolvidos nas etapas do ciclo de desenvolvimento do produto, como clientes internos da organização produtora.

Os requisitos de sistema são as características do produto final que, por exemplo, são encontradas em catálogo. Elas podem normalmente ser expressas por um número e uma unidade, sendo ainda chamadas por alguns autores de características da qualidade ou requisitos do projeto (Akao, 1990) e (Mirshawka, 1990).

Esta primeira matriz ilustra o relacionamento entre a qualidade real ou verdadeira - conforme as solicitações do cliente - com a função e as características da qualidade ou requisitos de sistema resultando nas características iniciais do produto (Akao, 1990). É a matriz básica para o projeto da qualidade, onde a qualidade desejada pelo consumidor é convertida, através de um completo processo, em características de contrapartida, por meio de raciocínio, tradução e transferência.

A matriz QFD é um gráfico que possibilita: analisar sistematicamente as estruturas da qualidade solicitada pelos clientes em suas próprias palavras, indicar a relação entre estas qualidades desejadas e certos requisitos de sistema, converter as necessidades dos clientes nas

características de contrapartida ou requisitos de sistema e desenvolver a qualidade do projeto (Akao, 1990).

Além dos itens *o que* e *como*, essa matriz *(figura 5.7)* contém os seguintes elementos:

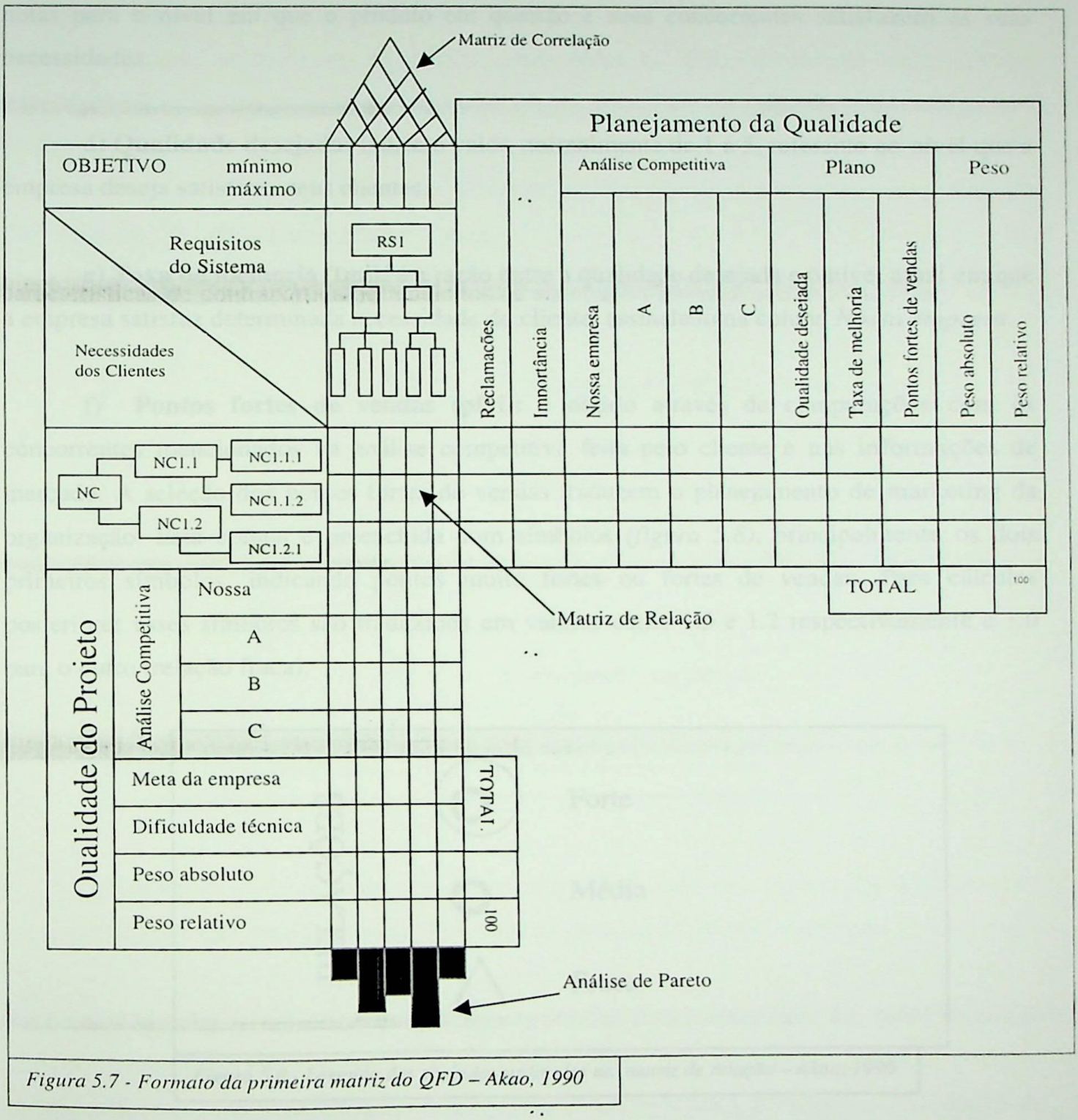


Figura 5.7 - Formato da primeira matriz do QFD – Akao, 1990

a) **Reclamações**: refere-se ao número de reclamações relativas à determinada necessidade dos clientes.

b) **Importância (impi):** é uma indicação qualitativa que representa quanto aquela necessidade do cliente influencia na sua decisão de compra. Pode assumir valores de 1 a 5.

c) **Análise competitiva feita pelo cliente:** através de um questionário o cliente atribui notas para o nível em que o produto em questão e seus concorrentes satisfazem as suas necessidades.

d) **Qualidade desejada:** indica o valor, normalmente de 1 a 5, referente ao nível que a empresa deseja satisfazer seus clientes.

e) **Taxa de melhoria (tmi):** é a razão entre a qualidade desejada e o nível atual em que a empresa satisfaz determinada necessidade do cliente, assinalado na coluna *Nossa empresa*.

f) **Pontos fortes de vendas (pfvi):** é obtido através de comparações com os concorrentes mencionados na análise competitiva feita pelo cliente e nas informações de mercado. A seleção dos pontos fortes de vendas traduzem o planejamento de marketing da organização. Esta coluna é preenchida com símbolos *(figura 5.8)*, principalmente os dois primeiros símbolos, indicando pontos muito fortes ou fortes de vendas. Para cálculos posteriores esses símbolos são traduzidos em valores como 1.5 e 1.2 respectivamente e 1.0 para o outro (relação fraca).

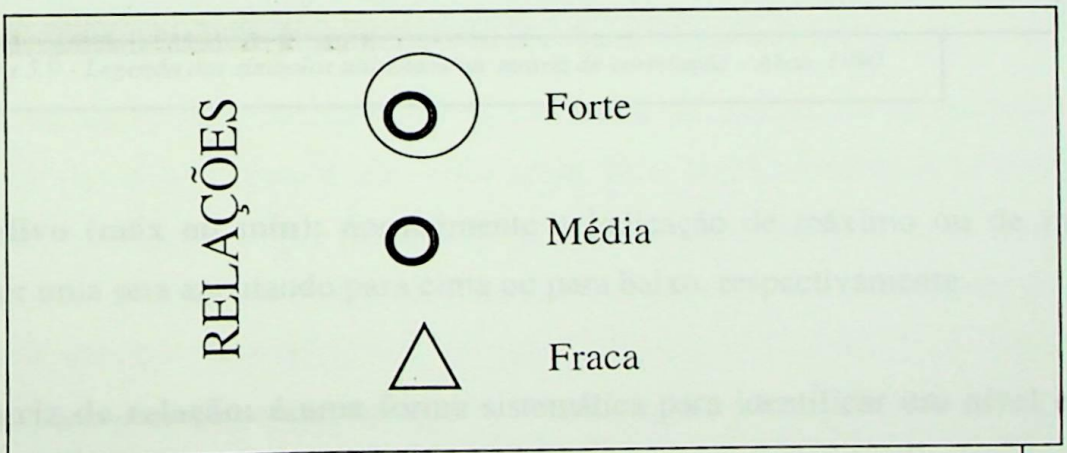


Figura 5.8 - Legenda dos símbolos utilizados na matriz de relação – Akao, 1990

g) **Peso absoluto das necessidades dos clientes (PANci)**

$$PANci = impi \times tmi \times pfvi$$

h) **Peso relativo das necessidades dos clientes (PRNci)**

$$PRNci = PANci \times 100 / iPANci$$

i) **Matriz de correlação:** é uma matriz triangular que apresenta a relação ou a dependência que existe entre os vários *como* entre si. Essa matriz de correlação é normalmente utilizada somente na primeira matriz do QFD. As células desta matriz são preenchidas com símbolos que expressam o grau de correlação *(figura 5.9)*. A correlação positiva ocorre quando os requisitos são diretamente proporcionais e a negativa ocorre quando os requisitos são inversamente proporcionais.

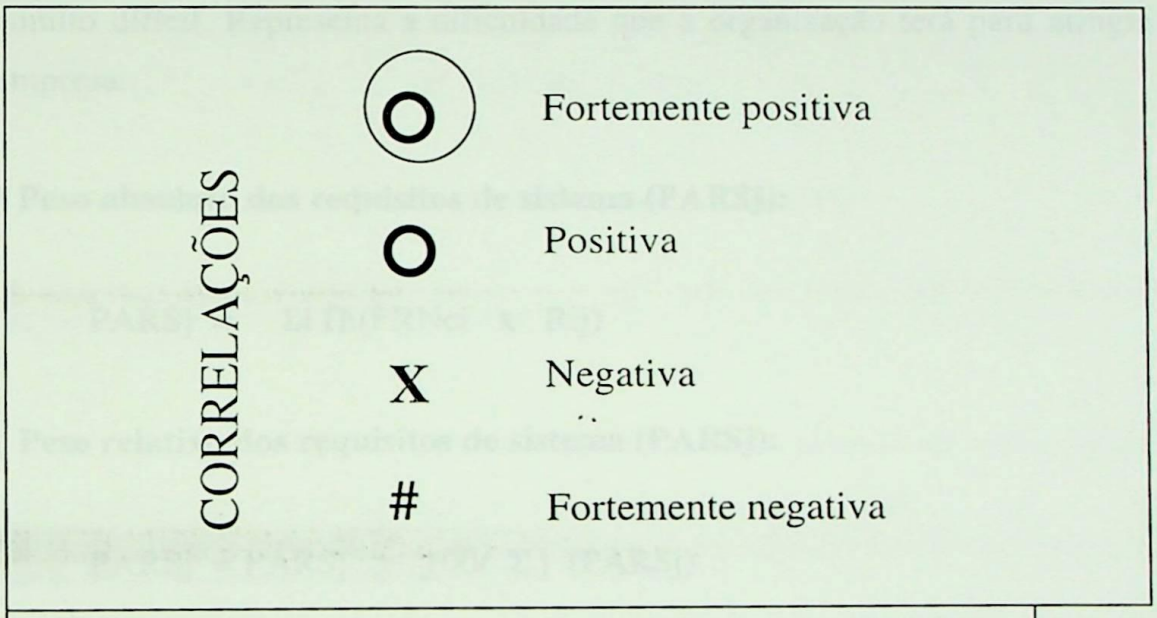


Figura 5.9 - Legenda dos símbolos utilizados na matriz de correlação – Akao, 1990

j) **Objetivo (máx ou mín):** normalmente a indicação de máximo ou de mínimo é representada por uma seta apontando para cima ou para baixo, respectivamente.

