REGINALDO ARAKAKI

METODOLOGIA PARA AVALIAÇÃO DE QUALIDADE E PRODUTIVIDADE EM PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO A OBJETOS

TESE APRESENTADA À ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO PARA A OBTENÇÃO DO TÍTULO DE DOUTOR EM ENGENHARIA.

SÃO PAULO
1997

REGINALDO ARAKAKI

METODOLOGIA PARA AVALIAÇÃO DE QUALIDADE E PRODUTIVIDADE EM PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO A OBJETOS

TESE APRESENTADA À ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO PARA A OBTENÇÃO DO TÍTULO DE DOUTOR EM ENGENHARIA.

ÁREA DE CONCENTRAÇÃO:
ENGENHARIA DE COMPUTAÇÃO E SISTEMAS DIGITAIS
ORIENTADOR:
PROF. DR. ANTÔNIO MARCOS DE AGUIRRA MASSOLA

SÃO PAULO
1997

DEDICATÓRIA

AOS MEUS PAIS KAORU E YOLANDA ARAKAKI.

À MINHA FAMÍLIA VALQUÍRIA, ROBERTA, NEJA E ALEX (IN MEMORIAN).

AGRADECIMENTOS

AOS MEUS AMIGOS DA SPECTRUM ENGENHARIA, PELO APOIO NA PREPARAÇÃO DO TRABALHO.

AOS MEUS AMIGOS DO DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E SISTEMAS DIGITAIS DA ESCOLA POLITÉCNICA DA USP.

EM ESPECIAL:

- PROF. MASSOLA, PELO PRECIOSO ESTÍMULO, ORIENTAÇÃO E PACIÊNCIA COMO ORIENTADOR;
- PROF. MOACYR MARTUCCI, PELA VALIOSA ORIENTAÇÃO E ESTÍMULO NOS ASPECTOS ACADÊMICOS DA PESQUISA;
- PROF. PAULO CUGNASCA, PROF. SELMA MELNINKOFF, PROF. KECHI HIRAMA PELO INCENTIVO E ORIENTAÇÃO;
- AOS AMIGOS JOSEF, MARISA, PATRÍCIA CALOVINI, CRISTINA ABREU, TERCILA RODRIGUES, MAURO GONDO E GIRLENE, PELO APOIO;
- ÀS MINHAS MULHERES VALQUÍRIA E ROBERTA QUE SEMPRE ME APOIARAM NESTA EMPREITADA;

ENFIM, A TODOS QUE DE ALGUMA FORMA ME AJUDARAM.

MUITO OBRIGADO.
REGINALDO.

PCS

ERRATAS E OBSERVAÇÕES ADICIONAIS

TESE:

METODOLOGIA PARA AVALIAÇÃO DE QUALIDADE E PRODUTIVIDADE EM PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO A OBJETOS.

DATA: 24/10/97

1) Pág. 3, 1º Parágrafo:

Considerar: O processo reorganizado tem grande probabilidade de estar adequado ao nível 2 e em seguida ao nível 3.

Ao invés de: ... ficará adequado ao contexto ... (nível 2 – SEI/CMM) ...

2) Pág. 19, 1º Parágrafo e pág. 182:

Considerar: A tecnologia OO é utilizada por alguns grandes fabricantes de software ...

Ao invés de: ... amplamente utilizada com sucesso por grandes fabricantes de software ...

3) Pág. 19, 1º Parágrafo e também em outras partes onde aparecem tais afirmações:

Considerar: Um processo reorganizado por esta metodologia pode ficar preparado para ser submetido à ...

Ao invés de: Um processo reorganizado por esta metodologia deve estar pronto para se submeter à ...

4) Pág. 21, 3° Parágrafo:

Considerar: A indústria referenciada é aquela cujos os processos são tais que os sistemas que os apoiam são classificados de missão crítica, devida a alta dependência destes sistemas na operacionalização dos referidos processos.

5) Págs. 17, 27, 28, 38, 39 e 40:

Considerar: Onde se lê arquitetura de software, entenda-se uma organização ao nível global de uma infraestrutura como o software deve ser organizado. Neste trabalho, a arquitetura considera a organização por objetos distribuídos considerando alocação de redes locais, remotos, servidores locais, remotos e computadores de grande porte (*mainframe*).

6) Pág. 74:

Considerar: A área chave de processo de contratação.

7) Sobre as referências bibliográficas:

a) A bibliografia contém mais referências que os citados no trabalho:

Considerar: As referências adicionais foram relacionadas com o objetivo de propor uma referência bibliográfica adicional para os leitores e pesquisadores interessados em evoluir e aprofundar estudos relacionados com os diversos temas citados neste trabalho.

b) pág. 84: Onde se lê OTTENSTEIN, leia-se OTTENSTEIN (1979).

c) Pág. 83: Onde se lê BULUT leia-se BULUT et. al (1974a, 1974b). Idem para HALSTEAD (1976a, 1976b).

d) Pág. 117: Leia-se PRESSMAN (1988).

e) Pág. 169: Leia-se KARUNANITHI e BIEMN (1993).

8) Alguns números adicionais, obtidos e registrados no período de abril a outubro de 1997, decorrentes da aplicação prática da metodologia:

- Abrangência dos Experimentos: Indústria financeira
- Qualidade dos Produtos e Processos
    - 300 desenvolvedores
    - 30 líderes e gerentes de projetos
    - 40 projetos
    - 20 a 40 objetos de negócios por sistema
    - "Fazer correto da primeira vez"
- Time boxing no planejamento

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

RESUMO

A APLICAÇÃO DE TÉCNICAS DE AVALIAÇÃO DE QUALIDADE DOS PROCESSOS DE OBTENÇÃO DE SOFTWARE NÃO GARANTE UM AUMENTO NA CAPACIDADE DE PRODUZIR SOFTWARE COM A QUALIDADE E PRODUTIVIDADE DESEJADAS.

A INDÚSTRIA FINANCEIRA RETRATA BEM ESTA DEMANDA POR QUALIDADE. NESTE MERCADO, MUITOS PROCESSOS PRODUTIVOS DE SOFTWARE SÃO VITAIS, POIS SERVEM DE BASE PARA O PROCESSO DE NEGÓCIOS, NO ENTANTO, A MAIORIA APRESENTA UM GRAU DE MATURIDADE BAIXO, RESULTANDO EM DESPERDÍCIOS E PRODUTOS DE BAIXA QUALIDADE.

ESTA TESE APRESENTA UMA METODOLOGIA PARA IMPLANTAR MECANISMOS DE AVALIAÇÃO DA QUALIDADE E DE PRODUTIVIDADE EM PROCESSOS DE OBTENÇÃO DE SOFTWARE COM BAIXA MATURIDADE (NÍVEL 1 - CMM/SEI), AQUI DENOMINADOS QUASI-CAÓTICOS.

AS BASES DESSA METODOLOGIA FORAM ESTABELECIDAS NA INCORPORAÇÃO DE CONCEITOS E PROCEDIMENTOS DA TECNOLOGIA ORIENTADA A OBJETOS E DA TEORIA DE MÉTRICAS EM SOFTWARE E NA SIMULTÂNEA INCORPORAÇÃO DE ASPECTOS DE QUALIDADE DE PROCESSO E DE PRODUTO (COMO O ISO 9000 OU MODELO CMM/SEI OU MALCOM BALDRIGE).

AS PRINCIPAIS RESULTANTES DE SUA APLICAÇÃO PRÁTICA FORAM A ORGANIZAÇÃO DO PROCESSO E A IMPLANTAÇÃO DE MECANISMOS DE AVALIAÇÃO QUE RESULTARAM REALMENTE EM QUALIDADE E PRODUTIVIDADE DO PROCESSO.

AVALIAR QUALIDADE E PRODUTIVIDADE DENTRO DE UM PROCESSO QUASI-CAÓTICO SOMENTE RETRATA A SITUAÇÃO CAÓTICA, SEM APRIMORAR. O PRODUTIVO É INCORPORAR MECANISMOS DE QUALIDADE E PRODUTIVIDADE AO PROCESSO, DE FORMA A, SIMULTANEAMENTE ORGANIZÁ-LO, E TORNÁ-LO PASSÍVEL DE AVALIAÇÃO.

ABSTRACT

APPLYING QUALITY EVALUATION TECHNIQUES IN SOFTWARE DEVELOPMENTS DOES NOT MEAN TO GET A PROCESS WITH THE EXPECTED LEVEL OF QUALITY AND PRODUCTIVITY.

FINANCIAL INDUSTRY WELL REPRESENTS THE DEMAND FOR QUALITY IN SOFTWARE PROCESS. IN THIS MARKET, MANY OF THE SOFTWARE DEVELOPMENT PROCESSES ARE VITAL, SUPPORTING BUSINESS PROCESSES, NEVERTHELESS, MOST OF THEM PRESENTS A LOW LEVEL OF MATURITY, RESULTING IN WASTES AND IN LOW QUALITY PRODUCTS.

THIS THESIS PRESENTS A METHODOLOGY TO IMPLEMENT MECHANISMS FOR QUALITY AND PRODUCTIVITY EVALUATION IN PROCESS WITH LOW LEVEL MATURITY (LEVEL 1 SEI/CMM), .

THIS METHODOLOGY USES CONCEPTS AND PROCEDURES FROM THE OBJECT ORIENTATION TECHNOLOGY AND FROM SOFTWARE METRIC CONCEPTS, ALSO INCLUDING, IN A SIMULTANEOUS WAY, ASPECTS OF QUALITY IN PRODUCTS AND IN PROCESS.

THE MAIN RESULTS OF THE APPLICATION OF THIS METHODOLOGY , IN REAL PRACTICE, WERE A ORGANIZED PROCESS (LEVEL 2 SEI/CMM) AND THE INCLUSION OF EVALUATION MECHANISMS WHICH REALLY INCREASED QUALITY AND PRODUCTIVITY IN THE PROCESS OF SOFTWARE DEVELOPMENT.

QUALITY AND PRODUCTIVITY EVALUATION IN PROCESS, WITH LOW LEVEL MATURITY, ONLY WILL OFFER A PORTRAID OF THE DESORGANIZED SITUATION, DO NOT IMPROVING ANYTHING. THE CORRECT ATTITUDE IS TO INCLUDE MECHANISMS OF QUALITY AND PRODUCTIVITY, IN SUCH AWAY THAT YOU, SIMULTANEOUSLY, GET THE ORGANIZATION OF THE PROCESS, HAVING, THEN, CONDITIONS TO EVALUATE.

1. INTRODUÇÃO E MOTIVAÇÕES

Objetivo do Capítulo:

Apresentar o objetivo da tese, motivações, justificativas, ineditismo e estrutura, assim como seu contexto de aplicação e sua contribuição na área de produção de sistemas de software.

Tópicos do Capítulo:

1.1 Objetivo da Tese

Esta tese de doutorado apresenta uma metodologia que otimiza tanto a qualidade dos produtos como a qualidade dos processos críticos de produção de software, implantando mecanismos que permitam avaliação de qualidade e produtividade e obtendo como resultado a organização de um processo de obtenção[1] de software que se apresenta num estado definido quasi-caótico[2] (nível 1, base SEI/CMM)

Os instrumentos básicos utilizados por esta metodologia são:

1. **Tecnologia orientada a objetos** - aplica-se conceitos de orientação a objetos em todo processo de planejamento, desenvolvimento e implantação;
2. **Mecanismos que permitam a avaliação da qualidade e da produtividade** - utiliza-se o conceito de métricas, resultando em avaliações objetivas e claras por serem mensuráveis.

A aplicação desta metodologia é indicada principalmente onde há necessidade de se organizar/reorganizar o processo de desenvolvimento de produtos que ocorre em ambientes de obtenção de software caracterizados por um nível de maturidade baixo (equivalente ao nível inicial - segundo critérios de categorização do Software Engineering Institute - Capability and Maturity Model - SEI/CMM - detalhes no Cap.2).

---

[1] Processo de Obtenção de Software - termo empregado nesta tese para denotar processos de desenvolvimento, aquisição ou manutenção de software (PRESSMAN, 1988).

[2] Quasi-caótico - termo criado e empregado nesta tese para denotar ambientes com processos de software desorganizados, com nível de maturidade em produção de software equivalente ao nível 1 (Inicial) do modelo SEI/CMM.

Como resultado, o processo organizado/reorganizado ficará adequado ao contexto de maturidade equivalente ao nível repetitivo (nível 2 - SEI/CMM) e estará também em condições de acessar, em seguida e rapidamente, o equivalente ao nível definido (nível 3 - SEI/CMM), pois, neste estágio, já possui requisitos de qualidade que o capacitam para tal categorização, conforme está descrito nos Caps. 5, 6 e 7.

O ambiente de desenvolvimento de software de indústrias financeiras, onde sistemas de software constituem infra-estruturas vitais aos seus processos de negócios, é o objeto de estudos e pesquisas específicas desta tese. O fato de estarem disponíveis normas, padrões, sistemas de qualidade (como o ISO 9000), a nível nacional ou internacional, não garante a melhoria efetiva dos produtos e dos processos, estando aí, portanto, a oportunidade de realização deste trabalho.

Porém, esta metodologia e os resultados aqui registrados podem ser adequados e aplicados em outros setores. Qualquer setor, onde sistemas de software são críticos, os processos de desenvolvimento e de manutenção praticamente indefinidos ou com atividades internas não visíveis em termos de andamento e evolução das tarefas, podem estar com um processo de qualidade insatisfatória, não otimizada, e, assim, a aplicação da metodologia aqui apresentada será adequada. Gerentes de projetos podem detectar um indício de não otimização quando, dada uma solicitação de desenvolvimento, os resultados produzidos só possam ser constatados ao final, sem possibilidades de verificações intermediárias. Em casos com tais características, o processo pode ser

reorganizado segundo esta metodologia, com ganhos em qualidade expostos ao longo desta tese.

1.2 Introdução

A presente tese é resultante de efetiva atuação na área de engenharia de software, atuação esta, que abarca pesquisas, atividades conceituais, atividades profissionais e práticas especialmente voltadas para metodologias de desenvolvimento e para tecnologia orientada a objetos, buscando formas de desenvolvimento de sistemas de software que fossem, além de racionais, técnicas, também fossem financeiramente viáveis e com nível de qualidade adequado ao contexto.

Por uma série de motivos, conforme apresenta PFLEEGER; FENTON (1994), a existência de padrões e normas como os estabelecidos, por exemplo, no ISO 9000, SEI/CMM ou Malcom Baldrige (ver Cap.2), não garantem por si só que ambientes de desenvolvimento de software das empresas consigam evoluir nos seus processos em capacidade e maturidade. Um dos principais motivos está ligado à alta administração, que deve se envolver no processo, na medida em que tomará decisões estratégicas que apoiarão a busca de requisitos que controlem, avaliem e analisem não somente a qualidade, mas também o processo, seus estágios, seu andamento e seu processo global. A alta administração deve atuar consciente de que a melhoria de seus sistemas de software está vinculada à melhoria dos processos de obtenção de software, com a incorporação de requisitos de qualidade. Outro motivo muito importante está relacionado com

aspectos culturais e comportamentais da equipe técnica, que, dada a tradicional criticidade dos prazos em suas atividades, apresentam pré rejeição a alterações de procedimentos que ameacem a rotina do seu dia-a-dia, mesmo que, no processo corrente, pela falta de critérios que garantam a qualidade, os produtos estejam apresentando deficiências e diversos problemas como os explicitados no Cap.2. A alta administração e as lideranças de equipes resolverão tais problemas através da conscientização de todos envolvidos, fazendo-os apreender que: reorganizar processos é, antes de tudo, uma melhoria de qualidade de trabalho, que resultará em aumento geral da produtividade, aumentará a competitividade da organização, trará benefícios globais, incluindo benefícios pessoais, na medida em que estarão ampliando conhecimentos e se aprimorando profissionalmente devido às novas experiências e práticas, sentindo-se motivados e valorizados ao produzirem efetivamente, eficientemente, eficazmente, com a qualidade adequada. Apesar de óbvio, importante é ressaltar a necessidade da participação e do esforço conjunto para elaborar procedimentos adequados e bem definidos, que possam ser executados por diversas pessoas treinadas e habilitadas. Porém, não é escopo desta tese aprofundar-se nos aspectos relacionados com a conscientização das lideranças administrativas.

O foco desta tese é a organização/reorganização, considerando aspectos de qualidade, dos processos de desenvolvimento de software, no que se refere à parte técnica, sendo, no entanto, necessário ressaltar a importância da participação e do apoio da alta administração para uma realização bem sucedida deste processo, uma vez que, implantar um processo metodológico necessita,

antes de tudo, de um posicionamento firme, estimulador e cobrador da administração, das gerências e das lideranças de equipes, conforme apresentado em ARAKAKI (1996a).

1.2.1 Justificativas e motivações

No mercado de produção de software, há uma carência técnica e metodológica, no que se refere a planejamento, condução, avaliação, análise, implantação e re - alimentação do processo. Uma metodologia que otimizasse e elevasse os níveis de qualidade e produtividade dos processos de produção de software era imperiosa e, elaborar tal metodologia foi, e é, a principal motivação deste trabalho. A identificação de tais carências e conseqüente necessidade metodológica dentro da indústria financeira foi o alvo destas pesquisas, ressaltando-se que não são exclusivas deste setor, conforme estatística apresentada em YOURDON (1991) apud ARAKAKI (1996d).

As carências, relacionadas com aspectos de qualidade de produto e de processos de obtenção de software, apresentam pontos comuns. Para melhor identificação de um ambiente de produção com as referidas carências, relacionamos alguns, conforme categorização apresentada em MCCONNEL (1996):

1) Pessoal

- ◊ Heroísmos - Alguns dirigentes estimulam de forma errada: os trabalhos são realizados de maneira forçada , sem planejamento, na base do fôlego. É um indício negativo, uma vez que as situações poderão ser

suportadas por heróis temporariamente, mas serão acompanhadas de crises de produtividade durante ou depois do processo;

◊ Colocação/alteração de pessoas durante o andamento do projeto - Tal medida, quando necessária, deve ser acompanhada por re-planejamentos e, se possível, com a antecipação necessária. Os transtornos podem ser de grande impacto e conseqüência, principalmente quando os envolvidos não forem devidamente consultados/informados e o prazo para os ajustes não sejam adequados.

◊ Conflito entre desenvolvedores e clientes - Prejudica o andamento geral dos trabalhos, especialmente nos aspectos de convergências das especificações técnicas referentes aos requisitos esperados pelos usuários;

◊ Expectativas irrealistas - Por deficiência na consolidação adequada de requisitos. A divergência de expectativas é altamente prejudicial ao processo de desenvolvimento de software, ocorrendo sempre quando há consolidação de requisitos inadequada;

◊ Falta de envolvimento dos usuários/cliente - Idem ao de conflito entre eles. A participação dos usuários é fundamental, somente com seu envolvimento pode-se garantir uma convergência, extraindo-se um modelo segundo suas expectativas e que será utilizado para adequação do processo em questão. Somente com seu envolvimento poder-se-á consolidar a funcionalidade esperada para o software;

2) Produtos

- ◊ Requisitos de "ouro" - Por preciosismo, às vezes, estabelecem-se requisitos não necessários, causando transtornos de alocação de esforços adicionais;
- ◊ Flutuação dos requisitos - Os requisitos que mudam durante o desenvolvimento, exigem alterações de esforços nos projetos. Muitas vezes, quando não se replaneja tais mudanças, surgem situações de conflito com o cliente/usuário;
- ◊ Desenvolvedor de "ouro" - Desenvolvedores fascinados com as novas tecnologias e novas ferramentas, importam-se apenas em realizar atividades extras relegando as suas obrigações técnicas de projeto;
- ◊ Desenvolvimento orientado a pesquisas - Um projeto dependente de pesquisas para prosseguir a execução de atividades, pode causar um atraso não previsível, de acordo com a complexidade desta pesquisa;

3) Processos

- ◊ Cronogramas otimistas - Estimativas demasiadamente otimistas podem levar a equipe a uma pressão inadequada, com implicações na qualidade dos trabalhos;
- ◊ Gerenciamento de riscos insuficientes - Ignorar os riscos de projetos, sem um gerenciamento próximo, também é fator de insucesso de projetos. Enfraquece planejamentos por falta de avaliação, não identificando indícios de falhas, nem tomando medidas adequadas, em prazos adequados, como, por exemplo, se algo ocorreu/pode ocorrer

e pode atrapalhar a velocidade do projeto, então, vamos convergir esforços, tomar medidas de forma a impedir que isto não ocorra;

◊ Falhas de sub-contratação - Contratações terceirizadas mal feitas, com descontrole de requisitos, levam fatalmente a projetos ruins;

◊ Planejamento insuficiente - Falhas de planejamento podem acarretar faltas de atividades a serem executadas, e, dependendo da gravidade, podem levar o projeto ao insucesso;

◊ Abandono do planejamento, quando sob pressão - Ao contrário do que se imagina, nesta situação, o planejamento é vital para minimizar erros, ganhando-se tempo e produtividade;

◊ Direto para codificação - "ir direto para a codificação" e depois ficar num ciclo de correção de código, em função das discrepâncias funcionais. Isso ocorre muito, e, ao contrário do que muitos consideram, esta técnica está longe de ser eficaz. Muito pelo contrário, se considerarmos a importância e a frequência com que ocorrem as manutenções durante o seu ciclo de vida em operação. Código desestruturado significa dificuldades de manutenção, gastos de recursos e tempo. Segundo FAGAN (1976) e BOEHM (1989), trabalhos não realizados em tempo de análise e projeto, serão necessários em tempo de codificação, porém gastarão de 10 a 100 vezes mais tempo;

◊ Projeto da Arquitetura inadequada - falhas em tempo de projeto, como por exemplo, de arquitetura do sistema, podem implicar em trabalhos em tempo de codificação com gastos de tempos e de recursos muito

maior do que o necessário devido às deficiências incorporadas ao sistema, decorrentes de projetos mal executados;

◊ Garantia de qualidade relegada - Falta de qualidade, inobservância de fatores de qualidade, é muito prejudicial para o projeto. Cada dia empregado em atividades que garantam a qualidade implica em ganho de 3 a 10 dias para atividades de implementação, segundo JONES (1994);

◊ Controle gerencial insuficiente - Quando o controle é fraco, qualquer problema faz com que o projeto não siga o planejamento, ficando à deriva nas suas atividades. Isto pode ser catastrófico no que se refere a alocação de recursos, tempo e resultados a serem obtidos;

◊ Codificação "porca" - Infelizmente, é muito freqüente. As conseqüências são péssimas para um sistema que vai entrar em produção, pois ele é desenvolvido na base do "codifica e corrige", sem projeto nenhum. O Cap.2 mostra mais detalhes sobre este tipo de características;

5) Tecnologia

◊ Elementos de "magia" - Não existe magia, como novas tecnologias, novas ferramentas para todos os tipos de contexto (JONES, 1994);

◊ Otimismo exagerado com novas ferramentas e produtos utilitários - faz com que se monte um planejamento irreal, podendo levar o projeto ao insucesso;

◊ Mudanças de ferramentas no meio do projeto - Pode ser prejudicial para o projeto;

◊ Falta de controle automatizado de códigos fontes - Não possuir um controle de documentos dos projetos constitui uma falha grave, indicando falta de qualidade, falta de infra-estrutura. É o caso, por exemplo, de um sistema que deve ser desenvolvido por várias pessoas, ou diferentes equipes sem controle de documentos.

Além dos já citados, os seguintes aspectos da qualidade devem ser considerados:

◊ Prazos - Prazos não cumpridos, além do custo direto do projeto, pode trazer prejuízos corporativos consideráveis, pois, como no caso da indústria financeira, pode prejudicar um processo de negócio com perdas de alta monta, por inviabilizá-lo, dentro de um mercado altamente competitivo;

◊ Arquitetura de software - os modelos de análises, as estruturas de projeto e a implementação devem ser elaborados com requisitos de qualidade suficientes para proporcionar facilidades e flexibilidade para atividades de manutenção;

◊ Confiabilidade - transações financeiras, apoiadas por sistemas de software, necessitam da máxima precisão e máxima confiabilidade, com execução sem falhas, com segurança de acesso e execução precisa. Evidentemente, tais características devem ser padrão para todo sistema de software, e, atividades como as transações financeiras

denotam claramente a importância vital do aspecto confiabilidade do sistema de software.

◊ Produtividade - o principal aspecto está relacionado com a necessidade de aumentar a capacidade de produção dos analistas, dotando-os com conceitos que devem ser continuamente aprimorados. Além disso, deve fazer parte da estratégia desse aprimoramento um planejamento no sentido de possibilitar o uso de técnicas, padrões e ferramentas CASE (*Computer Aided Software Engineering*).

Algumas constatações justificam as citações até aqui colocadas. Em atividades realizadas no desenvolvimento e manutenção de sistemas de software, notadamente na indústria financeira, detectou-se uma carência no que se refere a qualidade no desenvolvimento dos produtos de software em dois aspectos:

1. Sistemas de software efetivamente produzidos. Nessa situação, lidar com os erros detectados nas fases integração dos módulos e nas fases de homologação (pré-produção) levam os projetos a custos proibitivos chegando a consumir de 100% a 300% do custo de projeto inicialmente planejado[3]. Adicionalmente, existe também a dificuldade de manutenção corretiva ou evolutiva após a colocação destes produtos em operação.

2. Processos de obtenção desses sistemas de software. Constatou-se falhas graves nos processos, tanto nas atividades técnicas de execução do

[3] Esses números foram constatados pelo próprio autor, através de avaliações de projetos conduzidos, e analisados a posteriori, tanto nos Laboratórios do Departamento da Epusp, como também em projetos de consultoria em instituições financeiras.

desenvolvimento, como também nas gerenciais, onde, administrativamente, não existe controle que possibilite tomar conhecimento ou avaliar o real estado de desenvolvimento, nem uma ação organizada no sentido de corrigir as falhas[4] [5].

1.2.2 Posicionamento da tese

Para posicionar a tese, a Fig.1.1 apresenta um esquema que representa o universo da área de engenharia de computação, onde estão relacionados temas, tecnologias, métodos, teorias e ferramentas.

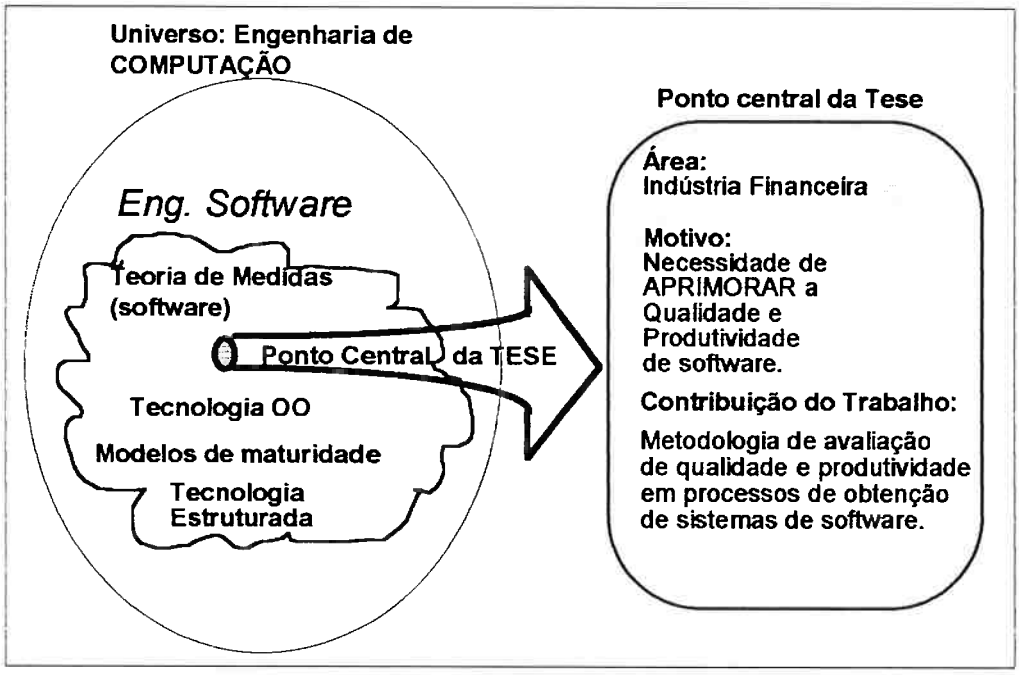

Fig.1.1 - Posicionamento do trabalho da tese de doutorado - foco do trabalho.

[4] Dentro da classificação de maturidade de software, constatou-se que nestas instituições, o nível CMM (Capacity and Maturity Model estabelecido pelo SEI) era o de nível 1, ou seja, inicial, onde a visibilidade do processo é praticamente nula, ou seja, como se os processos fossem caixas pretas onde de um lado entra alguns requisitos/necessidades e de outro sai um sistema de software.

[5] SEI - Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

O esquema da Fig.1.1 indica que o universo maior é a área de engenharia de computação. A tese baseia-se em conceitos de Engenharia de Software, especificamente com relação às metodologias de produção de software. O ponto central do trabalho é a proposição de uma metodologia para reorganizar ambientes de obtenção de software críticos, que apresentem características de maturidade em software equivalente ao nível inicial da categorização do SEI/CMM.

Esses ambientes, no caso da indústria financeira, incorporam processos críticos, vitais na construção e manutenção de infra-estrutura de software que apoiam processos de negócios de relevante importância para manter a competitividade no mercado. Porém, durante o período de alta inflacionária, esses processos não eram avaliados por itens como custos e prazos, ou seja, o objetivo era produzir sistema, independentemente se houvesse desperdícios de recursos humanos ou materiais (ARAKAKI, 1996a).

A metodologia caracteriza-se por incorporar práticas da tecnologia orientada a objetos, ajudando no planejamento e análise de um sistema até a sua construção final. Em especial, estimula a reutilização, um tema já antigo, porém com grande potencial de aplicação, por meio de um recurso que deve impactar produção de software neste final de década - a componentização - onde um aplicativo de software é organizado como uma integração de componentes, de forma similar aos que são conduzidos os projetos de circuitos eletrônicos digitais. Além disso, a metodologia prevê a incorporação de mecanismos de avaliação do processo e do produto através do uso de métricas.

Um processo, organizado e operacionalizado segundo esta metodologia, se caracterizará por tornar visíveis as atividades do projeto e por ter recursos de avaliação da qualidade com o uso de métricas. Tais características estarão presentes em todas as etapas de obtenção do software, desde a formalização de uma necessidade de negócio por uma solicitação, até a sua liberação final para a produção, incluindo estratégias para as atividades de manutenção corretiva ou evolutiva que certamente os sistemas de software necessitarão. Como conseqüência, o processo pode se categorizar como o do nível 2 (repetitivo), do modelo SEI/CMM, e rapidamente passar para o nível 3 de maturidade e capacidade em software.

A metodologia, incorporando as referidas características no processo, estabelece o conceito de desenvolvimento rápido, eficaz e simples. Este conceito corresponde a desenvolver sistemas de forma controlada, racional, direta, sem muitos retrabalhos e com a confiabilidade adequada - ou seja - "fazer do jeito certo da primeira vez".

A adoção da orientação a objetos se mostrou eficaz neste ambiente pela possibilidade de uma modelagem de sistemas através de objetos de negócios, o que permite uma participação direta do cliente (usuário do futuro sistema) em praticamente todo o ciclo de desenvolvimento. Mas isso não é exclusividade desta técnica: o modelo estruturado (funcional) também pode ser eficaz na construção de um sistema de software, porém, constatou-se que a orientação a objetos é superior por apresentar mecanismos e técnicas de abstração através de objetos de negócios que possibilitam o particionamento de um sistema e facilita

lidar com complexidades através de recursos como encapsulamento (ver Cap. 4). A uniformidade de notação e de conceitos em todas as fases do desenvolvimento, da modelagem à implementação, facilita sobremaneira a integridade e uniformidade das informações de projetos, minimizando erros e retrabalhos que consomem recursos e tempo de um projeto.

Outro fator importante a ser considerado é a tendência de se utilizar os recursos das redes de computadores como a rede Internet (ARAKAKI, 1996a). Organizar o software em componentes que refletem partes do processo de negócio facilita a distribuição de software por tais redes. Mais detalhes podem ser vistos em MOWBRAY (1995).

Com relação aos mecanismos de avaliação de qualidade do processo e dos produtos de software, é inegável a importância do uso de métricas de software, onde vale fazer a seguinte citação do cientista Lorde KELVIN (1989) apud OTT (1996): "*When you can measure what are you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stages of science.*". Para aprimorar um processo, é melhor conhecê-lo através de medidas, que podem ser registradas em base de dados históricos, modelar o comportamento desse processo para então propor mudanças.

1.3 Contribuição

Há dois pontos importantes de contribuição: O primeiro refere-se à parte conceitual, teórica; o segundo refere-se à parte prática, a ser empregada pela comunidade, em especial, nas áreas que lidam com o desenvolvimento e manutenção de software.

1.3.1 Contribuição 1: parte teórica

A tese contribuirá para o aprimoramento de avaliação de qualidade e de resultados na área de desenvolvimento de sistemas de software, na medida em que entrosa conceitos e procedimentos de três áreas: Engenharia de Software, Teoria de Medidas e Tecnologia Orientada a Objetos. A Fig.1.2. esquematiza a tese em relação aos conceitos teóricos utilizados, que são comentados nos itens que seguem.

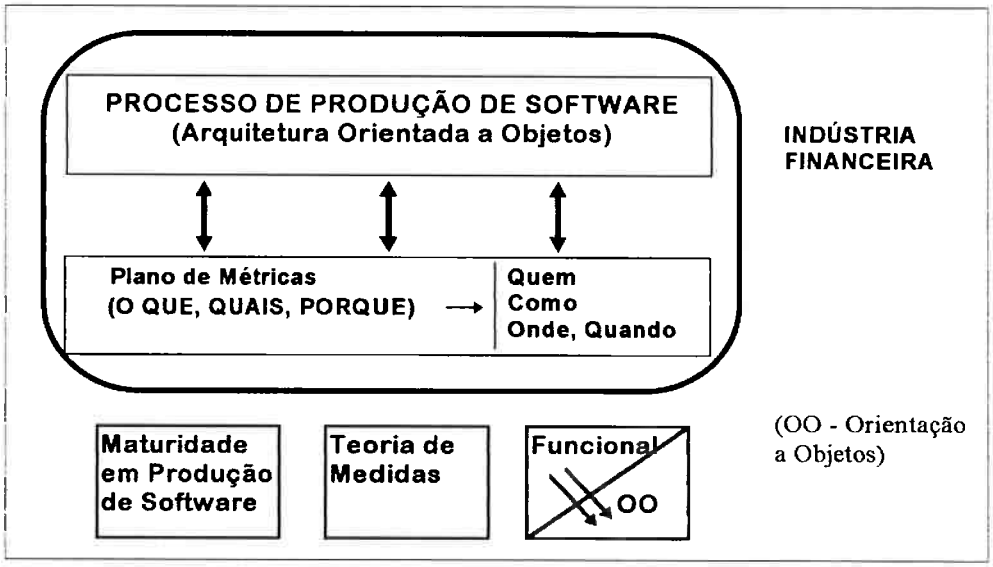

Fig.1.2 - Esquema da tese e os conceitos teóricos.

Dos Conceitos:

1. **Engenharia de Software** (PRESSMAN (1988), JACOBSON (1995)): como o princípio básico é a obtenção de software racional e com qualidade, tanto do ponto de vista técnico, como do financeiro, seus conceitos são direcionados para estabelecer técnicas, atividades, ferramentas e fluxo de trabalho que atendam todo ciclo de construção do software, abrangendo especificação, projeto, implementação, operação e manutenção.
2. **Teoria de Medidas** (MELTON, 1995): extrai-se a base teórica para conceituar métricas e especializá-las na área de engenharia de software. A principal contribuição desta teoria é a de estabelecer diretrizes para definir métricas, estabelecer um significado para elas e como obtê-las (como coletá-las), com base matemática capaz de subsidiar a elaboração de modelos de medição direta e indireta e de modelos de análise matemática.

   Um fato levantado é a constatação de que muitos programas de métricas em software falham (JONES, 1994) devido a uma série de fatores. São os casos, por exemplo, onde os dados de medidas são extensamente coletados e não são analisados adequadamente, por falta de significado semântico deles dentro do ambiente, denotando falhas de planejamento de implantação de métricas. Esta constatação nos ensina a não só coletá-los, mas, principalmente analisá-los e interpretá-los adequadamente dentro do ambiente no qual estamos trabalhando.

3. **Tecnologia Orientada a Objetos**: amplamente utilizada com sucesso por grandes fabricantes de produtos de software dentro de um mercado extremamente competitivo, está sendo incorporada na indústria financeira de forma lenta, gradativa e irreversível, conforme levantamentos recentes (BOOCH, 1996). Um dos principais pontos da tecnologia é a possibilidade de componentização: a expectativa de fortes melhorias na produção de software por reutilizar componentes de software estáveis, por aprimorar a robustez dos sistemas de software, e contribuir para o aumento da produtividade.

1.3.2 Contribuição 2: o lado prático

A tese deve contribuir na organização de processos de software da seguinte maneira:

- ◊ Contribuição prática 1 - Metodologia para organizar processos de produção de software através da incorporação da tecnologia orientada a objetos em todo o ciclo de desenvolvimento de software, incorporando o mecanismos de avaliação de qualidade, por métricas;
- ◊ Contribuição prática 2 - Acelerar o amadurecimento dos processos de produção de software, guiados pelas definições estabelecidas pelo SEI, no modelo CMM. Um processo reorganizado por esta metodologia deve estar pronto para se submeter à classificação equivalente ao nível repetitivo (nível 2 - modelo SEI/CMM);

◊ Acelerar a produtividade através da criação de mecanismos de componentização que evidencia e separa dois tipos de especializações dentro do ambiente de desenvolvimento:

- *Desenvolvedor especialista em processo de negócio:* São pessoas, normalmente internas da empresa que detêm o conhecimento dos processos de negócios e estruturam as aplicações de software com componentes de software. São os responsáveis pela definição dos componentes[6];

- *Desenvolvedor especialista em implementação dos componentes:* São pessoas não necessariamente da empresa (terceirizados) com competência para implementar os componentes, com base nas especificações elaboradas por desenvolvedores de aplicação.

1.4 Ineditismo

Atingir um grau de maturidade e capacidade de software, segundo os padrões internacionais, é um processo difícil, conforme atestam as estatísticas (ARAKAKI, 1996d). Buscar uma estratégia agressiva, ajustada ao ambiente alvo, e respeitando o seu atual quadro de pessoal técnico, resultou numa metodologia, prática para ser usada em especial pela indústria financeira. Nessa indústria, uma característica deve ser ressaltada: apesar de haverem excessos em aplicação de recursos materiais e humanos (Gurovitz, 1997), os sistemas de software são

[6] Componente - Implementação física de um conjunto de objetos.

efetivamente construídos, ainda que os processos não sejam adequados. Dificilmente um projeto é abandonado, pelo contrário, esforços são alocados até a concretização final do mesmo. A adoção de uma metodologia de avaliação e realização tornará possível a identificação das falhas existentes no processo, otimizando e adequando todos recursos de forma racional, objetiva, precisa e econômica.

Assim, a parte inédita da tese é o estabelecimento de uma metodologia para organizar um processo de obtenção de software através da incorporação de mecanismos de avaliação de qualidade em ambientes da indústria financeira, onde o nível de maturidade e capacidade pode ser considerado inicial (SEI/CMM). Para isso, são incorporados e aplicados conceitos da tecnologia orientada a objetos, conceitos e recursos de métricas aplicados para avaliação de qualidade. O ambiente reorganizado fica pronto para se classificar como sendo de nível 2 e com condições de atingir o nível 3 (SEI/CMM).