k) **Matriz de relação:** é uma forma sistemática para identificar um nível e relação (Rij) ou dependência entre os itens *o que* e os itens *como*, que nesta primeira matriz referem-se respectivamente às necessidades dos clientes e aos requisitos de sistema. As células da matriz de relação são preenchidas com os símbolos já apresentados na figura 2.8. Para cálculos futuros é atribuído o valor 9 à relação forte, o valor 3 à relação média e o valor 1 à uma relação fraca.

l) **Análise competitiva técnica:** é estabelecida uma comparação de como os produtos da organização produtora *Nossa empresa* e seus concorrentes *A,B,C* preenchem os requisitos de sistema. Essa comparação pode ser quantitativa, registrando-se valores numéricos e unidades para cada requisito de cada produto ou qualitativa, atribuindo-se notas de 1 a 5 para o produto em questão e seus concorrentes.

m) **Meta da empresa:** é um valor do requisito de sistema e sua unidade que a organização deseja atingir.

n) **Dificuldade técnica:** é uma indicação qualitativa, atribuindo-se notas entre 1 e 5, onde o valor 1 corresponde à muito fácil, 2 fácil, 3 dificuldade mediana, 4 difícil e 5 significa muito difícil. Representa a dificuldade que a organização terá para atingir o valor meta da empresa.

o) **Peso absoluto dos requisitos de sistema (PARSj):**

$$PARSj = \Sigma i \, \Pi i(PRNci \times Rij)$$

p) **Peso relativo dos requisitos de sistema (PARSj):**

$$PARSj = PARSj \times 100 / \Sigma j \, (PARSj)$$

q) **Análise de Pareto:** seleciona-se os requisitos de sistema de maior peso relativo cuja soma destes pesos seja igual a um valor entre 50 e 100%, determinado pelo usuário (Mirshawka, 1990). Espera-se que poucos requisitos possam fornecer essa soma. Esses requisitos serão então desdobrados em características das partes na segunda matriz do QFD.

2) Desdobramento das partes

Os requisitos do sistema identificados e selecionados na etapa anterior permitirão agora a identificação das características críticas das partes, ou seja, os itens *como*. Nesta matriz serão identificadas estas características com os requisitos de sistema a ela relacionados, neste caso, os itens *o que*.

No entanto, a obtenção sistemática das características das partes a partir dos requisitos de sistema não é direta, principalmente quando o projeto trata de um novo produto. O resultado desse processo é apresentado na *figura 5.10.*

Essa matriz contém os seguintes elementos:

a) **Grau de importância (GCPj):** indica a importância técnica da característica para a obtenção do requisito de sistema sob o ponto de vista de quem produz. É normalmente expressa numa escala de 1 a 5 onde 1 significa muito importante, 2 pouco importante, 3 uma importância média, 4 grande importância e 5 uma enorme importância.

b) **Peso absoluto das características das partes: (PACPj)**

$$PACPj = GCPj \times \Sigma i \, \Pi i(PRrsi \times Rij)$$

c) **Peso relativo das características das partes: (PRCPj)**

$$PRCPj = PACPj \times 100 / \Sigma j \, (PACPj)$$

d) **Valor atual:** é o valor da característica da parte do produto até então fornecido pela organização. Normalmente é designado por um número e uma unidade.

e) **Valor desejado:** é o valor ou nível desejado da característica da parte do produto atual melhorado ou de um novo produto a ser fornecido pela organização. Normalmente é designado por um número e uma unidade.

f) **Grau de dificuldade:** é uma indicação qualitativa, atribuindo-se notas (por exemplo, de 1 a 5, sendo 1 - muito fácil, 2 - fácil, 3 - dificuldade mediana, 4 - difícil, e 5 - muito difícil) para a dificuldade que a organização terá para atingir o valor desejado da característica da parte.

g) **Características *gargalo*:** são as características que possuem dificuldade de engenharia ou uma nova tecnologia ainda não dominada pela organização. As características gargalo são simplesmente assinaladas nessa linha.

h) **Peso das características *gargalo*:** é o produto do grau de dificuldade pelo peso absoluto da característica da parte.

Método para conformidade: indica o caminho a ser seguido para eliminar o *gargalo*.

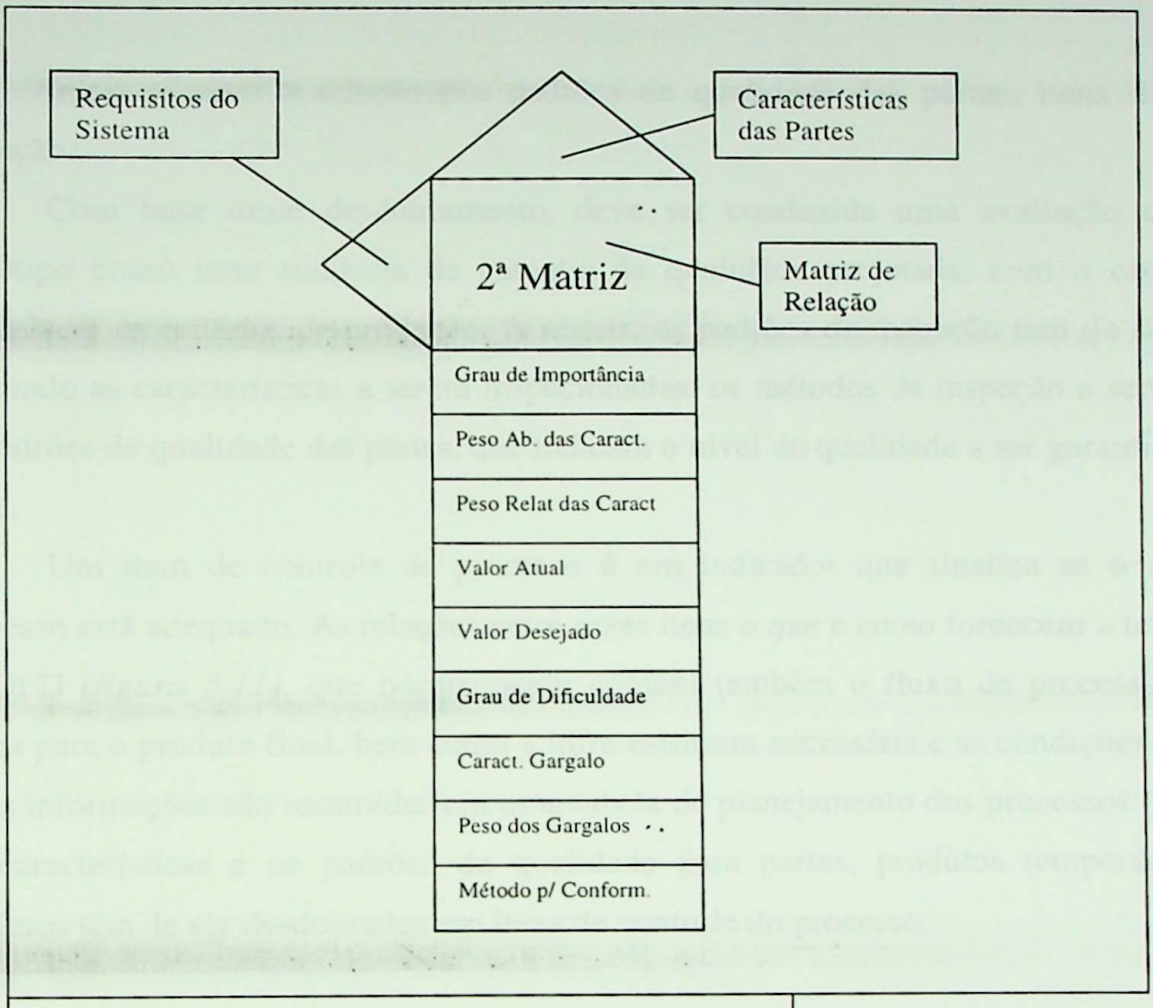


Figura 5.10 - Formato da segunda matriz do QFD – Akao, 1990

As células da matriz de relação devem ser preenchidas com os mesmos símbolos indicados na *figura 5.9*, podendo-se acrescentar ainda flechas indicando se os itens *o que* e *como* são direta ou inversamente proporcionais (Akao, 1990).

3) Planejamento dos processos

Para obtenção dos processos de fabricação e respectivos itens de controle (itens *o que* da terceira matriz), são necessários as características críticas das partes (itens *como* da segunda matriz).

A terceira matriz do QFD, também chamada por alguns autores de carta de Garantia da Qualidade identifica os itens de controle que têm de ser incorporados no projeto do processo

para implementar as características críticas das partes que estão sendo desdobradas. Participam preferencialmente da elaboração dessa matriz as equipes de engenharia do produto, de engenharia de processo e de produção, além do pessoal de projeto composto anteriormente (Akao, 1990) e (Mirshawka, 1990).

Inicia-se com a seleção dos padrões de qualidade das partes, itens e padrões de inspeção.

Com base neste desdobramento, deve ser conduzida uma avaliação completa do protótipo como uma auditoria de garantia da qualidade projetada, com o objetivo de se estabelecer os critérios de produção. A seguir, os padrões de inspeção tem de ser definidos, incluindo as características a serem inspecionadas, os métodos de inspeção a serem usados e os padrões de qualidade das partes, que indicam o nível de qualidade a ser garantida para cada parte.

Um item de controle de processo é um indicador que sinaliza se o resultado do processo está adequado. As relações entre esses itens *o que* e *como* fornecem a terceira matriz do QFD *(figura 5.11)*, que normalmente contém também o fluxo de processo da matéria prima para o produto final, bem como a infra-estrutura necessária e as condições de processo. Estas informações são resumidas em uma tabela de planejamento dos processos *(figura 5.12)*. As características e os padrões da qualidade para partes, produtos temporários e semi-produtos têm de ser desdobrados em itens de controle do processo.

Os critérios para desdobrar os processos são os maiores pesos relativos e o grau de dificuldade.

As células da matriz de relação devem ser preenchidas com os mesmos símbolos indicados na *figura 2.9*, podendo-se acrescentar ainda flechas indicando se os itens *o que* e *como* são direta ou inversamente proporcionais (Akao, 1990).

O formato apresentado na *figura 2.11* para a Matriz de Planejamento dos Processos não é consensual.

Alguns autores (Akao, 1990) e (Mirshawka, 1990) apresentam a terceira matriz como uma tabela de planejamento de processos representada na *figura 2.12*.