Várias áreas da indústria estão em um estágio avançado no uso de métricas como estratégia de controle de projetos de software. As principais normas americanas como os publicados pela DoD (U.S Department of Defense) e a NASA são amplamente utilizadas nestes setores (DoD2167 (1985) e MELTON (1995)).

O mesmo não acontece na grande maioria de empresas que compõem o segmento da indústria financeira, onde prevalece os processos de negócios que se apoiam em sistemas de software muito antigos, escritos em linguagens como Cobol, Assembler, Clipper e C. Nesses casos, o fato de haverem normas, padrões,

metodologias genéricas não implica na melhoria de processos conforme discutido por PFLEEGER (1994). Em geral, as instituições financeiras não conseguiram estabelecer planos de amadurecimento dos seus processos de execução de projetos apoiado em números (medidas) como em outros setores da indústria. Com a retração global da economia, os investimentos devem ser racionalizados e a área de informática, que serve de apoio aos processos de negócios, deve ser eficaz e com baixo custo. Com isso, as empresas buscam reorganizar, otimizar seus processos, sendo que o uso de técnicas de orientação a objetos é um fato relevante na maioria das empresas e deverá ser também para as instituições financeiras. A presente tese pretende contribuir para isso.

Na indústria financeira, especialmente no Brasil, cada área responsável pelo processo de produção de software tem a sua característica operacional que reflete as diretrizes da empresa. Algumas mantêm o total controle do desenvolvimento, permitindo-se eventuais trabalhos de terceirização na implementação, mas mantendo o controle dos projetos. Em outro extremo, existem aquelas que terceirizam completamente, fazendo apenas o controle gerencial[7]. No atual estágio, o controle gerencial e técnico das atividades tem se mostrado bastante frágil, especialmente em projetos de sistemas com arquitetura Cliente-Servidor, com reflexos diretos na qualidade da implementação dos software produzidos: prejuízos em custos e os prejuízos decorrentes de prazos especialmente na validação[8] e na manutenção destes sistemas. As perdas normalmente acontecem

[7] Essas situações são constatadas através de participações de trabalhos de consultoria junto a iniciativa privada, mostra inúmeras situações de composição de equipes próprias de desenvolvimento com equipes de técnicos alocados por terceiros.

[8] Homologação do sistema: Verificação completa do sistema para atestar a sua correta funcionalidade.

por falta de um respaldo metodológico no processo de execução e controle dos projetos. Isso acontece porque a grande maioria das empresas estão nos níveis iniciais, que equivale ao nível 1 do modelo de maturidade e capacidade em software (SEI/CMM). Porém, é vital para as empresas a evolução para o nível equivalente ao nível 2 do modelo SEI/CMM, onde o processo se caracteriza por ser repetitivo, visível e com vários fatores de qualidade incorporadas no que se refere à confiabilidade, manutenibilidade e produtividade. Esta tese pretende suprir esta necessidade de evolução.

1.5 Estrutura da Tese

A metodologia, objeto desta tese, é apoiada por um conjunto de textos, de teoria e também apresenta resultados práticos relevantes ao trabalho. Veja a Fig.1.3.

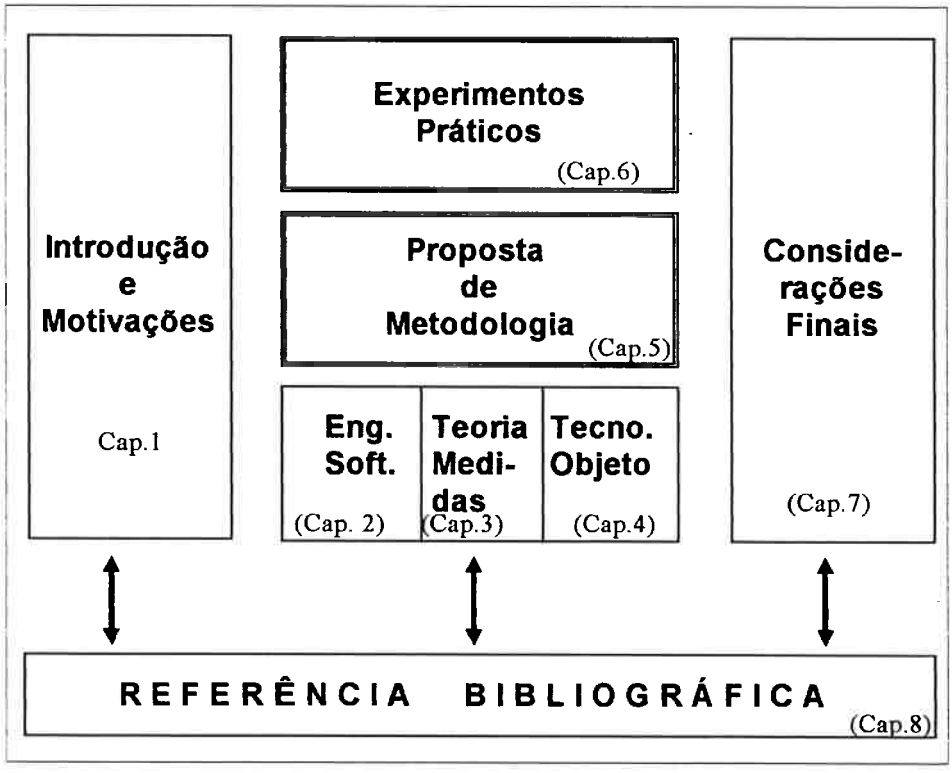

Fig.1.3 - Esquema de organização do texto da Tese.

O Capítulo 1 descreve as oportunidades e as motivações que direcionaram a pesquisa relacionada com esta tese. Os capítulos 2, 3 e 4 formam a base teórica da metodologia: o Cap.2 refere-se aos aspectos da engenharia de software, especialmente relacionados com a qualidade e produtividade; o Cap.3 descreve os conceitos relacionados com as métricas de software, extraídos da teoria de medidas; e o Cap.4 apresenta conceitos da tecnologia orientada a objetos relevantes aos processos de produção de software, sem se prender a nenhuma metodologia.

O Cap.5 apresenta a metodologia que serve de base para a definição e organização de um processo de software. O Cap. 6 apresenta um processo organizado e em implantação dentro da indústria financeira, como exemplo de aplicação da metodologia. O Cap. 7 apresenta os resultados até aqui obtidos e os próximos passos a serem seguidos.

O Cap.8 é constituída da referência bibliográfica utilizada na tese.

2. ASPECTOS DA QUALIDADE DE SOFTWARE

Objetivo do Capítulo:

Apresentar aspectos relacionados com a avaliação de produtos e de processos de produção de software. Discutir alguns padrões de referências de mercado como o ISO 9000, CMM e o Malcom Baldrige.

Tópicos do Capítulo:

2.1 Desenvolvimento de Sistemas como um Processo Industrial
2.2 Qualidade de Produto e de Processo de Software
2.3 Processos de Produção na Indústria
2.4 Capacitação e Maturidade em Produção de Software

2.1 Desenvolvimento de Sistemas como um Processo Industrial

A seguir discute-se como obter produção de software com qualidade através do estabelecimento de processos similares aos processos de produção de outros setores industriais.

2.1.1 Maturidade no desenvolvimento de software

A indústria de desenvolvimento de software é bastante jovem, da ordem de 40 a 50 anos e está em busca de uma maturidade ainda não alcançada como ocorrido em outras atividades industriais muitas vezes centenárias tal qual a área de construção civil. A maior consequência disto é a falta de práticas consolidadas para o desenvolvimento e operação (manutenção) de software.

Deve se considerar que o desenvolvimento de software deve acompanhar os métodos e as tecnologias que permitem a elaboração de projetos criativos, porém também deve fornecer mecanismos para a realização de software como um processo industrial racional, onde o objetivo não é apenas a construção de um sistema com excelência em engenharia, mas que seja um produto viável de ser explorado industrialmente, com todos os aspectos de competitividade que o mercado impõe, tanto a nível de qualidade e custos e como também de estar sempre atual, acompanhando as necessidades dos usuários.

Vários organismos internacionais buscam a definição de processos de obtenção de software capazes de garantir um nível de requisitos de qualidade, através de estabelecimentos de processos que detalham atividades, infraestrutura,

preparação das equipes técnicas e estratégias de verificações de evolução. Alguns dos mais importantes são detalhados no item 2.4 deste capítulo.

Para entender a importância de estabelecer processos bem definidos, JACOBSON (1995) faz uma analogia com área de construção civil, com a ajuda de um especialista da área. Por que a área da construção civil? Porque é uma área (centenária) onde o conceito de processo industrial está bastante maduro e serve como um paralelo bastante importante para o nosso contexto de avaliação da qualidade do software.

As fases de uma construção são definidas por princípios que refletem os conceitos de uma boa engenharia de construção muito bem definidos, com guias de procedimentos nas várias atividades de um projeto de desenvolvimento. Para representar estes princípios, atividades e procedimentos, podemos representar através de uma figura esquemática (Fig.2.1).

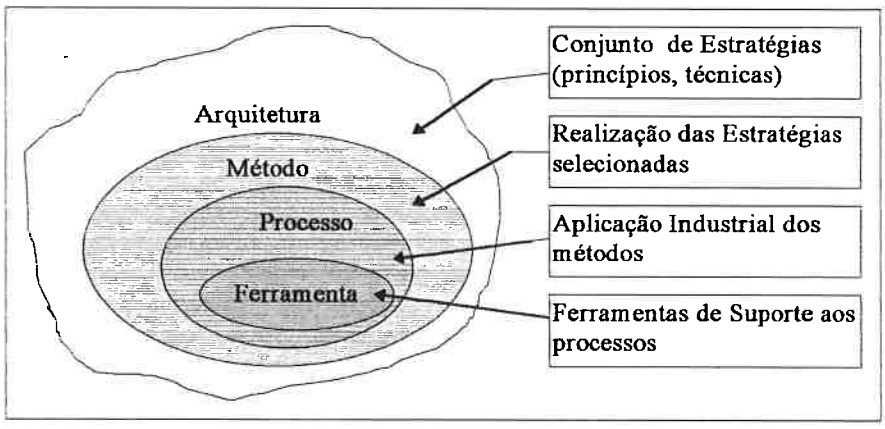

Fig.2.1 - Mecanismo industrial de processos de contrução.

A camada de **arquitetura** refere-se ao conjunto de princípios, conceitos e técnicas, selecionadas de um universo que refletem a característica estrutural de

todas as construções a serem adotada numa organização Por exemplo, construções que utilizam pré-moldados (poderia ser o conjunto de construções customizadas).

O método refere-se ao conjunto de procedimentos, para aplicar os princípios definidos na arquitetura para a obtenção da construção. O **processo** permite que o método seja escalável, ou seja, possibilita a sua aplicação em todos os detalhes dos projetos que são constituídos de muitas partes, muitas atividades interativas e diversas pessoas envolvidas. O nível **ferramenta** refere-se aos aspectos de recursos técnicos com os quais os processos são implementados.

Para ilustrar essa a organização dos processos em escala de aplicação, a tabela a seguir apresenta um detalhe maior resumindo cada fase de uma construção (Fig.2.2). Observe que em cada etapa de uma construção civil, os níveis Arquitetura, Método, Processo e Ferramentas estão presentes dentro da linha de produção de uma construção civil, desde a identificação das necessidades até a sua liberação para o uso.

A fase de operação de uma construção também deve ser cuidadosa, pois se for considerada que a construção deva existir por muitos anos (é diferente de uma construção temporária para shows, por exemplo), ela deve sofrer manutenções ou alterações para acompanhar as novas necessidades. Para isso, essas características devem ser incorporadas no nível de arquitetura desde a fase de projeto, ou seja, os métodos, os processos e as ferramentas devem ser executadas de tal maneira que ela possa ser adequadamente mantida, conforme requisitos.

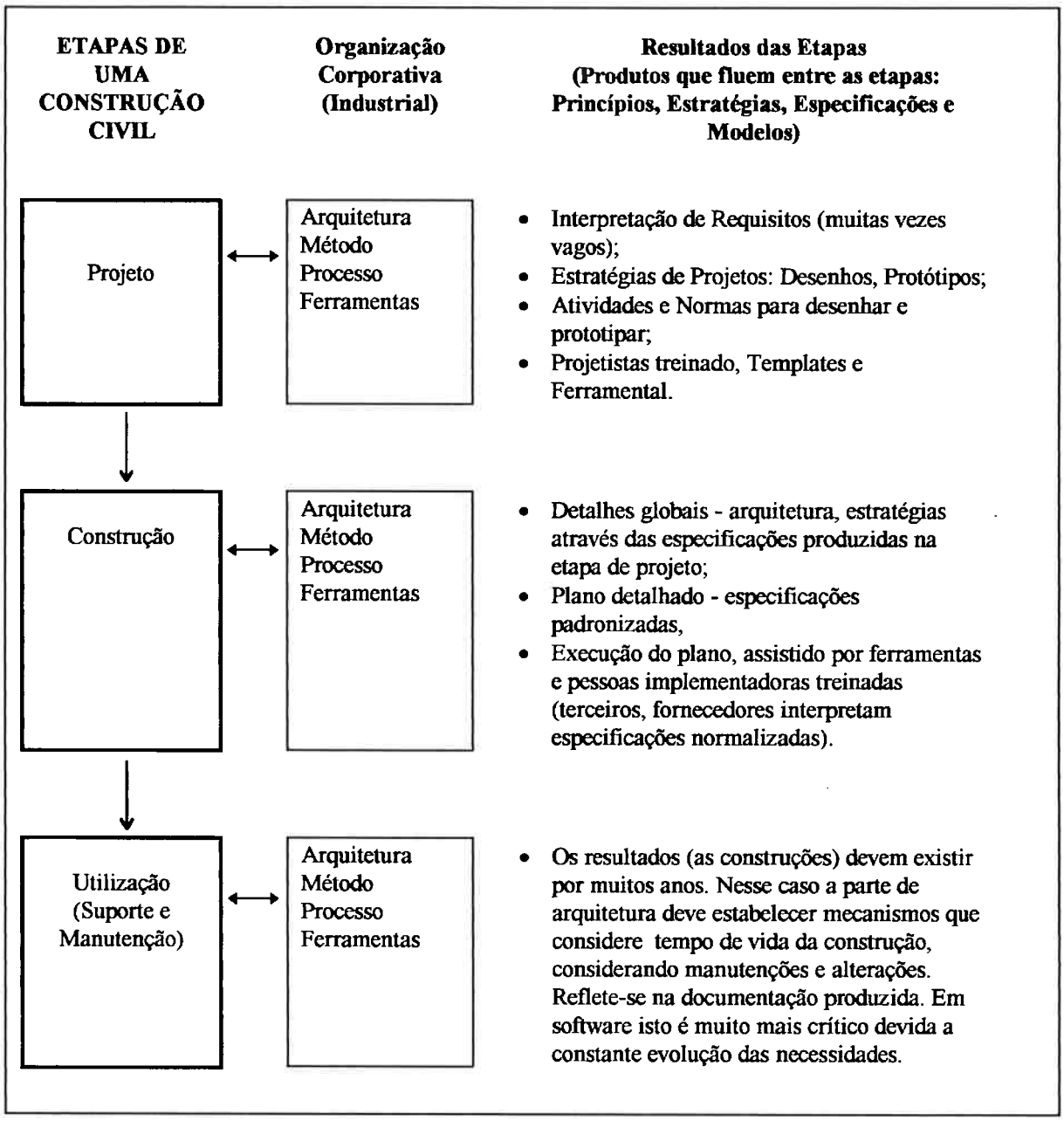

Fig.2.2. Exemplo dos níveis ARQUITETURA, MÉTODO, PROCESSO E FERRAMENTA presentes em todas etapas de desenvolvimento de uma construção civil.

Nessa indústria, todo o ciclo de desenvolvimento de uma construção, envolvendo desde a fase de requisitos iniciais até a liberação da construção, o elemento fundamental é a documentação que deve ser mantida atualizada, em ordem, e serve como um meio de comunicação entre todos os participantes, de forma independente de pessoas. Nessa indústria, as ferramentas de CAD/CAM

(Computer Aided Design e Computer Aided Manufacturing - ARAKAKI (1991)) constituem-se em poderosas ferramentas de qualidade e produtividade, pois ajudam a manter de forma controlada e atualizada a documentação que serve de base para os projetos em todo o ciclo de vida das construções, incluindo a fase de manutenção. Outro aspecto relevante que essa indústria pode oferecer como referência para a área de software é o processo de reutilização - elementos prontos são exaustivamente utilizados promovendo muitos ganhos em produtividade pois reduz prazos, custos e melhorias na qualidade geral.

A área de desenvolvimento de software apresenta características muito parecidas com as da área de construção civil. Observa-se que desde a década de 1980, movimentos em busca de reutilização em software foram iniciadas e ainda estão em fase de amadurecimento. Portanto, a fundação básica que deve apoiar a construção de sistemas de software deve ser estruturada de maneira muito semelhante, pois as etapas mostradas na Fig.2.2 são caracterizadas num processo sistemático de desenvolvimento de um sistema. JACOBSON (1995) resume os pontos que devem nortear um processo considerado industrial:

◊ O processo deve apresentar resultados visíveis, independentemente de quem o executa;

◊ O volume de saída de produção não deve afetar o processo;

◊ Partes do processo devem ser repassáveis por sub-contratos para vários fornecedores o terceiros;

◊ Deve ser possível o uso de blocos pré-definidos (por exemplo, componentes);

◊ Deve ser possível planejar e calcular o processo com a maior precisão possível;

◊ Todas as pessoas treinadas no processo devem apresentar performances similares.

2.1.2 Disciplina da engenharia de software (ciclo de vida de software)

Na indústria, o desenvolvimento de software faz parte de uma organização mais ampla. É o caso de um banco, onde o processamento de um departamento serve para os demais departamentos. Numa indústria o planejamento da produção tem influência direta com o setor de compras e também com a área de *marketing* e vendas responsáveis por atender aos clientes.

Desenvolvimento de Sistemas - faz parte de uma grande atividade

O desenvolvimento não é um elemento isolado dentro de uma organização. Muito relacionado com o desenvolvimento de sistemas, normalmente existe uma área com a tarefa de identificar necessidades e oportunidades de negócios (Marketing) e uma outra área responsável por transformar em realidade as oportunidades identificadas (Área de Produtos). A área de desenvolvimento de software interage corporativamente com essas grandes áreas, de modo que vale o seguinte esquema (Fig.2.3).

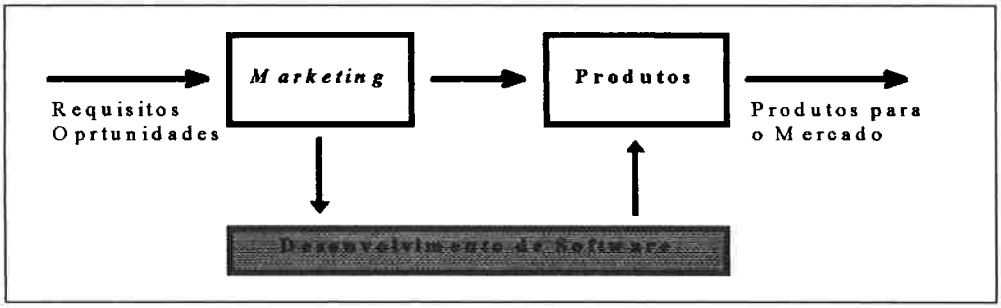

Fig.2.3 - Relacionamento de um desenvolvimento de um sistema dentro de uma organização.

Atividades de um desenvolvimento de um sistema

Detalhando o processo de desenvolvimento (Fig.2.4), ele pode ser vista como uma composição de três grandes atividades: uma está relacionada com análise, cujo o foco de tarefas é orientado pelo processo de negócio, de forma independente da implementação computacional; outra atividade está relacionada com a construção do sistema de software, conforme especificações obtidas nos modelos das atividades de análise. Ela envolve as etapas de projeto e implementação computacional (ao nível de código). A última atividade corresponde à verificação final do sistema desenvolvido para verificar e refinar as implementações até que ele atenda aos requisitos definidos no início do processo.

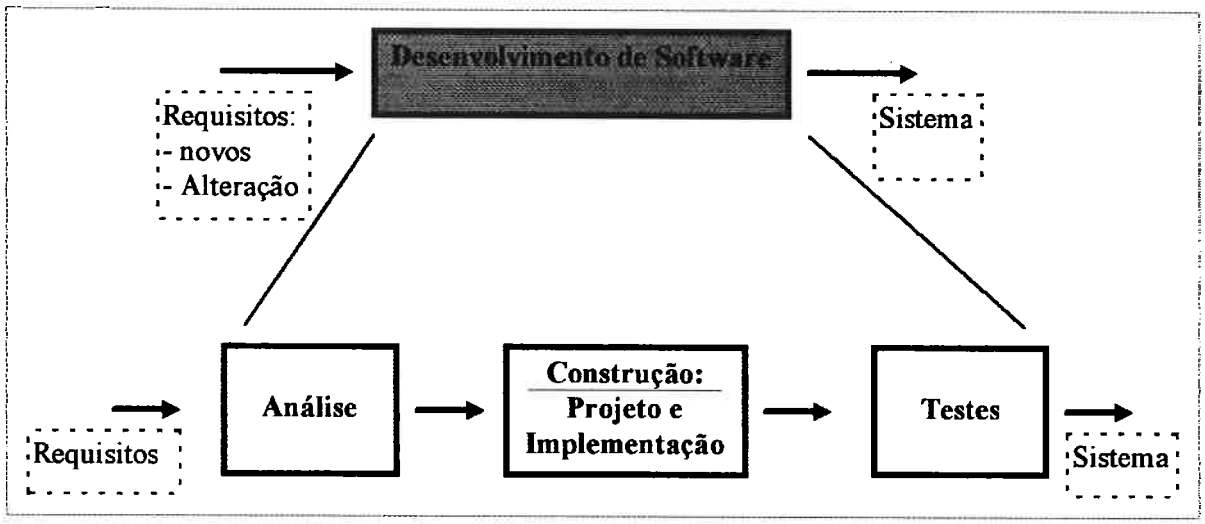

Fig.2.4 - Grandes atividades de desenvolvimento de um sistema.

Requisitos como entrada de um processo de desenvolvimento

A entrada básica para um processo de desenvolvimento é a especificação de requisitos. Por exemplo num sistema técnico como um controlador de um sistema de telecomunicações, o comportamento do sistema junto aos sensores e atuadores constituem o conjunto de requisitos de entrada. Num administrativo,

como controle de pessoal de um departamento de recursos humanos, é preciso realizar uma análise de necessidades, problemas e tendências de processo em questão, normalmente denominado como o processo de negócio. A especificação do processo de negócio, com as suas regras determinam a especificação de requisitos. Pode existir o caso em que a atualização tecnológica de um sistema de software já existente, por exemplo, reimplementar um sistema operacional, segundo uma nova tecnologia. Nesse caso, a especificação de requisitos pode se apoiar no próprio conjunto de recursos funcionais do sistema antigo.

A saída de um processo de desenvolvimento de um sistema é obviamente o próprio sistema executável em uma plataforma computacional, porém complementado por conjunto de elementos. São as documentações de software de apoio: Alguns deles devem atender aos usuários operadores do sistema; outros devem atender aos técnicos responsáveis pela administração e suporte técnico do sistema durante o seu período operacional apoiando o suporte a manutenção; e por fim um conjunto da documentação deve servir de registro da interface deste sistema com outros sistemas técnicos.

As partes interessadas num desenvolvimento de um sistema

Lembrando que além da área de desenvolvimento, existem outras áreas envolvidas no processo de desenvolvimento que devem ser observadas no processo de software bem consolidado. Do ponto de vista do desenvolvimento, normalmente, a área que faz o pedido de desenvolvimento é diferente da área usuária do sistema. Nesse caso, deve-se criar uma estratégia para que as

especificações de requisitos considere o usuário do sistema (direto - aquele que opera e indireto - aquele que usufrui os benefícios proporcionados com o sistema). As atividades devem ser orientados (ou centrado) no usuário para garantir uma boa aceitação e uma boa assimilação por parte dos usuários e como conseqüência, obter o sucesso operacional do sistema.

2.1.3 Software como resultado de uma linha de produção

Processos industriais consideram o fato de que o resultado da produção deve entrar em operação e geralmente deve sofrer processos de manutenção e suporte técnico. Assim, um produto[1] de software deve ser desenvolvido tendo uma consideração especial no que se refere à manutenção, ou seja, um software deve ser desenvolvido para mudanças.

Desenvolvimento incremental

Geralmente um software demora alguns anos em operação, e o seu desenvolvimento também demora mesma grandeza de tempo pois sistematicamente sofre alterações evolutivas ou corretivas. Muitas indústrias de produtos de software para computadores pessoais também lançam produtos de software na forma de versões, onde através de estratégias eficazes de marketing conseguem liberar novas versões de software a cada 4 a 5 meses[2]. Do ponto de vista de processos industriais, organizar um sistema de software em versões, é uma estratégia denominada desenvolvimento incremental. Nesse caso, sempre

[1] Neste texto, entenda-se produto de software ou sistema de software como sendo uma referência ao produto de um desenvolvimento de software.
[2] Produtos de fabricantes como a Microsoft ™ lançam versões de um mesmo produto em tempos bastante curtos, como uma eficaz técnica de marketing.

existe um ponto de partida de um software que é construído a partir de alguns requisitos e transformada no que se chama de versão 1.0. A partir daí, cada nova versão corresponde a um incremento de especificações que adicionadas aos requisitos já existentes na versão anterior, estabelecem o desenvolvimento de uma nova versão. Normalmente um processo de definição de uma versão é feita em função das características funcionais que atendam ao usuário final ou de suporte técnico, ou seja, dificilmente em função de uma arquitetura interna de implementação. Definir versão caracterizada pelo conjunto de funções é a maneira mais prática e é o que tem sido realizado (JACOBSON (1995)). A elaboração de cada versão sofre o mesmo conjunto de atividades como o relatado na Fig.2.4. Assim, uma representação possível deste fato pode esquematizada como mostrado na Fig.2.5. O circulo representa o ciclo de vida do software e cada grande fatia é a liberação de uma versão. O esquema representa um exemplo onde um sistema hipotético de CAD apresenta um conjunto de funções de desenho interativo, um conjunto de funções para construir bibliotecas de símbolos específicos e também um conjunto de funções para preparação de dados que sirvam para a produção como lista de materiais, planilhas de custos e outros. Uma maneira de organizar o cronograma desenvolvimento algo como mostrado na Tab.2.1

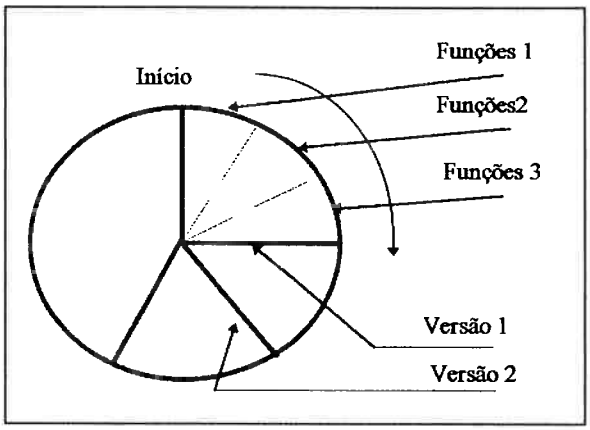

Fig.2.5 - Processo de desenvolvimento incremental JACOBSON (1995).

Funções	Tempo
Núcleo Interativo de Desenho - Funções 1	
Editor de Símbolos para Bibliotecas - Funções 2	
Pós-Processador CAM para Produção - Funções 3	

Tab.2.1 - Cronograma global de um Sistema CAD/CAM hipotético, organizado em grupo de funções afins. A versão 1 é implementada através da execução destes três grupos de funções.

Uso de Prototipação

O desenvolvimento incremental é uma tendência dos software da década de 90. Com o aprimoramento de recursos gráficos na implementação de GUI, o uso de prototipação pode ser um recurso eficaz dentro deste conceito de organizar a implementação voltada para usuários. Nessa situação, o protótipo, se bem usado constitui-se numa poderosa técnica para refinar requisitos de sistema junto ao usuário, pois complementa documentos de especificações com protótipos computacionais que permitem sensibilizar a percepção do usuário quanto aos recursos do futuro sistema a ser desenvolvido, facilitando a sua participação dentro das atividades de análise, quando os requisitos do sistema espelham as funções esperadas para o sistema. O uso desta técnica deve ser cuidadosa, com a

conscientização do próprio usuário quanto as vantagens e limitações para evitar que o usuário crie uma expectativa inadequada em relação a um protótipo e queira, por exemplo, utilizá-lo em operação real. Esta técnica é excelente, porém a sua má aplicação pelos desenvolvedores valeu-lhe uma reputação de ser uma técnica "rápida e suja". Porém a sua principal utilização é servir de recurso para convergência de uma especificação de requisitos de um sistema, considerando o fato de que muitas vezes o usuário sabe das necessidades e da importância de um novo sistema, porém, sem muita riqueza (ou nenhuma) de detalhe. O protótipo serve para um contato prévio, que concretiza de forma parcial o sistema a ser implementado, o que estimula o detalhamento mais aprofundado, importante para elaboração da especificação de requisitos do sistema a ser desenvolvido.

Reutilização/Componentização

Reutilizar (COX(1990) E JACOBSON (1995)) partes previamente prontas na construção de um sistema é um modo significante de redução de custo dentro ciclo de vida por uma série de fatores. Um dos principais é o fato de usar elementos prontos, testados, obtém-se ganhos de tempo na validação, além de se minimizar retrabalhos (evita-se o "reinventar a roda"). A área de engenharia eletrônica utiliza estes conceitos há muito tempo: o projetista de um circuito tem disponível catálogos de componentes prontos para o uso com os quais elabora o seu projeto. A tecnologia orientada a objetos pode contribuir para o uso de componentes, pois as suas características são bastante adequadas para usar técnicas de abstrações e encapsulamento, o que facilita lidar com complexidades, como pode ser visto nos capítulos subsequentes desta tese.

Organizar a estruturação de sistemas por componentes exige uma disciplina relacionada com a administração de componentes, especialmente nas atividades de análise e projeto, onde os elementos de software devem ser projetados tendo a visão de preparação de novos componentes, ou seja, generalizações devem ser consideradas para dar espaço de utilização dos componentes. Ver o Cap. 4, 5 e 6.

Outro fator a ser considerado é que, contrariamente aos que imaginam que a reutilização só acontece apenas na codificação, ela deve ser considerada em todos os níveis, desde a etapa de análise. Nessa situação, desenvolver sistemas voltadas para o reuso pode exigir atividades adicionais de generalização de componentes e administração do reuso, ao invés de ser totalmente polarizado para a implementação específica de sistemas (JACOBSON (1995), MICROSOFT (1995)).

2.1.4 A infraestrutura de um processo de desenvolvimento de software

Pirâmide de Jacobson

Desenvolvimento de um sistema de software deve ser realizado com base num **processo** constituídos de vários passos de atividades que podem ser assistidos por um conjunto de **ferramentas**. O processo por sua vez tem como base o **método** que corresponde a um conjunto de estratégias, princípios, técnicas selecionadas de um universo de métodos adequados e coerentes com uma determinada classificação de **arquitetura** de sistemas de software. A Fig.2.6 mostra a base de uma metodologia, segundo JACOBSON (1995).

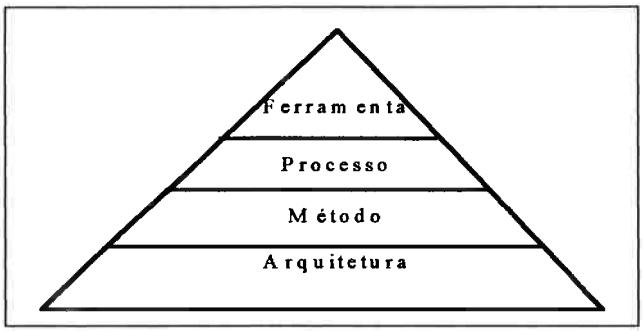

Fig.2.6 - A organização de uma metodologia, com os seus elementos de sustentação: Ferramentas, Processos, Métodos e a Arquitetura.

As propriedades estruturais de um sistema de software é um elemento chave. Uma boa estrutura permite uma boa documentação, fácil entender, testar e modificar, ou seja, define a complexidade de se lidar com o sistema durante o seu ciclo de vida. A **arquitetura**, conforme esquematizado na Fig.2.6, reflete as propriedades estruturais de um software. Assim, organizar um software com funções e dados em separados define uma arquitetura. Organizar o software como um conjunto de objetos autônomos, que se comunicam por mensagem define uma outra arquitetura. Os modelos de análise, projeto e implementação definem a arquitetura de software.

Método aqui entendido como um conjunto de procedimentos planejados para obter uma meta específica de desenvolvimento de software não deve ser confundida com metodologia que corresponde à ciência de métodos. Um método é baseado numa arquitetura específica e o requisito fundamental de um método é simplificar o desenvolvimento de um sistema de uma particular arquitetura.

Um **processo** é a aplicação do método em larga escala, por exemplo, em escala industrial, de forma racional aplicável de forma efetiva por diversas pessoas. Para

se fazer um paralelo, tomemos o modelo de desenvolvimento de um automóvel. Ao se construir o protótipo de série (incorporando, por exemplo novos combustíveis, novos mecanismos anti-impacto), feito segundo um método que reflete uma arquitetura do carro. Uma vez aprovado o protótipo desenvolvido, a sua industrialização deve ser realizada através de processos que reflitam o método utilizado para o seu desenvolvimento, porém em escala industrial, onde se planeja não só na criação do veículo, mas a estruturação do seu ciclo de vida, desde a montagens de partes, desenvolvimento de fornecedores de componentes específicos, testes em separados, mecanismos de controle de qualidade, e infraestrutura de apoio à manutenção. Em software não é diferente. Os processos devem ser quebrados em sub-processos, sempre refletindo os métodos, que por sua vez refletem a arquitetura específica de estruturação dos sistemas a serem desenvolvidos.

Um método define sempre um caminho básico para a obtenção de um software. Os processos são organizados de tal maneira que podem flexibilizar atividades de acordo com as necessidades corporativas. Por exemplo a definição de roteiros distintos de desenvolvimento de um sistema novo exige um roteiro diferente de uma manutenção de um sistema existente, que por sua vez pode ter um roteiro próprio para aquisição de módulos de software do mercado de fornecedores.

2.2 Qualidade de Produto e de Processo de Software

Produzir um produto de software com qualidade é essencialmente uma atividade de gerenciamento com ações específicas para que os produtos resultantes atendam a padrões aceitáveis.

A forma mais comum de se implementar mecanismos de garantia da qualidade é através da formação de equipes independentes de qualquer grupo de desenvolvimento e diretamente ligada aos níveis administrativos acima da gerência de desenvolvimento.

2.2.1 Qualidade de produto e de processo

Lidar com a qualidade de software exige que se estabeleça mecanismos de garantia da qualidade que assegurem a eficácia nos processos de desenvolvimento de software de modo a obter produtos de software confiáveis. Esta tem sido a maneira com que as empresas competitivas têm se posicionado no mercado onde a disputas por segmentos de negócios são intensas e sistemas de computacionais são elementos estratégicos e que fazem a diferença. Neste contexto vale a definição estabelecida por BERSOFF (1984) apud SOMMERVILLE (1989):

> *"Garantia da qualidade consiste de procedimentos, técnicas e ferramentas aplicada por profissionais para garantir que um produto atende ou excede aos padrões pré-estabelecidos durante o ciclo de desenvolvimento de um produto; se não existir as especificações padrões pré-estabelecidas, a garantia da qualidade assegura que o produto*

atende ou excede um nível de excelência mínima industrial ou comercialmente aceita."

Normalmente interpreta-se de forma errada que garantir a qualidade é acionar procedimentos de qualidade no final do processo de desenvolvimento. Na verdade a qualidade de desenvolvimento de software inicia-se desde as fases anteriores de um processo, já na fase de planejamento é possível se medir a qualidade ao nível de estratégias de obtenção de software estabelecidas.

Outra experiência trazida da indústria de manufatura é que a qualidade de produto de software é decorrente da qualidade do processo. Assegurar a qualidade medindo diretamente os atributos de qualidade no produto pode ser difícil e contra-producente. Normalmente a estratégia adotada pela garantia de qualidade é prover um processo muito bem planejado e gerenciado.

Em resumo, o objetivo básico de grupo da garantia da qualidade é estabelecer padrões de processos e de produtos adequados para a corporação (SOMMERVILLE, 1989) onde:

◊ Estabelecer padrões para produtos corresponde a estabelecer atributos que todos os produtos de software devem apresentar;

◊ Estabelecer padrões de processos corresponde como o processo de software deve ser conduzido para produzir produtos de software.

2.2.2 Sistema da garantia da qualidade

Existem muitas definições sobre a qualidade de software. A definição apresentada por PRESSMAN (1988) é bastante adequada:

> *Conformidade com requisitos explícitos de funcionalidade e desempenho, com padrões de desenvolvimento explicitamente documentados e com características implícitas, é o que se espera de todos os software profissionalmente desenvolvidos.*

Desenvolvendo um pouco mais, PRESSAMN (1988) extende para uma definição bastante interessante para o escopo da tese, ou seja:

- ◊ Requisitos de software é a base sobre a qual a qualidade é medida. Falta de conformidade com os requisitos é falta de qualidade;
- ◊ Padrões específicos definem um conjunto de critérios de desenvolvimento, que orientam a maneira como um software deve ser projetado. Se os critérios não são seguidos então é falta de qualidade que certamente refletirá nos resultados finais;
- ◊ Existe um conjunto de requisitos implícitos que não são mencionados. Se um software apresenta conformidade com os requisitos explícitos e falha em atender aos requisitos implícitos, então a qualidade do software é suspeita.

Fatores de Qualidade de Software

A qualidade de um software pode ser medida através de uma série de fatores que variam de acordo com as aplicações e com os clientes. São chamados de fatores de qualidade de software. Uma boa classificação foi a montada por McCALL et. al. (1977), cuja categorização é mostrada na Fig. 2.7. Os fatores de qualidade são classificados segundo os aspectos de alteração, migração de ambientes e caracterização operacional.

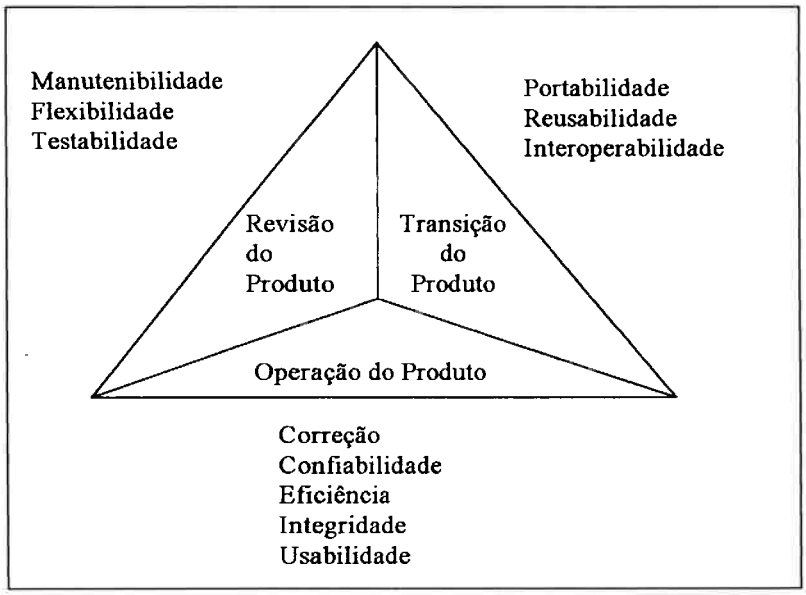

Fig.2.7 - Categorização dos fatores de qualidade (McCALL (1977)).

Onde cada fator pode ser entendido por:

◊ Revisão de Produto, composto pelos fatores:

- Manutenibilidade - Possibilidade de corrigir defeitos;
- Flexibilidade - Possibilidade de alterar o software;
- Testabilidade - Possibilidade de testar o software.

◊ Operação do Produto, composto dos fatores:

- Correção - A funcionalidade do produto atende às expectativas do cliente;
- Confiabilidade - Ele realiza as funções esperadas sempre, todo o tempo;
- Eficiência - Ele executa da melhor maneira possível no *hardware* onde está instalado;
- Integridade - Indica a extensão que uma pessoa inautorizada pode acessar o sistema;
- Usabilidade - Indica o esforço para aprender, operar, preparar entrada e entender e interpretar as saídas produzidas do produto;

◊ Transição do Produto, apresenta os seguinte fatores:

- Portabilidade - Indica o esforço para transferir um programa de um hardware (ou ambiente de software) para outro;
- Resusabilidade - Indica a extensão do programa, ou parte do programa que pode ser reutilizada em outra aplicação;
- Interoperabilidade - Indica o esforço necessário para acoplar um sistema a outro.

Para completar o mecanismo de medida, os fatores são associados através de fórmulas com métricas que podem ser diretas (como número de erros, número de linhas, tempo, etc.) ou indiretas (pontos de função, usabilidade, complexidade. As principais métricas são:

◊ Auditabilidade - Mede a facilidade de verificação de conformidade com padrões;

◊ Precisão - Indica a precisão de cálculo e controle;

◊ Conformidade de comunicação - Grau de conformidade aos padrões de comunicação como protocolos, taxa de transmissão e outros;

◊ Completeza - Mede o grau de completeza funcional entre o esperado e o disponível;

◊ Concisão - Mede a compactação do programa em linhas de código;

◊ Consistência - Mede o grau de uniformidade de utilização das técnicas e documentação de projeto para todo o sistema;

◊ Conformidade de dados - Mede o grau de conformidade em relação às estruturas de dados e tipos por todo o programa;

◊ Tolerância a erros - Mede o impacto quando acontece um erro do programa;

◊ Eficiência de execução - Desempenho de um programa em tempo de execução;

◊ Expansibilidade - Mede o grau que a arquitetura, os dados e procedimentos podem ser extendidos a nível de projeto;

◊ Generalização - Mede a abrangência de potenciais aplicações dos componentes de programas;

◊ Independência de hardware - Mede o grau em que um software é desacoplado do hardware em que ele opera;

◊ Instrumentação - Mede o grau com um programa monitora a sua própria operação e identificas os erros que ocorrem;

◊ Modularidade - Mede o grau de independência dos componentes do programa;

◊ Operabilidade - Mede a facilidade de operação de um programa;

◊ Segurança - Mede a extensão dos mecanismos que controlam ou protegem programas e dados;

◊ Auto-Documentação - O grau de documentação dos programas fontes para se tornar mais legível;

◊ Simplicidade - Mede o grau com um programa pode ser entendido sem dificuldades;

◊ Independência de sistema de software - Mede o grau de independência em relação a linguagem (comandos não padronizados), sistemas operacionais e outras restrições de ambiente de software;

◊ Rastreabilidade - Mede a facilidade de rastrear as representações de projetos ou trechos de implementação aos requisitos do sistema;

◊ Treinabilidade - Mede o grau de assistência dados pelo próprio programa para os usuários iniciantes;

Os fatores de qualidade e de métricas são relacionados, onde fatores são calculados como composição de métricas, conforme mostra a Tab.2.2. Vale lembrar que o grau de associação depende do contexto da aplicação. Além disso, conforme descrito na Norma IEEE Standard for a Software Quality Metrics

Methodology, (IEEE1994), *"... o uso de métricas de software não elimina a necessidade de julgamento humano na avaliação de software ..."*.

Tab.2.2 - Associação de fatores com métricas, segundo McCall (1977).

CATEGORIZAÇÃO	FATORES DE QUALIDADE	MÉTRICAS ASSOCIADAS
OPERAÇÃO DE PRODUTO	CORREÇÃO	Completeza Consitencia Rastreabilidade
	CONFIABILIDADE	Precisão Complexidade Consistência Tolerancia a erros Modularidade Siplicidade
	EFICIÊNCIA	Concisão Eficiência de execução Operabilidade
	INTEGRIDADE	Auditabilidade Instrumentação Segurança
	USABILIDADE	Operabilidade Treinamento
REVISÃO DE PRODUTO	MANUTENIBILIDADE	Concisão Consistência Instrumentação Modularidade Auto-documentação Independência de sistemas
	FLEXIBILIDADE	Complexidade Concisão Consistência Expandibilidade Generalidade Modularidade Auto-documentação Simplicidade
TRANSIÇÃO DE PRODUTO	Portabilidade	Generalidade Independência de Hardware Modularidade Auto-documentação Independência de sistemas
	Reusabilidade	Generalidade Independência de Hardware Modularidade Auto-documentação Independência de sistemas
	Interoperabilidade	Conformidade de comunicação Conformidade de Dados Generalidade Modularidade

Garantia da Qualidade

Funções de controle e da garantia da qualidade foi implantada formalmente na Bell Labs, em 1916, dando início a um rápido espalhamento do modelo em toda a indústria da manufatura. Atualmente, as empresas incorporam nos seus processos de produção mecanismos de garantia da qualidade. Na área de software, uma das primeiras iniciativas em controle de qualidade surgiu com a preparação de padrões de qualidade dentro de um contrato de desenvolvimento na área militar, isto aconteceu na década de 1970 (PRESSMAN (1988)).

A importância de um sistema com garantia de qualidade pode ser representado esquematicamente. Os clientes, os analistas de negócios, apoiados nos mecanismos de qualidade têm a possibilidade de obter níveis mínimos, padronizados, de software de forma mais corporativo, por processo. Internamente, o software envolve equipes especializadas em manutenção, desenvolvimento, testes, gerenciamento. Estes também, de forma análoga, podem dispor de mecanismos de qualidade inserida ao nível de processo. Veja a Fig.2.8.

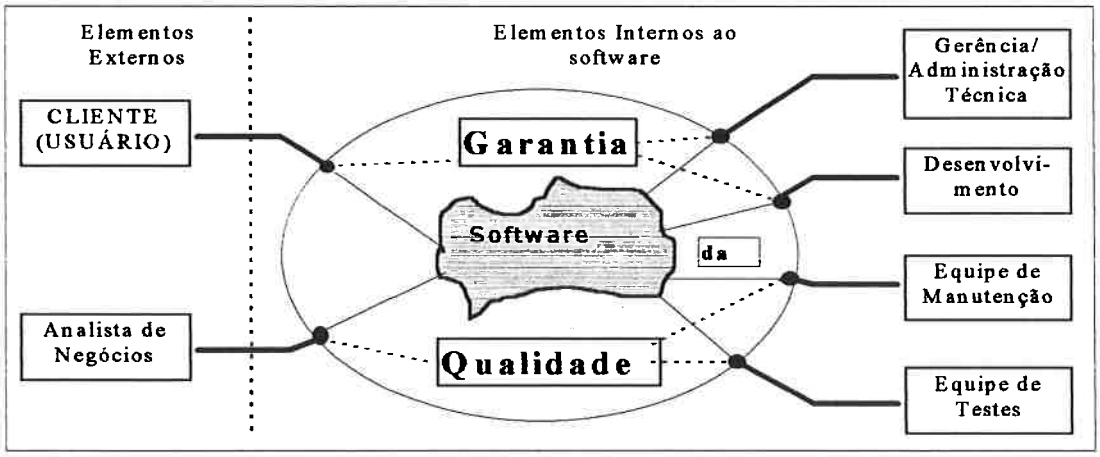

Fig.2.8 - Sistema da Garantia da Qualidade incorpora mecanismos de apoio em qualidade para os elementos externos e internos ao software.

Normalmente o grupo de qualidade representa o cliente e o próprio vendedor (ou analista de negócios) dentro do ambiente de software, muitas das atividades relacionadas com a qualidade é definida do ponto de vista do usuário (cliente).

Atividades da Garantia da Qualidade

Segundo PRESSMAN (1988), um sistema da garantia da qualidade compreende uma variedade de tarefas que podem ser agrupadas em um conjunto de sete atividades: Aplicação de métodos; Executar revisões formais; Testes de software; Garantir o uso de padrões; Controle de alterações; Realizar medições e Manter o arquivamento e a disseminação de informações de qualidade, onde:

- ◊ Aplicação de Métodos - Garantir que especificações e projetos de alta qualidade reflitam a busca de qualidade de um sistema desde o início. Daí a importância da implantação de métodos e ferramentas para prover infra-estruturas ao desenvolvedores;
- ◊ Revisões técnicas formais - Reuniões com as equipes técnicas, para revisar os produtos elaborados, como especificações, projetos, protótipos. O propósito deste tipo de atividade é buscar situações de erro antes da implementação física (codificação). Revisões de especificações e de projeto pode ser comparado aos testes realizados sobre programas executáveis;
- ◊ Teste de Software - Envolve a realização prévia de planos de teste combinados com a aplicação em vários níveis como testes unitários, de módulo, de integração e de validação. Os testes têm o objetivo de detectar problemas do software para correção antes de liberação para uso operacional, em produção;

◊ Procedimentos e padrões - Os procedimentos e padrões são a implementação ajustada e adequada de uma metodologia dentro de uma corporação. Estando eles formalizados, o sistema de garantia da qualidade verifica a conformidade dos trabalhos produzidos através de revisões técnicas formais, na forma de algum tipo de auditoria interna;

◊ Controle de Alterações - Relaciona-se ao controle de versões de software, considerando que uma das maiores fontes de erros são as mudanças realizadas no software na forma de manutenção. São mecanismos de qualidade para formalização de requisições de alteração, avaliação da natureza de mudanças e controle dos impactos da mudança. Esse controle deve acontecer em todo o ciclo de vida, desde o desenvolvimento até a fase de manutenção;

◊ Medições - Corresponde a criação de medidas, atributos das métricas de capaz de tornar a visualização da qualidade através de medições de elementos técnicos e gerenciais;

◊ Arquivamento e Disseminação de informações - São procedimentos para coletar e disseminar informações de qualidade. O resultado das revisões, as auditorias, controle de alterações, testes e outras atividades da qualidade devem fazer parte do arquivamento do projeto e da base de conhecimento corporativo, na forma de informações históricas, para divulgação por toda a equipe de desenvolvimento.

Revisão de Software

As revisões técnicas formais podem contribuir para minimizar erros funcionais, de lógica ou de implementação. Permitem verificar se o software atende aos

requisitos técnicos ainda no seu estágio inicial de desenvolvimento. E um item fundamental é que permitem um gerenciamento mais efetivo do projeto.

O foco de uma revisão técnica formal é o produto e suas atividades são registradas em documentos apropriados. Para entender o escopo destas revisões, listamos os produtos que devem ser submetidos durante, por exemplo, o desenvolvimento de um sistema: Engenharia de Sistemas; Planejamento de software; Análise de requisitos; Projeto de software; Codificação; Testes de software; e Manutenção;

2.3 Processos de Produção na Indústria

Os sistemas de software são hoje elementos importantes e muitas vezes essenciais de infra-estrutura aos processos de negócios. São elementos diferenciadores da competitividade do mercado, especialmente, na área objeto desta tese, a indústria financeira. Essa área, há muito tempo carece de mecanismos de qualidade eficaz e adequado aos seus processos críticos (ARAKAKI (1996a)). Os sistemas de software, além da pré-condição implícita e crítica de confiabilidade como o seu principal fator de qualidade, eles devem ser produzidos em tempos cada vez menores, em média em *time-boxes* de 3 a 4 meses. O fator tempo tem sido o ponto principal de definição e de desafio no aprimoramento dos processos de desenvolvimento desta indústria. Vale lembrar que esse desafio não é exclusivo da área, pois a indústria mundial de produtos de software de "prateleira" também está com o fator tempo nas suas estratégias de consolidação de mercado.

2.3.1 Técnicas de desenvolvimento rápido

Lidar com o fator tempo está exigindo algumas adaptações de estratégias de implantação dos processos de obtenção software. Essa estratégias são objetos de estudos e a principal é aquela denominada na literatura como "*Rapid Development*" (Desenvolvimento Rápido) MCCONNELL (1996), cujo o foco é a correta implantação de processos com mecanismos de qualidade capazes de minimizar erros de especificação, retrabalhos de projetos e ineficácia de codificações.

O conceito de desenvolvimento rápido poderia ser interpretado por alguns como: implementar um software em 36 horas; ou usar um produto de 4ª Geração usando rápida prototipação; ou simplesmente cortar atividades de um planejamento criterioso para simplesmente produzir um software em tempo menor. Não é nada disso. Não existe mágica. Sistema com planejamento simplificado pode ter péssimas implementações, sistema com baixa cobertura de testes apresentará baixa qualidade funcional, sistema com estrutura de projeto deficitária apresentará custos na manutenção, com implicações negativas na confiabilidade.

O termo desenvolvimento rápido está relacionado com enfatizar a velocidade de desenvolvimento. Significa buscar meios para fazer mais rápido que a atual forma de implementar. Segundo MCCONNELL (1996) existem muitas práticas de desenvolvimento de software. Aquelas que são eficientes e as que não são eficientes. Dentre elas, que são muitas, existem um conjunto específico que pode

ser definido como os que apresentam um foco para o planejamento de atividades no tempo (Schedule Oriented), conforme mostra a Fig.2.9.

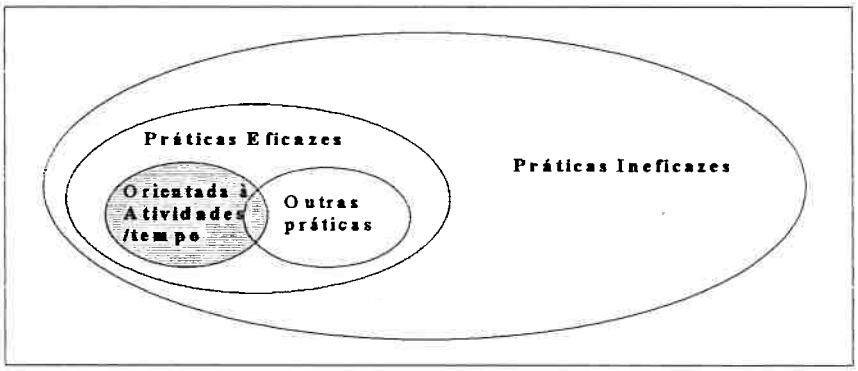

Fig.2.9 - A velocidade de desenvolvimento depende do uso de uma prática adequada.

As práticas de desenvolvimento que estão relacionadas com a otimização atividades no tempo, podem ser categorizadas em:

◊ Práticas de otimizam a velocidade de desenvolvimento, permitindo liberação de software mais rápido;

◊ Práticas que reduzem riscos de atividade, o que permite que se evite *overrun*; e

◊ Práticas que tornam visíveis os progressos para os clientes/contratantes.

A escolha ou a definição deve ser realizada de acordo com o contexto. Pode haver uma situação, por exemplo, em que a equipe técnica entende que o software está sendo desenvolvido na velocidade correta, porém não é o que pensa o contratante que acha que está com velocidade aquém do desejado. Nessa situação, uma maneira de corrigir esta divergência é usar a prática que privilegia a

visibilidade dos produtos de desenvolvimento permitindo assim que o contratante possa mudar a sua percepção.

As dimensões do Desenvolvimento Rápido

Para atuar na velocidade de desenvolvimento, significa fazer certo, sem muitos erros. A Fig.2.10 procura ilustrar o fato de ganhar tempo percorrendo a trajetória de desenvolvimento de forma correta. Um ambiente organizacional para obter sucesso no desenvolvimento deve incorporar mais do que uma prática conforme orienta os trabalhos de JONES (1991) e BOEHM (1987), de modo adequado e coerente ao seu contexto. A velocidade de desenvolvimento pode ser atuada em quatro dimensões: Pessoas, Processos, Tecnologia e Produto. Cada uma das dimensões é detalhada a seguir:

◊ Pessoas - Dados estatísticos mostram que pessoas apresentam diferenças de produtividade da ordem de 10 para 1 e equipes apresentam diferenças de 4 até 5 para 1 (BOEHM (1981)) demonstrando que equipes influem muito intensamente na produtividade. Nessa situações, ações devem se organizadas para trabalhar a motivação, trabalho em grupo, seleção de equipe e treinamento.

◊ Processo - Está diretamente associado com gerenciamento e métodos. Os processos devem apresentar mecanismos que facilitem os trabalhos e estabeleçam os níveis mínimos de qualidade e padronização. Trabalhar em processos é um dos principais fatores de ganhos da atual indústria conforme demonstram esses indicadores. Empresas como a Motorola, Xerox, NASA e Rayethon focaram de forma explícita no processo de desenvolvimento e

obtiveram ganhos de 50% no tempo de desenvolvimento e redução de custos em defeitos em 300% a 1000% dentro de um período de 7 anos (PUTNAM (1992)).

Atuar em processo é importante para: Evitar retrabalhos melhorando a qualidade das especificações e de projetos fazendo com as atividades prossigam sempre em frente (sem voltas - retrabalhos). A empresa Rayethon obteve redução de retrabalhos de 41% a menos de 10% e com a produtividade geral aumentada em 300 % MCCONNELL (1996)); Garantir a qualidade é um fator básico para implementar o desenvolvimento rápido considerando que, conforme foi mostrado nos itens anteriores deste capítulo, porque lida com a qualidade final do software, produto do processo de desenvolvimento e também porque busca a identificação de erros e inconsistências do sistema desde o seu estágio inicial do desenvolvimento; Princípios de Desenvolvimento, independentemente das técnicas utilizadas, devem ser seguidas da engenharia de software para evitar que um desenvolvimento se mantenha sob controle gerencial; Gerenciamento de Riscos também é importante para evitar percalços e surpresas durante o desenvolvimento, através de um gerenciamento refinado em torno de atividades que ofereçam riscos potenciais dentro de um desenvolvimento; Outros fatores a serem atuados nos processos são gerenciamento de recursos para um uso eficaz, Planejamento de todo o ciclo de vida do software buscando identificar de acordo com o contexto os roteiros mais adequados de desenvolvimento, e finalmente a orientação aos clientes onde as expectativas deles devem ser gerenciados. Se a sua expectativa baseia-se numa definição concreta, estável,

em termos de recursos esperados para o sistema, a estratégia de desenvolvimento é uma. Caso o cliente não tenha muito claro todos os recursos funcionais do sistema, então vale outra estratégia como por exemplo a de desenvolvimento incremental. Neste exemplos citados de inúmeros possíveis, a participação e o nível de conscientização do cliente em relação ao processo de desenvolvimento que será executado para o sistema é diferente e ele deve ser informado para adequar as expectativas ao nível técnico bem como gerencial.

◊ Produtos - Tamanho de produto e características de produto pode contribuir sobremaneira na redução de atividades. Para isso a regra muito válida é o 80/20, que diz que oitenta por cento dos recursos do sistema pode ser desenvolvido em vinte por cento do tempo, e desenvolve os vinte por cento restante mais tarde. Com relação as características de produto, fatores de qualidade como performance, robustez, uso de memória apresentarem níveis muito ambiciosos acima do efetivamente esperado, tomam muito tempo de desenvolvimento. Adequar essas características incorporando tais fatores de forma incremental no planejamento pode possibilitar muito ganho no cronograma de desenvolvimento;

◊ Tecnologia - A escolha da tecnologia pode influir fortemente na velocidade. Por exemplo, usar componentes de software pode produzir altos ganhos em redução de tempos.

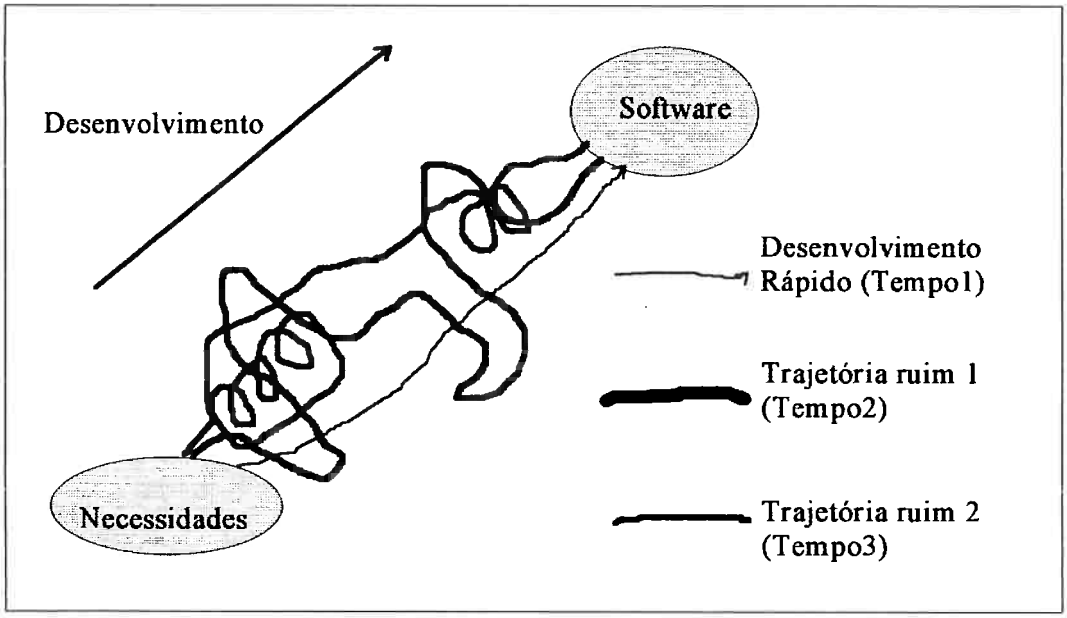

Fig.2.10 - Desenvolvimento rápido corresponde obter uma trajetória como o ilustrado como Tempo 1. (Em termos de desenvolvimento, a trajetória mais rápida, mais eficaz é a de tempo1, pois Tempo1 < Tempo2 < Tempo3, onde os tempos citados correspondem às durações de desenvolvimento do software).

Os erros comuns "modernos" em desenvolvimento de software

Vale citar alguns erros mais comuns cometidos por equipes, na ânsia de desenvolver trabalhos de forma muito rapidamente. A Tab.2.3 mostra um quadro resumido dos principais erros cometidos e para alguns casos apresenta-se dados estatísticos. Os erros apontados na Tab.2.3 devem ser expandidos com base nos *Post-mortens* de projetos já realizados para adequar ao contexto da empresa.

ERROS - PESSOAS	ERROS - PROCESSO	ERROS - PRODUTO	ERROS - TECNOLOGIA
1. Baixa motivação	14. Cronogramas irrealisticamente otimista	28.Requisitos de "ouro-platina".	33. Síndrome de "*Silver-Bullet*" (milagres)
2. Pessoas fracas tecnicamente	15. Gerenciamento de risco insuficiente	29. Requisitos "flutuantes".	34. economia superestimadas no uso de métodos e ferramentas.
3. Problemas incontroláveis entre empregados	16. Falhas do contratante	30. Desenvolvedor de "ouro-platina".	35. Troca de ferramentas no meio do projeto.
4. Heróis	17. Planejamento insuficiente	31. Negociação unilateral de planejamento	36. Falta de controle automatizado de programas (fonte e código)
5. Adição de pessoas no meio do projeto	18. Abandono do planejamento sob pressão	32. Desenvolvimento orientado a pesquisas.	
6. Ambiente inadequado de desenvolvimento	19. Tempo gasto com relacionamento "fuzzy"de negociação de projeto		
7. Conflito enter os desenvolvedores e clientes	20. Corte de atividades de desenvolvimento		
8. Expectativas irrealistas	21. Projeto inadequado.		
9. Falta de lideranças efetivas no projeto	22. Corte de atividades de controle de qualidade		
10. Falta de compromentimento	23. Controle de gerenciamento insuficientes		
11. Não envolvimento de usuários	24. Convergência de produto prematuramente		
12. Excesso de política	25. Omissão de atividades usadas nas esimativas do planejamento		
13. Postura displicente (diferente de postura otimista)	26. Falta de replanejamento adequado		
	27. Codificação ruim ("misesrável")		

Tab.2.3 - Resumo dos principais erros de projetos, na tentativa de aumentar a velocidade de desenvolvimento (MCCONNELL (1996).

Com relação à Tab.2.3, valem alguns comentários. Um refere-se ao erro 20 que fala sobre o corte de algumas atividades com o objetivo de aumentar a velocidade. O mais comum é aquele em que a equipe dá um salto para ir direto para a codificação. Pois bem, números estatísticos mostram um desvio para pior de 10 a 100 vezes mais em custos pela tomada de tal decisão (FAGAN (1976) e BOEHM (1989)). Outro comentário refere-se ao erro 22 da Tab.2.3. que fala sobre cortar atividades de controle da qualidade, por exemplo cortar revisões técnicas, planejamento de testes, eliminação de revisão de projetos e de codificação. Pois bem, nesse caso, JONES (1994) mostra que 1 dia eliminado de controle de qualidade produz em média de 30 dias de atividades adicionais.

2.3.2 As principais práticas da indústria

Algumas práticas mais conhecidas e utilizadas na indústria valem ser referenciadas uma vez que referem-se a práticas bem sucedidas de obtenção de software por desenvolvimento rápido, onde o objetivo é desenvolver com maior velocidade, explorando a eficácia das atividades dos processos.

A seguir apresenta-se uma breve caracterização de algumas práticas mais usadas na indústria (ARAKAKI (1996a), McCONNELL (1996)):

◊ Quadro de Mudanças - Foco no controle de mudanças de um produto de software. Pelo fato de lidar constantemente com todos os produtos relacionados com o software, mapeando as mudanças solicitadas, apresenta a vantagem de explicitar, tornando visíveis as flutuações de requisitos, de modo

a reduzir o número de mudanças não controladas. Permite combinação de outras práticas;

◊ Construção diária e teste fumaça - O software é construído completamente a cada dia e submetido a uma série de testes para validação. Baseia-se no fato de construção incremental, com resultados diários, incluindo testes básicos realizados no mesmo dia;

◊ Projetado para mudanças - É orientado a mudanças. Incluem atividades de identificação de pontos passíveis de mudanças, relacionados com principais problemas, lógica de negócios, portabilidade de hardware etc.;

◊ Entrega evolucionária - Essa prática fornece meios para liberar partes do software mais rapidamente, mas não garante o tempo final de entrega muito menor. A sua grande vantagem é possibilitar uma certa flexibilidade para mudanças no meio do projeto de modo a atender às mudanças de requisitos pelo cliente. ;

◊ Prototipação evolucionária - Essa prática faz baseia-se na montagem de protótipos que são submetidos aos clientes e por um processo interativo de análise e revisão do protótipo este vai convergindo de acordo com as expectativas do cliente. Todos os demais módulos previstos para o sistema vão sendo agregados em torno dos protótipos e cuidados especiais devem ser tomados com relação às expectativas do cliente, gerenciamento de riscos de projetos e estruturação da implementação, uma vez que a implementação não é descartada;

◊ Orientado por Metas - As atividades sã realizadas com base em metas estabelecidas pela gerência e pelo próprio cliente. A maior dificuldade é o estabelecimento de metas adequadas (resultado x Prazo);

◊ Inspeção - Essa prática baseia-se num tipo de revisão técnica formal, na qual os participantes se concentram em descobrir erros em estágios iniciais de projetos, em atividades prévias às reuniões formais de revisão. O ganho em tempo dessa prática é buscar erros em estágios iniciais do projeto;

◊ Joint Application Development (JAD) - Essa prática baseia-se na construção de interfaces de usuários e no refinamento de requisitos, através de um envolvimento intensivo de usuários, executivos, desenvolvedores em reuniões específicas. Sua economia está em refinar os requisitos muito rapidamente. Depende bastante do líder que conduz as reuniões e as peças chaves como usuários, executivos e desenvolvedores através de uma interação com sinergia para maior eficácia. Essa prática deve ser combinado com aquelas que apresentam características de evolução incremental como prototipação e entregas por partes;

◊ Seleção do Modelo de Ciclo de Vida - Cada produto de software pode apresentar um modelo de ciclo de vida e uma escolha inadequada do modelo pode ser causa de insucesso, pois pode resultar em faltas de tarefas e ordem inadequadas de tarefas. Ao contrário, uma escolha correta organiza os trabalhos de forma eficaz;

◊ Medidas - Produz benefícios relacionados com a qualidade. Ela procura apresentar meios para combater estimativas falhas, cronogramas falhas e verificação de progressos imprecisos. Organização que apresenta

procedimentos efetivos de medidas tem muitos ganhos, especialmente na competição. Usar essa prática é ponto de diferenciação na indústria para melhoria da qualidade, pelo fato de transformar em indicadores numéricos várias características de desenvolvimento. Detalhes nesta tese estão nos itens anteriores deste Cap.2 e no Cap.3;

◊ Mini-Milestones - Caracteriza-se por definir pontos de liberação de pequenos resultados no cronograma. Possibilita melhoria de quatro fatores: Visibilidade e verificação do progresso; Controle fino do cronograma; Redução de riscos e Motivação da equipe;

◊ Outsourcing - É uma prática de encomenda de implementação para empresas externas à organização. Usa especializações muitas vezes não disponíveis internamente, porém oferece um risco grande de transferência de conhecimento de negócio, muitas vezes críticos e sigilosos;

◊ Princípio da Negociação - Baseia-se no processo de interação do tipo "win-win", onde o cliente e o fornecedor negociam as tarefas de forma que todos ganhem, nas várias fases de um projetos, desde a definição dos requisitos. O foco das negociações baseiam-se nos interesses mútuos, não em posições de cada um;

◊ Ambiente de Produtividade - Baseia-se na disponibilização sempre crescente de ambientes confortáveis para o desenvolvimento, considerando que esse tipo de tarefa de desenvolvimento exige muita concentração e o ambiente deve ser livre de interrupções e barulhos. Aumentar a taxa de retenção da mão de obre é fator de produtividade;

◊ Linguagens para Desenvolvimento Rápido - Essa prática produz economia através da utilização de ferramentas capazes de produzir aplicações. São as linguagens de 4ª geração, como as linguagens de programação visual como as especializadas para sistemas operacionais como o MS-Windows[3];

◊ Rascunho de Requisitos - Concentra-se em minimizar esforços fazendo análises cuidadosas no conjunto de requisitos, buscando eliminá-los , considerando que o tamanho do sistema é dependente do conjunto de recursos funcionais previstos nos requisitos;

◊ Reuso - Essa estratégia de longo prazo. A organização constrói repositórios de componentes úteis, mais usados, permitindo que novos programas sejam montados através componentes pré-existentes. O seu uso efetivo faz com que seja uma das práticas das mais eficazes. O tema central dessa da tese baseia-se no uso dessa prática;

◊ Modelo Espiral - Apresenta um foco sobre o gerenciamento de riscos. O seu ganho não está em diminuição do prazo global, mas na redução de riscos por comprometer pequenos ciclos de desenvolvimentos orientados pela análise constantes dos riscos. Exige um gerenciamento bastante atento e consciencioso;

◊ Liberação por estágios - O software é desenvolvido em estágios, priorizando grupos de funções mais importantes, mas deforma muito semelhante ao espiral, não diminui o prazo global, porém lida com os riscos. Baseia-se na estruturação do software em partições, que podem ser mapeados diretamente nos processos de negócios, ou seja, as partições pode entrar em produção;

[3] MS-Windows - É marca registrada de sistema operacional da Microsoft Corporation.

◊ Gerenciamento por Teoria W - Baseia-se no gerenciamento onde todos os participantes devem ganhar (*win*). O gerente tem como base uma negociação constante entre as partes, onde todos devem ganhar. Isto define diretrizes para eliminar conflitos, e o ganho produzido por esta prática está em aumentar a eficiência dos relacionamentos, da visibilidade dos progressos e da redução de riscos;

◊ Desenvolvimento *Time-Box* - É uma prática baseada em construção de sistemas por fases (tempo), onde a urgência é tratada em focos adequados nos requisitos mais importantes "encaixados" em módulos de tempo pré-estabelecidos que são as fases. Depende muito do cliente final, uma vez que ele deve contribuir nas reduções de requisitos nas montagens das atividades do cronograma;

◊ Protótipo de Interface de Usuário - A interface com o usuário é desenvolvido rapidamente para ser utilizado na convergência da especificação de requisitos;

Tab.2.4 - Resumo das principais práticas de desenvolvimento rápido.

Nome da Prática	Potencial de Redução do Cronograma	Melhoria na visibilidade do progresso	Efeito nos riscos do cronograma	Chance de Sucesso na 1ª vez	Chance de sucesso de longo tempo
1. Quadro de mudanças	Médio	Médio	Diminui	Muito bom	Excelente
2. Construção diária e teste fumaça	Bom	Bom	Diminui	Muito bom	Excelente
3. Projeto para Mudanças	Médio	Nenhum	Diminui	Bom	Excelente
4. Entrega evolucionária	Bom	Excelente	Diminui	Muito bom	Excelente
5. Prototipação evolucionária	Excelente	Excelente	Aumenta	Muito bom	Excelente
6a. Orientado por metas (diminuir cronograma)	Muito bom	Nenhum	Aumenta	Bom	Muito Bom
6b. Orientado por metas (Diminuir riscos)	Nenhum	Bom	Diminui	Bom	Muito bom
6c. Orientado por metas (Máxima visibilidade)	Nenhum	Excelente	Diminui	Bom	Muito Bom
7. Inspeção	Muito bom	Médio	Diminui	Bom	Excelente
8. JAD	Bom	Médio	Diminui	Bom	Excelente
9. Seleção do Modelo do ciclo de vida	Médio	Médio	Diminui	Muito bom	Excelente
10. Medidas	Muito bom	Bom	Diminui	Bom	Excelente
11. Mini-milestones	Médio	Muito bom	Diminui	Bom	Excelente
12. Outsourcing	Excelente	Nenhum	Aumenta	Bom	Muito bom
13. Princípio da Negociação	Nenhum	Muito bom	Diminui	Muito bom	Exececelente
14. Ambiente de Produtividade	Bom	Nenhum	Sem efeito	Bom	Muito bom
15. Linguagem de Desenvolvimento Rápido	Bom	Nenhum	Aumenta	Bom	Muito bom
16. Rascunho de Requisitos	Muito bom	Nenhum	Diminui	Muito bom	Excelente
17. Reuso	Excelente	Nenhum	Diminui	Pobre	Muito bom
18. Modelo Espiral	Médio	Muito bom	Diminui	Bom	Excelente
19. Liberação por Estágios	Nenhum	Bom	Diminui	Muito bom	Excelente
20. Gerenciamento por Teoria W	Nenhum	Muito bom	Diminui	Excelente	Excelente
21. Desenvolvimento Time-box	Excelente	Nenhum	Diminui	Bom	Excelente
22. Protótipo de Interface de Usuário	Bom	Médio	Diminui	Excelente	Excelente

2.4 Capacitação e Maturidade em Produção de Software

A competitividade dentro de um contexto onde a globalização expõe as empresas numa constante disputa de segmento de mercado ao nível mundial exige da sua infraestrutura de informática eficácia e confiabilidade no apoio aos processos de negócios, flexibilidade para acompanhar a dinâmica de novos negócios e um uso racional de recursos do ponto de vista administrativo e financeiro. Esses pontos estão hoje canalizado nas empresas competitivas dentro de uma linha ação denominada Sistemas de Garantia da Qualidade, onde o foco é melhoria da qualidade.

Vale citar uma definição a respeito do sistema da qualidade. Neste caso, mais especializado ainda, em relação ao sistema de qualidade total (SQT): "SQT é uma filosofia e um conjunto de princípios que representam uma base para uma contínua melhoria da organização. SQT é a aplicação quantitativa de métodos e recursos humanos para melhoria do material e dos serviços fornecidos para uma organização, para a melhoria de todos os processos dentro da organização, e o grau de satisfação das necessidades dos clientes para o agora a para o futuro. SQT integra técnicas, esforços de melhoria e ferramentas técnicas sob uma sistemática linha focada na continuada melhoria". Esta definição é utilizada pelo Departamento de Defesa dos EUA (TINGEY (1997)). Os itens que se seguem apresentam algumas das referências técnicas mais aceitas internacionalmente que são: Malcom Baldrige Nacional Quality Award (MB); International Organization for Standardization 9000 (ISO 9000); e Software Engineering Institute (SEI) Capability Maturity Model (CMM) for Software.

2.4.1 A necessidade de avaliação do sistema de qualidade

Considerando que um sistema da qualidade busca (e necessita) uma melhoria contínua, ele deve ser sistematicamente avaliado através algum mecanismo de monitoração. Esta garantia é muito importante para aqueles que compram produtos ou serviços, ou seja os clientes. Para eles o que interessa é a certeza de que os produtos e os serviços que compram têm a dose correta de controle que garanta que os produtos e serviços ofertados sejam livres de falhas, ou melhor, no mínimo reduzidas a níveis admissíveis.

2.4.2 O ISO 9000

A série ISO 9000 (International Standardization Organization) é composta de dois tipos de padrões. O primeiro forma uma conjunto de diretrizes e não auditáveis. O segundo tipo que incluem a ISO 9001, ISO 9002 e ISO 9003 que são padrões que uma vez implantados nas organizações são auditáveis.

O primeiro tipo, não auditável, incluem:

- ISO 9000-1: 1994, Quality Management and Quality Assurance Standards - part 1: Guidelines for Selection and Use;
- ISO 9000-2: 1993 Quality Management and Quality Assurance Standards - part 2: Generic Guidelines for the Application of ISO 9001, 9002 e 9003;
- ISO 9000-3: 1991, Quality Management and Quality Assurance Standards - part 3: Guidelines for the Application of ISO 9001 to the Development, Supply, and Manutenance of Software;

◊ ISO 9000-4: 1993 Quality Management and Quality Assurance Standards - part 4: Guidelines for Dependability Programmer Management;

◊ ISO 9004-1: 1994 Quality Management and Quality System Elements - part 1: Guidelines;

◊ ISO 9004-2: 1991 Quality Management and Quality System Elements - part 2: Guidelines for Services;

◊ ISO 9004-3: 1993 Quality Management and Quality System Elements - part 3: Guidelines for Processed Materials;

◊ ISO 9004-4: 1993 Quality Management and Quality System Elements - part 4: Guidelines for Quality Improvement;

O segundo tipo de padrões são auditáveis e apresentam o seguinte conjunto:

◊ ISO 9001: 1994, Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation, and Servicing;

◊ ISO 9002: 1994, Quality Systems - Model for Quality Assurance in Production, Installation, and Servicing;

◊ ISO 9003: 1994, Quality Systems - Model for Quality Assurance in Final Inspection na Test;

Esses modelos apresentam escopo de aplicação que pode ser representado através de um esquema, conforme ilustra a Fig.2.11.