Escola Federal de Engenharia de Itajubá - EFEI
Biblioteca
MAUÁ
BM

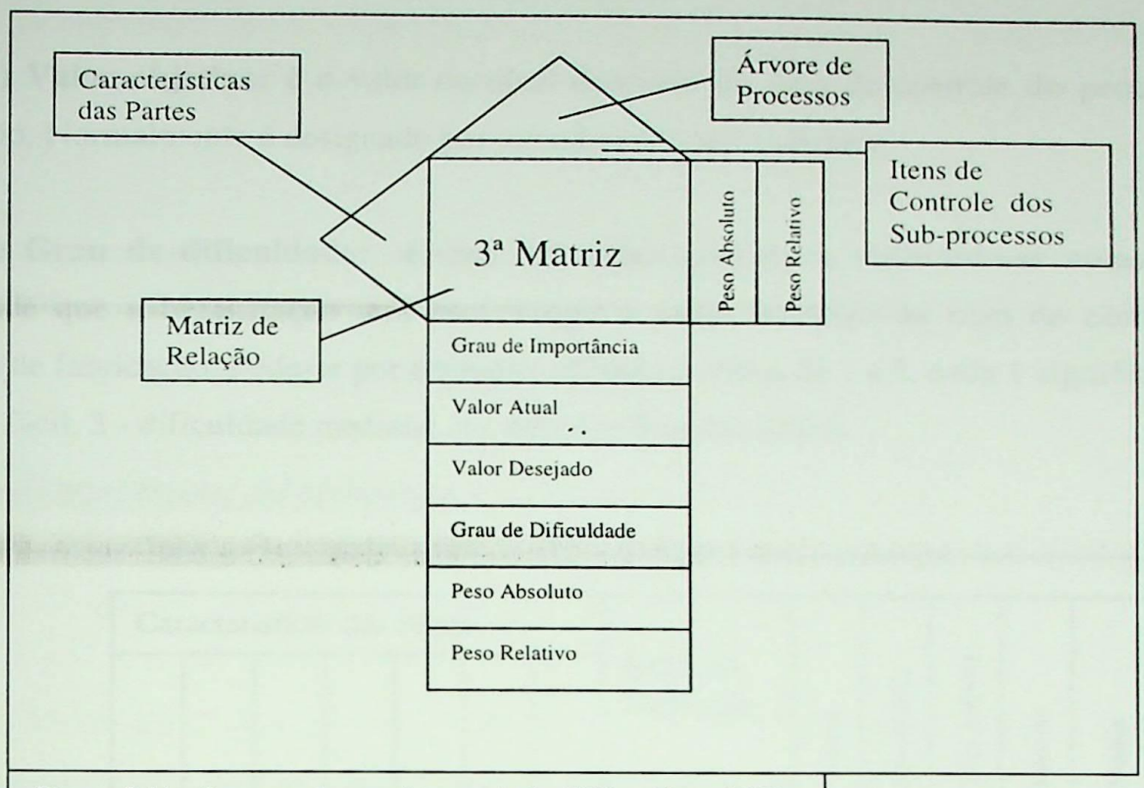


Figura 5.11 - Formato da terceira matriz do QFD – Akao, 1990

Essa matriz é um importante dispositivo para transmissão de informação entre projeto e produção, contendo ainda os seguintes elementos:

a) **Grau de importância (GPCj):** indica a importância técnica dos itens de controle do processo para a obtenção da característica das partes, sob o ponto de vista de quem produz. É normalmente expresso numa escala de 1 a 5, onde 1 é muito pouco importante, 2 pouco importante, 3 importância média, 4 grande importância e 5 enorme importância.

b) **Peso absoluto das processos de fabricação (PAPCj):**

$$PAPCj = GPCj \times \Sigma i \, \Pi i(PRcpi \times Rij)$$

c) **Peso relativo dos processos de fabricação (PRPCj):**

$$PRPCj = PAPCj \times 100 / \Sigma j \, (PAPCj)$$

d) **Valor atual:** é o valor ou nível do item de controle do processo de fabricação. Normalmente é designado por um número e uma unidade.

e) **Valor objetivo:** é o valor ou nível desejado do item de controle do processo de fabricação. Normalmente é designado por um número e uma unidade.

f) **Grau de dificuldade:** é uma indicação qualitativa, atribuindo-se notas para a dificuldade que a organização terá para atingir o valor desejado do item de controle do processo de fabricação. Pode-se por exemplo, utilizar-se notas de 1 a 5, onde 1 significa muito fácil, 2 - fácil, 3 - dificuldade mediana, 4 - difícil, e 5 muito difícil.

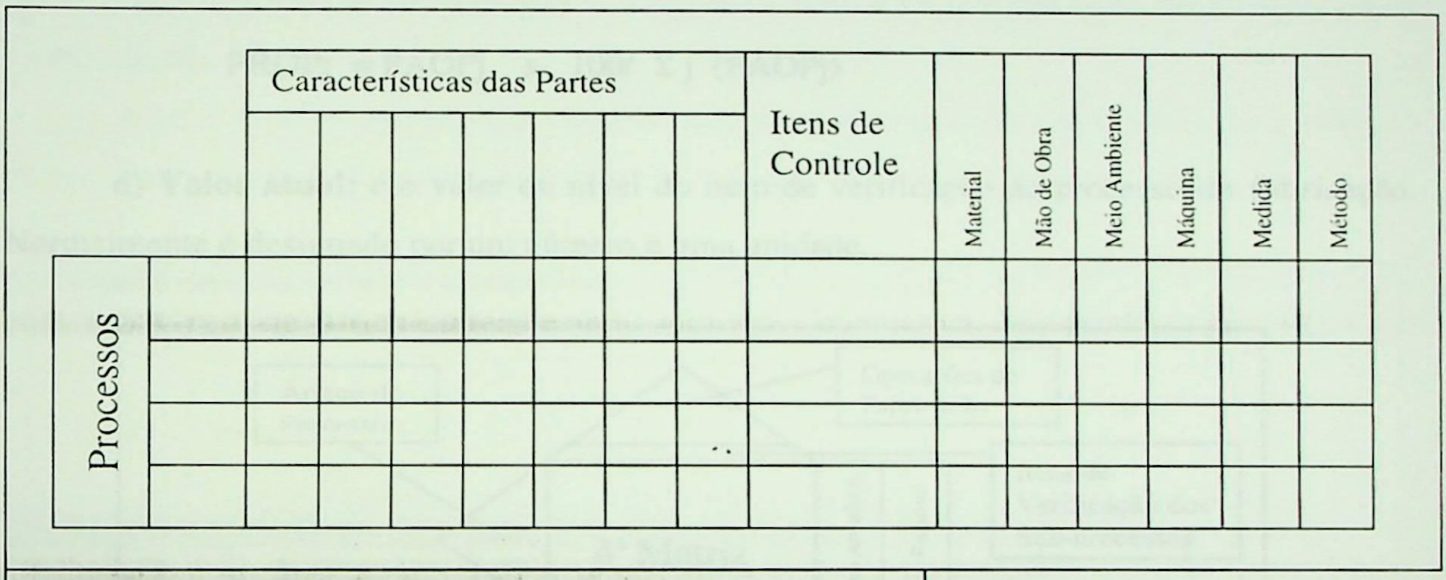


Figura 5.12 - Formato alternativo da terceira matriz do QFD – Akao, 1990

4) Planejamento da produção

Cada item de controle de processo da terceira matriz (itens *o que* da quarta matriz) é conseqüência de um conjunto de itens de verificação realizados durante as operações de manufatura (itens *como* da quarta matriz) que compõem o processo e realizam o produto. Um item de verificação de um processo é um fator que deve ser monitorado e que influencia o resultado do processo.

O relacionamento entre esses itens *o que* e *como* compõem a quarta matriz do QFD *(figura 5.13),* possuindo ainda os seguintes elementos:

a) **Grau de importância (GPCj):** indica a importância técnica dos itens de verificação do processo para a obtenção dos itens de controle de processo, sob o ponto de

vista de quem produz. É normalmente expresso numa escala de 1 a 5, onde 1 é muito pouco importante, 2 - pouco importante, 3 - importância média, 4 - grande importância e 5 enorme importância.

b) **Peso absoluto das operações de manufatura (PAOPj):**

$$PAOPj = GOPj \times \Sigma i\ \Pi i(PRpci \times Rij)$$

c) **Peso relativo das operações de manufatura (PROPj):**

$$PROPj = PAOPj \times 100 / \Sigma j\ (PAOPj)$$

d) **Valor atual:** é o valor ou nível do item de verificação do processo de fabricação. Normalmente é designado por um número e uma unidade.

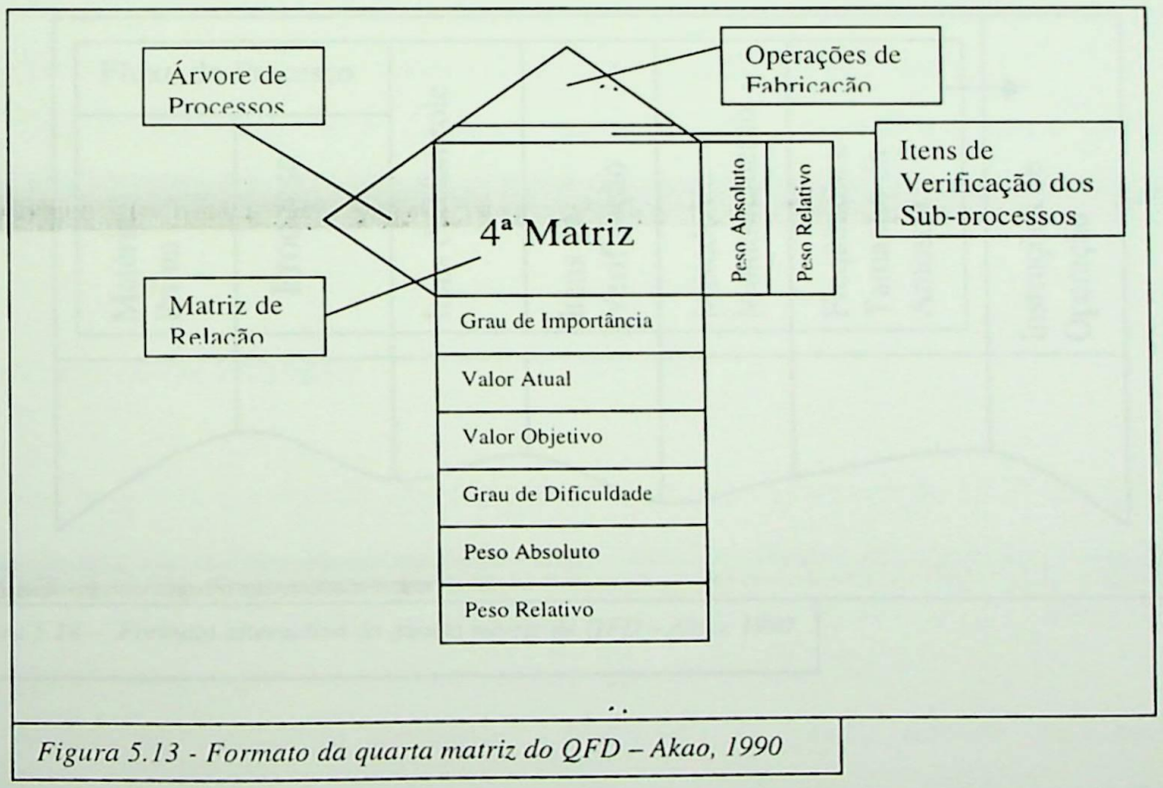


Figura 5.13 - Formato da quarta matriz do QFD – Akao, 1990

e) **Valor objetivo:** é o valor ou nível desejado do item de verificação do processo de fabricação. Normalmente é designado por um número e uma unidade.

f) **Grau de dificuldade:** é uma indicação qualitativa, atribuindo-se notas para a dificuldade que a organização terá para atingir o valor desejado do item de controle do

processo de fabricação. Pode-se por exemplo, utilizar-se notas de 1 a 5, onde 1 significa muito fácil, 2 - fácil, 3 - dificuldade mediana, 4 - difícil, e 5 muito difícil.