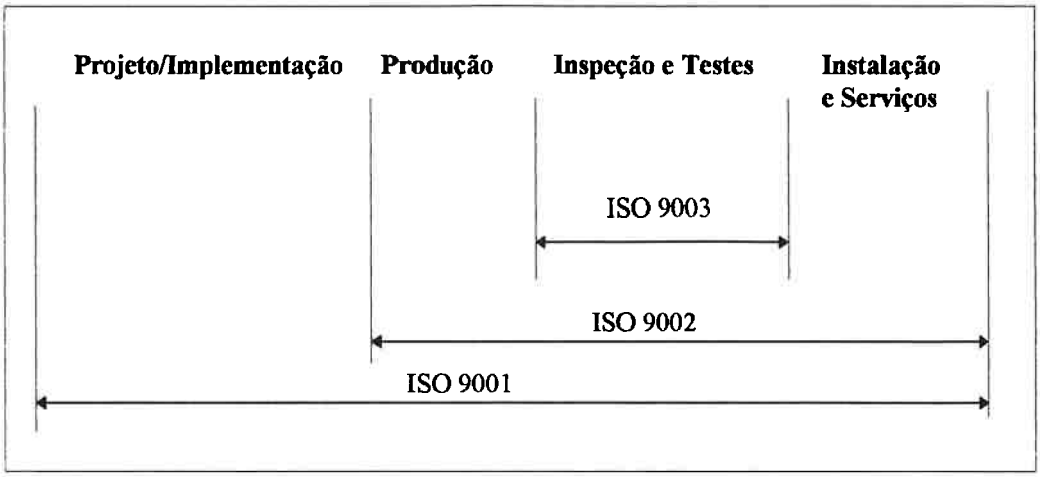

Fig.2.11 - Escopo das Normas ISO 9000.

A norma ISO 9001 é a que apresenta maior escopo em relação aos três padrões. A ISO 9002 não trata a parte de projeto e desenvolvimento com tanta extensão como a ISO 9001. Já a ISO 9003 procura especializar-se na detecção de falhas e testes.

A avaliação da ISO 9000 é conduzida como processos de registros, onde uma organização se torna registrada em um dos padrões (9001, 9002, ou 9003) e a mantém através da seguinte dinâmica: Faz pré-auditoria para detecção preliminar de problemas, efetua a auditoria oficial, obtém o registro que é revalidado continuadamente através de auditorias periódicas de seis em seis meses em média.

Os requisitos (são 20 itens) que compõem a ISO 9001 pode ser visto na Tab.2.5.

Tab.2.5 - Estrutura da Norma ISO 9001 - Requisitos.
(Baseado em TINGEY (1997), Normas ABNT NBR 9000).

REQUISITOS	SUB-ITENS / SUB-SEÇÕES
4.1 Gerenciamento de Rsponsabilidades	4.1.1 Política de qualidade 4.1.2 Organização Responsabilidade e autoridade Recursos Gerencias 4.1.3 Revisão do gerenciamento
4.2 Sistema da Qualidade	4.2.1 Gerais 4.2.2 Procedimentos do sistema da qualidade 4.2.3 Planejamento da qualidade.
4.3 Revisão de Contrato	4.3.1 Gerais 4.3.2 Revisão 4.3.3 Modificações no contrato 4.3.4 Registros
4.4 Controle de Projeto	4.4.1 Gerais 4.4.2 Planejamento de projeto e desenvolvimento 4.4.3 Interfaces organizacionais e técnicas 4.4.4 Entrada de projeto 4.4.5 Saída de projetos 4.4.6 Revisão de projeto 4.4.7 Verificação de projeto 4.4.8 Validação de projeto 4.4.9 Alterações de projetos
4.5 Controle de Dados e Documentos	4.5.1 Gerais 4.5.2 Aprovação e distribuição de documentos e dados 4.5.3 Alteração de documentos e dados
4.6 Aquisição/Compra	4.6.1 Gerais 4.6.2 Avaliação de sub-contratos 4.6.3 Dados de compra 4.6.4 Verificação do produto comprado
4.7 Controle de Produto e de Cliente	
4.8 Identificação e Rastreabilidade deProduto	
4.9 Controle de Processo	
4.10 Inspeção e Testes	4.10.1 Gerais 4.10.2 Inspeção de recebimento e testes 4.10.3 Inspeção e Testes dentro do processo 4.10.4 Inspeção final e testes 4.10.5 Registros de inspeção e testes
4.11 Controle de Inspeção, Medidas e Testes de Equipamentos	4.11.1 Gerais 4.11.2 Procedimentos de controle
4.12 Status de Testes e Inspecção	
4.13 Controle de Produtos não Conformes	4.13.1 Gerais 4.13.2 Revisão e disposição de produtos não conformes
4.14 Ações Corretivas e Preventivas	4.14.1 Gerais 4.14.2 Ações corretivas 4.14.3 Ações preventivas
4.15 Manipulação, Armazenamento, Empacotamento, Preservação e Entrega	4.15.1 Gerais 4.15.2 Manipulação 4.15.3 Armazenamento 4.15.4 Empacotamento 4.15.5 Preservação

	4.15.6 Entrega
4.16 Registros de Controle de Qualidade	
4.17 Auditorias Internas de Qualidade	
4.18 Treinamento	
4.19 Serviços	
4.20 Técnicas Estatísticas	4.20.1 Identificação de necessidades 4.20.2 Procedimentos

Tab.2.5 (Continuação) - Estrutura da Norma ISO 9001 - Requisitos.
(Baseado em TINGEY (1997), Normas ABNT NBR 9000).

2.4.3 O modelo CMM/SEI

O Software Engineering Institute (SEI) é uma instituição federal de pesquisas e desenvolvimento que opera na Universidade Carnegie Mellon University, sob contrato patrocinado pelo Departamento de Defesa dos EUA (DoD/USA). O SEI desenvolveu uma estrutura para avaliação de processos de software chamado CMM (Capability Maturity Model for Software). Ela descreve uma estrutura de processos de maturidade em cinco níveis.

Cada nível de maturidade (exceto o 1° nível) é dividido em 18 áreas chaves de processo (*Key Process*). Dentro de cada área chave, existem 52 metas e 316 práticas chaves. As metas (GO) resumem as práticas de uma área chave. As práticas chaves orientam por atividades, como atingir a metas.

Características comuns definiram a categorização das dezoito áreas chaves. São elas:

◊ Commitment to Perform (CO) - Descreve as ações da organização para garantir que o processo está estabelecido e deve durar;

◊ Ability to Perform (AB) - Descreve as pré-condições que devem existir para o projeto ou organização para implementar o processo de software de forma competente;

◊ Activities Performed (AC) - Descreve regras e procedimentos necessários para implementar a área chave do processo;

◊ Measurement and Analisys (ME) - Descreve a necessidade de medir e analisar medidas;

◊ Verifyng Implementation (VE) - Descreve os passos para garantir que as atividades são executadas em conformidade com o processo estabelecido. Inclue revisões, auditorias e garantia da qualidade de software.

A estruturação dos níveis de maturidade CMM é caracterizado pelos processos chaves, conforme mostra a Tab.2.6.

NÍVEL DE MATURIDADE	ÁREAS CHAVES DE PROCESSOS
Nível 1 - Inicial Existem poucos processos e é, as vezes, caótico. O sucesso depende de esforços individuais.	
Nível 2 - Repetitivo Processos básicos de gerenciamento de projetos para acompanhar e controlar custos, prazos, cronograma e funcionalidade.	Requisitos de Gerenciamento (RM) Planejamento de Projeto de Software (PP) Acompanhamento do projeto de software Garantia da Qualidade de Software Gerenciamento de Configuração de Software
Nível 3 - Definido O processo de software para gerenciamento e para a engenharia são documentados, padronizados e integrado com o processo organizacional. Existe um processo padrão de desenvolvimento e manutenção de software.	Foco de Processo (PF) Definição de Processo (PD) Programa de Treinamento (PT) Gerenciamento Integrado de Software (IM) Engenharia de Produto de Software Coordenação Inter-grupos (IC) Peer Reviews (PR)
Nível 4 - Gerenciado Medidas detalhadas da qualidade de produto e de processo são coletadas. Os processos e os produtos são bastante controlados.	Gerenciamento Quantitativo do Processo (QP) Gerenciamento da Qualidade de Software (QM)
Nível 5 - Otimizante Os processos são continuamente melhorados, mudando os processo com a incorporação de novas idéias e tecnologias.	Prevenção de Defeitos (DP) Gerenciamento de Mudanças de Tecnologias (TM) Gerenciamento de Mudanças de Processos (PC)

Tab.2.6 - Áreas chaves de processos requeridos para cada nível de maturidade CMM. (Baseado em Software Engineering Institute, CMU/SEI-93-TR-24, Capability Model for Software, Version 1.1, Carnegie Mellon University, 1993).

2.4.4 O modelo Malcom Baldrige National Quality Award

Foi criado por uma lei pública em 1987, com a meta específica de melhoria de qualidade nos EUA.

Malcom Baldrige National Quality Award (MB) forma uma estrutura composta de quatro elementos:

◊ Driver - Incorpora as diretrizes executivas: cria valores, metas e as sistematicas; e diretrizes para buscar atender as expectativas do cliente e da melhoria da companhia;

◊ System - O sistema é composto de um conjunto de processos bem definidos e projetados que atendam aos clientes da organização e aos requisitos de performance;

◊ Measures of Progress - Medidas de Progresso disponibiliza uma base orientada a resultados para canalizar as ações para a melhoria de atendimento externo e a própria organização interna; e

◊ Goals - Estabelece os alvos para dirigir uma melhoria no atendimento do cliente e na disputa de segmento de mercado.

A estrutura do MB inclui categorias de critérios para agrupar os itens de avaliações e pontuação. São sete categorias (Fig.2.12) com 24 itens de avaliação.

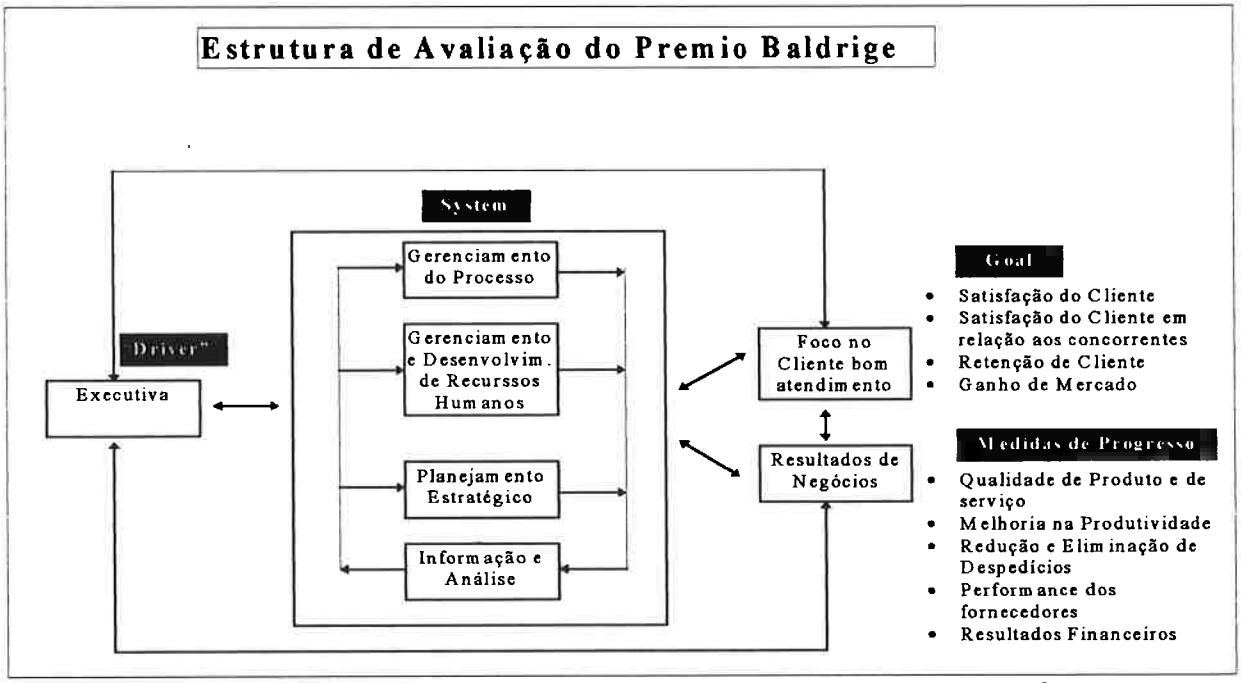

Fig.2.12 - Estrutura de Avaliação do Malcom Baldrige (Baseado em TINGEY (1997)).

A estrutura de exame do MB e a avaliação é executada por um grupo de 5 membros de revisores e avaliadores, que realizam a avaliação no local para então documentar o julgamento e as recomendações finais.

Essa avaliação apresenta uma estrutura conforme mostra a Tab.2.7.

Categoria de Exame	Itens
1. Condução Executiva	1.1 Condução executiva 1.2 Organização e sistema executivo 1.3 Responsabilidade pública e corporativa
2. Informação e Análise	2.1 Gerenciamento de dados e informações 2.2 Comparações e *benchmark* de concorrentes 2.3 Análise e uso de dados corporativos
3. Planejamento Estratégico	3.1 Estratégia de Desenvolvimento 3.2 Estratégia de Particionamento
4. Gerenciamento e Desenvolvimento de Recursos Humanos	4.1 Planejamento e avaliação de recursos humanos 4.2 Sistema de trabalho de alta performance 4.3 Desenvolvimento, educação e treinamento de funcionários 4.4 Satisfação e bem-estar dos funcionários
5. Gerenciamento de Processos	5.1 Projeto e introdução e serviços e produtos 5.2 Gerenciamento de processos: Produção e liberação de produtos e serviços 5.3 Gerenciamento de Processos: Serviço e suporte 5.4 Gerenciamento de performance de fornecedores
6. Resultados de Negócios	6.1 Resultados de qualidade de produtos e de serviços 6.2 Resultados operacionais e financeiros da corporação 6.3 Resultados de performances de fornecedores
7. Foco e Satisfação do Cliente	7.1 Conhecimento do cliente e do mercado 7.2 Gerenciamento do relacionamento com o cliente 7.3 Determinação da satisfação do cliente 7.4 Resultados da satisfação do cliente 7.5 Comparação da satisfação do cliente

Tab.2.7 - Itens de avaliação do Prêmio MB, categorizados conforme Fig.2.12.
(Baseado em TINGEY (1997)).

2.4.5 Outros padrões de qualidade

Vale citar dois outros modelos de sistemas de gestão de qualidade, mais notáveis, segundo levantamento realizado por Tingey (em TINGEY (1997)): Deming Prize e European Quality Award.

Deming Prize

Este modelo empresta o nome de W. Edwards Deming, experiente cientista da qualidade com trabalhos reconhecidos nos EUA e no Japão. A sua estrutura de avaliação apresenta os seguintes itens:

1. Política
2. Gerenciamento da Organização
3. Education
4. Sincronização de Informação
5. Análises
6. Padronização
7. Controle
8. garantia da Qualidade
9. Resultados
10. Planejamento futuro

Tab.2.8 - Estrutura de avaliação do Deming Prize.
(baseado no TINGEY (1997)).

European Quality Award

A European Foundation for Quality Management (EFQM) foi criada em 1988 por 14 companhias e iniciou a premiação em 1992. A sua premiação é semelhante ao Malcom Baldrige, com uma estruturação de avaliação de 9 itens (ao invés de 7), conforme mostra a Tab.2.9.

1. Executiva
2. Política e Estratégias
3. Gerenciamento de Pessoal
4. Recusos
5. Processos
6. Satisfação de Clientes
7. Satisfação da Equipe Interna
8. Impacto na Eociedade
9. Resultados de Negócios

Tab.2.9 - Estrutura de avaliação da European Quality Award. (baseada em TINGEY(1997)).

3. MÉTRICAS DE SOFTWARE

Objetivo do Capítulo:

Apresentar uma base teórica a respeito de métricas, procurando esclarecer conceitos sobre medidas diretas, medidas indiretas, estimativas e eliminar alguns conflitos de terminologia a respeito de métricas. O assunto é detalhado com enfoque específico de aplicação na área de engenharia de software.

Tópicos do Capítulo:

3.1 Histórico

Para entender a evolução de estudos relacionadas com métricas, com especial ênfase em software, apresenta-se um breve relato histórico a respeito do assunto.

3.1.1 Primeiras pesquisas

A base teórica sobre métricas está inserida neste contexto para subsidiar a definição de atributos de medidas, estabelecer mecanismos de medidas em processos. E dessa maneira, organizar a terminologia sobre o assunto necessário ao desenvolvimento da tese.

Para posicionar o atual contexto sobre o assunto métrica, vale um resumido retrospecto, percorrendo um período de 20 anos, até chegar o presente contexto da indústria de software, onde a qualidade de produto e de processo é hoje uma fator de muita influência na competitividade das empresas, considerando ser os sistemas computacionais elementos fundamentais na composição da infraestrutura de apoio aos processos de negócios do mundo moderno. Muitas vezes essa infra-estrutura é um fator de diferenciação na disputa de segmento de mercado. Os principais registros históricos estão apresentados a seguir.

1968, R. J. Rubey e R. D. Hartwick, na Conferência da ACM National Computer Conference

Apresentaram um sistema compreensivo de atributos de software, definidos por eles como qualidade de software. Métricas quantitativas foram associadas com cada atributo. O atributo de medida calcula a proporção de instruções que não são

modificadas durante a execução do programa. Se eles modificam durante a execução, tais modificações são claramente identificadas. Assim vale a fórmula:

$$M = \max [\ 0,\ 100(I - 2N + C)/I\ ]$$

onde:

I = Número de instruções no programa;

N = Número de instruções modificadas durante a execução do programa;

C = Número de instruções modificadas que tem comentários que indicam a fonte da modificação, o tipo e o resultado que pode ocorrer durante a execução;

Uma das consequências dessa medida é um fator externo que indica quanto bem o programador as partes do programa sobre a qual ele não tem controle. Além disso, o modelo de qualidade fornece subsídios para relacionar todos os fatores necessários para se julgar a qualidade de um programa:

- ◊ Medida absoluta;
- ◊ Medida normalizada indicado quanto bom o programa foi montada durante o desenvolvimento;
- ◊ Medida com fatores de pesos levando em conta a importância relativa dos atributos;

Neste trabalho, os autores encerram indicando a importância de se realizar tais medidas durante o desenvolvimento para garantir maior qualidade de medida e prevenir futuros gastos.

1972, Maurice Halstead

Software science foi o trabalho mais significativo focado em métrica de software mais detalhadamente. Neste trabalho, os pesquisadores partiram de analogia de termodinâmica que relaciona volume, pressão, temperatura e energia:

- ◊ Analogia 1: algoritmos é uma destilação do pensamento, isto é, sua natureza está relacionada com o esforço mental, assim como a temperatura está relacionada com a energia ou esforço físico;
- ◊ Analogia 2: os quatro processos (Compilação, otimização, decomposição e canonização) formam o ciclo ideal de software e é análogo ao Ciclo de Carnot, de onde surge um método para medir a temperatura de um "algoritmo";
- ◊ Dizia-se que os algoritmos deveriam ser medidos, não em termos de execução em qualquer máquina real, mas com métricas observadas diretamente dos programas. Considerou-se ainda que os programas traduzidos para as máquinas são compostas de comandos simples estruturadas de operador e operando. Assim, uma métrica possível é a contagem do número de operandos e operadores.
- ◊ Aplicou-se essa técnica de medida para um mesmo algoritmo implementado em quatro linguagens de programação, de onde mediu-se para cada versão os números n1 e n2 (vocabulário de componentes) e N1 e N2, quantidades totais de componentes. Descobriu-se uma clara relação entre comprimento total, como função dos vocabulários:

N = N1 + N2 pode ser estimado por

Nest = n.log n, onde

n = n1 + n2

n1 = número de tipos de operadores (vocabulário de operadores)

n2 = número de tipos de operandos (vocabulário de operando)

N1 = Número total de operadores no módulo;

N2 = Número total de operandos no módulo.

Além disso, determinou as seguintes relações:

Volume

$V = N \log n$

Nível de Implementação

$Lest = (2/n1).(n2/N2)$

Qualidade interna

$I = V.L$

Esforço

$E = V/l$

Com base nestes estudos, muitos trabalhos foram conduzidos na Universidade de Purdue, das quais podemos citar:

EXPERIMENTOS:

◊ HASLTEAD (1973b), BULUT et.al (1974), CORNELL; OTTENSTEIN (1976), GORDON; HALSTEAD (1976), WOODFIELD (1979);

APLICAÇÕES:

◊ HALSTEAD (1973a, 1976b, 1977a, 1978, 1979), COMER; HALSTEAD (1979), FUNAMI; HALSTEAD (1976), OTTENSTEIN (1976), BAKER; ZWEBEN (1979), GORDON (1979a, 1979b), OLDEHOEFT; BASS (1979).

REFINAMENTO DE TEORIA:

◊ BULUT; HALSTEAD (1974), HALSTEAD (1976a), OLDHOEFT (1977).

NOVAS RELAÇÕES:

◊ HALSTEAD (1976b), SCHENEIDER (1978), OTTENSTEIN (1979).

Nestes estudos, Halstead resumiu uma série de aplicações relacionada com as métricas até então estudadas, como por exemplo:

- Determinação do tempo para que o programador de manutenção se familiarize com o novo programa;
- Determinação de quando que programas pode ser melhorado com o objetivo de facilitar a manutenção;
- prever o número de BUGS restante no código, no início da fase de testes;
- Estimar o número de execuções de programas por dia, por programador, durante os testes;
- Prever números do programa, a partir da especificação;
- Graduação automática do estilo de progrmação de estudantes;
- Detecção de automática de programas com plágio;

No seu último trabalho, Halstead declarou algumas definições que devem estar em qualquer disciplina da ciência. São cinco:

1. *Métricas* - que podem ser aplicadas para qualquer item no seu domínio;

2. *Experimentos reproduzíveis* - que podem ser reproduzidos e repetidos com resultados comparáveis em laboratórios independentes, por diferentes pesquisadores;

3. *Relações derivadas* - a partir de várias propriedades dos itens do domínio;

4. *Explicação* - possibilidade de explicar fenômenos observados a partir de propriedades básicas;

5. *Previsão* - possibilidade de prever o resultado de um experimento, antes mesmo dele ser executado, ao invés de simples explicações após a sua execução.

1981, Perlis, Sayward e Shaw - OFFICE NAVAL RESEARCH

Esses pesquisadores discutem a importância da métrica em software na Office of Naval Research, onde concluíram que é importante que a ciência da computação não foque em desenvolvimentos prematuros ou incorretos, pois a complexidade de software deverá crescer muito em função da crescente necessidades em software; Até este momento, as métricas estavam relacionadas com os produtos de software obtidos e não sobre o processo;

1976, McCabe - Modularização, Testabilidade e Manutenibilidade

Como modularizar sistemas de software tal que os módulos resultantes são testáveis e manuteníveis? Quais as técnicas matemáticas que deverá fornecer uma base quantitativa para modularização e identificar partes que apresentarão dificuldades nos testes ou na manutenção.

McCabe (MCCABE, 1976) utilizou a técnica de medir e controlar o número de caminhos de um programa e defendeu a complexidade do grafo de fluxo de controle como sendo um indicador primário de testabilidade e manutenibilidade, que acabaram se tornando conhecido como métrica de complexidade.

1976, Elshoff - Fluxos de controle

Elshoff (em ELSHOFF, 1976a), usou o número de referência a dados, número de fluxos de controle como métricas de complexidades de programas.

1977, Walston e Felix - Processos industriais

Walston e Felix (em WALSTON; FELIX, 1977) registraram o esforço industrial para estabelecer medidas de programas: iniciaram medidas de sistemas de software em 1972, quando decidiram analisar os efeitos da programação estruturada nos processos de desenvolvimento de software. Para isso estabeleceram um rigoroso programa para medir as metodologias então disponíveis, de modo a preparar medidas para introdução de novas metodologias que certamente viriam no futuro próximo. Nessa situação, atingiram os seguintes objetivos:

- Fornecer dados para avaliação das novas tecnologias de programação;

- Fornecer suporte para propostas e contratos de desempenho;

- Sincronizar e manter registros históricos dos trabalhos executados em engenharia de software;

- Fornecer dados de programação para gerenciamento.

- Estabelecer uma terminologia comum (padrão) de programação.

1978, Basili e Zelkowitz - Avaliação de Processo e Produto

Basili e Zelkowitz, realizaram estudos (BASILI; ZELKOWITZ, 1978) com base nos resultados obtidos no Laboratório de engenharia de software da NASA Goddard Space Flight Center, juntamente com a Universidade de Maryland para atender a seguinte meta:

◊ Analisar o processo de desenvolvimento de software e o software produzido de modo a entender o processo de desenvolvimento, os produtos de software, os efeitos de melhorias no processo e desenvolver medidas quantitativas que correlacione com a intuição com aquilo que entende por um bom software.

3.1.2 Proliferação de métricas

Nas décadas de 70 e 80 espalhou-se esforços e conceitos para estudos e criação de novas métricas, das quais podemos citar (MELTON, 1996):

◊ 1979, Schneiderwind e Hoffman evolui as métricas baseadas em fluxos de processamento propostos por McCabe;

◊ 1979, Benyon-Tinker define a métrica de complexidade baseada nas árvores de chamadas do programa;

◊ 1978, Hansen sugere aumentar o ciclo proposto por McCabe através de operador de contagem;

◊ 1977, Meyers extende o modelo de McCabe para incluir o número relacionados com estruturas condicionais d eprogramas;

◊ 1979, Woodward, Hennell e Hedley, definem métricas de complexidade com "Knots" como sendo cruzamentos de dois caminhos de controle;

◊ 1979, Gaffney usa *jumps*, de código como métrica de controle de complexidade;

◊ 1981, Harrison e Magel, descrevem métrica de complexidade baseada em níveis de encadeamento;

◊ 1981, Henry, Kafura, e Harris, definem métricas de complexidades de fluxo de informações como comprimento (fan-in * fan-out);

◊ 1979, 1983, Albrecht, definem métricas usando pontos de funções - é a preocupação de prever processos e produtos nas fases anteriores de desenvolvimento - Pontos de funções são baseadas na contagem de números de entradas externas por usuários, funções, saídas, arquivos básicos a serem utilizados no projeto. Pontos de função foi considerado viável devido ao fato de essa contagem pode ser feita de forma fácil através de reuniões envolvendo usuários e clientes nas fases iniciais de desenvolvimento. São dados mais fáceis de serem entendidos pelos clientes do que SLOC (Source Lines of Code), por exemplo;

◊ 1978, Yin e Winchester desenvolveram métricas baseadas e em diagramas de projetos;

◊ 1979, Chapin, estabeleceu métricas baseadas em regras de dados numa função: ele relacionou os tipos de entradas e saídas com os tipos de funções do programa. Foi um dos primeiros pesquisadores a lidar com a confiabilidade de métricas, onde ele disse: "quando diferentes pessoas usam o mesmo procedimento computacional de forma consistente, devem obter o mesmo resultado";

◊ 1981, Mohanty, apresentou uma métrica baseada nos conceitos da teoria da informação de Shannon, como medida da quantidade de informação compartilhada por subsistemas de um sistema;

◊ 1981, Whitworth e Szulewski métricas para a fase de projeto (design phase), através do uso de dígrafos;

◊ 1981, Troy e Zweben selecionou métricas capazes de auxiliar a avaliação de projetos;

Pode se dizer que a maturidade da disciplina de métricas se refletiu a através da quantidade de livros publicados na década de 70. Em especial, vale ressaltar as seguintes publicações: HALSTEAD (1977a), GILB(1976) E BOEHM (1978). Adicionalmente, uma publicação especial da IEEE Workshop on Quantitative Software Models, em Outubro de 1979, apresentou seções de confiabilidade, complexidade e custos. Nesta publicação, Shooman declarou: " Modelamento de custos é o que está melhor desenvolvida de três áreas; a complexidade está numa

fase inicial; Assim, as discussões estão focadas em novas medidas e na aplicabilidade práticas delas e de outras existentes".

3.1.3 Principais críticas às pesquisas das métricas (Antes da década de 80)

As principais críticas relacionadas com as pesquisas conduzidas na década de 70 estão relacionadas com os aspectos psicológicos incorporados por Halstead, onde no detalhamento de seus estudos incluem números mágicos como 7 mais ou menos 2 e 18 discriminações mentais por segundo. Esses números estão relacionados com os processos cognitivos referentes aos processos de percepção e retenção de um estímulo, ao invés de atividades complexas que envolvem a programação.

Outra crítica forte está relacionada com os métodos estatísticos utilizados, especialmente as técnicas de validação da correlação de métricas com as ocorrência de erros.

Enfim, as críticas ficaram por conta da aplicabilidade das métricas em grandes sistemas industriais. Uma avaliação forte foi emitida em trabalho publicado por DeMillo e Lipton em 1981, onde declaram que é mais importante analisar e selecionar medidas para construção de bases históricas para "prever" o futuro do estudar métricas. Ou seja, é melhor analisar as propriedades observáveis de software, especialmente aquelas que podem ser numericamente caracterizadas e objetivamente registradas.

3.1.4 Tendências

Como foi descrito, muito se fez em torno do assunto e conforme demostraram as dúvidas, sabe-se muito sobre como não realizar métricas, porém permanece as dúvidas de como fazer. A seguir apresenta-se um panorama geral das frentes relacionadas com métodos e técnicas de realização de métricas.

◊ **Data Gathering (BASILI; WEISS, 1984)** - É baseada na coleção de dados que devem satisfazer metas estabelecidas por uma série de questões a serem respondidas , considerando a implementação desta coleção e validação dos dados. Este processo tem também o nome de paradigma GQM (Goal-Question-Metric). Esse mecanismo exige uma sistematização da estruturação dos dados e programas para viabilizar a coleta dos dados;

◊ **Teoria do Comportamento (KEARNEY et. al. 1986)** - Neste contexto, defende-se a importância da especificidade da aplicação e principalmente a experiência e habilidades dos desenvolvedores. As propriedades robustez, padronização, especificidade, e documentação relacionada com a prescrição são importantíssimos para a definição de métricas.

◊ **Examinação analítica de Métricas (BAKER; ZWEBEN, 1980)** - Esta técnica é executada através de exames detalhados de métricas específicas. Baker e Zweben analisaram as transformações aplicadas sobre as representações de fluxos de controle para geração de códigos.

◊ **Teoria de medidas (DeMILLO; LIPTON, 1981)** - De Millo e Lipton discutiram a relevância da teoria de medida nos anos 80. Para eles é importante

a base matemática para subsidiar a montagens de métricas e realização de medidas;

◊ **Técnica Axiomática (KAFURA; HENRY, 1981)** - Essa técnica reza que as métricas e as suas avaliações devem ter base formal estruturada por um conjunto de axiomas capazes de formar uma framework teórica comparar as complexidades de métricas e analisar novas métricas à medida em que vão sendo desenvolvidas.

◊ **Técnica da Estatística e Experimentos (BASILI, 1986)** - Basili e equipe propôs o TAME (Tailoring a Measurement Enviroment) - uma framework para ajudar a estruturar processos experimentais.

◊ **Processos e Métricas (HUMPHEREY, 1988)** - Vários pesquisadores iniciaram relacionar métricas com processos. Humphrey enfatizou (apud MELTON, 1996): "é essencial limitar as medidas para os poucos itens que realmente serão usados. Medidas são muito caras e corruptíveis ...". Neste trabalhos, defende-se que para cada nível de maturidade de produção de software, a SEI estabelece o conjunto de medidas apropriada para o seu ambiente.

3.2 Teoria de Medidas

Considerando que medida de software não deixa de ser medida (MELTON, 1996), é importante discutir base teórica sobre medidas para então especializar para as medidas de software.

3.2.1 O que é medida?

Medida é definida como um processo na qual entidades de um determinado conjunto recebem símbolos ou números que indicam atributos destas entidades. Os atributos recebem os valores (números ou símbolos) codificados que permitem diferenciar uma entidade da outra.

Uma **entidade** é um objeto do mundo real, como por exemplo, uma pessoa, um carro, uma especificação de software ou mesmo um evento que representa uma fase de um sistema de software.

Um **atributo** é uma propriedade ou característica da entidade. Por exemplo, altura, pressão de uma pessoa, o tamanho de uma especificação de software, duração de uma fase de testes de um sistema.

Normalmente, utiliza-se os valores de medidas como meio de comparação de entidades. Por exemplo, vale a propriedade de que sempre uma pessoa X é maior que uma pessoa Y, então a medida de altura (X) é maior que a medida de altura (Y). Ou seja é importante o conceito de relação:

Relacionamento entre os valores de medida, ou melhor, relacionamento entre entidades através dos atributos;

Assim, certamente a medida de altura da pessoa X é maior que a medida de altura da pessoa Y, quando a altura real de X é maior que de Y. O relacionamento entre entidade é então mapeada para o relacionamento entre as medidas.

O relacionamento entre atributos são modelados através de relações matemáticas, valendo para entidades e atributos.

3.2.2 Medidas Diretas, Indiretas e Previsão

Uma medida pode ser direta, indireta ou uma previsão. A medida direta é aquela em que a partir de observação de uma entidade, aplica-se uma ferramenta de medida diretamente sobre a entidade. Por exemplo a altura e o peso de uma pessoa, ou o tempo.

Em medidas indiretas, não é possível, ou não é conveniente aplicar uma ferramenta de medida diretamente na entidade. Nesse caso, a medida indireta é aquela obtida através de cálculos baseados em relação matemática de atributos de medidas diretas, ou seja, mede-se alguns atributos diretos e então através de cálculo sobre estas medidas obtém a medida desejada.

As medidas do tipo previsão é quando se pretende estimar a medida de uma entidade que pode não existir naquele momento, porém é necessário a sua medida. Por exemplo, saber uma estimativa de custo de desenvolvimento de software, antes de tomar a decisão de realmente desenvolvê-lo.

Existem medidas que podem ser feitas de forma direta ou indireta. Um exemplo típico é a da velocidade, onde:

v = distancia/tempo;

Neste caso, calcula-se geralmente a velocidade em termos de medidas direta de distância e tempo. Discussões mais detalhadas podem ser vistas em KRANTZ (1971) e KYBURG (1984).

3.2.3 Focando as atividades de medidas

As teorias de medida indicam que as atividades de medidas devem ser executadas com base em objetivos claros das metas a serem atingidas. Por exemplo, ao se mudar um processo para otimizar os custos de manutenção, então certamente medidas de custos do processo corrente devem estar adequadamente estabelecidas. É preciso saber exatamente quantas entidades são de interesse. A seguir, um exemplo mais.

Imagine que por algum motivo as entidades definidas de análise foram fase de manutenção e código-fonte. Definido isto, é importante dizer quais atributos destas entidades devem ser medidas. Exemplo:

◊ ENTIDADE - Fase de Manutenção

◊ ATRIBUTOS

- Tempo gasto;

- Esforço dispendido.

◊ ENTIDADE - Código-fonte

◊ ATRIBUTOS

- Tamanho

- Funcionalidade

Na teoria é importante entender esta necessidade. Na prática, as vezes acontece muito de se realizar as métricas sem ter definido antes a sua aplicabilidade objetiva, ou pior, associa-la de forma errada, podendo levar a conclusões errôneas. Por isso que até hoje questiona-se sobre quais métricas deve-se usar.

3.3 Fundamentos em Medidas de Software

A incorporação da teoria de medida nos processos de engenharia de software é recente, onde os principais trabalhos que devem ser listados são DeMILLO; LIPTON (1981), BAKER (1990), FENTON (1991), ZUSE (1991). A teoria de medidas deve ser capaz de responder perguntas do tipo:

a) Quanto se deve conhecer sobre um atributo antes de realizar as medidas? (Conhece-se o suficiente sobre complexidade de programas para medição dos mesmos ?)

b) Como saber se medimos corretamente o atributo desejado? (A quantidade de erros encontrados durante a fase de integração mede a confiabilidade do sistema? Se não, o que medir?)

c) As declarações sobre medidas são significativas? Tem significado dizer "aumentamos a qualidade em 30%"?

d) Que tipo de operação pode ser feito sobre medida? Tem sentido dizer calcular a produtividade média de um grupo de pessoas?

Os itens que se seguem fornecem a base teórica para responder os tipos de questionamentos acima.

3.3.1 Relações empíricas

As relações empíricas possibilitam relacionar entidades na forma de comparação, associação, ou elaborar conjuntos de entidades com atributos comuns, e até mesmo distinguir subconjuntos de atividades com base em observações de medidas de atributos destas entidades. Por exemplo, se num subconjunto de entidades, definido através de um determinado atributo, as entidades não podem ser diferenciadas, então eles apresentam a mesma medida de atributo para todos os elementos deste conjunto.

Por outro lado, se duas entidades são diferentes segundo valores de um atributo, qual a conclusão? As medidas desses atributos, certamente diferentes, permitem comparar as duas entidades ou os valores diferentes indicam que as entidades são diferentes?

A teoria de métricas indica que é preciso duas coisas para estabelecer relações empíricas: uma é a observação empírica sobre as entidades; a outra é estabelecer

uma organização empírica através de descrição formal, por meio de relação matemática, segundo formalismo apresentado no item 3.3.2.

3.3.2 Condição matemáticas para relações e representações

Para medir um atributo de um conjunto de atividades, é preciso montar o sistema de relação numérica que reflita a relação empírica que o atributo gera sobre o conjunto de atividades. Uma relação empírica é estabelecida sobre um conjunto de números ou símbolos. Assim, de acordo com a teoria de representação, um sistema de relação numérica deve coincidir exatamente com a relação empírica, ou seja, observar a seguinte definição:

Definição 3.1:

Seja E um conjunto de entidades e $R = \{ R_1, R_2, \ldots , R_n\}$ seja o conjunto de relações empíricas definidas sobre E, com respeito a um dado atributo. $<E,R>$ é o sistema de relação empírica. Seja N um conjunto de números e símbolos, e seja $S = \{S_1, S_2, \ldots, S_n\}$ um conjunto de relações numéricas definidas sobre N. $<N,S>$ é o sistema de relação numérica. M é a medida para E com respeito a um determinado atributo, ou melhor, M é uma medida para $<E,R>$ se:

1) $M : E \text{----} > N$

e

2) $R_i (e_1, e_2, \ldots, e_n)$ se e somente se $Si (M(e_1), M(e_2), \ldots, M(e_n))$.

A primeira condição define que M é função de E para N. A segunda condição diz que os elementos em E estão relacionados com por uma relação R_i em R exatamente quando a imagem destes elementos são relacionadas por uma relação em S_i de S. É necessário que R e S tenham o mesmo número de relações e para i=1,2,...,n, e que R_i corresponda ao S_i.

Exemplo: Suponha que se deseja medir as pessoas pelo tipo de lojam que frequentam num shopping center. Este tipo de medida é de classificação. Devem ser analisados três tipos de lojas: doceria, eletrônicos e roupas. A codificação da classificação poderia ser:

1 - Doceria

2 - Eletrônicos

3 - Roupas

Nessa situação não existe nenhuma relação naturalmente pessoas que frequentam a loja de eletrônicos e as que frequentam as lojas de roupa. Assim as representação adotada não permite nenhuma relação sobre os números 1,2 ou 3. A relação $1 < 3$ não serve como relação para estes casos.

Outro exemplo: Suponha que se deseja medir pessoas baseadas na respostas a uma determinada piada. Definiu-se 5 tipos de respostas:

1 - Choroso,

2 - Triste (Descontente),

3 - Sério,

4 - Sorridente, e

5 - Gargalhando.

é natural neste caso que existe uma relação entre pessoas, no que diz respeito a essas respostas: Triste é melhor que choroso; Sério é melhor que triste; Sorridente é melhor que sério; e gargalhando é melhor que sorridente. Assimpode ser montar relações empíricas denominadas R que incluem os pares ordenados (choroso, triste), (triste, sério), (sério, sorridente) e (sorridente e gargalhando). Assim, de acordo com a segunda definição dada, o sistema de numeração inclue a correspondente relação, que neste caso pode ser a relação "menor que" (<).

3.3.3 Significados de Medidas

O significado de uma medida é determinada pela riqueza da relação que ela determinada. Essa riqueza de informação é calibrada por escalas, cujos os tipos podem ser determinadas por transformações admissíveis.

Suponha que M e M' são dois tipos diferentes de medidas de altura de uma pessoa. Nesse caso sempre é possível achar uma constante $K > 0$, tal que $M = K.M'$. Se, por exemplo, M' está em polegadas e M está em centímetros, então $K = 2.54$. A transformação de uma medida para outra é chamada de transformação admissível. No caso de altura de pessoa, cada transformação admissível é uma

multiplicação escalar. Em geral, a classe de transformações admissíveis determinam a tipo de escala da medida.

Podemos relacionar cinco tipos de escalas bem conhecidas, que são: a nominal, ordinal, intervalo, proporção e absoluta. Esses tipos são caracterizadas por transformações admissíveis:

Medidas nominais usam transformações admissível do tipo função um-para-um. Exemplos deste tipo de medidas são as classificações como o do exemplo de pessoas que frequentam as lojas de shopping;

◊ Medidas ordinais são caracterizadas por funções do tipo

F: X ---> Y,

se x < x' então F(x) < F(x'). Exemplos de medidas ordinais são medidas de preferência ou de qualidade como o do exemplo de piadas;

◊ Medidas de intervalo usam funções lineares do tipo

F: X ---> Y, onde F(x) = a.x + b, onde a > 0 é uma constante. Exemplo, intervalos de temperatura Celcius e Fahrenheit.

◊ Medidas de proporção usam funções do tipo lineares do tipo

F: X ---> Y, onde F(x) = a.x, e a >0 é uma constante. Como exemplo, a medida de comprimento.

◊ Medida absoluta só admite a função identidade como transformação admissível. Por exemplo, contagem é uma medida absoluta.

Normalmente o entendimento e o significado inicial de um atributo de medida é bastante pobre. Somente com a acumulação de dados e análises de resultados permitem uma reavaliação do atributo. Isto pode levar ao refinamento ou criação de novas relações empíricas e melhoria na precisão das medidas e normalmente com a obtenção de escalas mais refinadas.

Muitos atributos de software ainda apresentam sistemas de relações empíricas muitos "crus". Por exemplo a criticidade de falhas de software usam um relação simples do tipo binária "mais crítico do que". Neste caso, qualquer duas medidas são relacionadas pela transformação de ordem (escala ordinária).

Formalmente, uma declaração tem significado exato quando é o nível de verdade ou de falso permanece constante sobre qualquer transformação admissível das medidas envolvidas. Por exemplo, José é mais alto duas vezes que Maria. Essa declaração permanece (com o mesmo nível de verdadeira ou falsa) qualquer que seja o método de medida de altura, em centímetros ou em múltiplos de centímetros.

Por outro lado a afirmação de que a falha x é duas vezes mais crítica que a falha y não tem significado formal se usa uma escala ordinária de medidas de criticidade. Nesse caso, uma escala poderia estabelecer que M(x) = 6 e M(y) = 3, então a afirmação é verdadeira. Porém, se uma outra escala ordinária determinar que M'(x) = 10 e M'(y) = 9, a afirmação "duas vezes..." não é mais válida.

A noção de significância também permite determinar quais tipos de operações podem ser realizadas sobre diferentes medidas. Por exemplo calcular a média de medidas do tipo proporção, mas não faz sentido calcular a média de medidas do tipo ordinal.

3.4 Formalizando Medidas Diretas, Indireta e Previsões

3.4.1 Medidas Diretas

Antes de realizar medidas diretas de um conjunto de entidades, em relação a um determinado atributo, é preciso estabelecer:

- Relação que existe entre as entidades e as que são determinadas pelos atributos em questão;
- Definição ou identificação de um sistema numérico ou simbólico equivalente as relações de atributos;
- Um função que mapeia o conjunto de entidades para o conjunto de números ou símbolos (R_i para S_i);
- Uma ferramenta de medida é desenvolvida. Esta ferramenta é uma implementação da função definida on item anterior.

3.4.2 Medidas Indiretas

Na medida indireta é preciso medir diretamente uma ou mais entidades, e em seguida calcular a medida indireta da entidade em questão. Para isso, é necessários estabelecer:

- A conexão entre os conjunto de entidades que serão medidas diretamente com o conjunto de entidades que serão medidas indiretamente;
- Esta conexão deve ser refinada e desenvolvida em uma fórmula, ou conjunto de fórmulas, que permite a uma tradução de medidas diretas para medidas indiretas.

3.4.3 Medidas de previsão (Estimativas)

Em medidas do tipo previsão ou estimativa é preciso medir diretamente uma ou mais entidades, e em seguida calcular a medida de previsão da entidade em questão. Para isso, é necessários estabelecer:

- A conexão entre os conjunto de entidades que serão medidas diretamente com o conjunto de entidades que terão suas medidas estimadas;
- Esta conexão deve ser refinada e desenvolvida em uma fórmula, ou conjunto de fórmulas, que permite a uma tradução de medidas diretas para medidas aproximadas de previsão.

O elemento chave para se fazer boas medidas de previsão é estabelecer uma boa função de mapeamento das medidas diretas para as medidas aproximadas. Em sistemas de software muito se fala sobre estimativas de custos, tempos de desenvolvimento ou confiabilidade de sistemas.

3.5 Fundamentos de Métricas em Software

Tendo como base a teoria de medidas apresentada nos itens anteriores, pode-se especializar os conceitos sobre medidas em software.

Antes disso, devemos reafirmar alguns termos definições, pois muitas vezes o termo métrica aparece com mais de um significado, como por exemplo nos três casos citados a seguir:

a) Um número obtido de um produto, processo ou recurso.

◊ Por exemplo, métrica de pontos de função, métrica de linhas de código (LOC) por programador.mês. Na verdade, não se trata de métrica e sim de medidas diretas.

b) Escala de medida.

◊ Uma classificação de falhas de software como métrica.

c) Um atributo específico.

◊ Por exemplo, métrica de portabilidade de programas ou métrica de acoplamento em projetos, mesmo sem relacionar com número de medidas. Nesse caso, portabilidade e acoplamento são atributos, ou seja características de entidades como programas e projeto.

Essas várias interpretações de métricas são admissíveis uma vez que elas podem ser acomodadas na base científica de medidas, usando-se conceitos e termos bem definidos conforme orientação da teoria apresentada nos itens anteriores.

Definição 3.2: Um atributo é uma característica ou propriedade de uma entidade;

Definição 3.3: Medição é um processo de atribuição de número ou símbolo empírica e objetivamente a atributos de entidades (que podem ser objetos ou eventos) do mundo real, afim de descrever estas mesmas entidades;

Definição 3.4: Medida caracteriza um atributo específico de uma entidade, através de número ou símbolo atribuído empíricamente;

Definição 3.5: Medição direta de um atributo é uma medição que não depende da medição de nenhum outro atributo;

Definição 3.6: Medição indireta de um atributo é uma medição que envolve a medição de outros (um ou mais) atributos;

Definição 3.7: Um modelo (segundo FENTON, 1991) é:

1) Uma abstração de um conjunto de objetos que apresenta certos atributos dos objetos e ignora outros atributos;
2) Uma fórmula matemática que relaciona duas ou mais medidas.

Definição 3.8: Um produto é resultado concreto ou um documento que é produzido durante o desenvolvimeto de software;

Definição 3.9: Um processo é um conjunto de atividades associadas a software;

Definição 3.10: Um recurso é uma entrada para um processo;

Definição 3.11: Um atributo interno de uma entidade é um atributo que pode ser medida completamente da própria entidade;

Definição 3.12: Um atributo externo de uma entidade é um atributo que pode ser medido no que se refere ao relacionamento da entidade com o seu ambiente;

Métricas de software e suas medidas devem ser usadas para determinar o estado de software e também guiar o desenvolvimento de software, através de uma base teórica de métricas onde dois termos (distintos) são fundamentais (STANLEY; HOPKINGS, 1972):

- ◊ Termo 1) **Confiável** - Uma medida é confiável se o método usado para a medição fornece resultados consistentes. Assim, duas pessoas diferentes usando o mesmo método de medição irão obter os mesmos resultados. Uma mesma pessoa usando o mesmo método em tempos diferente obterá o mesmo valor;
- ◊ Termo 2) **Válido** - Uma medida é valida se reflete o mundo real de forma precisa. Por exemplo, se as medições de dois programas A e B indica que o programa A tem um maior grau de acoplamento que o programa B, então espera-se que na realidade a codificação de A é mais acoplada que a de B.

A montagem de uma base teórica de métrica de software (citada na literatura como *Foundation for Software Measurements*) é importante para a engenharia de

software, devida a formalidade que oferece. Na verdade se torna núcleo dos processos de engenharia de software. A evolução da maturidade de software é centrada exatamente nesta capacidade de observar e registrar formalmente através de medidas, os produtos e processos de software.

3.5.1 Validação de métricas de software

A falta de confiança em medidas é um dos maiores problemas da aplicação industrial de medição de software, quer por falta de conhecimento, quer por falta de processos de validação mais aprofundada.

A validação de métricas pode ser discutida em termos de medidas, onde as entidades são caracterizadas através de números ou símbolos através de atributos específicos, ou em termos de sistemas de previsões que envolvem modelos matemáticos juntamente com procedimentos de previsão.

Assim, considerando as entidades de software como sendo produtos, processos e recursos, o quadro abaixo mostra uma organização de atividades sobre entidades e atributos para a produção das medidas de interesse. Veja a Tab.3.1

Atividade de medida	entidade	Exemplo de entidade	Atributo	Medida
Estimativa de custos	Processo	Ciclo de desenvolvimento da Especificação a operação	Custo	Previsão (Estimativa)
Modelo de Produtividade	Recurso	Pessoal	Produtividade	Valor medido
Modelo de Confiabilidade	Produto	Código executável	Confiabilidade operacional	Previsão (Estimativa)
Métrica de Complexidade	Produto	Listagem do código-fonte	Estruturação	Valor medido

Tab. 3.1 - Algumas atividades de medidas típicas de software.

Existem, basicamente, dois tipos de modelos para medidas de software. Um deles corresponde aos modelos de produto, processo e recurso que definem métricas sem ambiguidade. Estes modelos devem capturar os atributos a serem medidos. Por exemplo, um diagrama de fluxo de controle ressalta os atributos relacionado com a estrutura de controle dos programas e não os relacionados com as transformações de dados.

O outro está relacionada com o modelo que relaciona dois ou mais métricas numa fórmula matemática. A maioria de medidas indiretas usam este tipo de modelamento.

Considere, por exemplo, uma fórmula que calcule o número de páginas de código-fonte de um programa pode ser definido por

$m = x / a$, onde

x é uma medida de número de linhas de código (LOC) e ***a*** é uma constante. Se existe o programa então ***x*** é uma medida direta do mesmo e ***a*** depende do ambiente (ex. a=55), então ***m*** pode ser calculado e fornece um valor.

Por outro lado, se o objeto para o qual ***m*** está relacionado não existe e o objeto relacionado com ***x*** também não existe, e a constante ***a*** deva ser determinado por procedimento empírico, então ***m*** é um valor de previsão, ao invés de medida.

Definição 3.13: Um sistema de previsão consiste de um modelo matemático juntamente com um conjunto de procedimentos de previsão para determinação de variáveis e parâmetros desconhecidos.

Para ilustrar os aspectos relacionados com os modelos de previsão, discutamos dois exemplos clássicos.

Exemplo 1 - O modelo criado por Boehm, COCOMO, estabelece que o esforço (medido por ***E*** - pessoa.mês) necessário para o desenvolvimento de um sistema de software, com o tamanho ***S*** (medido em milhares de comandos-fonte) pode ser calculado por:

$$E = a \cdot S^b$$

onde ***a, b*** são parâmetros determinados pelo tipo de sistema de software a ser desenvolvido (BOEHM, 1981). Nesse caso a previsão de esforço realizada na fase

de especificação do sistema, necessita como pré-condição a determinação dos parâmetros ***a, b*** por previsão e também de ***S***, também por previsão.

Observe que neste exemplo é correto o modelo COCOMO. Muitas vezes se utiliza erradamente o termo método COCOMO (ou métrica), pois como pode ser visto, ela depende de procedimentos para estabelecer parâmetros que são dependentes de contexto.

Exemplo 2 - Os pontos de função de Albrecht (ALBRECHT, 1979) planejou medir o atributo de funcionalidade em documento de especificações, conforme percepção do usuário. A validação deste tipo de medida deve capturar pelo menos uma escala de ordem, ou seja, se uma especificação E1 apresenta mais funcionalidade que uma outra especificação E2, então a medida de FP (S1) é maior que FP(S2). Normalmente os pontos de função fazem parte de sistemas de previsões de custos e portanto validação de pontos de função refere-se a validação destes sistemas de previsão.

A validação de métricas, deve-se falar inicialmente sobre a validação de sistemas de previsão (estimativas).

Definição 3.14: A validação de um sistema de previsão em um dado ambiente é o processo de estabelecer a precisão do sistema de previsão utilizando meios

empíricos, através de comparação da performance do modelo com dados conhecidos no referido ambiente.

No modelo de cálculo de esforço do modelo de COCOMO, Boehm indica que a precisão de medida, dentro de certas circunstâncias chega a uma margem de ± 20%. Isto define uma faixa de aceitação no processo de validação (BOEHM, 1981).

Outro foco de validação é a de medidas.

Definição 3.15: Validação de uma medida de software é um processo de ceritificação de que a medida é um número que caracteriza adequadamente um atributo em questão, ou seja, mostrar a validade das condições de representação.

3.6 Integração de Processos e Medidas

Medida é essencial para entender, definir, gerenciar e controlar os processos de desenvolvimentos e de manutenção. Não é possível caracterizar os vários aspectos de desenvolvimentos de forma quantitativa sem ter um entendimento detalhado das atividades e os seus relacionamentos. As medidas ajudam a estabelecer métricas de produtividade e qualidade e também uma base para avaliar e comparar otimizações realizadas nos processos. Durante a manutenção, medidas pode refletir efeitos de mudanças em tamanhos, complexidade e manutenibilidade. As métricas também podem apoiar o planejamento, as projeções e previsões sobre futuros projetos,

com base em dados coletados de projetos passados. Ou seja, as medida são úteis para:

- Visibilidade dos processos e dos relacionamentos entre os produtos, recursos, tecnologias, métodos e padrões;
- Planejamento, monitoração e controle de projetos.

Trabalhos mostram que a adoção de processos de coleção e análise de dados aumentou em 7% do custo do projeto segundo relatórios emitidos pelo Laboratório de Engenharia de Software da NASA (Goddard Space Flight Center), descrito no trabalho de Card e Glass (CARD; GLASS, 1990). Adicionalmente, De Marco indica que há um aumento de 5 a 10% nos custos de desenvolvimento (DeMARCO, 1990). Ou seja, é preciso técnicas e ferramentas para apoiar a realização de medidas.

Lembrando que as métricas devem ser definidas com objetivos claros e operacionalmente viável de coletar e entender para suas medidas possam ser incorporadas como parte de desenvolvimento e manutenção de software, é preciso fazer um planejamento de medidas. O plano deve explicar o que deve ser medido, quando medir (no processo) e quem fará a coleta dos dados. Deve indicar ainda o porque cada medida é realizada. Esse plano de medida deve ser montada com base nos modelos de processo, combinada com as metas de desenvolvimento ou manutenção. O gerente de projeto deve então (MELTON, 1996):

- Decidir as metas mais importantes;
- Determinar quais questões devem ser realizadas e respondidas para auditar o cumprimentos das metas; e
- Medir itens que ajudam a responder estas questões.

O modelo de processo é usado para decidir se os itens necessários são visíveis o suficiente para ser medido adequadamente.

3.6.1 Análise dos níveis de maturidade do ponto de vista de métricas

Uma organização, com ausência de métodos, padrões, com ambientes de desenvolvimento diversificados, com projetos dependentes de pessoas que compõem as equipes não suporta um conjunto padronizado de medidas. Atrelar as necessidades de medição com os níveis de maturidade dos processos de software é uma estratégia mais viável para ser utilizada na indústria. Qualquer estratégia diferente disto é bastante contra-producente.

Nos trabalhos empreendidos pela Software Engineering Institute (SEI) foram definidos os níveis de maturidade (HUMPHREY (1989) E PAULK (1991)). Nestes níveis, um elemento discriminador dos níveis de maturidade é a quantidade de visibilidade que os desenvolvedores e gerentes têm em todo o processo de desenvolvimento. No nível mais baixo de maturidade, o processo todo não é bem conhecido ou entendido. À medida que a maturidade evolui, o processo é melhor entendido definido. Quanto maior a maturidade, ou seja, maior visibilidade do processo, maior é a oportunidade de medidas: um desenvolvedor pode medir

apenas aquilo que é visível no processo, e medidas ajudam a aumentar a visibilidade. O cinco níveis de maturidade da SEI fornece um contexto conveniente para determinar o que medir antes e como planejar um programa de aumento gradual de medidas.

Usando o critério de visibilidade, Pfleeger e Mcgowan (PFLEEGER; McGOWAN, 1990) examinaram cada nível, e em cada um eles tentaram estabelecer o que um processo típico pode visualizar naquele nível, do ponto de vista de medidas. A Tab.3.2 mostra uma visão geral dos tipos de métricas sugeridas para cada nível de maturidade.

MATURIDADE	CARACTERÍSTICAS	MÉTRICAS
1. Inicial	--	Baseline
2. Repetível	Processo dependente em indivíduos	Gerenciamento de projeto
3. Definido	Processo definido, institucionalizado	Produto
4. Gerenciado	Processo medido	Processo e Feedback para controle
5. Otimizante	Melhoria, com feedback para os processos	Processo e feedback para mudar os processos

Tab.3.2 - Visão geral do processo de Maturidade (Pfleeger e Mcgowan, 1990).

No nível inicial, acontece situações, projetos similares podem apresentar um características de qualidade e produtividade com grande variação de um projeto para outro, por falta de estrutura e controle adequados. Nesse caso, os gerentes tem metas para melhoria de qualidade e produtividade, porém eles não conhecem em que nível de qualidade e produtividade se encontram num determinado instante. Nesse nível de maturidade, recomenda-se que a medida de baseline seja

coletada para estabelecer o processo inicial de medidas e que possa ser incrementada a medida em que a maturidade evolui.

O segundo nível, chamado de repetível, identifica as entradas e saídas do processo, as restrições (prazos e financeiros), e recursos a serem usados para produzir o produto final. O processo é repetido da mesma maneira que uma sub-rotina é repetidamente executada: entradas apropriadas produzem saídas apropriadas, porém não há visibilidade como as saídas são produzidas. O diagrama SADT mostrada na Fig. 3.1. Observe que o que é visível está associada às setas que entram e sai do processo, ou seja métricas de entrada podem incluir tamanho e volatilidade de requisitos, a saída em termos de tamanho de sistema (física e funcional), os recursos como esforço de recursos humanos e as restrições em termos de custos e prazos.

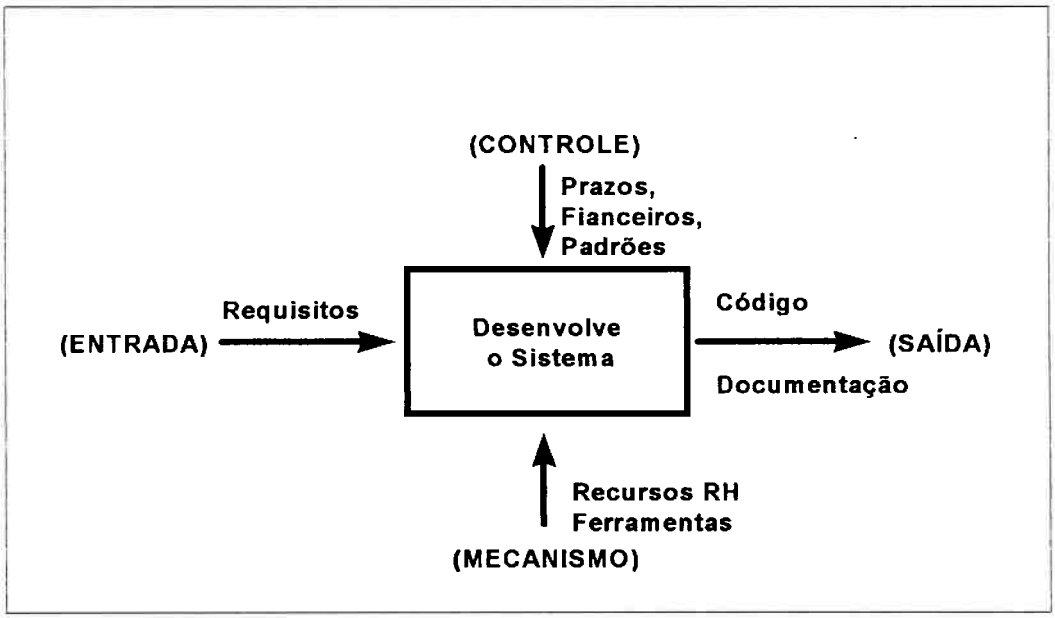

Fig.3.1 - Nível 2 - Processo repetitivo.

O nível 3 de maturidade difere do nível 2 porque oferece um uma visibilidade maior da caixa de desenvolvimento, oferecendo assim maior oportunidades de medidas. No nível 3 as atividades intermediárias são bem definidas e a nível de entradas e saídas, podendo ser examinadas e medidas considerando que os produtos intermediários são bem conhecidos.

Cada atividade é bem delineada uma da outra, num processo bastante definido, de modo que as medidas de produtos deve ser executada de acordo com a visibilidade dos processos. As métricas podem ser obtidas em produtos de desenvolvimento, em muitas situações, análise de medidas realizadas em etapas iniciais pode ser útil nas métricas dos produtos finais. Por exemplo, densidade de defeitos encontrados na fase de especificação pode influir na estratégia de desenvolvimento, exigindo uma postura gerencial para refazer e corrigir esta especificação. Esta ação normalmente acontece devido uma previsão de futuros problemas nos códigos gerados, considerando que corrigir erros em estágios anteriores do desenvolvimento sempre é mais barato (PRESSMAN, 1989). A Fig.3.2 mostra um exemplo de um processo que se encontra no nível 3, onde a visibilidade sugere mais métricas que os possíveis adotados no processo de nível repetitivo, esquematizado na Fig.3.1.

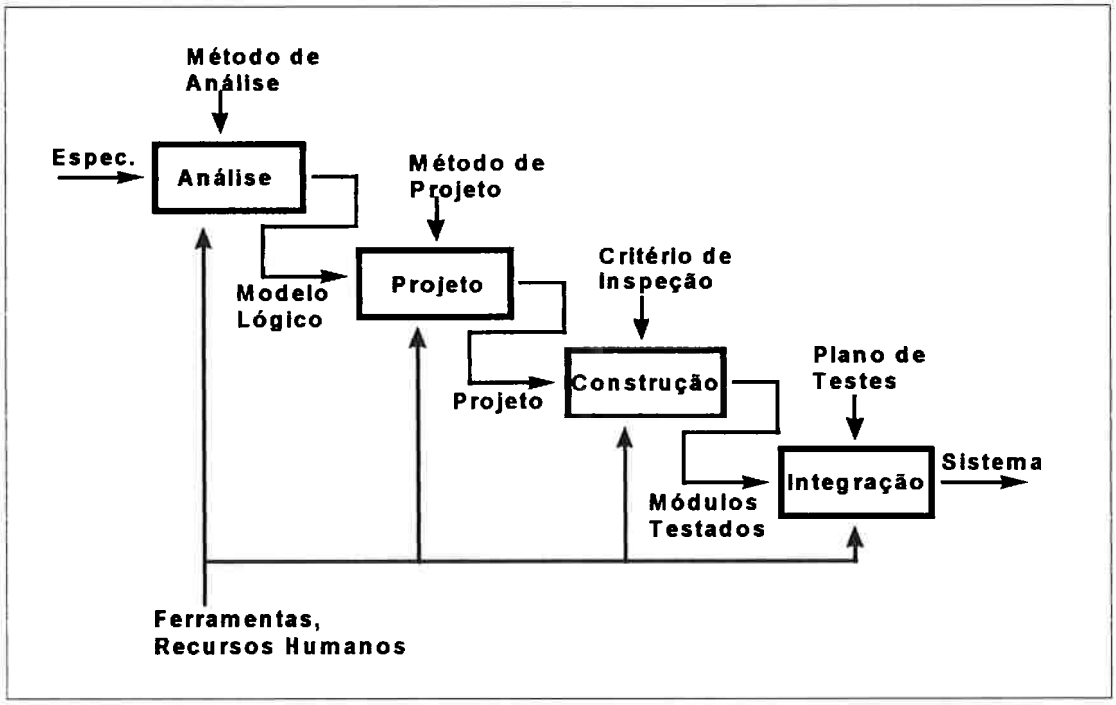

Fig. 3.2 - Um processo de nível 3.

No nível 4 (processo gerenciado), existe um mecanismo de monitoração para fins de gerenciamento. Neste caso, formalmente, existe uma mecanismo de realimentação de medidas obtidas das atividades de desenvolvimento anteriores que permitem identificar atividades críticas, atividades que pelo número de defeitos de seus produtos exigem retrabalho antes de prosseguir nas fases seguintes. As medidas observadas podem ainda ser comparadas com referências. Nessa situação, as medidas coletadas ajudam a estabilizar o processo, até que se atinja as metas de qualidades e produtividades desejadas.

A grande diferença entre os níveis 3 e 4 é que no nível 4 as métricas refletem características dos processos e das interações entre as grandes atividades do processo. Outro fator fundamental é o mecanismo de monitoração, que se apoia

numa base de dados de métricas capaz de fornecer informações sobre características de distribuição de defeitos, produtividade, eficiências das atividades, alocação de recursos de modo que os valores planejados e os esperados se aproximem bastante.

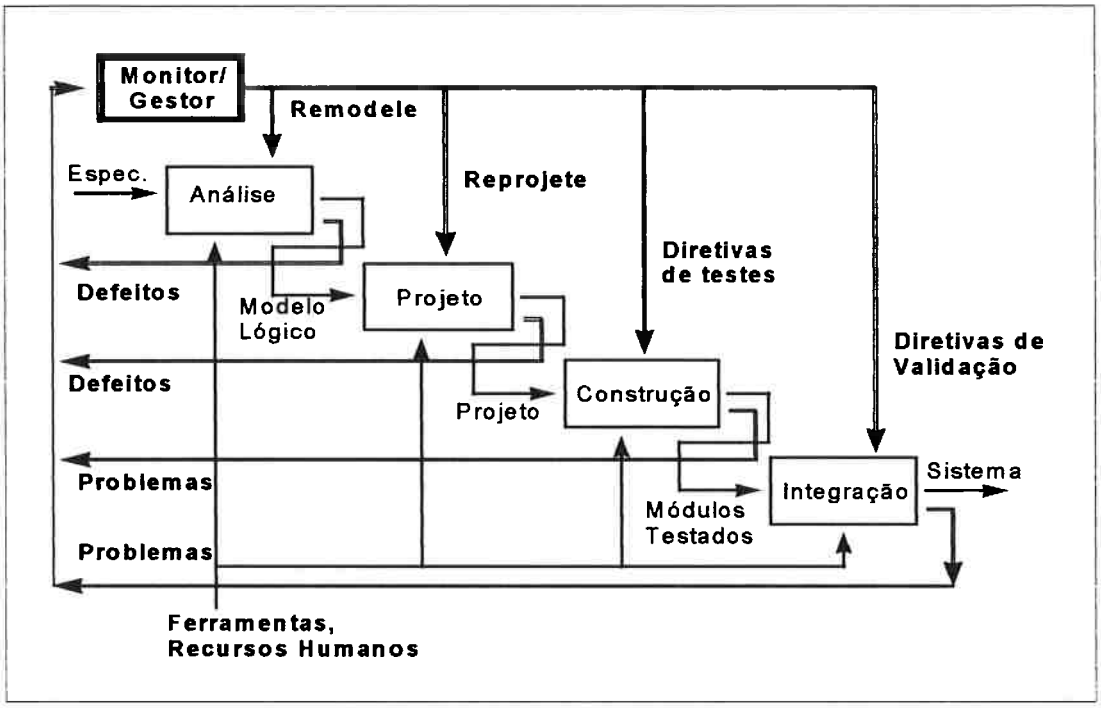

Fig.3.3 - Um processo de nível 4 - gerenciado (mecanismo de monitoração).

Um processo otimizante é o último nível de maturidade, onde as métricas são utilizadas para otimizar o processo, com inclusão e retirada de atividades se necessário. Ele permite uma alteração dinâmica do processo, de acordo com as métricas obtidas com o processo em pleno funcionamento.

Os processos indicados na Fig.3.4 como P0, P1, P2, ... Pn, indicam a dinâmica alteração sofrida por um processo. Por exemplo, suponha que um determinado

sistema tenha sido planejado para ser desenvolvido "waterfall" e no meio da análise de requisitos perceba-se uma especificação falha. Neste caso é preciso mudar o roteiro no processo de modo a implementar protótipos de especificação que facilite o levantamento, eliminando as incertezas, antes de prosseguir para a fase de projeto. Esta mudança, com reflexos em toda a organização do projeto, é efetuada de forma dinâmica e direta.

No nível 5, as métricas agem como sensores e monitores e os processos tem alta capacidade de realizar mudanças em respostas às sinalizações de alerta ou de otimizações. Assim, as cinco categorias de métricas citadas até aqui: baseline, gerenciamento de projetos, produto, processo para monitoração e processo para monitoração e controle estão encadeadas, ou seja, todas as métricas do nível 5 e dos níveis inferiores devem ser capturadas. É importante considerar que os níveis não são níveis discretos de classificação dos processos. Ela é uma graduação contínua, onde em grandes casos se verifica que determinadas partes do processo são mais ou menos maduros (ou visíveis) que os outros. Pode existir por exemplo, processos no nível repetitivo, onde as atividades intermediárias não são visíveis, porém a atividade de projeto é medido e gerenciado.

Rifkin e Cox (RIFKIN; COX, 1991) defendem o fato, na prática bastante adequado, de que o programa de medidas inicia modestamente, especializada em algum particular projeto. Lembram ainda que as medidas devem ser focados em

problemas críticos ou metas do projeto, com base na significância e na necessidades.

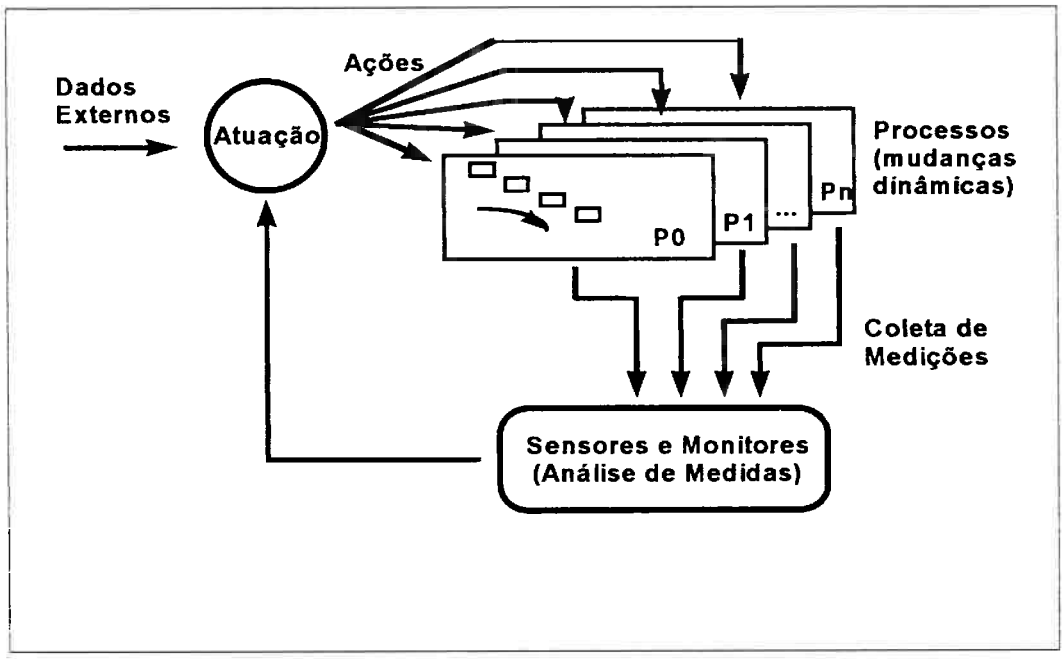

Fig.3.4 - Um processo no nível otimizante.

Entendido os mecanismos de medidas e da arquitetura que permite a sua incorporação nos processos, de acordo com o nível de maturidade do processo de obtenção de software, como analisar e decidir qual a métrica a ser utilizada? É o que se apresenta a seguir.

3.6.2 Metas, Questões e Métricas

O sucesso de um programa de medição depende da sua aplicação prática. A maioria de falhas ocorridas neste tipo de iniciativa é a dificuldade de realização prática das métrica e seu efetivo uso, conforme constata-se nas empresas que produzem sistemas de software (ARAKAKI, 1996a).

Um mecanismo simples e eficaz para estabelecer métricas é o que proposto por Basili e Rombach, denominada técnica GQM (Goals, Questions, Metric - Metas, Questões e Métricas), onde as métricas são planejadas, projetadas e implementadas a partir de um foco adequado e direto sobre as metas desejadas para o projeto de medição (BASILI; ROMBACH, 1988). Esta técnica envolve três passos:

- **Passo 1)** Listar as metas (diretrizes) para o projeto (de aquisição, desenvolvimento, ou manutenção);
- **Passo 2)** Derivar questões que ao serem respondidas indicam se as respectivas metas foram atingidas;
- **Passo 3)** Decida o que deve ser medido para responder às perguntas adequadamente;

A Fig.3.5 mostra um exemplo de execução dos três passos, onde a partir de uma meta pode-se chegar a um conjunto de métricas e de questões. A definição de métricas feita desta maneira é bastante prática pois tem como virtude o foco adequado que são as metas. Tudo deriva delas, possibilitando objetivos muito claros e também justificável nos vários aspectos, como por exemplo a relação custo/benefício.

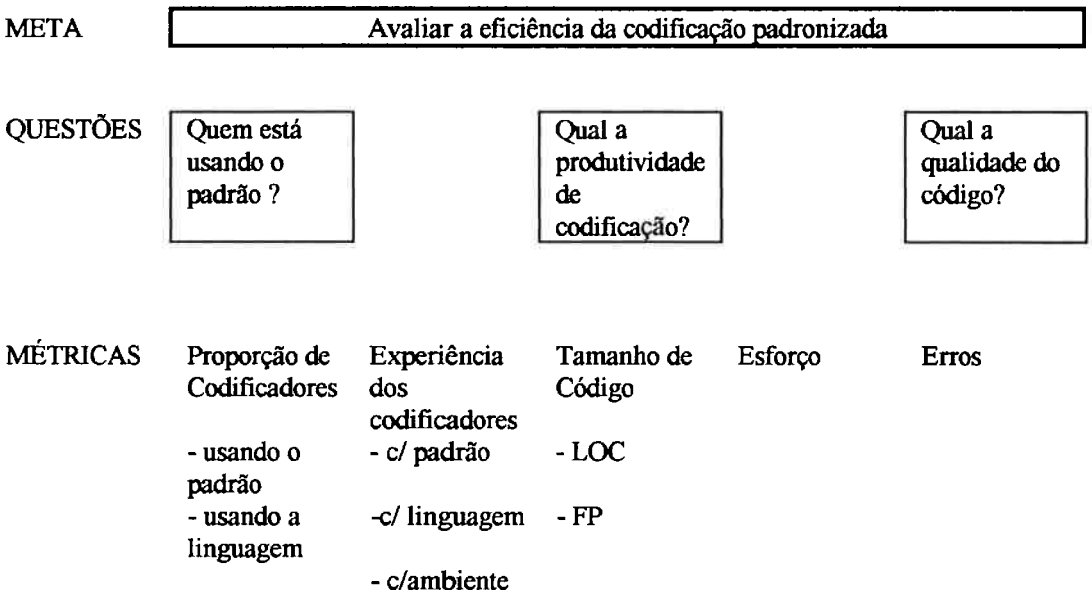

Fig.3.5 - Técnica GQM para seleção de métricas a partir de uma meta.

No exemplo esquematizado na Fig.3.5, a meta é "avaliar a eficácia de usar codificação padronizada". Para decidir se o padrão é eficiente, pode-se montar uma série de questionamentos. Por exemplo um gerente pode querer saber quantos desenvolvedores estão usando o padrão e quantos não estão, para medir qual a produtividade dos desenvolvedores. Pode querer avaliar qual a qualidade do código gerado com padronização.

Identificadas e elaboradas as questões, cada questão pode ser analisada para determinar a resposta da questão. Uma questão maior pode ser decomposta em questões menores mais diretas (ou explícitas). Por exemplo o questionamento "quem está usando o padrão?" pode ser subdividido em questões manores:

- Qual a proporção de codificadores usando o padrão?
- Existe diferenças entre o código produzido por codificadores experientes com o padrão e pelo produzido por iniciantes no padrão?
- Idem com relação a experiência na linguagem?
- Idem com relação a experiência no ambiente de trabalho?

Destas perguntas mais explícitas derivam as métricas, conforme atesta o exemplo esquematizado na Fig.3.5. Por exemplo a questão de produtividade pode ser respondida através de medidas de esforço por alguma medida de tamanho, por exemplo LOC ou FP, ou qualquer outra métrica, desde que útil para o projeto. Por exemplo de qualidade poderia ser a quantidade de erros, quantidades de cabeçalhos de comentários de módulos ou outros, desde que o gerente entenda como útil para o projeto. A técnica GQM fornece um conjunto de sugestão de medidas e fica de responsabilidade da liderança do projeto adequar as restrições de custos e prioridades para definir um planejamento de medidas úteis e possíveis de serem executadas dentro do orçamento do projeto.

Observe-se que a matriz de métricas apresenta uma característica interessante. Uma delas é que para responder a uma pergunta, várias métricas são necessárias. Em contrapartida, cada métrica pode ser usada para responder a mais de uma pergunta. Esta matriz indica como organizar a coleção de dados (métricas) e também fornece uma orientação de como usar estes dados (são as perguntas).

Esta técnica permite generalizar o suficiente conforme discute o trabalho de Austin Melton (em MELTON, 1996). Neste trabalho, o que chama a atenção é a associação dos níveis de maturidade com a técnica GQM utilizado pelo DoD (U.S. Department of Defense).

Como organizar um planejamento de medidas, uma vez estabelecida matriz de métricas usando o GQM? Ou melhor, ao se determinar o conjunto de métricas, responde-se às seguintes perguntas:

- O QUE medir?
- QUAIS métricas usar?
- POR QUE medir?