Os critérios para que itens *como* de uma etapa tornem-se itens *o que* da fase seguinte são: importância sob o ponto de vista do cliente, nova tecnologia e alto risco para a organização.

As células da matriz de relação devem ser preenchidas com os mesmos símbolos indicados na figura 2.9, podendo-se acrescentar ainda flechas indicando se os itens *o que* e *como* são direta ou inversamente proporcionais (Akao, 1990).

O formato apresentado na *figura 5.14* para a Matriz de Planejamento da Produção não é consensual.

Alguns autores apresentam a quarta matriz como uma carta de controle da qualidade de processos como representada na *figura 5.14* (Akao, 1990) e (Mirshawka, 1990).

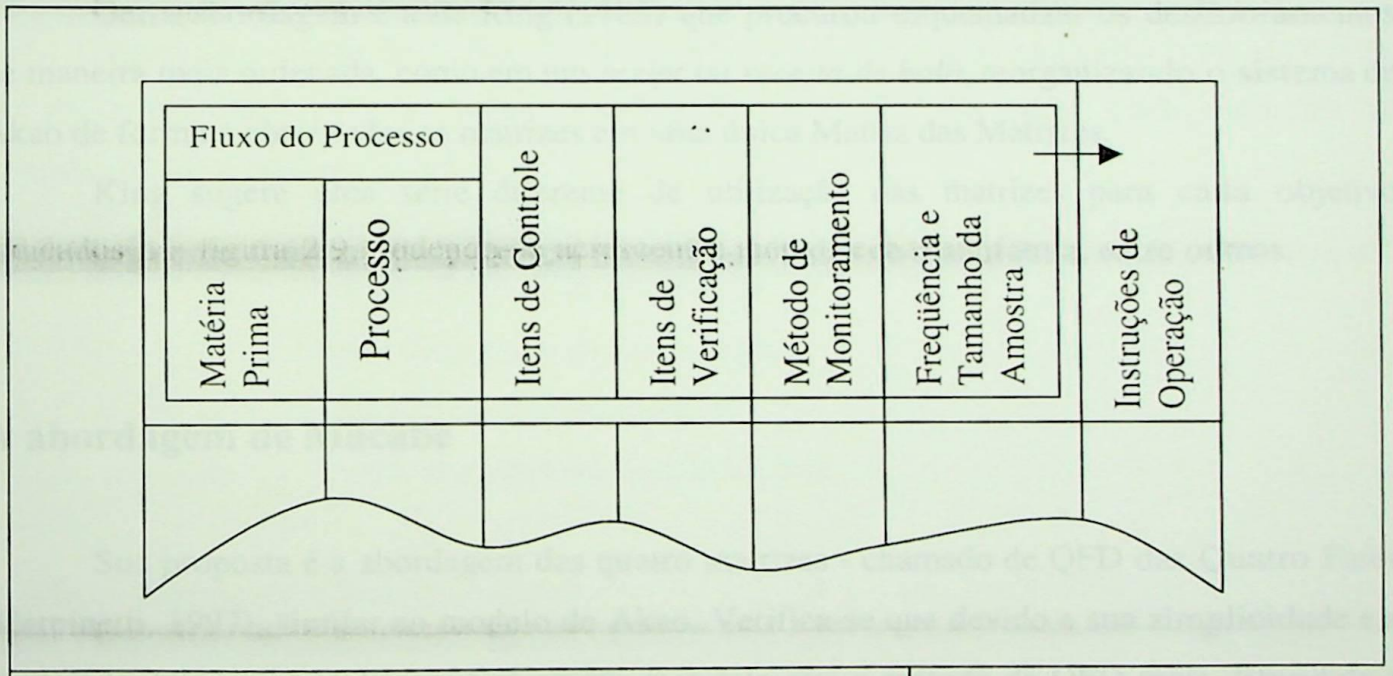


Figura 5.14 – Formato alternativo da quarta matriz do QFD – Akao, 1990

5.4 Outras abordagens de QFD

As matrizes originais de Akao foram sofrendo algumas adaptações conforme a necessidade específica do tipo de indústria e seus processos ou devido características muito particulares de um ou outro produto. Por isso surgiram novas abordagens do QFD, diferindo entre si nos aspectos de determinação dás etapas propostas para a execução dos desdobramentos, sendo mais ou menos abrangentes, no número de matrizes utilizadas para cada uma dessas etapas e no uso de diferentes ferramentas auxiliares.

Em síntese, a diferenciação básica está no modelo conceitual - caminho que o estudo deve percorrer para alcançar o objetivo desejado - desenvolvido em cada uma das abordagens.

A abordagem de King

Outra abordagem é a de King (1989) que procurou esquematizar os desdobramentos de maneira mais ordenada, como em um *script* ou *receita de bolo*, reorganizando o sistema de Akao de forma a obter todas as matrizes em uma única Matriz das Matrizes.

King sugere uma série diferente de utilização das matrizes para cada objetivo específico, como confiabilidade, custos, inovação métodos de manufatura, entre outros.

A abordagem de Macabe

Sua proposta é a abordagem das quatro matrizes - chamado de QFD das Quatro Fases (Carpinetti, 1997), similar ao modelo de Akao. Verifica-se que devido a sua simplicidade e a facilidade de acesso a bibliografias americanas este seja o método de QFD mais difundido e que no Brasil este é o que tem recebido maior número de adeptos.

As quatro fases desta abordagem constituem em quatro matrizes que direcionam o desenvolvimento do produto ou serviço, desde os requisitos dos consumidores até a fabricação, sem no entanto, preocupar-se com objetivos mais específicos como desdobramento de custos ou de confiabilidade considerados no método de Akao. Este fato tem sido alvo de críticas pelos estudiosos japoneses devido a sua limitação, mas de qualquer forma, tem tido grande aceitação junto às empresas.

A abordagem de Carpinetti

Nesta abordagem Carpinetti (1997) ressalta duas das versões de QFD existentes: o QFD-estendido e o QFD das quatro ênfases. O QFD-estendido foi criado a partir da versão das Quatro Fases proposta por Macabe enquanto o QFD das Quatro Ênfases foi criado por Akao e Mizuno e difundida pela JUSE (Union of Japanese Scientists and Engeneers).

O QFD-estendido se diferencia pela forma com que conduz a tomada de decisões, através das relações de extração enquanto o QFD das Quatro Ênfases privilegia a identificação das relações e a conversão de parâmetros.

Carpinetti propõe uma abordagem de aplicação da metodologia QFD que sintetiza a lógica das versões do QFD-Estendido e do QFD da Quatro Ênfases. Essa abordagem poderá potencializar o poder da metodologia QFD como gestora do processo de desenvolvimentos de produtos e serviços, pois o QFD-Estendido pode ser um condutor da determinação de parâmetros e o QFD das Quatro Ênfases um condutor da identificação das prioridades.

Nessa proposta os desdobramentos são executados unindo-se as quatro fases da versão do QFD-Estendido com a versão das Quatro Ênfases. Isso gera uma estrutura de desdobramentos com cinco ênfases:

1. Desdobramento do Produto;
2. Desdobramento da Qualidade,
3. Desdobramento da Tecnologia,
4. Desdobramento do Custo e
5. Desdobramento da Confiabilidade.

A abordagem de Cheng

Cheng (1995) propõe uma abordagem de QFD para a indústria de serviços, onde o detalhamento do projeto de serviços deve responder os seguintes questionamentos:

- Quais os processos necessários para atender à Qualidade Projetada?
- Quais os materiais ou recursos necessários para a execução desses processos?
- Como devem ser os procedimentos para a execução desses processos?

Sua proposta apresenta um método de QFD composto de três matrizes, onde a qualidade exigida pelo cliente e/ou pelas normas relacionam-se com as características da qualidade ou atributos que o serviço deve ter.

Os processos são então identificados e relacionados com as características da qualidade, ou seja, estabelece-se uma relação das características da qualidade (itens o que) com os processos (itens como). Esses processos são desdobrados em procedimentos ou atividades, porém não constituirão uma nova matriz.

A terceira matriz refere-se aos materiais ou recursos necessários para o cumprimento dos processos. Os recursos físicos e humanos são identificados e relacionados com os processos.

Os componentes das matrizes de Cheng são os mesmos já descritos das matrizes de Akao.

A abordagem de Cheng *(figura 5.15)* foi selecionada para a elaboração do esquema de implementação do SCM - objetivo deste trabalho - devido ao fato de que o gerenciamento da configuração de sistemas possui características de produção de serviços (Nóbrega, 1997).

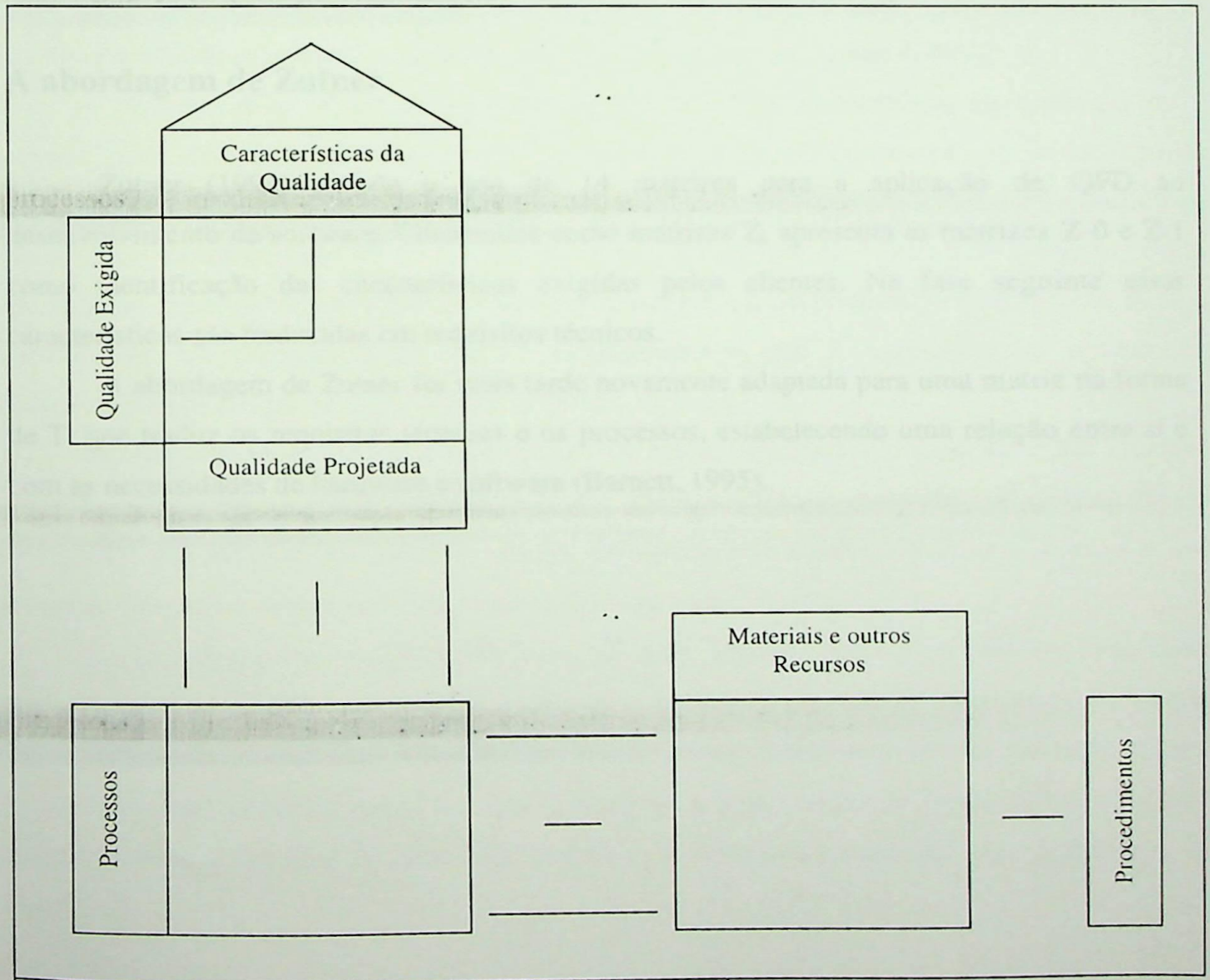


Figura 5.15 - As matrizes do QFD de Cheng - Cheng, 1995

Juran (1988) define serviço como sendo o trabalho desempenhado por alguém. O trabalho pode ser dirigido a um consumidor, a uma instituição ou a ambos.