Se além destas perguntas, forem respondidas as seguintes perguntas, o plano estará completo:

- QUEM é responsável por coletar as dados?
- QUANDO os dados podem ser coletados?
- ONDE, no processo, serão coletados os dados?
- COMO os dados serão coletados?

Responder estas simples perguntas exige um subsídio forte, uma base, que é o processo de produção de software sobre o qual o projeto do software será conduzido. O gerente deve esquematizar o seu processo de software em fluxo de atividades e produtos de forma mais visível possível e sobre ele mapear as

atividades de medidas sobre os elementos do processo, de modo a responder as perguntas. Então resumindo, vale o seguite roteiro para a montagem de um plano de medição:

1. O modelo do processo deve incluir todas as grandes atividades que envolvem as medidas;
2. O modelo deve ser detalhado o suficiente para mostrar os produtos, recursos e os sub-processos que serão medidos;
3. O modelo deve explicitar as entradas e saídas de cada atividade do processo que sofrerão medições;
4. O modelo deve identificar claramente as restrições das atividades como ganhos esperados, gastos orçados e padrões;
5. O modelo deve identificar recursos que serão usados em cada atividade do processo tais como hardware, software e pessoas;

Consagrado o modelo do processo, deve-se mapear as métricas e determinar onde elas devem acontecer dentro do processo, em seguida o plano de medição deve estabelecer quem realiza as tarefas de coletas e registro de dados e como serão coletados, indicando a sistemática (se manual, ou com ferramentas). Este tipo de mapeamento também força uma revisão dos processos, uma vez que é importante tornar cada vez mais visíveis para facilitar (muitas vezes viabilizar) as medidas.

3.6.3 Métricas nos processos

O que fazer com os dados de métricas coletadas? Muitos programas de medidas falham porque nunca usam os dados. São montados extensos relatórios mas não se extrai nenhuma informação delas ou usada para avaliar o processo. Além dos tradicionais relatórios de análise, usando algumas técnicas estatísticas podem ser usadas, porém algumas técnicas de visualizar as métricas são sugeridas. Uma delas é a arvore de classificação descrita por Porter e Selby (em PORTER; SELBY, 1990), que permite visualizar analisar conjunto de dados por tendências. A grande vantagem desta forma de análise é de verificar o produto e o processo, através de várias audiências, cada um com a sua necessidade de análise (ex. Analistas, programadores, gerentes). Adicionalmente, suportar com facilidade novas configurações de analises sem alterar as existentes.

Outra técnica de análise e visualização de métrica é a descrita por Pfleeger (em PFLEEGER, 1992) que é uma variação do diagrama de Kiviat adaptado para visualizar múltiplas métricas num mesmo gráfico, usando como base um gráfico de pizza. Ele relaciona num mesmo gráfico, a importância relativa de diversas métricas sobre o processo, permite uma quantificação global das métricas obtidas contra as metas estabelecidas e uma quantificação de uma métrica em especial contra a sua meta. Assim, um gerente pode verificar o desempenho do processo e do produto de uma forma global, assim como os analistas podem analisar um produto do processo em particular e verificar o que precisa melhorar. Nesta técnica comparar projetos significa sobrepor estes gráficos.

4. CARACTERIZAÇÃO DE SISTEMAS ORIENTADOS A OBJETOS

"Object-Oriented is a way of thinking, not a just programming language."

Brian Hederson-Sellers, 1997.

Objetivo do Capítulo:

Apresentar uma conceituação básica sobre a tecnologia orientada a objetos, o seu estado da arte no mercado e a sua aplicação na indústria financeira; e

Discutir o uso de métricas para avaliação de processos de desenvolvimento de sistemas que usam esta tecnologia.

Tópicos do Capítulo:

4.1 Conceitos Básicos da Tecnologia Orientada a Objetos

Este capítulo procura apresentar conceitos básicos necessários ao detalhamento dos assuntos da tese, não sendo escopo do mesmo apresentar toda a conceituação a respeito do assunto, nem tampouco mostrar todas as metodologias disponíveis no mercado. Para isso, os conceitos apresentados são essenciais para a discussão da qualidade de software orientado a objetos.

4.1.1 Um paradigma de desenvolvimento de sistemas

Entenda-se aqui, paradigma de desenvolvimento como sendo um modelo de desenvolvimento baseado num conjunto de conceitos, ideias e ferramentas para organizar o processo de conceber, organizar, projetar e implementar um software. Dentro deste ponto de vista, a tecnologia objetos constitui um paradigma de desenvolvimento.

Orientação a objetos não é dependente de uma particular linguagem de programação. Ela define uma maneira diferente de organizar o pensamento. Nos dias atuais existem linguagens que suportam bem o tipo de abstração conceitual incorporado pela tecnologia. Independentemente de qualquer definição de notação, é importante a apresentação dos conceitos relacionados com a análise e projetos para o aprofundamento das ideias proposta nesta tese.

Neste paradigma, o menor dos elementos (objeto) de um sistema de software é autônomo e é constituído de uma estrutura de dados e procedimentos na forma de um bloco compacto, independente, com uma interface de acesso clara, coesa e bem definida. Esses blocos são conhecidos ou utilizados dentro da organização

de um sistema de software pela responsabilidades ou serviços que oferecem. Essas características propiciam mais flexibilidade, mais facilidades de manutenção, e incentiva a reutilização que é capaz de diminuir tempos de desenvolvimento e conferir mais robustez aos sistemas através do uso de elementos disponíveis (e portanto mais estáveis, ou maduros).

Um sistema com organização orientado a objetos é composto de um conjunto de objetos que interagem entre si. O sistema então é definido como uma topologia de objetos interligados, interagindo através de mensagens, que passam informações ou acionam procedimentos disponíveis nos objetos. Dentro deste paradigma, a organização do software é detalhada na forma de diagrama de objetos (ou classes), hierarquias de herança e de agregação, diagramas de interação entre objetos, especificação de classes e diagramas de estados. Estes conceitos são descritos nos itens que se seguem.

4.1.2 O três conceitos básicos

Orientação a objetos considera três conceitos fundamentais que utilizados de forma integrada compõem a base da tecnologia. São eles:

- ◊ Identidade e modularidade;
- ◊ Abstração e classificação; e
- ◊ Polimorfismo e herança.

Identidade e modularidade

Os conceitos de encapsulamento, privacidade de informações e modularização não são novos, porém a tecnologia objetos utiliza esse conceitos de forma

bastante forte de modo que eles contribuem bastante na organização de um sistema. Encapsular e modularizar permitem constituir os objetos como verdadeiras caixas pretas, isto é, somente as interfaces de acesso são visíveis. O seu interior que armazena informações e o método são implementados e os seus procedimentos internos ficam invisíveis para o restante do sistema. Isto é fundamental para o modelamento objetos, tanto na análise quanto no projeto, pois a o software é organizado com o foco na visão externa (pública) dos objetos.

Identidade relaciona o fato dos objetos serem elementos independentes e identificáveis. Por exemplo, o objeto conta-corrente tem identidade, considerando que cada conta apresenta atributos de identificação como número, nome do cliente e agência.

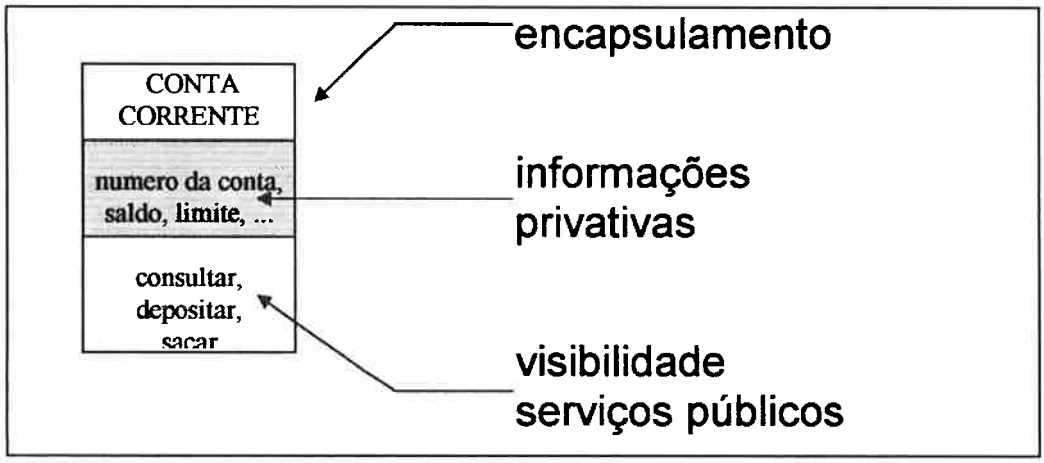

Fig.4.1 - Exemplo de Identidade e Modularidade numa conta corrente.

Abstração e Classificação

Abstrair corresponde ao levantamento e à identificação das características dos objetos, relevantes ao contexto da aplicação. Classificar é agrupar conceitos

comuns, de acordo com as informações tratadas, ou operações executadas. Por exemplo uma cidade: arquiteto planejador urbano pode visualizá-la como uma composição de ruas, pontes, quadras, bairros e regiões. Já um engenheiro construtor de casas visualiza a cidade como um conjunto de prédios e casas e a sua estruturação interna de projeto. Um proprietário pode visualizar as casas e apartamentos e suas constituição interna nos aspectos de moradia como sala, quarto e outras dependências.

Outro fator importante da classificação é a possibilidade de se agrupar as classes de objetos através da especialização e da generalização. Com isso os objetos podem ser refinados, sempre com base nos objetos do mundo real. Por exemplo pode-se classificar os membros da classe mamífero em cachorro e não cachorro (Fig. 4.2). As classes cachorro e não cachorro podem apresentar refinamentos através de um particionamento mais detalhado para a classe cachorro.

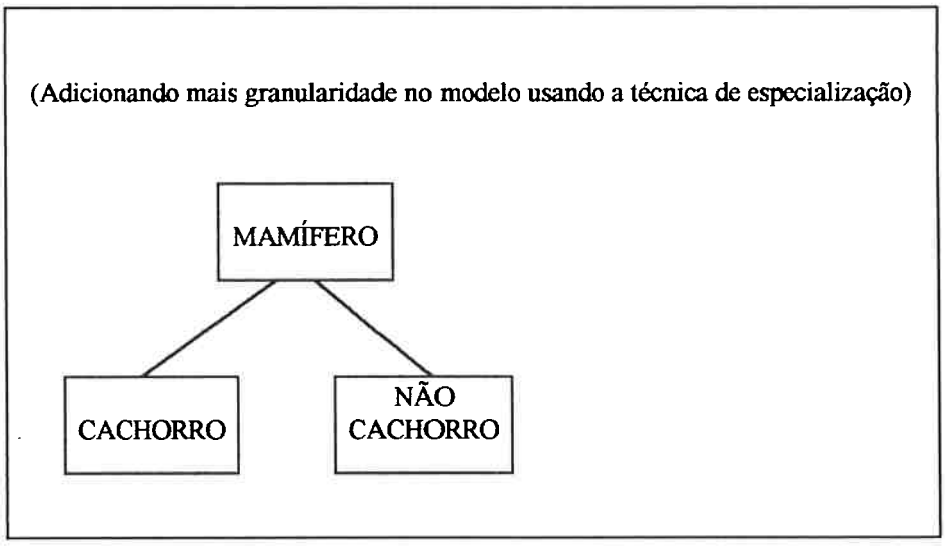

Fig. 4.2 - Enriquecendo a classificação usando a especialização.

Outra possibilidade para refinar mais ainda o modelo é a técnica de generalização, que organiza classes através de criação de classes de mais alto nível capaz de combinar conceitos e características comuns das classes de mais baixo nível. Veja a Fig. 4.3 que esquematiza um exemplo de generalização onde as classes cachorro e ave são agrupadas através da identificação de uma classe mais genérica ANIMAL que contém as caracterizações comuns das duas classes MAMÍFERO e AVE.

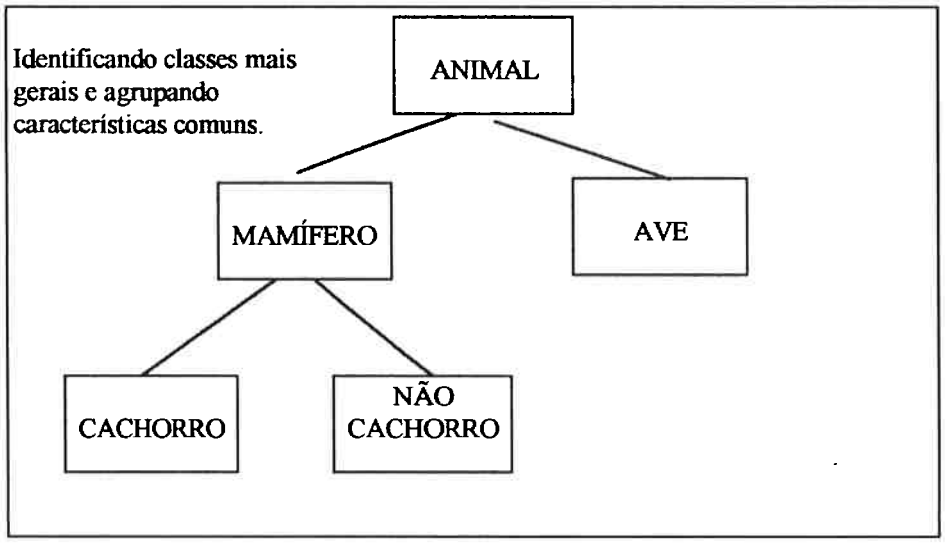

Fig. 4.3 - Criação de classe mais genérica através da generalização.

Polimorfismo e herança

Polimorfismo e herança completam a base de apoio da tecnologia orientada a objetos. O polimorfismo indica várias formas. Em orientação a objetos, o polimorfismo está diretamente associado aos conceitos de herança. Por exemplo, imagine-se a seguinte hierarquia de classes de tipos de arquivos de dados, conforme esquematizado na Fig.4.4. Observe que a classe básica arquivo é especializado em arquivo do tipo binário, do tipo ASCII. Assim, essas classes

podem atender um mensagem *imprimir*, porém, cada uma com método próprio de tratamento de dados para o tipo de arquivo a que se referem. A mensagem imprimir é dita polimórfica, isto é, uma classe externa pode solicitar o serviço de impressão através de uma mensagem única para a classe arquivo, que por polimorfismo aciona o algoritmo de impressão adequada para o tipo de arquivo solicitado.

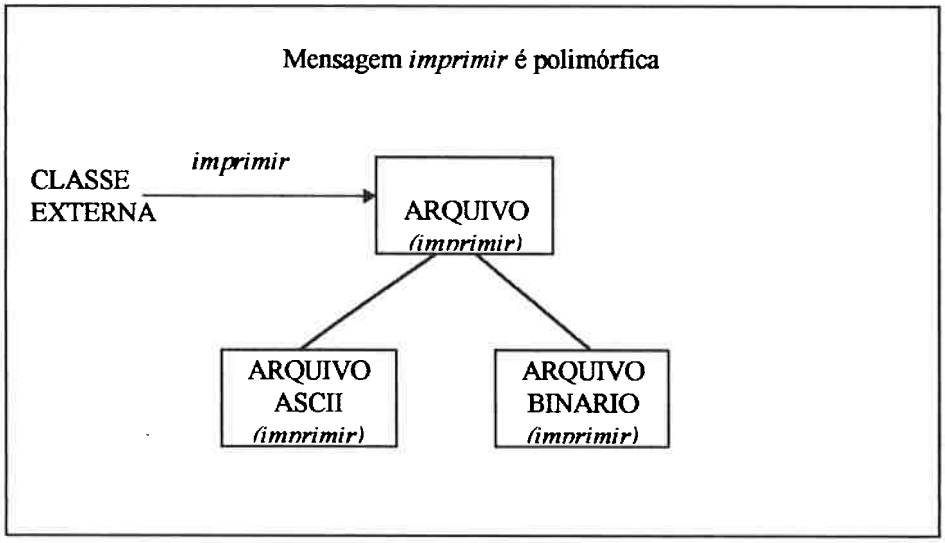

Fig.4.4 - O serviço de impressão de cada classe é implementada diferentemente e acessada de forma única através de uma mensagem imprimir.

A herança acontece em classes relacionadas e obtidas através de processos de especialização ou generalização. A Fig.4.3 mostra uma relação de hierarquia entre as classes ANIMAL, MAMÍFERO e AVE. Pode se dizer que as classes MAMÍFERO e AVE herdam características da classe básica ANIMAL. Por exemplo, os animais apresentam características comuns como se alimentar, se procriar. Assim essas características são herdadas pelas classes MAMÍFERO e AVE. Porém cada uma delas apresenta a suas próprias características como a de

apresentar penas e voar para a classe AVE, enquanto que os mamíferos têm a característica de mamar.

Outro exemplo de relação de hierarquia é da área bancária. Como relacionar as classes CONTA BANCÁRIA, CONTA CORRENTE, CONTA POUPANÇA e INVESTIMENTOS. Veja a Fig. 4.5.

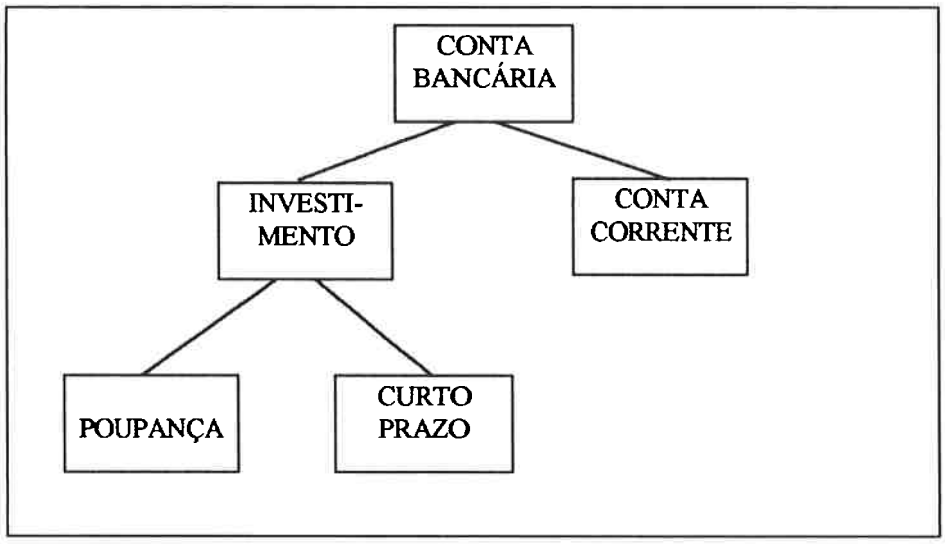

Fig.4.5 - Exemplo de relação de herança entre classes de um sistema bancário.

As classes INVESTIMENTO e CONTA-CORRENTE são dois tipos de contas possíveis de uma conta bancária. Na relação de herança, essas classes herdam as propriedades da classe-pai, CONTA-BANCÁRIA. Elas podem ser vistas como descendente da classe-pai. Essa relação pode ser interpretada como sendo "PODE SER UM DOS TIPOS". Dessa maneira, CONTA-CORRENTE herda todas as características da classe-pai, ou seja, é um tipo da classe CONTA-BANCÁRIA. Além disso, a classe-filha CONTA-CORRENTE pode apresentar propriedades especializadas de uma conta corrente que não existem na classe-pai,

como por exemplo, permitir que se use o cartão para retirada de dinheiro, através de um equipamento de um auto-atendimento. Diz-se então que CONTA-CORRENTE é uma especialização da classe CONTA-BANCÁRIA.

Ainda no exemplo de relação de herança entre classes de um modelo para um sistema bancário (Fig. 4.5), a classe INVESTIMENTO que é um tipo especializado da classe CONTA-BANCÁRIA é especializada no modelo em dois outros tipos: a classe POUPANÇA e a classe CURTO-PRAZO. As duas refletem tipos de investimentos, porém com especializações diferentes. Por exemplo, a poupança apresenta regras de datas de depósitos, retiradas e taxas de juros diferentes da classe CURTO-PRAZO. Assim a classe INVESTIMENTO foi especializada em duas classes com diferenças claras de regras de negócio, sem deixar de serem investimentos, isto é, herdam regras de investimentos do banco, porém com regras diferenciadas que justificam a separação em dois tipos.

Sobre o modelo da Fig.4.5, podemos supor dois de muitas possibilidade de manutenção de um sistema de automação baseado neste modelo. Um deles seria o caso em que uma transação de poupança apresente um problema. Nesse caso, o rastramento do problema ficará mapeada para as classes que representam o modelamento da transação poupança, que é a classe POUPANÇA.

Suponha o caso em que o banco crie um novo produto de investimento, especial para o período de férias. Por exemplo, "super-investimento-férias". Neste caso, um modelamento possível será o de criar mais uma classe especializada SUPER-FÉRIAS, abaixo da classe INVESTIMENTO (Fig. 4.6). Do ponto de vista de

programação, escrever uma especialização pode significar a escrita de algumas linhas de código que represente a especialização e aproveitando por herança todos as funções básicas como extrato, transferências, saldo, e outros. Essa é a maior contribuição da tecnologia, no que se refere ao aspecto da reutilização, que neste exemplo acontececeu desde a especificação de requisitos até a programação.

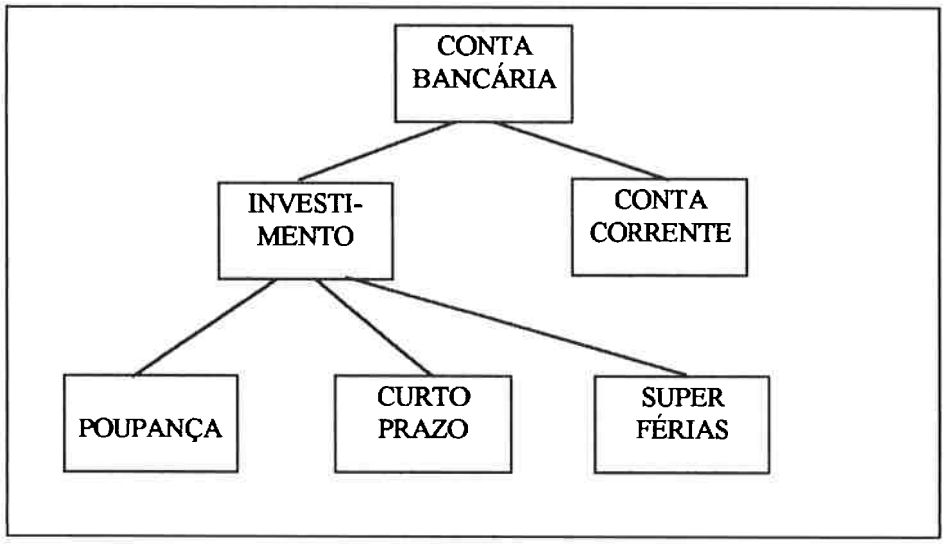

Fig.4.6 - Criando um novo produto SUPER-FÉRIAS.

4.1.5 Programação orientada a objetos

A programação orientada a objetos baseia-se em instância de objetos, classes, mensagens, herança e polimorfismo. Instância de objetos são ocorrências de seus templates que são as classes que apresentam a definição dos atributos (dados) e as operações (funções/procedimentos). As mensagens são a maneira de se enviar solicitações (num esquema de relação cliente-fornecedor) de serviços de um objeto para outro. É o equivalente ao acionamento de funções e rotinas dentro do paradigma procedural.

Conceitos como classificação, abstração e encapsulamento não são exclusivos da orientação a objetos. Ele aparecem de alguma forma nas linguagens mais antigas. Classificação e abstração aparecem no recurso de *structure* da linguagem C e *record* da linguagem Cobol. Encapsulamento aparece fortemente nas linguagens Modula 2 e ADA. Já a herança aparece somente nas linguagens orientadas a objetos, ela tem sido o recurso de diferenciação das linguagens orientadas a objetos da outras. Porém é importante lembrar que a herança não é questão única do paradigma orientada a objetos. Conceitos importantes do paradigma estão relacionados com abstrações de comportamentos autônomos e encapsulamento de informações.

Utilizar estruturas de heranças é uma das muitas maneiras de prover reutilização, extensibilidade e baixo custo de manutenção, que são preocupações constantes da engenharia de software. A herança pode ser vista como uma maneira de se compartilhar trechos de código. Por exemplo, suponha-se que já esteja modelado a classe animal. Ao se especializar um animal na classe gato, não preciso reescrever o código referente às operações básicas do gato como comer, andar e dormir. Aproveita-se o código existente, aproveitando por herança as características de animal e somente se codifica a parte especializada do gato, como miar, saltar com agilidade e outras especialidades do animal gato.

Outro exemplo, considere uma especialização da conte corrente de um banco. Suponha-se a criação da conta corrente especial, que é caracterizado com um limite de saldo que o correntista pode se beneficiar nas operações de saque de dinheiro. Nesse caso, as operações de extrato e depósito se mantém, bastando

para tanto herdar da classe conta corrente o código existente utlizando os recursos de herança. E daí, devem ser especializadas as operações de saldo e saque de modo a considerar os valores de limites da conta corrente.

4.1.6 Notação e o ciclo de vida

A técnica orientada a objeto apresenta uma característica bastante importante e diferente do paradigma procedural. Na orientação a objetos é muito difícil diferenciar a fase de análise, de projeto e de implementação. Por isso, diz-se que essas fases de desenvolvimento no ciclo de vida de software orientado a objetos são ditas transições sem perdas. Isso significa que os objetos identificados durante a análise aparecem na fase de projeto e também na fase de codificação das classes. Dessa maneira a auditibilidade para a equipe de desenvolvimento como também para os usuários/clientes. O usuário deve visualizar as características do problema original no projeto e na implementação final, diferente do paradigma procedural que utiliza notações intermediárias como DFD (Diagrama de Fluxo de Dados), DER (Diagrama de Entidade-Relacionamento) e DHP (Diagrama Hierárquico de Projeto).

4.1.7 Nomenclatura de objetos e classes

Considerando que se usa notação uniforme em todo o ciclo de desenvolvimento de um software vale resumir as terminologias e conceitos que se utilizam nas fases de desenvolvimento do software.

CICLO DE VIDA	NOMENCLATURA	EXEMPLO
Mundo Real	Objetos e tipos de objetos	Conta corrente, Poupança
Análise	Classes/Objetos que representam o processo de negócio	Modelo de conta-corrente e poupança
Projeto/Codificação	Classes detalhadas pelos atributos e interfaces públicas	Conta-corrente e poupança modeladas através de atributos e serviços/operações
Execução (em Produção)	Instância de Objetos	Conta-corrente específica, pertencente a uma pessoa; Idem para uma conta de poupança.

Tab. 4.1 - Apresenta a terminologia OO usada durante o ciclo de desenvolvimento de um software.

Observe que na fase de análise, as classes (objetos) modelam os objetos do mundo real. Por exemplo, a classe conta-corrente modela, através de atributo e operações, todas as conta-correntes do banco. Nessa fase, o modelo mapeia diretamente os objetos do processo de negócio referente ao mundo real, operação bancária. Já durante a fase de projeto e codificação, esse objetos identificados na fase de análise são detalhados como pequenos elementos "caixa-preta", ou seja como protocolos de acesso bem definidos que são as operações acessíveis através de mensagens. Detalha-se ainda a estruturação de seus atributos internos. Na fase execução de um software OO, as classes/objetos aparecem na forma de instâncias: as efetivas conta-correntes dos diferentes clientes do banco, porém com características idênticas modeladas pela correspondente classe.

4.1.8 Interação entre os objetos

Os objetos se interagem na forma de cliente-fornecedor. Na modelagem dos objetos, utilizam as seguintes formas de relacionamento: AGREGAÇÃO, HERANÇA e ASSOCIAÇÃO. Agregação representa a relação "consiste de". Uma sala é constituída de quatro paredes, portas e janela. Herança representa a relação "é um tipo de" (*is-a*). Uma pessoa pode ser do tipo criança ou adulta. Associação representa um objeto utilizando os serviços de um outro. Um objeto cliente de um banco pode acessar os serviços dos objetos do banco.

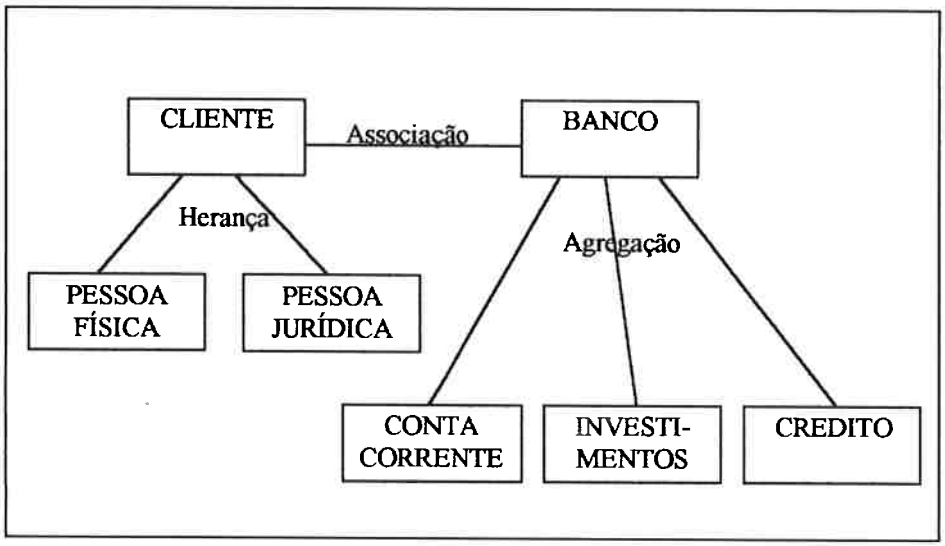

Fig.4.7 - Exemplo contendo as relações de herança, agregação e associação.

Observe na Fig.4.7, a relação de associação entre o objeto cliente e objeto do banco, que modela o relacionamento do cliente com os produtos do banco. Além disso, a relação de herança aparece para classificar os tipos especializados para os tipos de cliente pessoa fisica ou jurídica (empresas). Já a agregação aparece no modelo para indicar que o banco é constituído de produtos como conta-corrente, investimentos e créditos.

4.1.9 Mudança na forma de pensar

A forma de conceber sistemas nos paradigmas procedural ou objeto é bastante diferente. A Tab.4.2 resume este fato. Enquanto que no paradigma procedural modela-se as funções a serem executadas pelo sistema, no objeto procura-se a identificação de objetos dos quais é composto o sistema. Utiliza-se ainda técnicas de abstração para identificar e separar as responsabilidades e autonomias desses objetos. O detalhamento do sistema busca detalhar de forma sistemática e intensiva as abstrações dos objetos com os seus atributos e operações, sem entrar em detalhes de implementação de algoritmo. Essa estratégia é bastante diferente do procedural, onde tudo é raciocinado e orientado em termos de como implementar os algoritmos. Pode-se fazer um paralelo forte com o processo de projeto de um microcomputador. Por exemplo, o fabricante de computadores no projeto de um microcomputador está preocupado com a interligação dos objetos CPU, memória, controladores de discos e elementos de interface, sem se preocupar com detalhamento de como são implementados internamente cada um desses objetos. A preocupação maior é interligar esses componentes respeitando de forma disciplinada os protocolos de interação desses objetos. Fica para o fabricante dos chips os detalhes internos de organização das centenas de milhares de transistores que implementam o referido componente.

Organizar software orientado a objeto é assim. Privilegia-se a organização do sistema como uma interação de objetos, cada um com responsabilidades, interfaces e autonomias bem definidas. O detalhamento dos algoritmos de cada classe fica postergada até no momento da efetiva implementação. A Tab.4.2

mostra os principais pontos de raciocínio conduzidos pelos paradigmas procedural e orientado a objetos.

Paradigma Orientado a Objetos	Paradigma Procedural
O sistema é composto de quais objetos ?	O que faz o sistema ?
Como organizar dinamicamente o sistema para que os objetos, com os seus comportamentos produzam os resultados esperados?	Qual é a finalidade do sistema? Como codificar esse sistema para realizar essa funções?
Os algorítmos são abstraídos e postergados.	O FOCO é nos algorítmos

Tab.4.2 - Diferenças de postura mental nos paradigmas procedural e orientado a objetos.

4.2 A Engenharia de Software e a Tecnologia Orientada a Objetos

Neste item do capítulo discute-se aspectos da engenharia de software relacionados com a organização orientada a objetos de um sistema de software, especialmente nos aspectos de qualidade e produtividade. Discute por exemplo, o impacto no uso dessa tecnologia com relação à manutenção corretiva e evolutiva.

4.2.1 A tecnologia orientada a objetos e a engenharia de software

Para discutir aspectos da engenharia de software, Henderson-Sellers (HEDERSON-SELLERS, 1995) apresenta em seu trabalho uma organização dos itens de qualidade e o correspondente mapeamento dos recursos oferecidos pela metodologia, conforme mostrado na Tab.4.3.

Tab.4.3 - Mapeamento dos requisitos de qualidade em engenharia de Software no paradigma OO.

REQUISITOS DE QUALIDADE	DESCRIÇÃO	RECURSOS DA TECNOLOGIA OO
Correção/Verificação	Está associado com a operação correta do software.	Obtido pela certificação dos objetos.
Robusto	O sistema deve ser robusto para situações de exceções.	Modularidade, encapsulamento e Abstração.
Extensível	O sistema deve suportar alterações ou evoluções, de acordo com as necessidades de negócios	"
Reusável	Reusabilidade de códigos e de projetos ajudam na qualidade e produtividade.	"
Integridade	Software deve manter integridade contra acessos não autorizados. Não se permite alterações acidentais.	"

Compatível	Compatibilidade com os sistemas existentes	--
Eficiente	Eficiência no desempenho, administração do projeto, depuração de códigos, entre outros.	--
Portável	Portabilidade do sistema entre ambientes e linguagem é importante para o software.	Abstração, Coesão semântica e Modelamento do mundo real.
Facilidade de uso	Facilidades de uso dos recursos do ambiente de desenvolvimento como CASE e os compiladores.	"
Manutenção	Estatísticas mostram o custo de 70% está mapeado para as manutenções.	"

Tab.4.3 (continuação) - Mapeamento dos requisitos de qualidade em engenharia de Software no paradigma OO.

4.2.2 Modularização

Modularidade não é um conceito exclusivo da Orientação a Objetos. Com esse conceito, o sistema é organizado em unidades menores e autônomas de código.

Objetos contemplam este conceito, pois se caracterizam por serem pequenos blocos construtivos compostos de dados e funções relativos a uma particular especificação. Em algumas linguagens, os módulos contém uma classe, em outras, uma classe pode ser dividida em mais de um módulo e em outras, um módulo pode conter mais de uma classe. A grande vantagem é que os módulos de objetos podem agrupar dados e funções de forma coesa do ponto de vista semântico, e se tornarem efetivos blocos construtivos de software, modulares, robustos (testados) e reutilizáveis em várias situações.

Detalhar o sistema como coleção de objetos é promover a modularização, através de classes, como resultado do uso da técnica, considerando que os objetos são pequenos e autônomos capazes de lidar com informações e serviços/operações específicas.

4.2.3 Privacidade de Informações

O encapsulamento e invisibilidade de informações também não exclusivo da orientação a objetos, porém igual a modularidade, facilita a utilização deste conceito: um objeto é definido como um agrupamento de dados e funções, onde os dados podem ficar totalmente invisíveis e o acesso ao objeto são possibilitados através de mensagens recebidas que acionam as suas operações.

Dessa maneira, as interfaces através de mensagens permite que os objetos interajam entre si, porém com um baixo acoplamento. Não há o perigo, por exemplo, de modificações acidentais de dados de um objeto por um outro, como acontecem com os dados globais implementados por linguagens como C e

FORTRAN. Nos objetos, se por acaso um dado for alterado de forma errada é possível se mapear de forma direta a porção de código responsável pelo erro, através da rápida identificação do objeto responsável por manter o referido dado.

As técnicas objetos permitem a definição do que é público e do que é privado para um objeto. As práticas de orientação a objetos indicam que o que se deve deixar públicas são algumas operações, de modo que todos os objetos externos solicitem os seus serviços através de mensagens. Dessa maneira, as demais partes do software não podem interferir na alteração dos dados. O baixo desacoplamento entre os objetos é mais um fator que permite o reuso, devida a autonomia que os caracterizam.

4.2.4 Produtividade

A produtividade na obtenção de software corresponde à produção de software no menor tempo possível, com recursos adequados e com custos aceitáveis, desde o desenvolvimento até o período de manutenção, onde o software fica em operação. A seguir relacionamos duas estratégias possíveis para obter o aumento da produtividade são dois: Um deles está associado ao de reutilização de software, onde a base do reuso é a utilização de objetos prontos, testados e robustos catalogados em bibliotecas de componentes. Os resultados obtidos com essa estratégia é de mais longo prazo, porém garante aumento de produtividade, pois o catalogo de objetos aumentam sistematicamente, melhorando o grau de reutilização. Quanto maior a reutilização, maior o nível de produtividade, pois

quanto maior o número de objetos prontos, menor é o tempo necessário para a preparação dos sistemas.

A outra estratégia é aquela obtida com a simples conversão de paradigmas procedural para o orientado a objetos. Nesse caso, com o maior esforço investido na análise, provocado pelo uso da técnica, e com a quebra do sistema em objetos (modularização) e as características de encapsulamento promovidas com o uso das técnicas orientada a objetos provocam um ganho substancial nas fases de implementação, através da redução dos tempos de testes. Provocam ainda um ganho de tempos na manutenção, pois devido ao baixo acoplamento entre os objetos, as correções de erros nos componentes produzem baixa perturbação no sistema.

4.2.5 Dependência de dados

A dependência de dados também um fator bastante considerado pela engenharia de software, especialmente no que se refere ao gerenciamento de base de dados. A forma como os dados são organizados e acessados num sistema orientado a objetos é melhor do que a forma tradicional, pois cada objeto tem a responsabilidade de cuidar, de forma autônoma, as informações a que se propõe, diferentemente da organização estruturada, onde o acesso aos dados ficam distribuídos em várias partes do sistema. Em especial, no caso da tecnologia orientada a objetos, esses dados são invisíveis para o restante do sistema através da técnica de encapsulamento. Para ilustrar essa dependência, tomemos um

exemplo, onde algumas transações bancárias são organizadas usando a técnica estruturada e a técnica orientada a objetos.

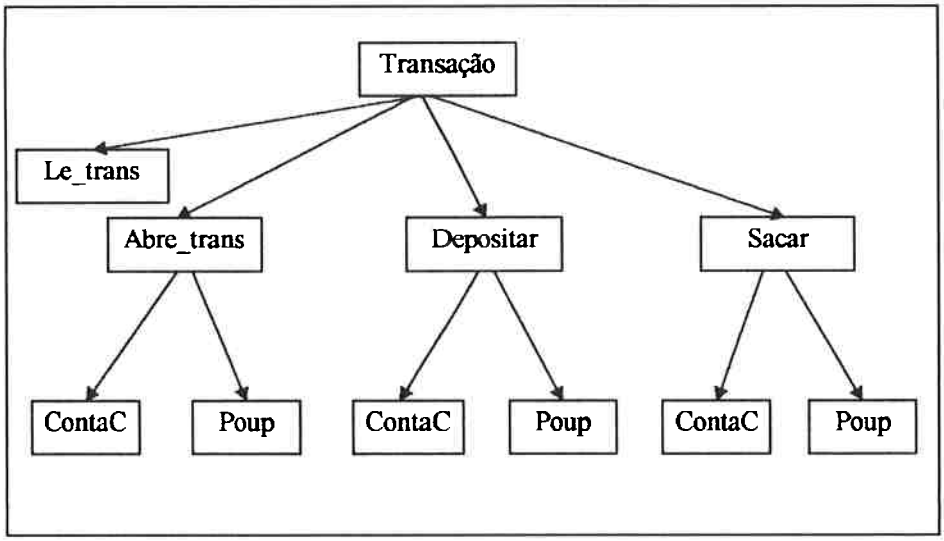

Fig.4.8 - Transações de saque e depósito bancário, organizado na forma procedural.