Ele existe para proporcionar melhor desempenho aos clientes na satisfação de suas necessidades. Algumas dessas necessidades (como energia elétrica) podem ser obtidas somente através de grandes empresas centralizadas que têm capacidade para congregar tecnologia e investimento. Sem elas, essas necessidades certamente não seriam atendidas.

Também pode-se dizer que outros serviços existem para satisfazer uma enorme variedade de necessidades humanas, psicológicas e fisiológicas.

Atividades de gerenciamento e controle são por essência consideradas *serviço*, pois não apresentam um produto final concreto.

O gerenciamento da configuração de sistemas é um suporte ao processo de desenvolvimento, sendo basicamente, um desempenho de atividades de serviço que auxiliam os desenvolvedores em suas atividades.

A abordagem de Zutner

Zutner (1991) propõe o uso de 14 matrizes para a aplicação de QFD ao desenvolvimento de software. Conhecidas como matrizes Z, apresenta as matrizes Z-0 e Z-1 como identificação das características exigidas pelos clientes. Na fase seguinte essas características são traduzidas em requisitos técnicos.

A abordagem de Zutner foi mais tarde novamente adaptada para uma matriz na forma de T, que traduz os requisitos técnicos e os processos, estabelecendo uma relação entre si e com as necessidades de hardware e software (Barnett, 1995).

6. A aplicação do QFD na determinação de um esquema para o SCM

6.1 Introdução

Foi escolhida a abordagem de Cheng para a elaboração do esquema de implementação do SCM - objetivo deste trabalho - devido ao desenvolvimento de sistemas ser uma atividade voltada para a produção de serviços.

Percebe-se que as matrizes originais de Akao foram sofrendo algumas adaptações, descritas anteriormente, conforme a necessidade específica do tipo de indústria e seus processos.

Com relação ao software, algumas tentativas de aplicação do QFD resultaram também em uma nova forma de utilização das matrizes, como a abordagem de Zutner (1991).

A primeira dificuldade na aplicação do QFD ao desenvolvimento de software ou à qualquer parte dele é que as necessidades dos clientes não são exatamente as características técnicas que o software deve ter. Os usuários possuem suas necessidades latentes de acordo com suas próprias características ou de sua organização e o software por outro lado também possui uma série de aspectos técnicos que necessitam estar presentes no produto final. Adota-se então a estratégia de em primeiro lugar definir tecnicamente o que é necessário ao software e posteriormente satisfazer as expectativas do cliente.

Sem dúvida, se as necessidades técnicas não forem consideradas, o software não atenderá o cliente, mesmo que se procure contemplar seus pedidos. Será um resultado que aparentemente corresponde aos desejos dos usuários, porém inconsistente e sujeito a inúmeras falhas, o que certamente levará ao descontentamento geral tão logo seja utilizado.

Conforme exposto, o QFD relaciona em suas matrizes itens ou elementos que serão selecionados e novamente relacionados com outros elementos nas matrizes seguintes. A seleção desses itens ou elementos depois dos cálculos é muitas vezes necessária quando sua quantidade é elevada ou ainda naqueles casos em que é inviável a implantação de todos elementos. Essa seleção obedece a critérios de cada organização, que normalmente escolhe alguns daqueles de maior peso.

Para efeito dessa pesquisa, as relações entre os itens e o seus respectivos pesos foram obtidos junto à um grupo de desenvolvedores e usuários de sistemas de software do INPE – Instituto Nacional de Pesquisas Espaciais, em São José dos Campos – SP.

Como nenhum dos elementos foi descartado na matriz seguinte, o esquema elaborado contempla todas as exigências dos clientes e características do produto, obtidas na bibliografia pesquisada.

Portanto, o esquema de implementação do SCM que será apresentado é completo, pois considera todas as necessidades dos clientes e características do produto.

No entanto, ele poderá ser otimizado conforme a conveniência de cada organização. Para tanto, poderão ser estabelecidas outras relações e outros pesos e ainda serem selecionados os elementos de maior interesse de cada organização, conforme seus próprios critérios, apresentando obviamente, resultados diferentes de uma organização para outra.

6.2 Matriz 1 - A Matriz da Qualidade

O CMM – modelo aplicável ao desenvolvimento de sistemas – ressalta a importância dos sub-sistemas e recomenda algumas das características que um software com qualidade deve apresentar. Os usuários de software – gerentes, analistas, programadores, cliente final, já descritos anteriormente – possuem basicamente suas necessidades agrupadas em prazo, custos e requisitos operacionais. Uma análise do CMM e das necessidades desses usuários resultam na primeira parte da primeira matriz de Cheng: a qualidade exigida.

O CMM recomenda para o SCM que as atividades de gerenciamento de configuração deverão ser planejadas e os produtos de trabalho de software identificados, controlados e disponíveis. Quaisquer mudanças nos produtos de trabalho identificados deverão controladas e todos os envolvidos informados da situação atual e projetada dos produtos de trabalho de software.

Para o usuário a qualidade do software estará assegurada se seus atributos, condições ou características de sua construção apresentarem facilidade de uso e de reuso, correteza, eficiência, flexibilidade, integridade, interoperabilidade, manutenibilidade, portabilidade e confiabilidade. Meyer (1988) e Arthur (1985).

Na verdade, um desdobramento das necessidades do CMM – prazos, custos e requisitos operacionais – resultam nessas características ou atributos citados: facilidade de uso e de reuso, correteza, etc...

A segunda parte da primeira matriz refere-se as características da qualidade ou requisitos funcionais do produto ou serviço. Os aspectos operacionais clássicos do SCM são identificar, controlar, acompanhar e auditar ou examinar (Bersoff, 1985). Desdobramentos desses requisitos resultam em registro de versões, identificação do grupo pelo SCM ou pelos ES's, elementos que serão controlados, fornecimento de meios de manipular repositórios ou bibliotecas onde serão armazenados os ES e as informações do SCM, definição do espaço de trabalho individual ou de grupos de trabalho, suporte à criação, controle e manutenção de um família de produtos, geração de facilidades de resolução de conflitos quando mesclando versões, instalação de acesso controlado aos ES do produto para não permitir alteração não autorizada e uma série de outros requisitos relacionados aos elementos de software que serão gerenciados pelo SCM, já descritos anteriormente.

A primeira matriz ou matriz da qualidade faz então a relação entre a qualidade exigida e as características da qualidade, obtidas através de desdobramentos. Esta relação pode ser forte, média ou fraca e foram atribuídos para efeito de cálculos, respectivamente, os valores 9, 3 ou 1. As relações fortes informam as principais características da qualidade.

Da matriz obtida *(figura 6.1)* seleciona-se os requisitos de sistema de maior peso relativo cuja soma destes pesos seja igual a um valor entre 50 e 100%, determinado pelo usuário (Mirshawka, 1990). Espera-se que poucos requisitos possam fornecer essa soma. Através dessa seleção, denominada análise de Pareto *(figura 6.2)*, observa-se os requisitos mais importantes ou essenciais.

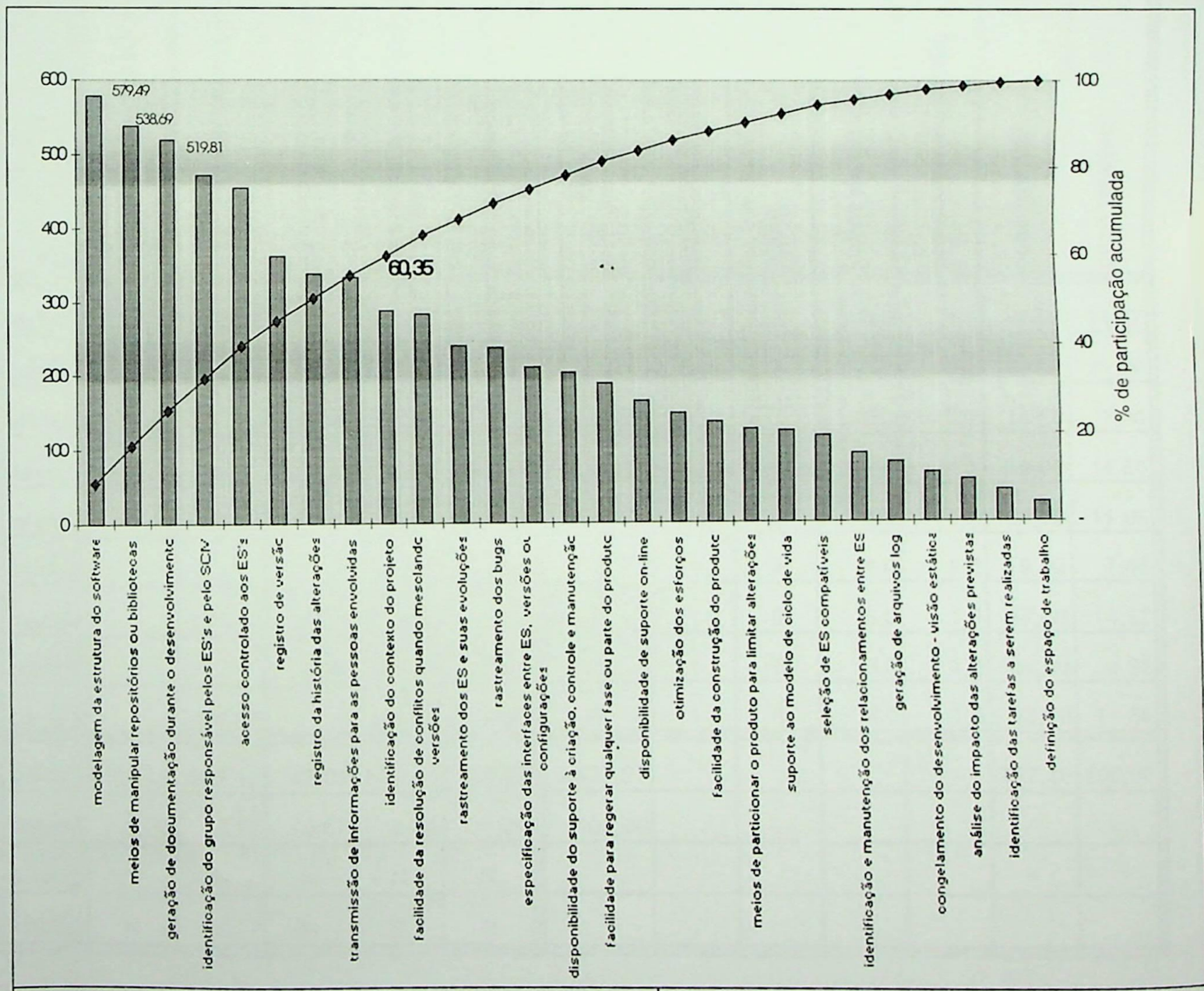


Figura 6.2 - Análise de Pareto da primeira matriz do QFD

6.3 Matriz 2 - A Matriz de Processos

Os processos para a obtenção de um SCM com qualidade farão relação com as características da qualidade da matriz 1, formando assim a segunda matriz do QFD ou matriz de processos.

A análise de Pareto referente as características da qualidade *(figura 6.2)* demonstrou que das vinte e sete características que um sistema de gerenciamento de configuração deve possuir, sete delas, ou menos de um terço, são fundamentais, pois encerram em mais de 50% de participação do total.

Um terço das características perfazem 60,35%. Sem dúvida as demais características são importantes, porém se os processos e todas as atividades contemplarem no mínimo estas sete selecionadas, teremos um resultado que certamente satisfará as necessidades do cliente.

As características fundamentais são: registro de versão, identificação do grupo responsável pelos ES's e pelo SCM, meios de manipular repositórios ou bibliotecas, acesso controlado aos ES's, modelagem da estrutura do software, registro da história das alterações, transmissão de informações para as pessoas envolvidas, geração de documentação durante o desenvolvimento e identificação do contexto do projeto, perfazendo o peso relativo total de 60,35%.

A segunda parte da segunda matriz refere-se aos processos que deverão ser desenvolvidos para que as características da qualidade estejam presentes no resultado final.

A segunda matriz ou matriz de processos apresenta um relação entre as características da qualidade e os processos *(figura 6.3)*.

Os processos podem ser desdobrados em atividades ou procedimentos, como acontece na quarta matriz do modelo de Akao. No entanto, no sistema de gerenciamento de configuração - que é uma atividade essencialmente de serviço - os diferentes procedimentos de um determinado processo normalmente não se relacionam ou pouco se relacionam com os outros diferentes processos, o que justifica a supressão desta matriz no modelo de Cheng. Porém, desdobramentos destes processos ou explicações mais aprofundadas se fazem necessários.

Os processos podem atender mais de uma característica da qualidade e aspectos inerentes a estes processos são descritos adiante, ficando os recursos necessários para serem discutidos na matriz 3 de Cheng.

Portanto, os processos identificados, seus desdobramentos e atividades são detalhados a seguir (Fiorini, 1998):

a) preparar plano de SCM: deve ser elaborado um plano para cada projeto de software, conforme um procedimento documentado. Um plano de SCM é desenvolvido tipicamente nas fases iniciais e em paralelo com o plano global do projeto, sendo revisado pelos grupos ou usuários envolvidos e principalmente controlado e gerenciado.

   Um plano de SCM deve conter as atividades a serem executadas, principalmente as que atendam os requisitos de um SCM, o cronograma dessas atividades, as responsabilidades de cada envolvido, os recursos necessários (pessoal, ferramentas, etc...).

b) executar atividades de acordo com o plano: isto significa que os usuários deverão efetuar as atividades previstas no plano, base para a execução.

c) estabelecer um repositório para as *baselines*: conforme IEEE, 1987, *baseline* é um conjunto de artefatos (ou produtos) formalmente aceitos (revisados e acordados), que servem de base para o desenvolvimento e que só podem ser alterados segundo um procedimento de controle de alterações formalmente estabelecido e documentado. O termo *baseline* pode também ser entendido como um marco de referência no desenvolvimento de software, que é caracterizado pela entrega de um ou mais itens de configuração e pela aprovação desses itens, obtida por meio de uma revisão técnica formal (Pressman, 1995).

   O desenvolvimento de software é uma seqüência de *baselines* acumuladas, sejam de itens novos ou revistos. No início as *baselines* são de documentos de especificações, depois de códigos, manuais e assim sucessivamente por todo o ciclo de vida do software.

   Estabelecer um repositório para as *baselines* significa estabelecer um sistema de gerenciamento de uma biblioteca de *baselines* de software, ou seja, um local como banco de dados, arquivos de dados ou de documentos em papel, que armazena as várias *baselines*. Este sistema de gerenciamento da biblioteca deve suportar os diferentes níveis de controle do SCM, possibilitar o armazenamento e recuperação dos elementos de software ou itens de configuração na versão desejada, possibilitar o

compartilhamento e a transferência de itens entre os grupos envolvidos e entre os níveis dentro da biblioteca, além de, obviamente, permitir a manutenção da estrutura da biblioteca.

d) identificar os ES's: a identificação dos ES's ou itens de configuração não é uma tarefa muito simples, uma vez que depende diretamente do paradigma de desenvolvimento do software, como análise estruturada ou orientada a objetos, entre outros.

Como exemplos de itens de configuração (Cunha, 1993) para a análise orientada a objeto temos:

PRODUTOS	*ITENS*	*RELACIONAMENTOS*
Classes – CLS	Métodos – MTD	Sub_classe – SUB ou Herda_de – HER
Componente do domínio do problema – CDP	Dispositivos de Armazenamento – DPA	Instância_de – INS
Componente de Interface Humana – CIH	Tarefas – TRF	Parte_de – PAR
Componente de Gerenciamento de Tarefas – CGT	Telas – TLA	Cliente_de – CLI
Componente de Gerenciamento de Dados – CGD		Define – DEF
Camada Assunto – CAS		Implementa – IMP
Camada Classe & Objeto – CCO		Usa – USA
Camada Estrutura – CET		Pertence – PER
Camada Atributo – CAT		Contém – COM
Camada Serviço – CSV		Testa – TST
Documento de Especificação de Requisitos – DER		
Plano de Integração e Testes – PIT		
Manual do Usuário – MUS		
Manual de Instalação – MIN		
Relatório de Testes – RLT		

Tabela 6.1 - Exemplos de itens de configuração para a análise orientada - Cunha, 1993

Como exemplos de itens de configuração (Cunha, 1993) para a abordagem estruturada temos:

PRODUTOS	*ITENS*	*RELACIONAMENTOS*
Arquivo de Dados – A	Data Flows – DF	Call – CAL
Funções – F	External Entity – EE	Delete – DEL
Arquivos Include – I	DFP Process – DP	Write – WRI
Command Procedures – O	Data Store – DS	Wait – WAI
Programas – P	Entity – ER	Use – USE
Subrotinas – S	Relationships – RR	Create – CRI
Specification and Analysis – SSD	Attribute – AR	Switch Off – SOF
System Design – SDD	Subsystens – SU	Generate – GEN
Integration and Test Plans – ITP	Task – TA	Read – REA
User Manual – UMD	Message Queues – MQ	Switch On – SON
Integration and Test Report – ITD	Shared Area – SA	Map – MAP
	Signal – SI	Compose – CMP
	Syncrhronism Message – MS	Simulate – SIM
	Module – MD	Test – TST
	Common Blocks – CB	
	Events Flags – EF	
	Cluster of Events Flags – CE	
	Global Sections – GS	
	Jobs – JB	
	Chain Subsystem – CS	
	Keyboards – KB	
	Logican Names – LN	
	Routine Parameters – RP	
	Global Variables – GV	

Tabela 6.2 - Exemplos de itens de configuração para a análise estruturada - Cunha, 1993

Os itens de configuração ou ES's são selecionados de acordo com critérios documentados e devem possuir identificadores únicos. Suas características são

especificadas bem como suas *baselines* e cada item é colocado sob a gerência de configuração após ser determinado o seu responsável, do ponto de vista do SCM.

Os ES's podem ser documentos relacionados ao processo de desenvolvimento, como planos, padrões e procedimentos ou ainda requisitos de software, arquitetura, unidades de código ou programas, procedimentos de testes, ferramentas de suporte, etc...

e) controlar a liberação de produtos: os produtos, artefatos ou itens existentes na biblioteca, novos ou modificados, devem ser testados e sua liberação para a geração de uma nova versão do software deve ser devidamente documentada.

f) gerenciar requisições de mudanças de todos os ES's: o gerenciamento envolve as atividades de criação, registro, análise, aprovação, revisão e acompanhamento de todos os itens através de procedimentos documentados.

g) controlar alterações nas *baselines*: envolve atividades de revisão e testes de regressão para garantia de que as alterações não causarão nenhum efeito inesperado na *baseline*. Somente os itens aprovados pela comissão do SCM serão incluídos na biblioteca. Os itens de configuração são verificados tanto na entrada como na saída da biblioteca, de forma a manter sua integridade.

h) registrar o estado dos itens de configuração: todas as atividades referentes a um item de configuração devem ser registradas em detalhes suficientes para que o seu conteúdo seja conhecido por todos os usuários, facilitando inclusive a recuperação de versões anteriores.

i) divulgar as atividades do SCM e o conteúdo das *baselines*: as atividades desenvolvidas e devidamente documentadas devem ser de conhecimento dos grupos ou pessoas envolvidos, bem como o conteúdo das *baselines*. Como exemplos de documentos ou relatórios utilizados na divulgação temos: atas de reuniões da comissão do SCM, estado e sumário das solicitações de alteração, dos relatórios de problemas e das alterações feitas nas *baselines*, histórico de revisões de ES's, estado das *baselines* de software e resultados das auditorias de software, entre outros.

j) conduzir as auditorias nas *baselines*: esse processo consiste nas seguintes atividades: preparação para a auditoria, avaliação da integridade das *baselines*, revisão da estrutura e das facilidades do sistema de biblioteca de gerência de configuração, verificação da completeza e corretude do conteúdo da biblioteca de *baselines*,

verificação da conformidade das *baselines* com os padrões e procedimentos do SCM e elaboração de documento para divulgação dos resultados da auditoria.

Pela análise de pareto *(figura 6.4)* constatamos que os processos de maior importância são controlar alterações nas *baselines,* controlar a liberação de produtos e gerenciar requisições de mudanças de todos os ES's.