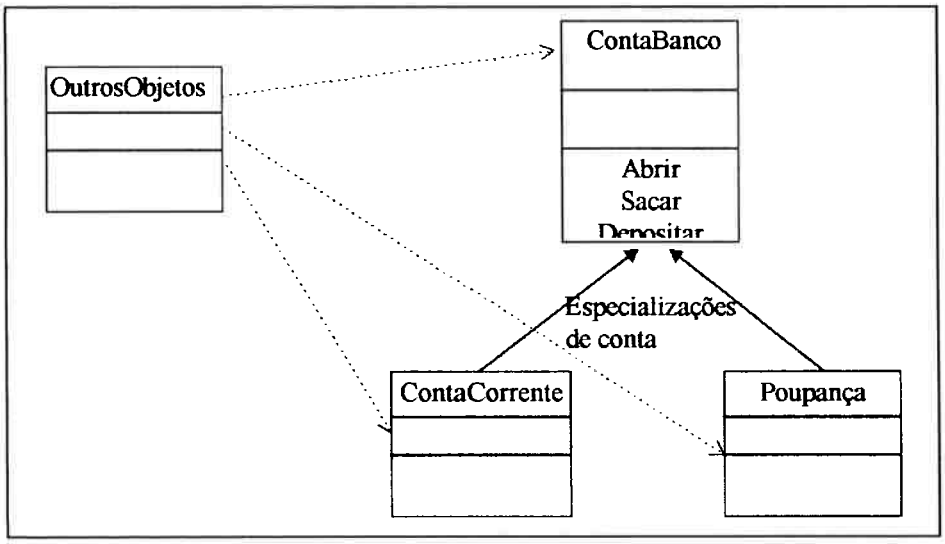

Fig. 4.9 - Transações bancárias de conta-corrente e poupança organizado em objetos.

A Fig.4.8 esquematiza uma estruturação possível de transações bancárias de conta corrente e poupança, através de diagramas hierárquicos de módulos

separados por funções: Um deles recebe e interpreta uma transação solicitada para saber se é saque ou depósito em conta corrente ou poupança. O outro abre a transação, cuidando para categorizar o tipo, se conta corrente (ContaC) ou poupança (Poup). Uma vez identificada a transação, ela é efetivada pelo módulo correspondente. Observe-se que a realização de uma transação exige a execução de partes de vários módulos até a sua execução total.

Diferentemente da procedural, na organização orientada a objetos a transação de saque ou depósito, aciona o objeto básico que responde pelas operações sobre conta bancárias (ver Fig.4.9) ContaBanco, que por sua vez, dependendo da especialização da conta, se poupança ou conta corrente, aciona as operações de saque e depósito de acordo com o solicitada pela transação. O acionamento do objeto básico ContaBanco pode ser realizado por outros objetos, como por exemplo, a partir de objetos que implementam a interface de um sistema de autoatendimento. Nessa organização das transações, observe-se que os objetos são elementos autônomos com relação às operações e aos dados específicos: o objeto ContaCorrente cuida das operações de saque e depósito de conta bancária do tipo conta corrente de forma autônoma e com os dados internos encapsulados. O mesmo vale para o objeto Poupança.

Uma diferença básica entre as duas formas de organização do programa está na forma como é executada, isto é, enquanto que no procedural a transação é executada em vários módulos, na orientada a objetos, o objeto responsável assume para si a realização completa da transação. Do ponto de vista de manutenção corretiva, a organização orientada a objetos é superior: ao se

detectar um erro de saque de poupança, por exemplo, os eventuais erros podem ser rastreados diretamente nos objetos e uma vez corrigidos, a perturbação sobre o sistema é baixo devido ao baixo acoplamento entre os objetos, diferentemente da organização procedural, onde uma correção pode ser espalhada pelo sistema produzindo perturbações de falhas (consequentes das correções) distribuídas pelos módulos.

Por outro lado, se o sistema exemplo exigir a incorporação de novas funcionalidades, como fica do ponto de vista de engenharia de software? É o caso de manutenção evolutiva, onde o software em produção é alterado para receber novos recursos funcionais. As alterações nos dois paradigmas são esquematizadas nas figuras Fig.4.10 e 4.11. Suponha-se que foi solicitada a incorporação de funções de saque e depósito de um tipo novo de conta, por exemplo, conta investimento especial, denominada InvMkt que apresenta funções de depósito e saque de forma semelhante às disponíveis em conta corrente e poupança.

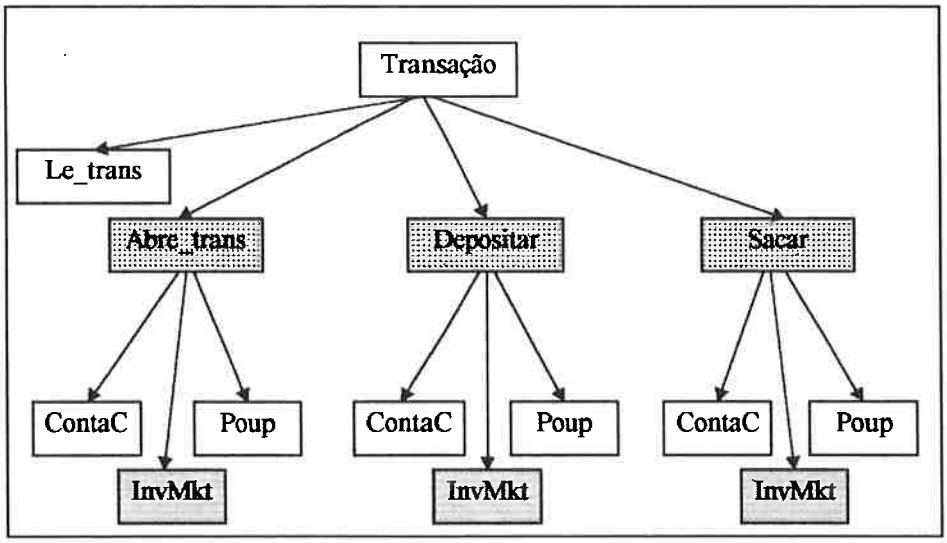

Fig.4.10 - Inserindo uma nova transação - conta tipo investimento InvMKT.

A Fig.4.10 mostra, indicando os módulos com hachuras, que para inserir as transações sobre a conta investimento especial é preciso alterar três módulos e criar três novos módulos. Ou seja as alterações são distribuídas pelo sistema, tornando mais difícil a validação da manutenção realizada. A Fig.4.11 mostra como ficaria a alteração no paradigma orientada a objetos.

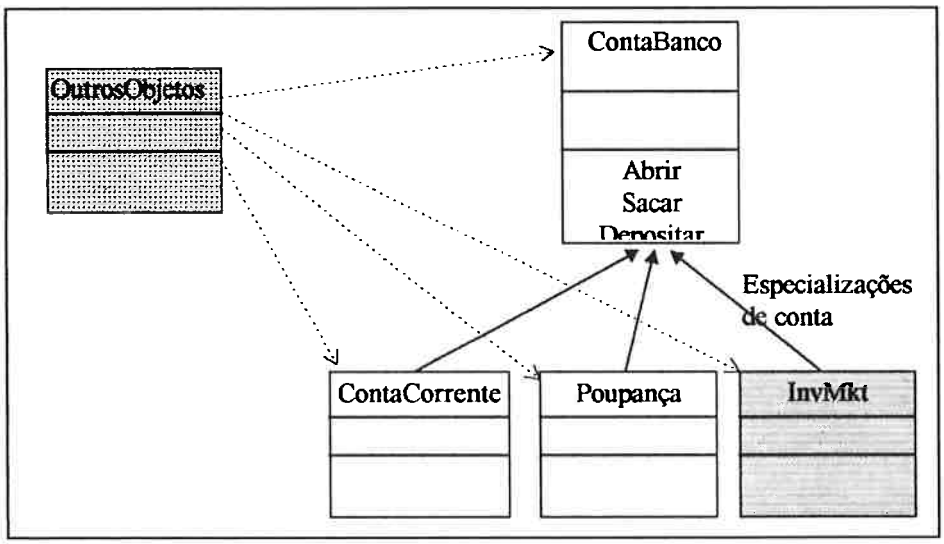

Fig.4.11 - Inserção de novas transações de depósito e saque para a conta investimento especial.

No paradigma orientado a objetos, a alteração está concentrada e sob a responsabilidade (encapsulamento) do objeto InvMkt. Não quer dizer que será escrito menos código para inserir a nova funcionalidade. A diferença está no fato de que neste paradigma, as alterações estão centralizadas e todo o processo de validação exige uma estratégia diferente da usada para validar as alterações no paradigma procedural. Além do objeto InvMkt, os demais objetos do sistema que acessam as novas transações também sofrem alterações, uma vez que passam a solicitar serviços do novo objeto incorporado ao sistema.

4.2.6 As vantagens da engenharia de software orientada a objetos

Do ponto de engenharia de software, a tecnologia orientada a objetos apresenta vantagens que podem ser constatadas através de alguns itens de qualidade: Robustez de software, associado ao comportamento do software sob condições de anormais; Extensibilidade para incorporação de novos recursos funcionais; Reusabilidade que indica a possibilidade de reaproveitamento de produtos de engenharia de software; Integridade, que relaciona o comportamento do software contra modificações acidentais de dados; Produtividade, associado com a otimização de recursos e prazos na construção de software; Dependência de dados, relacionado com perturbações causadas pelas manutenções; e Qualidade da modelagem do sistema em relação ao mundo real. Assim:

- ◊ **Robustez** - está associado com o comportamento do software sob condições anormais, devendo apresentar controlabilidade até mesmo em condições de falhas, através de indicações adequadas nessa situações de falhas. A orientação objeto contribui para a obtenção da robustez à medida em que se pode mudar partes do programa, sem causar perturbações de forma distribuída no software. Encapsulamento providos pelos objetos contribui para melhorar este item de qualidade;
- ◊ **Extensibilidade** - Extender a funcionalidade pode ser bastante explorada por mecanismos de herança, através da criação de objetos especializados a partir de objetos existentes. Com estratégia, extender significa aumentar funcionalidade de forma incremental, aproveitando porções já existentes (robustos por maturidade de uso);

◊ **Reusabilidade** - A idéia de "não reinventar a roda" é bastante utilizada na engenharia de hardware, ou seja, não se perde tempo nem esforços reinventado circuitos através de seus elementos básicos. Aproveita-se os componentes existentes para a construção de novos circuitos eletrônicos mais complexos. A tecnologia orientada a objetos permite tal estratégia, através da implementação de repositórios de objetos genéricos voltados para a reutilização. Para isso deve se adequar a metodologia para o reuso, considerando sempre que componente disponível é maduro por apresentar robustez e portanto pode contribuir para minimizar esforços em testes e validações;

◊ **Integridade** - Interface restrita de acesso e bem definida e os aspectos de encapsulamento provida pela tecnologia OO minimiza os acessos acidentais de dados;

◊ **Produtividade** - A tecnologia orientada a objetos desloca esforços de engenharia para a fase de análise, minimizando esforços nas fases de projeto e implementação. Um dos motivos da produtividade mais acentuada é aquela que considera que a correção de uma especificação (análise) evita que se projete e se implemente com erros que custam muito mais em termos de esforço de tempo e recursos;

◊ **Dependência de Dados** - A tecnologia orientada a objetos provê uma organização de objetos com baixo nível de acoplamento, o que reduz a dependência dos dados (ver itens anteriores). Essa independência é provida pelo encapsulamento;

◊ **Qualidade de Modelagem** - A tecnologia orientada a objetos provê uma melhor modelagem do mundo real e mais, nos vários estágios de um desenvolvimento como análise, projeto e implementação, os objetos de negócios são os mesmos. A estrutura em objetos da fase de análise permanece a mesma na fase de projeto, enriquecida de alguns objetos adicionais da implementação, facilitando a medição e o monitoramento das atividades de desenvolvimento do sistema.

4.3 A Tecnologia Orientada a Objetos na Indústria Financeira

Para discutir aspectos de incorporação da tecnologia orientada a objetos na indústria financeira, discute-se a seguir o contexto de geral a nível de sistema, os processos de produção e manutenção e software, as arquiteturas típicas de software e os sistemas legados.

4.3.1 O contexto de aplicação da indústria financeira

Para entender o contexto de aplicação da metodologia proposta nesta tese, apresenta-se algumas características da indústria financeira do ponto de vista da estrutura de hardware, software e da equipe técnica envolvida.

Estrutura de Hardware

Uma aplicação da indústria financeira baseia-se fortemente em sistemas de grande porte, com recursos de comunicação provendo mecanismos de interligação com os seus departamentos, agências e mesmos os clientes através de aplicações denominadas Home/Office Banking, onde os recursos são acessados remotamente. Ver a Fig.4.12.

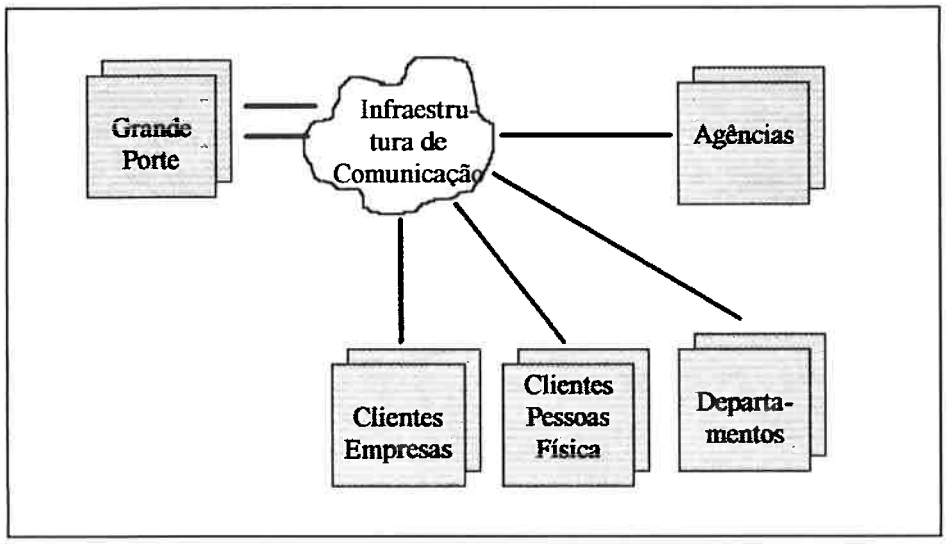

Fig.4.12 - Esquema de Hardware de uma instituição financeira.

Estrutura de software

Uma aplicação típica da indústria financeira apresenta pode ser modelado através de uma arquitetura em três camadas, conforme mostra a Fig.4.13, com processamento distribuído

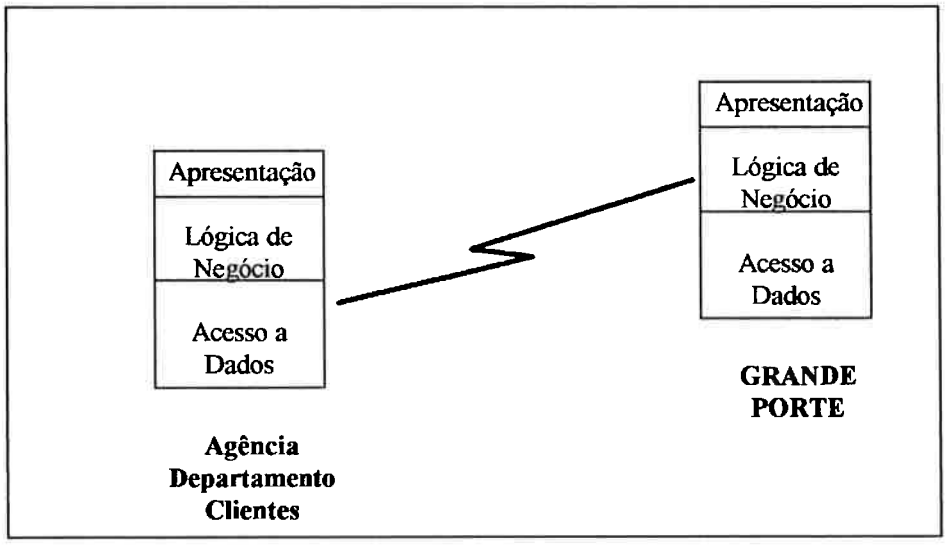

Fig.4.13 - Arquitetura típica dos sistema de software da área financeira.

Equipe Técnica (Peopleware)

As equipes técnicas se caracterizam por conhecerem profundamente os processos de negócios, independentemente do nível de maturidade dos processos de produção de software. Essa característica é o principal fator na estratégia para incorporar a tecnologia orientada a objetos. O conhecimento dos processos de negócios facilita a abstração e a modelagem dos processos de negócios através de objetos de negócios. As pessoas apresentam um nível de maturidade equivalente ao nível inicial, segundo a caracterização do SEI/CMM (ARAKAKI (1996a), MCCONNELL, 1996).

4.3.2 A inserção da tecnologia no processo de obtenção de software

Inserir técnicas de orientação a objetos dentro deste conceito deve atender a duas necessidades básicas:

- ◊ ANÁLISE - por técnica, a orientação a objetos desloca esforços de engenharia para a fase de análise (BOOCH, 1996), o que dentro deste contexto é uma vantagem muito grande, pois as maiores dificuldades dos produtos e dos processos estão em produzir implementações deficitárias por não atenderem aos requisitos, ou seja, os requisitos não foram devidamente consolidados antes de iniciar esforços de implementação.

◊ PRESERVAR O CONHECIMENTO DE NEGÓCIO - preservar os processos de negócios é um desafio, devido ao grau de terceirizações que ocorrem nas grandes corporações. Aproveitar os conceitos de componentização de objetos é uma maneira de minimizar este problema, ou seja, mesmo havendo uma terceirização muito forte, os processos de negócios podem ser preservados pela seguinte estratégia: a equipe interna define e especificam a arquitetura da aplicação e os componentes e terceirizam-se a implementação física dos componentes.

4.3.3 A arquitetura de aplicações usando orientação a objetos

Os sistemas ficam organizados em componentes de software de software que são compostos por um conjunto de objetos. Assim, cada módulo de um sistema podem ser organizados por "scripts" de programas que acionam os componentes da aplicação, de acordo com o modelo elaborado nas fases de análise e projeto.

Assim, no ambiente Cliente-Servidor, um sistema pode ser agrupado em módulos especializados, segundo o modelo das camadas:

◊ Lógica de negócios;

◊ Apresentação;

◊ Acesso aos dados.

4.3.4 O novo paradigma e o legado

Dentro deste ambiente, a convivência com os sistema legados, deve usar uma estratégia básica: procurar integrar com tais sistemas de acordo com as

necessidades de negócio, utilizando-se de componentes especializados que promovam uma integração com alto grau de desacoplamento entre o sistema e o legado, quer podem ser no Mainframe ou no mesmo ambiente local (departamento e agências). Esses componentes têm a responsabilidade de prover o interfaceamento. Dessa maneira, as próprias características e estruturas providas pela orientação a objetos promove o desacoplamento para manter a independência técnica entre os sistemas.

4.4 Métricas em Sistemas Orientados a Objetos

É por volta de 1990 (BIEMANN, 1992) que a orientação a objetos aparece tão forte na indústria e estabelecer métricas neste paradigma exige uma revisão conceitual em alguns pontos como acoplamento e coesão de dados. A maior dificuldade no uso de métricas orientadas a objetos está no fato de que várias empresas de alta competitividade no mercado de produtos de software estão aplicando a tecnologia que lhes permite lançamentos de produtos a uma média de três a quatro meses. Muitas das métricas não são de acesso público mas é notório o ganho em produtividade obtidas, considerando especialmente através de dois indícios (complexidade e prazos de lançamento) que também servem como métricas: versões de produtos de software para uso pessoal como editores de texto, planilhas eletrônicas, interfaces de navegação na internet se apresentam cada vez mais complexos com um número de funções cada vez mais complexos em prazos pequenos da ordem de três meses.

4.4.1 Métricas para sistemas orientados a objetos

A tecnologia orientada a objetos utiliza uma variedade de estruturação da abstração na programação, no projeto e na análise. Vale ressaltar três importantes mecanismos dessas abstrações: um deles é abstração de dados e funções, outro é a herança e por fim o mecanismo de mensagem para controle e comunicação entre objetos. Variáveis não acessados diretamente e sim através de solicitações de serviços realizados através de mensagens.

Software dentro do paradigma tradicional utiliza procedimentos e funções como aos maiores mecanismos de abstração, ou seja, um conjunto de procedimentos e um conjunto de dados acessados por eles. Nesse caso o sistema é uma coleção de procedimentos e funções.

Num sistema de software orientado a objetos, o software é composto de uma coleção de elementos onde a menor unidade é a classe ao invés de procedimentos. Uma classe é uma especificação encapsulada de tipo abstrato de dados e de operações. Para alterar o estado interno de um objeto, é preciso enviar uma mensagem para o objeto. As respostas às mensagens são especificadas através de métodos que são, na essência, procedimentos. Eles tem variáveis locais, enviam mensagem para outros objetos. Outro ponto específico é o de herança que fornece suporte para criar classes especializadas, com base numa já existente através de mecanismo de herança.

De acordo com Tegarden (TEGARDEN, 1995) existem os alguns níveis de métricas a serem aplicadas tomando como base a caracterização estrutural de sistemas orientados a objetos, que são:

- ◊ Nível de método - Este nível refere-se às operações definidas para uma classe. Este nível corresponde aos procedimentos e funções de uma software tradicional. Assim os conceitos de medidas do software tradicional podem ser aplicados neste nível (OTT, 1995);
- ◊ Nível de objeto - As classes correspondem ao elemento unitário, assim como os procedimentos correspondem ao elemento unitário de um software tradicional. O problema central aqui é definir como estabelecer o modelo de métricas para uma classe. Um dos caminhos é definir uma classe como uma unidade autônoma, cuja a métrica é obtida a partir da composição de métricas dos métodos que compõe cada classe (MELTON, 1996); e
- ◊ Nível de sistema - Cada classe esconde detalhes através da técnica de encapsulamento para preservar a integridade das informações de cada uma delas. As mensagens passadas de um objeto para outro estabelece um grafo de chamada de métodos que também contribui para análise de complexidade de chamada. Assim, um sistema pode ser modelado como um conjunto de classes conectados com recursos de hierarquia de herança e canais de mensagens que são as relações entre as classes;

◊ Estrutura de herança - Herança é unica nos sistemas orientados a objetos. As hierarquias de classes, sub-classes e super-classes podem ser representados por um gráfico de hierarquia de classes. Esse gráfico deverá permitir que se derive as métricas referentes às classes e às sub-classes.

Métrica de Visibilidade de Pacotes/Classes

Gannon, Katz e Basili (GANNON, 1986) propuseram métrica denominado de *package visibility metric*, que é baseada na identificação e contabilização de módulos unitários. Essa métrica é baseada na contagem de :

◊ Unidades (de uma *package*) que acessam informações numa determinada unidade;

◊ Unidade onde as *packages* são visíveis;

◊ Unidades onde uma *package* pode ser visível adicionando-se a cláusula *with*; e

◊ Unidades onde os packages são visíveis depois de se usar a clausula with nas unidades locais mais específicas (Proposta).

Essas métricas baseiam-se na contagem do número de *packages*. A métrica de acesso a *packages* indica um número que relaciona o número de packages de objetos não locais que são acessados em uma *package*. Ela é calculada como uma proporção de número de objetos não locais (declarados como pacotes externos) que são acessados pelo pacote pelo comprimento do pacote (medido em linhas de códigos). Os autores mostram que a métrica de acesso a *package* dá indícios de acoplamento e dificuldades de alterar ou manter um pacote. Entenda-se uma

package como um conjunto de objetos. Produtos de mercado para micromputadores conhecidos como componentes são exemplos de elementos cabíveis de serem submetidos a estas métricas.

Métricas Propostas por Sheetz

Sheetz, Tegarden e Monarchi (SHEETZ, 1991) derivaram um conjunto de métricas baseadas na definição de classes e as relações de herança. Algumas delas:

Métricas de Classes:

- ◊ Utilização: Indica o número de classes usadas por uma classe;
- ◊ *Fan-in/Fan-out*: Número de mensagens recebidas/enviadas por uma classe;
- ◊ *Fan-down*: Número de sub-classes de uma classe;
- ◊ *Fan-Up*: Número de super-classes de uma classe;
- ◊ Profundidade classe-pai: Máximo número de níveis acima de uma classe numa relação de hierarquia de herança;
- ◊ Profundidade classe-folha: Máximo número de níveis abaixo de uma classe numa relação de hierarquia de herança;
- ◊ Conflito classe-pai: Numero de propriedades definidos em
- ◊ Conflito pai-pai: Número de propriedades definidos em múltiplos pais da classe com o mesmo nome (herança múltipla);

Métricas de estrutura de herança:

- ◊ Profundidade máxima: Número de níveis entre a raiz e folha da árvore de classes na maior distância de hierarquia possível;
- ◊ Largura máxima: Número máximo de classes num dado nível;
- ◊ Número de associações de herança: Número de links de herança.

Um estudo encaminhado por Lake e Cook (LAKE, 1992) indicou que é mais fácil para os programadores executar mudanças nas classes próximas das raízes da árvore de hierarquia. O estudo foi feito tomando como base em levantamento em torno de 10 sistemas, através de uma ferramenta capaz de analisar um programa em C++ e gerar uma árvore de hierarquia de classes e indicar o número de classes em cada nível, a profundidade de cada classe na árvore e o número de sub-classes de cada classe. Este estudo foi apresentado com resultados classificados como preliminares uma vez que estudos estão prosseguindo num campo maior de análise de sistemas existentes.

Métricas de Chidamber[92]

Chidamber e Kemerer (CHIDAMBER, 1992) investiram esforços no desenvolvimento de métricas estruturais em projetos orientado a objetos. Eles desenvolverem um conjunto de métricas que incluem:

- ◊ Métrica de tamanho baseado no número de métodos por classes;
- ◊ A profundidade de herança de uma classe;
- ◊ O número de filhos;
- ◊ Número de associações sem herança com outras classes;
- ◊ Número de métodos que podem ser invocados por outras classes;

◊ Não-coesão de classes baseados em variáveis de instâncias não compartilhadas.

Essas métricas foram aplicadas em dois sistemas, um em C++ e outro em *Smaltalk*. Um dos fatos relevantes neste estudo foram que as métricas aplicadas sobre esses sistemas não utilizavam muitas estruturas de herança. Uma evolução destas métricas foram incrementadas por Li e Henry (LI, 1993) que adicionaram entre outras técnicas de métricas. Eles adicionaram métricas de acoplamento através de mensagens passadas, através de declarações de envio, e também medido pelo número de dados ADT (*Abstract Data Types*) definidos na classe. Eles definem vários tipos de métricas de tamanhos de classes incluindo o número de métodos locais e número de comandos. Eles também mostram que as métricas podem ser estatisticamente relacionadas com os esforços de manutenção.

Bieman conclui que muitos das métricas até citadas ainda devem ser melhorados em termos de conceituação e aplicação. Segundo este pesquisador (MELTON, 1996), as métricas orientadas a objetos: (a) combinam métricas com diferentes atributos elaborando composições ou combinações com perdas de sensibilidade de medida; (b) Ajusta simples contagem com pesos, tornando difícil senão impossível de existir numa determinada escala; (c) confundem sistema de medidas com sistemas de estimativas. Essas considerações, segundo o autor devem melhorar com o aprofundamento nas pesquisas de medidas sobre software orientada a objetos.

Métricas de Reuso de Karunanithi (KARUNANITH, 1993)

Um grande campo de estudo com relação a métricas de sistemas orientada a objetos, especialmente nos aspectos de projetos e programação, está no impacto da elaboração de sistemas de software usando componentes reusáveis. São componentes que podem ser reusados diretamente ou especializados através de construções de sub-classes. Brian Hederson-Sellers aponta em suas pesquisas (HEDERSON-SELLERS, 1992) as diferenças de métricas para elaboração de estimativas, através da comparação de métricas de sistemas procedurais e sistemas integrados usando recursos de reutilização de código.

As métricas atualmente utilizadas para quantificar reuso não estão diretamente relacionadas com a orientação a objetos, de modo que há muitas oportunidades de se realizar pesquisas neste campo. Um centro de pesquisa que não deve deixar de ser referenciada é a CSU (Colorado State University) onde existe grupo especializado neste estudos. Pesquisas registradas por Bieman (em BIEMAN, 1991) classifica a reutilização de software, identifica perspectivas de reuso, propõe abstrações para reuso e sugere atributos de reuso e métricas associadas com sistemas orientada a objetos. No trabalho de Karunanithi e Bieman em BIEMAN (1993 a), estas definições são extendidas para a linguagem ADA, ou seja, projetos de software baseado em objetos, patrocinado pela NASA, CTA e Colorado Advanced Software Institute).

Valem algumas definições, segundo estes estudos:

- ◊ CLASSIFICAÇÃO DE REUSO: A reutilização pode ser classificado segundo o modo em que ela acontece: Público ou Privado; Puro ou Genérico ou por níveis; e de modo direto ou indireto;
- ◊ Reuso Público/Privado: Reuso público acontece quando um sistema se utiliza de software costruído externamente a ele, enquanto que o reuso privado acontece dentro do prórpio sistema;
- ◊ Reuso Puro/Genérico/Por níveis: O reuso Puro é quando se utiliza software sem nenhuma modificação; Reuso por níveis acontece com modificação do módulo a ser utilizado (independente se a linguagem suporta ou não a modificação por técnica de herança); e o genérico refere-se a uso de pacotes genéricos - eles são templates genéricos que sofrem processos de adequação e preparação através de compilações parametrizadas;
- ◊ Reuso Direto/Indireto: Reuso direto refere-se ao reuso de um software, sem auxílio de uma entidade intermediária; Reuso indireto usa entidades intemediárias. O número de indireção depende do número de entidades intrmediárias. Numa arquitetura Cliente-Servidor acontece tais situações.

Segundo Bieman (MELTON, 1996), linguagens orientada a objetos suportam reuso da seguinte maneira:

- ◊ Reuso puro, através da instanciação e uso de classes pré-definidas;

◊ Reuso genérico através de uso de templates genéricos; e

◊ Reuso por níveis, através da técnica de herença.

4.4.2 Métricas relevantes nos processos da indústria financeira

As aplicações, objeto desta pesquisa, caracteriza-se por utilizar uma arquitetura cliente-servidor, onde módulos de software se relacionam como cliente de um serviço fornecido por um outro módulo. Nessa situação o reuso pode ser observado do ponto de vista de servidores, de clientes e do próprio sistema como um todo. Quando uma classe reusa uma outra, pode acontecer por herança, por instanciação de uma classe genérica ou por simples uso de outra classe. Desta forma, métricas do número de servidores usados, números de vezes em que os servidores são usados, o número de clientes para um servidor, o tamanho de cada servidor, ou de cada cliente ou outras medidas são importantes.

4.4.3 Avaliações possíveis a serem obtidas

As métricas devem ser cuidadosamente projetadas para que possam ser eficaz como elemento de administração e controle de processos de software. Vale ressaltar o trabalho efetuado por Karunanithi e Bieman, especialmente no que se referem a definição de métricas relacionadas com a reutilização. Como resultado de suas pesquisas, obtiveram uma lista de métricas relacionada com a organização cliente/servidor(fornecedor). As Tab. 4.4 e Tab. 4.5. registram essas métricas em separado, segundo o ponto de vista de servidores e clientes (BIEMAN, 1993b), das quais apresenta-se na forma de tabela. Ver Tab.4.4.

#	MÉTRICAS (Do ponto de vista de Classes CLIENTES)	DEFINIÇÃO
	Por nível	Métrica de herança.
1	#Classes servidoras diretas	Número de superclasses diretas.
2	#Classes servidoras indiretas	Número de classes servidoras com relacionamento indireto com a classe nos aspectos de uso, instanciação e herança.
3	#Servidores pai indiretos	Número de superclasses indireto da classe cliente.
4	#Métodos de servidor herdados diretamente	Número de métodos de classes servidoras disponíveis para clientes.
5	#Métodos de servidores extendidos	Número de métodos extendidos dos métodos das classes servidoras.
6	#Métodos de servidor diretamente redefinidos	Número de métodos redefinidos, através de especialização, os correspondentes métodos das classes servidoras.
7	#Métodos de classes servidoras diretamente sobrecarregadas	Número de métodos em classes clientes que sobrecarregam métodos das classes servidoras.
8	Tamanho: de cada método de classe servidora que é reusada ou extendida	
9	Tamanho: da interface da servidora	
10	Tamanho:das definições globais da interface da classe servidora	
11	Tamanho: das definições do corpo da classe servidora	
12	Tamanho: de cada método cliente	
13	Tamanho: da interface cliente	
14	Tamanho: das definições globais na interface cliente	
15	Tamanho: das definições globais do corpo da classe cliente	
16	#Caminhos de acesso indireto às classes servidoras	Número de caminhos conectando a classes cliente e as classes servidoras.
17	Comprimento dos caminhos para os servidores indiretos	Número de segmentos de acesso indireto conectando as classes cliente e servidoras.
18	#Caminhos de acesso indireto as classes servidoras-pai	Número de caminhos conectando cliente e classes servidoras pai.
19	Tamanho: do caminho até as classes servidoras-pai	Número de segmentos de acesso indireto conectando a classe cliente às classes servidoras-pai.

Tab. 4.4 - Métricas de reuso do ponto de vista de classes cliente (Karunanithi e Bieman, 1993b).

Doponto de vista de classes servidoras, vale a seguinte relação de métricas de reuso, conforme mostra a Tab. 4.5.

#	MÉTRICAS (Do ponto de vista de Classes SERVIDORAS)	DEFINIÇÃO
	Por nível	Métrica de herança.
1	#Classes cliente diretas	Número de superclasses diretas.
2	#Classes cliente indiretas	Número de classes cliente com relacionamento indireto com a classe nos aspectos de uso, instanciação e herança.
3	#Cliente-filhos indiretos	Número de subclasses indiretos da classe servidora.
4	#Métodos de servidor acessados pelas classes cliente	Número de métodos de classes servidoras acessadas pelas classes clientes.
5	Tamanho: dos métodos das classes servidoras	
6	Tamanho: da interface da classe servidora	
7	Tamanho: das definições globais na interface da classe servidora	
8	Tamanho: das definições globais do corpo da classe servidora	
9	#Caminhos de acesso indireto às classes clientes	Número de caminhos conectando classes servidoras e clientes.
10	Comprimento dos caminhos indiretos para as classes clientes	Número de segmentos de acesso indireto conectando classes clientes e servidoras.
11	#Caminhos de acesso indireto as classes cliente-filhas	Número de caminhos conectando a servidora e as classes clientes-filhas.
19	Tamanho: do caminho até as classes clientes-filhas	Número de segmentos de acesso indireto conectando a classe servidora às classes clientes-filha.

Tab.4.5 - Métricas de reuso do ponto de vista de classe servidora (Karunanithi e Bieman, 1993b).

Como pôde ser visto, existem muitas métricas para sistemas orientado a objetos e nesse caso é necessário um planejamento em cada contexto de aplicação. Nesse caso é importante a aplicação do paradigma GQM (Goal-Question-Métric) de Basili e Rombach (BASILI (1988), define as medidas de software em relação a uma meta do processo de engenharia de software. Certamente estas métricas devem estar centradas em classes.

5. METODOLOGIA PARA AVALIAÇÃO DE QUALIDADE

Objetivo do Capítulo:

- Apresentar uma metodologia para implantar mecanismos de avaliação de processos de produção de software, a partir da identificação, classificação e reorganização do processo atual, usando a técnica de orientação a objetos, capacitando o ambiente de produção a se inserir em um nível de maturidade equivalente ao nível 2 do modelo SEI/CMM.
- Essa metodologia define estratégias para reorganizar um processo quasi-caótico, classificado como nível inicial de capacidade e maturidade em produção de software, através de mecanismos que incorporam requisitos de qualidade e pemitem a avaliação por métricas.

Tópicos do Capítulo:

5.1 Visão Global da Metodologia

5.1.1 Objetivo da metodologia

O objetivo desta metodologia é implantar mecanismos de avaliação de qualidade e produtividade[1] (isto é, mecanismos que permitam a avaliação) em processos de desenvolvimento de sistemas de software cujo ambiente de obtenção de software apresenta-se não transparente, quasi-caótico, obtendo como resultante (simultânea e consequente) a organizaçãodo ambiente do processo de desenvolvimento de sistemas de software. Consequência simultânea da implantação de mecanismos de avaliação é a implantação também de procedimentos para planejamento, controle, acompanhamento e desenvolvimento (ou aquisição ou manutenção).

5.1.2 Características da metodologia

Esta metodologia foi elaborada com base em ambientes da indústria financeira, o que não invalida sua aplicação em ambientes de outros setores empresariais, fazendo-se, evidentemente, os devidos ajustes/adequações com base na análise de suas características específicas (Fig.5.1).

As características que identificam esta metodologia são:

1. Aplicação em ambientes de desenvolvimento de sistemas de software quasi-caóticos (nível 1).

[1] Nesta tese, entenda-se por mecanismos de avaliação de qualidade e produtividade ao conjunto de procedimentos, conceitos, padrões, ferramentas e recursos que possibilitem obter informações de produtos e de processos que sirvam de subsídios para análise e aumento de qualidade e produtividade de um processo.

2. Organiza o ambiente e prepara-o para se enquadrar no nível 2, e em seguida se enquadrar rapidamente no nível 3.
3. A estratégia básica é a implantação de mecanismos de avaliação da qualidade e da produtividade (que permitam tal avaliação)- com conseqüentes análise, controle e acompanhamento da qualidade, da produtividade, das partes do processo (andamento) e do processo como um todo.
4. Organiza o ambiente e chega a mecanismos de avaliação considerando quatro fatores: Pessoas, Produto, Processo e Tecnologia.
5. Aplica conceitos de particionamento.
6. Aplica conceitos da Tecnologia Orientada a Objetos.
7. Estabelece uso de métricas ao longo do processo de organização, desde seus estágios iniciais

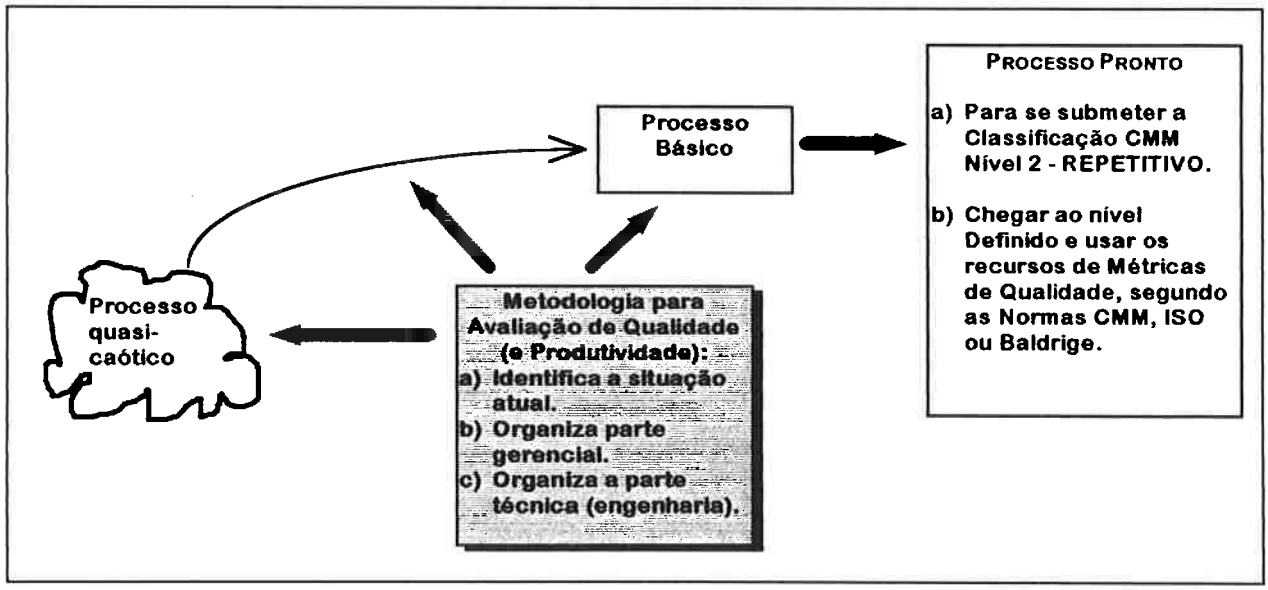

Fig.5.1 - Visão Global da Metodologia.