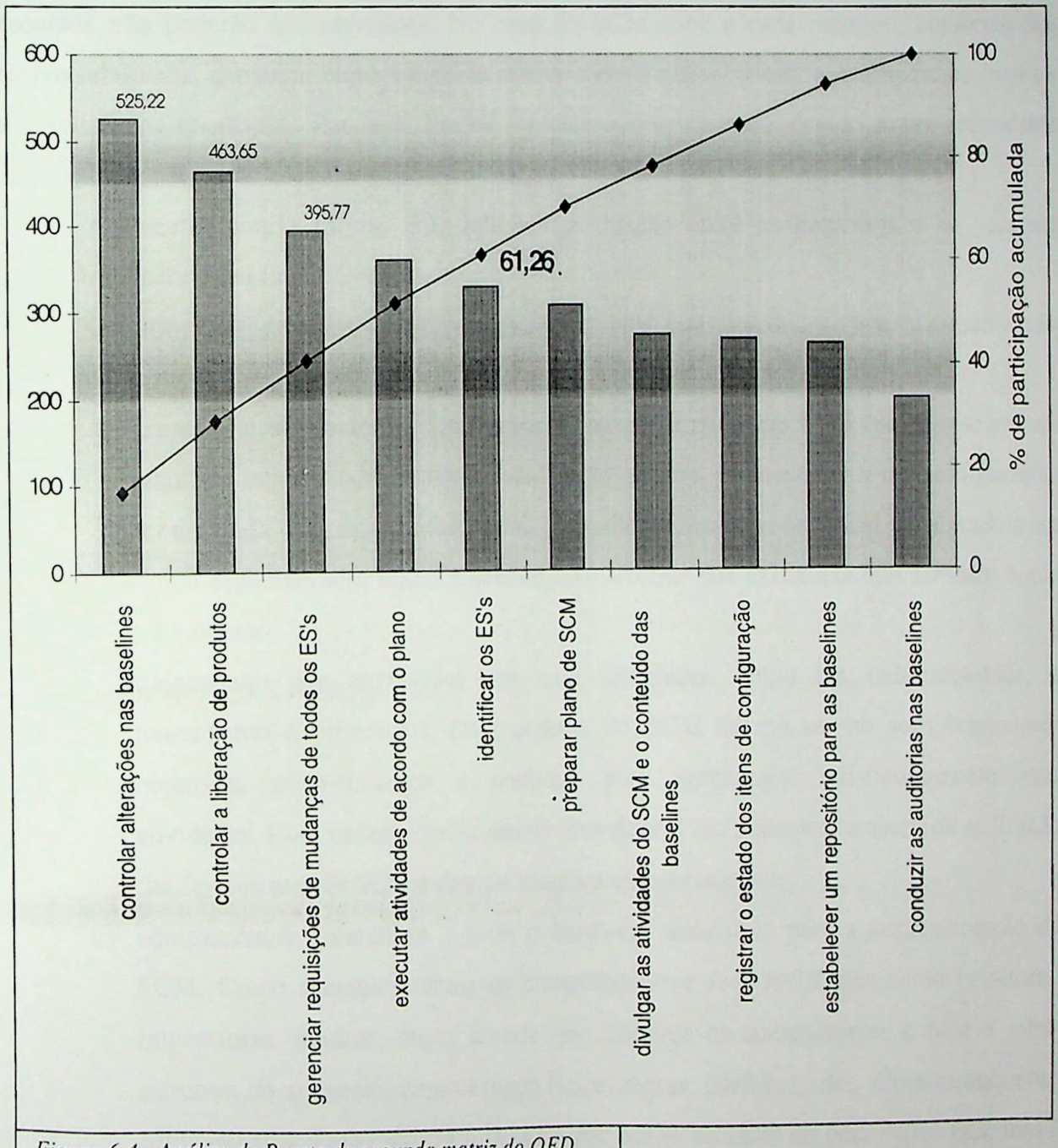


Figura 6.4 - Análise de Pareto da segunda matriz do QFD

6.4 Matriz 3 - A Matriz dos Recursos

Os recursos físicos e humanos necessários para a implementação do SCM devem ser previstos no *plano de SCM* e com certeza as atividades para a aquisição desses recursos farão parte do processo *executar atividades de acordo com o plano*.

Fundos ou recursos financeiros sem dúvida são necessários para a execução dos processos, seja na implantação de um SCM ou em qualquer outra atividade. Sem eles, os demais recursos não poderão ser adquiridos. No caso do SCM cabe a cada membro, conforme sua responsabilidade, gerenciar direta ou indiretamente estes fundos, buscando minimizar os custos e maximizar os resultados. Por isso, fundos ou recursos financeiros, devido a sua redundante necessidade, não serão considerados nessa análise.

A terceira matriz *(figura 6.5)* refere-se a relação entre os processos e os recursos necessários para a sua implementação.

Os principais recursos ou grupo desses recursos considerados na terceira matriz são detalhados a seguir:

a) grupo de usuários do SCM: Os recursos humanos ou grupo SCM constitui-se em um recursos imprescindível para a realização de um gerenciamento de configuração, sendo ainda uma das características da qualidade, ou seja, um SCM deve possuir um grupo especializado, sendo essencial não somente sua existência mas também o seu treinamento.

b) treinamento dos envolvidos em suas atividades: como em toda atividade, o treinamento é primordial. Os membros do SCM devem ser no seus respectivos objetivos, procedimentos e métodos para executarem satisfatoriamente suas atividades. Com certeza, todos envolvidos devem ser treinados também na utilização das ferramentas de SCM e demais recursos computacionais.

c) equipamentos: referem-se a todo o hardware necessário para a implementação do SCM. Como exemplos temos os computadores e seus periféricos como monitores, impressoras, modens, etc..., a rede que interliga os computadores e toda a infra-estrutura do ambiente, como espaço físico, mesas, telefones, etc... Com certeza estes equipamentos serão também utilizados por outros sistemas ou para outros fins, mas é importante destacar-se que sem o hardware e equipamentos de suporte um SCM não

poderá valer-se de outros recursos computacionais, como linguagens, ferramentas, etc..., que são de grande ajudar no cumprimento das atividades.

d) sistema de suporte ao controle de ES's: são linguagens, ferramentas, aplicativos e demais software's que dão suporte ao controle e gerenciamento. As atividades de desenvolvimento de software costumam utilizar-se de diferentes recursos computacionais tais como arquivo de memória secundária, mecanismos de comunicação e de interrupção, recursos de controle de processo, utilitários, etc... Além desses, os sistemas de controle necessitam de outros recursos, tais como pacotes de software para controle de versões e para geração de sistema, banco de dados para armazenar os elementos controlados e sistema de suporte para controle de elementos. O maior ou menor nível de automatização das atividades de controle depende da análise de custo-benefício, das características do software a ser desenvolvido e das ferramentas de software e do hardware disponíveis.

e) ambiente de SCM: em um sentido mais amplo, o ambiente é que faz a integração entre os recursos já citados. Pessoas, fundos, equipamentos e ferramentas são agora reunidos em um único ambiente. Quanto ao ambiente de SCM pode-se afirmar que suas atividades estão relacionadas com a Geração e Manutenção das Versões do Sistema, Manutenção das Bibliotecas Controladas, Transferência de Elementos entre Ambientes, Controle do Banco de Dados de Elementos Controlados (BDEC) e são realizadas pela Equipe de Controle de Configuração. Neste Ambiente a Equipe de Controle de Configuração ficará a cargo de realizar a geração e a manutenção das versões do sistema, definindo não somente os elementos que compõem uma versão do sistema, mas também o conjunto de regras de dependências e de ações para a construção de uma determinada versão do sistema. Fica a cargo também desta equipe o gerenciamento das Bibliotecas Controladas que armazenam os produtos aprovados pela Equipe de Sistema ou pela Equipe de Qualidade, tais como, documentos de especificação, planos de teste, arquivos de dados, classes etc. Como o controle das bibliotecas faz parte deste Ambiente, as transferências de elementos de software entre os ambientes de desenvolvimento são feitos pela Equipe de Configuração de Software, devendo manter inclusive um registro de todas estas operações de transferências.

A análise de Pareto da Matriz de Recursos *(figura 6.6)* destaca a importância das ferramentas computacionais e dos recursos humanos do SCM.

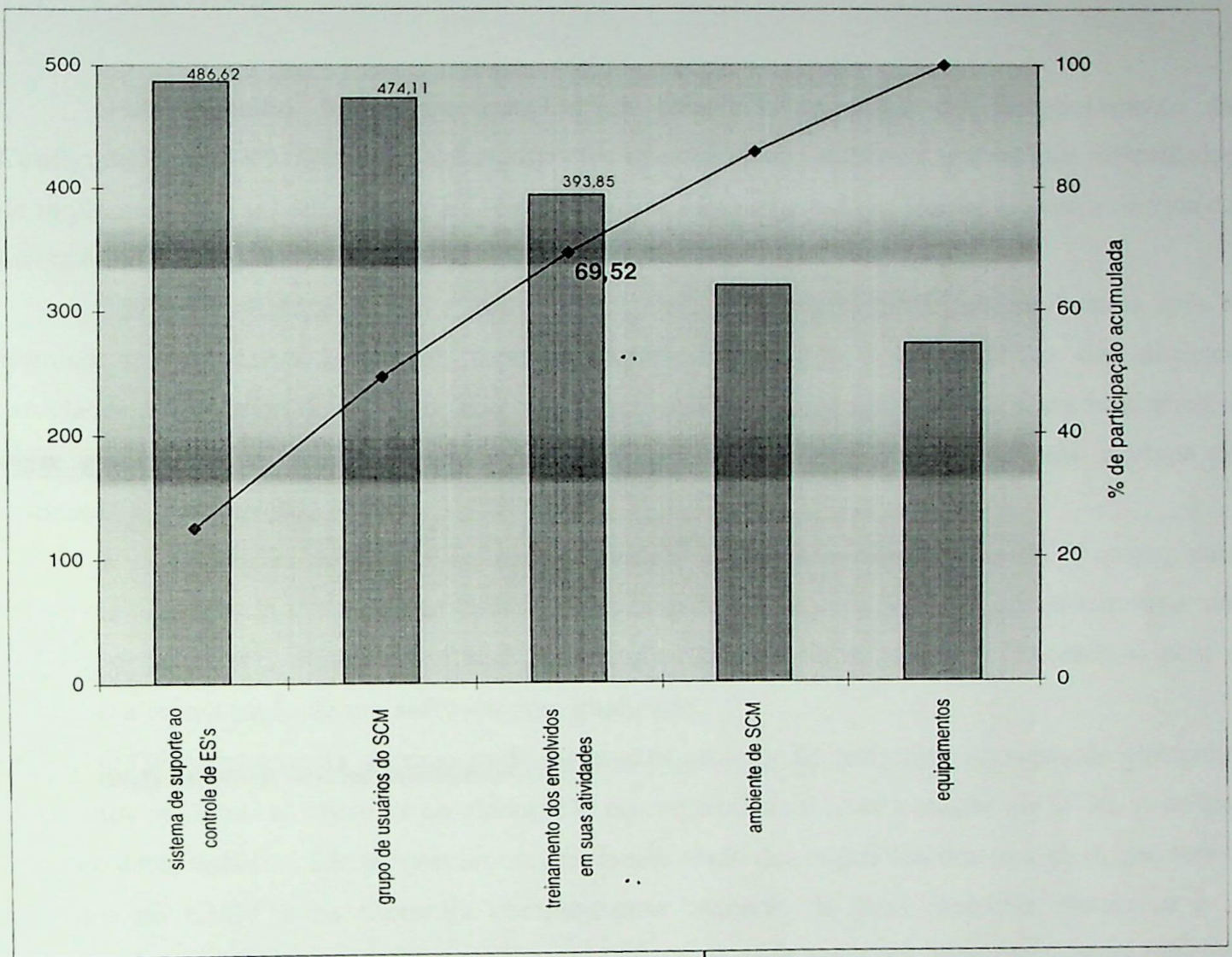


Figura 6.6- Análise de Pareto da terceira matriz do QFD

Escola Federal de Engenharia de Itajubá - EFEI
Biblioteca MAUÁ BIM

7. Conclusões e Recomendações

7.1 Conclusões

Neste trabalho foram apresentados os principais aspectos do Gerenciamento da Configuração de Software: abordagens, aspectos operacionais, usuários e até mesmo dificuldades de implementação, porém o que mais se destacou foi a sua importância dentro do ciclo de vida de um sistema.

Como a qualidade de um sistema de software dificilmente pode ser melhorada após o término de seu desenvolvimento, torna-se imprescindível que o desempenho das diversas atividades do processo de software seja constantemente gerenciado. O SCM é parte fundamental deste gerenciamento, que continua após o término do desenvolvimento, durante a etapa de utilização e manutenção.

A contribuição do CMM foi que o modelo além de estabelecer o SCM como uma exigência para a melhoria da maturidade e da capacidade de uma organização em desenvolver um sistema de software, ofereceu também informações básicas sobre o que é necessário para a obtenção e manutenção de um software com qualidade.

O QFD, ferramenta já consagrada em muitos projetos de bens e/ou serviços de diferentes segmentos, mostrou-se eficiente na elaboração do esquema de implementação do SCM, principal objetivo deste trabalho. Ele proporcionou uma maior visão dos requisitos dos usuários, que foram baseados no CMM e na literatura complementar. Através de suas matrizes constatou-se a importância de cada requisito, de cada característica e de cada processo, mostrando suas relações e interdependências, com o objetivo principal de proporcionar aos usuários um software de qualidade.

O esquema para implementação do SCM, obtido através da utilização do CMM e do QFD, é um esquema abrangente, que pode atender diferentes tipos de organizações e de metodologias de desenvolvimento de sistemas. Ele apresenta o que deve ser feito para a obtenção de um SCM com qualidade e ainda como deve ser feito, ou seja, quais são os processos e procedimentos que devem ser desempenhados pelos usuários.

No entanto, conforme a organização, poderá surgir a necessidade de otimização do esquema, de acordo com a sua política de investimentos, filosofia de desenvolvimento,

capacidade e maturidade dos usuários e outros fatores peculiares a cada organização. Logo, não só os elementos selecionados para cada matriz podem ser diferentes de uma organização para outra, mas também as relações estabelecidas entre esses elementos poderão ser outras e possuírem outros pesos. A partir dos elementos apresentados no esquema obtido cada organização selecionará aqueles de maior interesse, conforme seus próprios critérios.

Concluiu-se também que o QFD apresenta diferentes abordagens devido principalmente à características muito particulares do produto ou serviço e a necessidade específica do tipo de organização e de seus processos. As abordagens do QFD diferem entre si basicamente nos aspectos de determinação das etapas propostas para a execução dos desdobramentos, sendo mais ou menos abrangentes, no número de matrizes utilizadas para cada uma dessas etapas e no uso de diferentes ferramentas auxiliares.

No caso da abordagem utilizada para a obtenção do esquema observou-se a necessidade de um detalhamento mais profundo dos processos identificados na matriz 2 *(figura 6.3)*. Este detalhamento não é proposto pelas matrizes de Cheng, como acontece na abordagem de Akao. Podemos então, neste caso específico de aplicação do QFD, observar mais uma necessidade de adaptação da ferramenta, solucionada através da decomposição de cada processo em procedimentos.

O resultado da pesquisa é um esquema para a implementação do SCM utilizando o QFD, conforme a proposta inicial, que contempla todas os requisitos dos usuários, podendo ser otimizado de acordo com os usuários ou organização.

7.2 Recomendações para trabalhos futuros

Através da pesquisa elaborada verificou-se que o QFD original de Akao tem sofrido adaptações e ainda que as técnicas de desenvolvimento de sistemas estão em constante evolução, exigindo cada vez mais qualidade dos resultados finais.

Como sugestão para futuros trabalhos de pesquisa podemos propor:

a) a aplicação de QFD em outros sub-sistemas do ciclo de vida de sistemas de software e
b) a elaboração de uma ferramenta computadorizada a para implementação de SCM, utilizando o esquema proposto.

8. Referências Bibliográficas

AKAO, Y. Quality function deployment: integrating product customer requirements into product design. Cambridge, Productivity Press, 1990.

ARMSTRONG A., PENNY Takang, GRUBB A. Software Maintenance: Concepts and Practice. LA, International Thomson Computer Press, 1996.

ARTHUR, L. J. Measuring programmer productivity an software quality. New York, John Wiley, 1985.

BARNETT, W.D., RAJA, M.K. Application of QFD to the software development process. Department of Information Systems and Management Science, University of Texas at Arlington, Texas, USA, 1995.

BARROS, 1998

BEDWORTH, D. D. Computer-integrated design and manufacturing. New York, McGraw-Hill, 1991.

BERLACK, H. Ronald. Software Configuration Management. New York, John Wiley & Sons, 1992.

BERSOFF, E. H. Measuring programmer productivity an software quality. New York, John Wiley, 1985.

BINGHAM, J. E., GARTH, W. P. D. Manual de Análise de Sistemas. Rio de Janeiro, Interciência, 1996.

BOEHM, B. W. Software risk management: principles and practices. IEEE Software, 1991.

BOOCH, G. Object Oriented Design with Applications. Redwood City, Benjamin/Cummings, 1991.

CARPINETTI, L. C. R. - Síntese do QFD das quatro ênfases e do QFD estendido: uma abordagem de aplicação - XVII ENEGEP - Gramado - 1997.

CHENG, Lin Chih - QFD, Planejamento da Qualidade - Fundação Christiano Ottoni - 1995.

COAD, P.; YOURDON, E. Análise baseada em objetos. 2ª ed., Rio de Janeiro, Editora Campus, 1992.

COAD, P. Object Models : Strategies, Patterns, & Applications. New Jersey, Prentice Hall, 1997. 515 p.

CUNHA, J. B. S.; NAKANISHI, T. O controle do desenvolvimento e manutenção de sistemas de software. Congresso Ibero Latino-americano de métodos computacionais em engenharia, 14°, São Paulo. ANAIS, São Paulo, IPT, v. 2, 1993.

DAI, K.; NAGATA, M. Object-Oriented Analysis Using Scenarios. Tokyo, Systems and Computers in Japan, June 1997.

DART, S.A. Spectrum of Functionality in Configuration Management Systems. Pittsburgh, Software Engineering Institute, Dec. 1992.

EUREKA, W.E. & RYAN, N.E. QFD – perspectivas gerenciais do desdobramento da função qualidade. Rio de Janeiro, Qualitymark, 1992.

FEIGENBAUM, A.V. Total quality control. New York, McGraw-Hill Book Company, 1983.

FEILER, P.H. Configuration Management Models in Commercial Environments. Pittsburgh, Software Engineering Institute, Mar, 1991. 35 p. (Technical Report CMU/SEI-91-TR-7).

FIORINI, S.T.; STAA, A.; BAPTISTA, R.M. Engenharia de Software com CMM. São Paulo, Brasport, 1998.

Furlan, 1998

GANE, C.; SARSON, T. Análise Estruturada de Sistemas. Rio de Janeiro, LTC- Livros Técnicos e Científicos Editora S. A., 1983.

HUMPHREY, W. Characterizing the Software Process: A Maturity Framework. IEEE Software, Mar, 1988.

HUMPHREY, W. Managing the Software Process. Wokingham, Addison-Wesley Publishing Co., 1989.

HUMPHREY, W. A Guide to the CMM. Florida, Sarasota, 1995

IEEE. Guide to Software Configuration Management. STD 729, 1983.

IEEE, Transaction on Software engineering. STD 1045, 1987.

JACOBSON, I.; CHIRSTERSON, M.; JONSSON, P.; OVERDAARD, G. Object-Oriented Software Engineering. Wokingham, Addison-Wesley Publishing Co., 1992.

JURAN, J.M. & GRYNA, F. M. Juran's quality control handbook. 4^A ed New York, McGraw-Hill International Editions, 1988.

KANEKO, N. - QFD implementation em hotel service - 5th ISQFD - Belo Horizonte - 1999.

KENNETH, M. Dymond. Understanding the Capability Maturity Model for Software Process. Transition International, Inc., 1996.

KING, Bob - Better Designs in Half the Time - Goal QPC - 3° edition.
KOURA, K. - Analysis of the service functions of the Asahi University Hospital using QFD (part 2) - 5th ISQFD - Belo Horizonte - 1999.

KRAMER, J.; MAGEE, J. Dynamic Configuration for Distributed Systems. IEEE Transactions of Software Engineering, SE-11(4):424-435, Apr. 1985.

MANZUR, G. H. - Service QFD - state of the art update - 5th ISQFD - Belo Horizonte - 1999.

MARION, Kelly. Configuration Management: The Changing Image. U.K., McGraw-Hill, 1996.

McCONNELL, Steve. Code Complete. LA, Microsoft Press, 1993.

MEYER, B. Object-oriented Software Construction. New York, Prentice Hall, 1988.

MILITELLO, K. Pare de Jogar Dinheiro no Lixo. Revista Exame Informática, v. 12, n° 135, 1997.

MIRSHAWKA, V. QFD – a vez do Brasil. São Paulo, McGraw-Hill, 1990.

NAKANISHI, T. Engenharia de Software: apostila usada durante o curso de Engenharia de Software no INPE - Instituto Nacional de Pesquisas Espaciais. São José dos Campos, 2o trimestre, 1995.

PAGE-JONES, M. Projeto Estruturado de Sistemas. São Paulo, Mc Graw-Hill, 1988.

PAULK, M.C., WEBER, C.V. Key pratices of the Capability Maturity Model for Software. Version 1.1, CMU/SEI-93-TR-25, 1993.

PRESMANN, R. S. Engenharia de Software. São Paulo, Mc Graw-Hill, 1995.

RAWLINGS, Joseph H. SCM for Network Development Environments. LA, McGraw-Hill, 1994.

ROCHKIND, M.J. The Source Code Control System. IEEE Transactions on Software Engineering, SE-1(4):364-370, 1975.

ROOK, P. Controling software projects. Los Alamitos, CA, Ed Tutorial, 1988.

RUMBAUGH, J.; et al. Modelagem e Projeto Baseados em Objetos. Rio de Janeiro, Editora Campus, 1994.

STAA, A. Engenharia de Programas. Rio de Janeiro, LTC- Livros Técnicos e Científicos, 1987.

WIRFS-BROCK, R.; WILKERSON, B. Designing Object-Oriented Software. New Jersey, Prentice Hall, 1990.

USA, DEPARTMENT OF DEFENSE. Configuration Management. DoD-Mil-Std-973, 1992.

ZULTNER, R.E. Software Quality Deployment Seminar: Aplying QFD to Software, seminar notes, Zultner and Company, 1991.

Escola Federal de Engenharia de Itajubá
Biblioteca
MAUÁ
BIM