Algumas considerações sobre terminologias utilizadas:

- Ambiente ou processo quasi-caótico: pode ser descrito como ambiente/processo de desenvolvimento de sistemas de software com características que o tornam obscuro, de difícil análise, avaliação, organização, precisão, não havendo transparência nem em seu todo, nem em suas partes. Alguns elementos exemplos que o identificam: ausência de processo repetitivo, produtos de software dependentes de pessoas/equipes, inexistência de procedimentos para avaliar qualidade/produtividade/processo, prazos restritivos, pré definidos, devido ao aspecto tempo ser vital para a validade do desenvolvimento ou ser vital para a competitividade da empresa.

- A classificação dos ambientes em níveis 1, 2 e 3, com base no modelo SEI/CMM, é adotada como padrão de referência técnica nesta tese, no entanto, outras referências poderiam ser utilizadas, baseando-se nas normas internacionais de engenharia de software, com perfis equivalentes - vide Cap.2 para maiores detalhes.

Situação das áreas de desenvolvimentos das indústria financeira

Em ambientes da indústria financeira, o fator prazo é característica peculiar, figurando normalmente como prioridade máxima, independente das dificuldades/problemas em qualquer das fases do desenvolvimento, independente do quanto se gasta em recursos materiais/humanos, independente de se terceirizar ou não. Há, por exemplo, casos em que a alta administração determina que um sistema x deve sair em um prazo y para garantir a competitividade no mercado.

Um fato observado é o de que, em grande parte da indústria financeira, os sistemas de software são considerados essenciais em seus processos de negócios, adquirindo um *status* maior que o de fator de competitividade.

As deficiências mais marcantes detectadas nos ambientes vivenciados referem-se a: gerenciamento técnico e administrativo, processos técnicos e a mecanismos de qualidade:

- ◊ Situação Gerencial: A ausência de planejamento com o detalhamento adequado de atividades, recursos, prazos e mapeamento falho dos riscos do projeto causam falta de visibilidade que dificulta o acompanhamento dos projetos. Com isso, é frequente acontecer atrasos de projeto, descontroles como gastos excessivos com recursos humanos e com recursos materiais;
- ◊ Engenharia de Software: Os processos técnicos são difíceis, deficientes, devido à falta de técnicas de engenharia de software na construção de um sistema. Especificação de requisitos incompletos ou inexistentes, ausência de atividades de análise e acompanhamento do projeto e, além disso, a falta de planejamento e realização de testes, são alguns dos pontos mais freqüentes, detectados por estes estudos;

◊ Sistemas de Qualidade: Faltam mecanismos de apoio à qualidade, como registros de documentação, uso de ferramentas CASE, revisões técnicas, em cada etapa do ciclo de desenvolvimento, dos produtos elaborados. Como conseqüência, são comuns os retrabalhos (decorrentes das falhas de especificações, detectadas somente durante a implementação) e também a dificuldade de aprovação de sistemas pelos usuários (devido a grande quantidade de erros não detectados em tempo de desenvolvimento).

Arquitetura Geral do Ambiente de Aplicação da Metodologia

O ambiente onde se aplica esta metodologia apresenta uma arquitetura Cliente-Servidor, que pode ser esquematizada como na Fig.5.2.

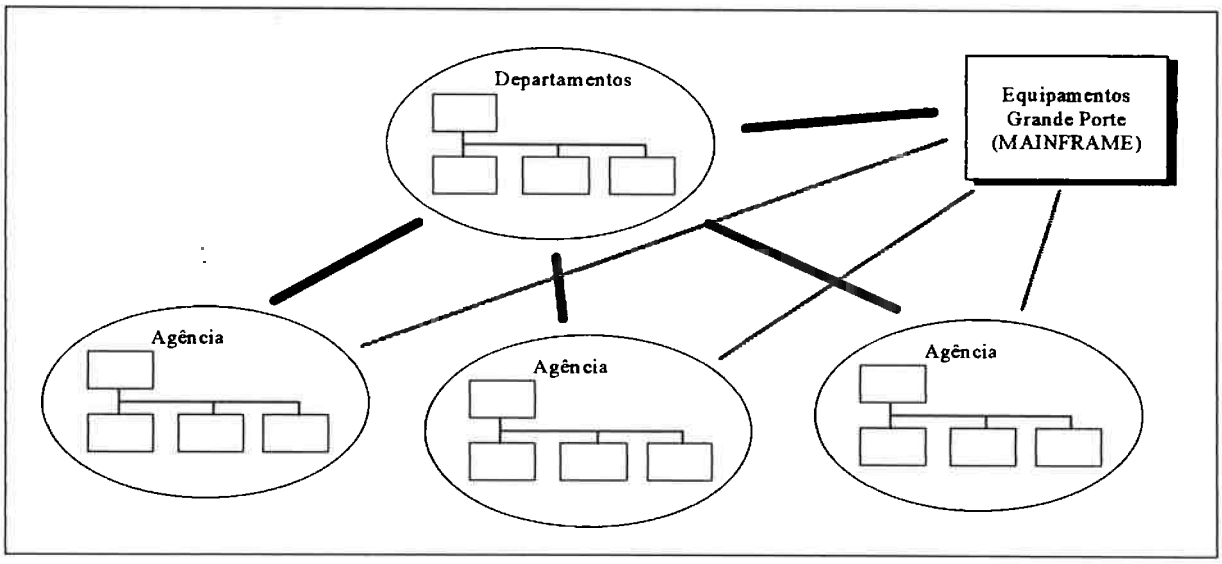

Fig.5.2 - A metodologia se aplica no desenvolvimento de aplicações Cliente-Servidor (Como ocorre nos departamentos e agências de uma organização da indústria financeira).

Há equipamento de grande porte (mainframe), departamentos da instituição e as agências, pontos de atendimento e comunicação com o cliente. A informação flui entre os três elementos direta ou indiretamente, estando, portanto, os três interagindo de forma dinâmica e inter-dependente.

5.1.3 Estrutura da metodologia

A Tab.5.1 mostra um esquema, em alto nível, da estruturação da metodologia. Ela é composta de: diagnóstico do ambiente (recursos para determinar estágio profissional de desenvolvimento de software - Nível 1 - SEI/CMM); controle gerencial (atuação sobre os processos gerenciais, especialmente nos mecanismos de planejamento, acompanhamento com requisitos de métricas, segundo princípios básicos da engenharia de software); processos de engenharia (em paralelo com o controle gerencial, tanto quanto possível, os processos técnicos são elaborados e implantados segundo métodos técnicos determinados, que garantam visão, análise, controle e avaliação das partes e do todo - nessa situação, um trabalho intenso e efetivo deve acontecer sobre a equipe técnica, elemento fundamental para esta reorganização); finalização: processo pronto para se classificar no nível 2 e 3 .

A METODOLOGIA (ETAPAS)	CARACTERÍSTICAS E RESULTADOS	ESTRATÉGIAS
Diagnóstico Global	Registro da Situação atual: • Gerencial • Técnico	• Identificação e classificação dos tipos de projetos
Controle Gerencial	Implanta Controle Gerencial: • Recursos, prazos e funções • Revisão formal • Métricas	• Categorização de projetos segundo visão do processo de negócio e sub-particionamento tecnológico. • Avaliar a qualidade de especificação e de planejamento
Roteiros de Engenharia	Implanta Métodos Técnicos: • Especificação, Particionamento, Planejamento • Roteiros de análise, projeto e implementação • Planejamento e registro de testes • Garantia da qualidade • Configuração de software	• Preparação de Pessoal *on-the-job training*; • Uso das tecnologia de Workflow, Orientação a Objetos, e reforço na infra-estrutura de testes. • Estimular a aplicação da reutilização
Pronto para o Nível 2 e 3 SEI/CMM	Maturidade direcionada: • Nível 2 - Repetitivo • Nível 3 - Definido • Segue conforme plano de qualidade • Avaliação de qualidade nos moldes ISO, SEI/CMM e Baldrige (Cap.2)	• Buscar pré-avaliação externa; • Formalizar base de dados histórico

Tab.5.1 - Estruturação da metodologia.

Essa metodologia, conforme estrutura mostrada na Tab.5.1, produz, ao final da sua aplicação, uma reorganização de um ambiente de obtenção de software com condições de se submeter a uma avaliação da sua maturidade e capacidade de produzir software, segundo o nível 2 da CMM (e posteriormente, nível 3). A sua aplicação é mais efetiva quanto maior for a sinergia obtida entre a equipe gerencial e a equipe técnica de engenharia de software do ambiente a ser reorganizado, o que, com certeza, será uma conseqüência, pois a reorganização baseia-se no envolvimento das pessoas, buscando, em conjunto, estabelecer mecanismos que garantam e melhorem a qualidade de trabalho e de produtos, incluindo no processo atual algumas práticas que facilitem a realização das tarefas com maior qualidade e simplicidade.

Etapas de implantação da metodologia:

- **Etapa 1: Diagnóstico Global** - Esse diagnóstico deve ser realizado em dois níveis: gerencial e técnico. No nível gerencial, devem ser registrados e analisados os procedimentos utilizados para elaborar o planejamento, o acompanhamento e a gestão dos recursos materiais e humanos. Quanto aos recursos humanos, verificar o nível de formação e a motivação com que trabalham nos projetos. Ainda no nível gerencial, é preciso definir os tipos de projetos sob o prisma de negócios e sob o prisma tecnológico. No nível técnico, é preciso ver como estão os processos, os produtos de software e os aspectos de qualidade, sendo que a tecnologia utilizada e sua adequação deve ser avaliada nesta etapa. Ver exemplos no Cap.6;

◊ **Etapa 2: Controle Gerencial** - Após o diagnóstico, define-se o controle gerencial de um projeto. Deve-se fazer o planejamento e o controle dos recursos, dos prazos e das atividades. Além disso, deve-se implantar um processo de revisão formal de produtos intermediários elaborados nos processos. Métricas, que possibilitem controle mensurável, quantitativo, devem ser empregadas para se fazer estimativas e para se verificar o andamento do projeto em relação ao prazo e tarefas executadas. Esta tarefa deve ser realizada em conjunto com administradores, gerentes e líderes de equipes;

◊ **Etapa 3: Roteiros de Engenharia** - Ocorre, tanto quanto possível, em paralelo com a etapa 2, pois se complementam. Etapa realizada junto à equipe técnica e aos líderes de projetos, deve-se reorganizar os processos internos, incluindo algumas práticas eficazes como os descritos no Cap.2. Deve-se reforçar as fases de produção de um sistema de software, principalmente as fases de especificação de requisitos, planejamento de testes, enfatizando a necessidade de aspectos que garantam a qualidade adequada ao ambiente alvo da aplicação desta metodologia (Cap.2 e 6). Nesta etapa aplica-se a técnica de orientação a objetos, que servirá como um elemento potencializador do nível de maturidade do processo. O conhecimento da equipe técnica, em relação aos processos de negócios, facilita o uso da técnica de orientação a objetos, especialmente devido ao grande poder de expressão da modelagem, usando, por exemplo, o método OMT (*Object Modeling Techniques*, RUMBAUGH et. al., 1991).

Também nesta etapa, a adoção da técnica de particionamento de grandes sistemas em módulos menores, deve ser rapidamente formalizada e implantada;

Os procedimentos de avaliação de qualidade devem ser planejados e incorporados nas etapas 2 e 3 que são realizadas em paralelo. Recomenda-se o uso da técnica de *on-the-job training* para implantar operacionalmente o processo (ver Cap.6 e 7). Ao final, o ambiente reorganizado pode passar por uma auditoria, e estará pronto para se enquadrar no contexto de avaliação de qualidade, segundo normas internacionais, como as que estão referenciadas no Cap.2, pois apresentará processos técnicos e gerenciais uniformes e pessoal preparado para desenvolvimento de sistemas com a qualidade estabelecida segundo a metodologia.

5.2 A Técnica Orientada a Objetos (OO)

A técnica orientada a objetos foi escolhida principalmente por oferecer mecanismos de abstração, encapsulamento e particionamento (Cap.4). Dessa maneira fica completa a metodologia, conforme estipula a definição de uma metodologia, como descrito no Cap.2, definição correspondente à camada da arquitetura. Assim, além do fato de os sistemas serem estruturados na forma de Cliente-Servidor, os sistemas de software devem ser projetados utilizando-se a técnica de orientação a objetos durante todo o ciclo de desenvolvimento, com uma ênfase muito forte nas fases de análise e de projetos.

5.2.1 Método orientado a objetos

Da tecnologia orientada a objetos, dentro do contexto da área financeira, foi escolhido o método OMT (*Object Modeling Techniques*, ver QUATRANI, 1996; RUMBAUGH et. al., 1991) devido à ênfase do método na fase de análise, onde os principais objetos, aqui denominados de objetos de negócios, podem ser modelados de forma eficaz e em grande conformidade com os processos de negócios a que se propõem os sistemas a serem desenvolvidos. Um fator importante proporcionado pelo método é a sua capacidade de modelagem sobre os objetos de negócios. Com esta modelagem, o desenvolvedor com base nos objetos de negócios pode validar todos os cenários funcionais, antes mesmo de serem implementados, com a ajuda até mesmo de especialistas dos processos, futuro usuário do sistema.

Outro fator importante é a possibilidade de montagem de componentes de software, ou seja, componentização. Os objetos de negócios são encapsulados em elementos executáveis, na forma de biblioteca de componentes, de forma análoga às bibliotecas de funções, muito utilizadas, na década de 70/80, em linguagens como FORTRAN. O uso de componentização é parte de uma estratégia para obter a reutilização, processo hoje bastante importante, e já consolidado por grandes fabricantes de software a nível mundial (ARAKAKI (1996 a).

Usar componentes prontos na reestruturação de sistemas de software traz benefícios diretos em robustez de software (uma vez que utiliza elementos de

software já amadurecidos, por estarem em uso por outros sistemas) e também resulta em ganho de tempo no desenvolvimento (pois evita trabalhos de desenvolvimentos repetidos - CHAPPELL, 1996). Não deve ser descartado ainda o ganho de tempo na fase de testes devido as funcionalidades corretas que o componente deve proporcionar.

5.2.2 Vantagens da Organização através de Objetos - As métricas importantes

Organizar o software através de objetos proporciona um efetivo ganho no processo de desenvolvimento, pois possibilita a estruturação de um sistema de software subdividindo-o em pequenos elementos de software, que são os objetos. Poder visualizar um sistema complexo como uma composição de partes simples, que são os objetos, traz benefícios do ponto de vista de controle gerencial, como:

- ◊ Planejar, administrar e executar pequenos elementos de software é mais fácil - é o que acontece no caso de uso de objetos, uma vez que um sistema maior pode ser analisado uma organização de objetos que se caracterizam por serem pequenos, de implementação bastante controlável;
- ◊ Os objetos de negócios podem ser modelados em conjunto com os especialistas de negócios e, conseguindo-se a uniformidade de notação e documentação, é possível que esses especialistas possam acompanhar boa parte da implementação do sistema, facilitando o controle do projeto e ajustes de expectativas junto aos usuários, pois os trabalhos podem ser visualizados;

- ◊ Os principais objetos de negócios estabelecidos durante a fase de análise constituem um meio de realizar estimativas mais precisas para o planejamento e o acompanhamento de implementação;
- ◊ Medir a complexidade de um objeto é facilitado devido a não só pelo fato de ser pequeno, mas por apresentar uma especialização bastante definida. A medida de complexidade de um objeto facilita a organização de um planejamento de implementação, tornando-o mais próximo da realidade;

Considerações devem ser feitas no que se refere às estimativas. Do ponto de vista gerencial, existe uma dificuldade no estabelecimento de prazos e recursos, principalmente neste ambiente de desenvolvimento. Com o uso da tecnologia orientada a objetos, isto pode ser facilitado: usando a modelagem de um processo de negócio a ser automatizado através de um conjunto de objetos de negócios, esses objetos são a base para estimar os prazos e recursos com maior precisão.

Como deve ser mostrado nos itens seguintes, qualquer sistema, por mais complexo que seja, pode ser sistematicamente subdividido de forma a obter uma estruturação de várias pequenas partições, através do uso de várias técnicas como a proposta no Caps.2 e 4. Cada partição, organizada como um conjunto de objetos de negócios, define uma estratégia de se lidar com a complexidade: esse objetos de negócios possibilitam o uso de abstração que separa a definição de processos de negócios dos aspectos de implementação, o que garante a especificação detalhada do sistema do ponto de vista do processo de negócio. A implementação computacional fica mais restrita à implementação dos objetos, que

se caracterizam por serem pequenos, especializados, bastante detalhados e de simples implementação.

5.3 Roteiros Gerenciais

Os roteiros gerenciais garantem o planejamento e o acompanhamento da execução de um projeto de desenvolvimento. Sugere-se que um planejamento deva ser obtido como decorrência de um particionamento sistemático de um sistema, primeiramente do ponto de vista de negócios e depois do ponto de vista técnico.

O acompanhamento de execução deve acontecer através da formalização de resultados intermediários que devem ser submetidos a avaliações formais de qualidade. Algumas métricas de apoio gerencial devem ser implantadas tanto no planejamento como na execução das atividades.

5.3.1 Particionamento de um sistema

Todo e qualquer particionamento deve ser realizado sobre o modelo de negócios do processo a ser automatizado. Para tanto, deve-se estabelecer meios para formalizar os processos de negócios detalhados ao nível que reflita o atual processo, ou seja, para cada tarefa do processo as entradas, as saídas, os mecanismos de controle de qualidade de cada tarefa, as ferramentas utilizadas e os operadores envolvidos. Este modelo deve ser aprovado formalmente por especialistas do processo de negócios, futuros usuários do sistema.

Particionar corresponde à identificação de partes autônomas do processo que podem ser automatizadas através de subsistemas de software. Esse particionamento pode ser sistematicamente aplicado até o nível adequado. No final, cada projeto pode ser ainda particionado segundo uma visão técnica. Por exemplo, dentro de uma aplicação cliente-servidor as camadas poderiam direcionar um particionamento, como por exemplo, uma subpartição específica de dados, outra poderia ser de apresentação e outra de lógica de negócios. O Cap.6 mostra de exemplo de um processo de particionamento.

5.3.2 Roteiro de planejamento e acompanhamento

O roteiro de planejamento deve ser obtido com base no particionamento obtido do sistema. Cada partição deve ser um item do cronograma do planejamento e a sua organização no tempo (se sequencial ou se em paralelo) depende da disponibilidade de recursos e interdependências das partições ou mesmo priorização estabelecida pelo usuário.

Cada partição é composta por um conjunto de subpartições. Neste instante deve-se elaborar uma estrutura preliminar de objetos de negócios, que pode ser realizado rapidamente com a ajuda de especialistas de negócios. Com base nos objetos de negócios, e nas experiências de desenvolvimentos anteriores, pode ser feita um memorial de estimativas por objetos de negócio, para então compor a estimativa global por partição, que por sua vez dará a estimativa final do sistema.

Ainda em tempo de planejamento, deve-se estabelecer alguns pontos no cronograma de acompanhamento, com base nos produtos intermediários

decorrentes de atividades de análise, projeto ou implementação que serão formalmente avaliados para o registro de andamento dos projetos. Deve-se estabelecer ainda uma política de acompanhamento do projeto por parte do cliente e também o processo de identificação de atrasos para o devido replanejamento do projeto.

5.4 Roteiros Técnicos de Engenharia

Os roteiros técnicos, num primeiro momento devem se adequar à forma de trabalho atual, porém não deve abrir mão de princípios básicos de engenharia de software, especialmente no que se refere aos passos básicos que garantem a obtenção de software com qualidade. Neste caso, a maior dificuldade está na preparação de pessoal, pois a introdução de tarefas formais são interpretados como elementos de atrasos. A conscientização acontece com estudos de casos reais que demostram as consequências perniciosas provocadas por falta de método, especialmente na qualidade do produto final gerado, como também na própria qualidade individual do trabalho, conscientizando-os sobre excesso de retrabalhos, falta de ferramentas, padrões de atividades, documentos e procedimentos.

5.4.1 Roteiro de análise

A análise, ao incorporar as técnicas de orientação a objetos, deve deslocar muito esforço de desenvolvimento, o que é uma característica da própria tecnologia. Uma ênfase especial deve ser dada aos processos de negócios, através da modelagem de objetos de negócios capazes de dar suporte às partições

especificadas no planejamento. Nessa situação, os aspectos de implementação devem ser deixados de lado, por ora. Porém, as lógicas de negócios e os atributos de informações devem ser formalizados nos objetos. A modelagem orientada a objetos deve ser completa, com os cenários dinâmicos que refletem a funcionalidade. A aprovação deve envolver os usuários, especialistas no processo de negócio. As métricas estão associados aos objetos de negócios.

5.4.2 Roteiro de projeto

O projeto deve mapear os objetos de negócios segundo a arquitetura de hardware e software onde será implantado o sistema. Além disso, deve incorporar o processo de componentização, pois é o caminho de se explorar a reutilização (Cap.2 e 4.) de forma sistemática.

No roteiro de projeto, os componentes são especificados como um conjunto de objetos para serem implementados sob encomenda, ou extraídos de um repositório de componentes reutilizáveis. Ou seja, procura-se utilizar um componente existente antes de implementar o componente provocando com isso um ganho em robustez, velocidade de desenvolvimento do sistema entre outras vantagens, conforme discutidas no Cap.4. Para isso, deve-se incorporar procedimentos de verificação sistemática de catálogos de componentes existentes para rever o projeto e tentar reutilizá-los, sem prejuízo do modelo de negócio obtido na análise. O Cap.6 mostra um exemplo de definição de roteiros de projeto que atende aos requisitos citados.

Aqui deve-se ressaltar um paralelismo de tarefas decorrentes da adoção da técnica orientada a objetos: Um ramo do paralelismo corresponde ao processo de implementar o sistema de software através da integração dos componentes de negócios, ou seja específico do processo de negócio; outro ramo de paralelismo é a da implementação dos componentes, cuja a especialização dos desenvolvedores é essencialmente técnica e genérica, pois recebem especificações técnicas que orientam a implementação mas que pouco trazem em relação aos detalhes de processos de negócios.

5.4.3 Roteiro de implementação e testes

O roteiro de implementação deve incorporar práticas atualizadas para implementar (ou reutilizar, de preferência) componentes e implementar os sistemas e subsistemas de software integrando tais componentes.

Neste roteiro, a adoção sistemática e efetiva de mecanismos de testes é um fator de aumento de qualidade e também ganho de tempo global no projeto (Cap.2. e Cap.4.). A estratégia é facilitada através do uso de orientação a objetos: o teste de implementação de objetos de um componente é facilitado considerando que cada objeto abriga pequenas e especializadas funcionalidades, portanto fáceis de testar e validar. O teste de integração de componentes é facilitado contando com um controle de qualidade na própria implementação dos componentes, ou seja, integrar componentes significa testar por métodos "caixa preta" (PRESSMAN, 1989; JACOBSON, 1995) de elementos previamente testados. Dessa maneira, a integração pode ser incremental acumulativa, ou seja, do processo de validação

de implementação dos objetos, passando pela validação de componentes e pela integração dos componentes até a validação final do próprio sistema de software.

5.4.4 Roteiro de integração

Este roteiro integra as várias subpartições elaboradas durante o planejamento como estratégia para o desenvolvimento do sistema. Para isso, considere-se que cada partição é um sistema independente em relação a outras partições, ou seja, do ponto de vista de cada partição, as demais partições são elementos/sistemas externos e a integração se restringe à compatibilização das informações trocadas entre elas.

5.5 Uso de Métricas para a Avaliação

O sucesso de um programa de medição depende da sua aplicação prática. A maioria de falhas ocorridas neste tipo de iniciativa é a dificuldade de realização prática das métrica e seu efetivo uso, conforme constata-se nas empresas que produzem sistemas de software (ARAKAKI, 1996a). Isso decorre de um planejamento deficitário das métricas, como falta de aplicabilidade prática e a ineficácia nos resultados obtidos como indicadores de apoio aos processos técnicos e gerenciais.

5.5.1 Escolha e definição de métricas

Um mecanismo simples e eficaz para estabelecer métricas é o que proposto por Basili e Rombach, denominada técnica GQM (Goals, Questions, Metric - Metas, Questões e Métricas), onde as métricas são planejadas, projetadas e

implementadas a partir de um foco adequado e direto sobre as metas desejadas para o projeto de medição conforme descrito no Cap.3.

5.5.2 Métricas de planejamento e controle

As métricas de planejamento e controle devem estabelecer:

- ◊ medidas indicadoras para apoiar os requisitos alocados ao sistema, e as mudanças que levam ao replanejamento;
- ◊ indicadores de atividades de planejamento como quantidade de atividades, prazos, complexidades, número de pontos de avaliação de resultados; e
- ◊ de acompanhamento, com indicadores da quantidade de revisões técnicas formais realizadas em relação ao planejado, quantidade de replanejamentos, com alteração de complexidades, custos, recursos e atividades alocadas, com medidas de sub-contratações de fornecedores terceirizados.

5.5.3 Métricas de análise

As métricas de análise devem estabelecer:

- ◊ medidas indicadoras complexidade de partições - através da quantidade de objetos de negócios envolvidos;
- ◊ medidas indicadoras de complexidade de objetos, através da indicação de pontos de função;

◊ medidas indicadoras de quantidade de produtos elaborados durante a fase de análise e também a quantidade de avalições técnicas formais realizadas sobre estes produtos.

5.5.4 Métricas de projeto

As métricas da fase de projeto devem estabelecer:

◊ medidas indicadoras do equilíbrio (ou alocação) de processamento entre os elementos de hardware, considerando a estrutura cliente-servidor;

◊ medidas indicadoras de quantidade totais de componentes especificados;

◊ medidas indicadoras do grau de reutilização, ou seja, quantidade de componentes reutilizados em relação aos componentes total;

◊ medidas indicadoras de quantidade de atividades de validação e testes planejados e realizados sobre os componentes;

◊ medidas indicadoras de tempos planejados e realizados.

5.5.5 Métricas de implementação

As métricas da fase de implementação devem estabelecer:

◊ medidas indicadoras de quantidade totais de componentes implementados;

◊ medidas indicadoras de quantidade de atividades de validação e testes realizados e registrados sobre os componentes;

◊ medidas indicadoras de revisões técnicas formais relacionadas com padrões e normas.

5.5.6 Métricas de testes

As métricas da fase de testes devem estabelecer:

◊ medidas indicadoras de quantidade totais de partições, subpartições integradas;

◊ medidas indicadoras de quantidade de classe de testes executadas para desempenho, volume, confiabilidade e integridade de processos de negócios;

◊ medidas indicadoras de revisões técnicas formais relacionadas com as especificações de requisitos alocadas ao projeto.

6. RESULTADOS EXPERIMENTAIS

Objetivo do Capítulo:

Apresentar um exemplo de aplicação da metodologia descrita no Cap. 5, abstraído de experiências práticas: incorporação de mecanismos de avaliação da qualidade e produtividade num processo produtivo de software.

Tópicos do Capítulo:

6.1. O Contexto do Exemplo

A aplicação do modelo descrito no Cap.5 adota um contexto muito típico da indústria financeira. Dessa maneira, considere um ambiente com características técnicas descritas nos Cap.2, 3 e 4.

Uma peculiaridade importante desse contexto é o grande poder de realização das equipes técnicas não importando o estágio em que se encontra o desenvolvimento do software. Um projeto, uma vez estabelecido como essencial ao processo de negócio, deve ser concretizado, independentemente dos atrasos, da quantidade de recursos, da gravidade dos problemas. Os atrasos quando acontecem são assimilados, procurando-se minimizá-los com alocação de recursos materiais e humanos, e/ou redefinindo-se os requisitos a serem implementados, que, portanto, são alterados durante o processo. E se ainda assim, houverem problemas técnicos no sistema resultante, e se esses problemas puderem ser contornados por procedimentos alternativos, manuais ou não, então o sistema entra em produção.

Essa capacidade de realização é uma virtude incorporada ao processo que, mesmo com um baixo nível de maturidade de produção, facilita a aplicação da metodologia apresentada no Cap. 5. Nesse processo, geralmente, um modelo válido é uma caixa preta de um passo único: A entrada é um pedido simplificado, incompleto, porém importante e/ou urgente. A saída é o sistema solicitado.

6.2. Diagnóstico Global

Conforme descrito no Cap. 5, a primeira etapa da metodologia é fazer o diagnóstico do ambiente. Assim, vale uma análise da situação geral. Na situação original, os projetos eram conduzidos de forma não sistemática e dependentes das pessoas envolvidas. Equipe com pessoas muito boas tecnicamente apresentava sistema com nível de qualidade bom. Equipe fraca tecnicamente, apresentava sistema com nível de qualidade inferior. A avaliação da qualidade baseia-se em parâmetros de usabilidade e confiabilidade. A situação pode ser resumida nos seguintes pontos:

- ◊ Pessoas: Líderes de projeto, gerências administrativas e técnicas têm ciência da necessidade de aprimorar conceitos. Equipes técnicas fortes nos processos de negócios e na criticidade, porém, fraca em bagagem relativa a princípios de engenharia de software, apresentando, dentro do universo do processo a seguinte estatística (Arakaki, 1996a): pessoas com conhecimento específico de linguagem de codificação - cerca de 50%, pessoas capacitadas em executar atividades de análise - cerca de 5%, e de projeto - cerca de 1%. O restante das pessoas com conhecimentos específicos do processo de negócio;
- ◊ Tecnologia: Estrutura básica em três camadas (conforme modelado no Cap. 4), ferramentas gerenciadores de base de dados e ferramentas de front-end com linguagens de 4ª geração (100% das aplicações). Alguns desenvolvedores (~40%) acreditavam que o conhecimento técnico em ferramentas garantiria a qualidade. As gerências e os líderes de equipe já são cientes desta crença equivocada. ;

◊ Processo: Processo de desenvolvimento sem transparência das atividades internas. Algumas deficiências comuns identificadas no processo:

- Definição de cronograma não atrelado a planejamento adequado;

- Acompanhamento e controle dos projetos deficitários, sem ferramentas para tal;

- Avaliação de produtos intermediários inexistente;

- Planos de testes inexistente;

- Dificuldade de homologação e validação de sistemas, com retrabalhos em grande amplitude.

◊ Produtos: Dificuldade na validação do produto de software final, devido a discrepâncias entre as expectativas dos usuários e as apresentadas, especialmente quanto aos requisitos funcionais implementados, exigindo muito retrabalho de codificação. Outros pontos importantes:

- Interface de interação com o usuário estruturalmente diferente para cada sistema, apesar da arquitetura das aplicações serem iguais ;

- Ausência de boa estruturação interna dos sistemas, uma vez que a prática utilizada é aquela em que se busca as atividades de codificação muito rapidamente, prematuramente;

- Dificuldade de manutenção corretiva ou evolutiva, decorrente de estruturação deficitária, de falhas nas etapas de análise de negócios e de análise do projeto, tornando difícil a implementação para correção, tirando-lhe a flexibilidade necessária para a evolução.

Ambientes com tais características no desenvolvimento de software podem ser categorizados como do nível 1 de maturidade do SEI/CMM, numa situação quasi-caótica de desenvolvimento.

6.2.1 Classificação de projetos

Neste ambiente, classificou-se os projetos, segundo três elementos: a sua natureza, a sua classe e o seu perfil, definidos nos itens que se seguem.

6.2.1.1 Natureza do Projeto

A natureza de um projeto é determinada de acordo com a maneira que se pretende obter o sistema de software. Assim, o projeto pode ser classificado como de:

Desenvolvimento

A ênfase é construir *software* novo. Nesse caso, o roteiro deve conter o conjunto de recomendações e procedimentos para orientar o desenvolvimento de aplicações, que envolvam concepção, projeto e codificação de programas e/ou integração de componentes de *software*;

Manutenção

A ênfase é alterar sistemas que estejam em operação, sanando erros detectados ou adequando a novas necessidades de negócios. A manutenção é fundamentalmente um trabalho pequeno, que pode ser realizado com um roteiro de trabalho simplificado. Uma manutenção ampla, pode exigir condução do projeto como um projeto de desenvolvimento; e

Aquisição

A ênfase é a aquisição de *software* disponíveis no mercado. Essa aquisição pode ser parte de um projeto de desenvolvimento, com ou sem customizações.

Cuidados especiais devem ser tomados no levantamento de alternativas: seleção através de criteriosos *benchmarks*, e customizações para sua perfeita integração.

O roteiro deve conter um conjunto de recomendações e procedimentos que orientam a especificação, avaliação qualitativa e quantitativa, seleção, fornecimento, alteração e adaptação, implantação e a operação destes sistemas de *software*.

6.2.1.2 Classe do Projeto

A classe de projeto reflete a criticidade referente a prazos, podendo ser urgente ou normal.

Normal

É um projeto que pode ser executado dentro de um prazo considerado normal, não emergencial, ou seja, todas as atividades, do planejamento à homologação, podem ser executadas completamente dentro de um padrão de tempo considerado rotineiro, tradicional, normal.

Emergencial

É um projeto que possui prazo emergencial, com exigências de tempo de desenvolvimento muito reduzidas para entrar em produção, é urgente e fora dos padrões rotineiros. Por exemplo, freqüentemente, o governo faz exigências legais

que impactam os sistemas, e o prazo de entrada em vigor é determinado e normalmente urgente, quando não imediato. O projeto emergencial tem prazos considerados urgentes e curtos quando comparados a situações e procedimentos padrões, tradicionais. Nesse caso exige a adoção de atividades não normais para a sua execução.

6.2.1.3 Perfil do Projeto

A arquitetura de sistemas Cliente-Servidor pode ser estruturada em três camadas distintas de software, conforme mostrado no Cap. 4, e novamente esquematizado na Fig.6.1.

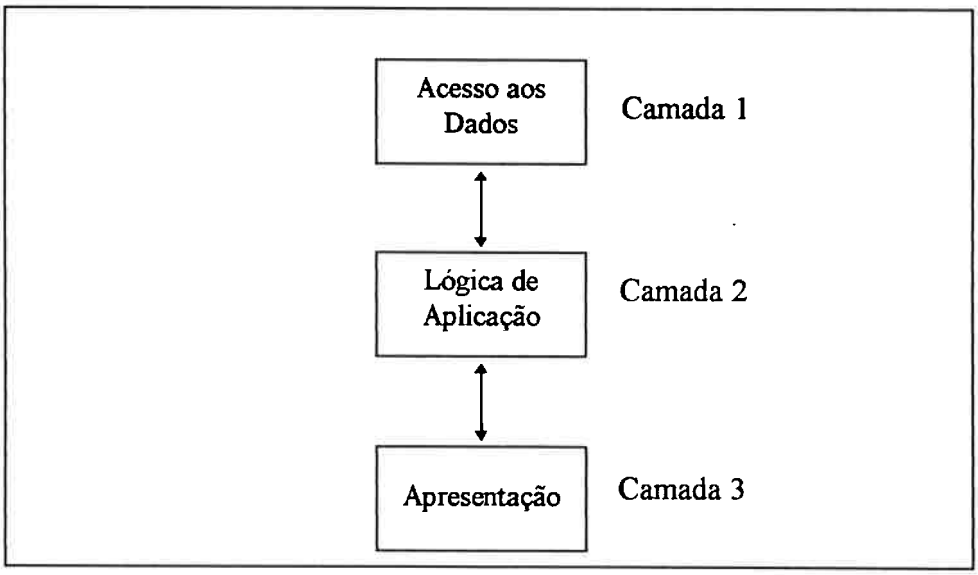

Fig.6.1. Arquitetura Cliente-Servidor em três camadas.

Acesso aos Dados

A camada de acesso aos dados se localiza em servidores e normalmente dispõe de uma ferramenta gerenciadora de base de dados. As atividades de

desenvolvimento desta camada se referem, fundamentalmente, ao modelamento dos dados e ao desenvolvimento de rotinas específicas (*Stored Procedures*) diretamente associadas, aos dados. Aqui são considerados aspectos importantes como a segurança de acesso e compartilhamento.

Lógica de Aplicação

A camada de lógica de aplicação (lógica de dados e de negócios) pode se localizar tanto no servidor, quanto no cliente, ou em ambos - tudo depende da arquitetura adotada e da existência de infra-estrutura (como o *middleware*) que a viabilize.

Apresentação

A camada de apresentação deve ser tratada de forma específica na arquitetura Cliente-Servidor. Ela se caracteriza por utilizar, em geral, mecanismos avançados de interação homem-máquina, usando recursos gráficos e dispositivos de entrada e saída cada vez mais ergonômicos como *mouse*, telas *touch screen*, placas gráficas de alta resolução e muitas cores, impressoras de alta resolução e ambientes gráficos consagrados (MOTIF®, WINDOWS®). É aqui que se dá a comunicação com o operador. Neste ambiente, a padronização da camada de apresentação foi fortemente considerada e uma das primeiras atividades de organização realizada (ARAKAKI, 1991).

6.3. Controle Gerencial

A ausência de controle e/ou planejamento falho são característicos dos ambientes com maturidade similar ao deste exemplo. Planejamento, controle e acompanhamento do projeto tornam-se difíceis pelo fato de os produtos e as atividades intermediárias das equipes de desenvolvimento estarem praticamente invisíveis, obscuras para avaliação.

6.3.1 Situação original

Na situação original, o planejamento consistia da montagem direta do cronograma e da alocação de recursos com base em documentos contendo registros das necessidades de negócio, emitidos pelas áreas cliente (usuária), sendo que tal prática era realizada não importando o porte do sistema.

O acompanhamento e o controle eram difíceis devido à dificuldade de se identificar produtos intermediários que atestassem as evoluções do processo. Os resultados negativos desta forma de operar eram sentidos no momento do replanejamento. Por exemplo, em caso de atrasos, a situação era tardiamente detectada, tornando a negociação com o cliente muito difícil, contribuindo assim para degenerar as relações com esses mesmos clientes, que se viam obrigados a aceitar os atrasos e com poucas alternativas de negociação.

6.3.2 Planejamento e controle de projetos

Reorganizar um processo de software tem como pré-condição o estabelecimento de uma sistemática para planejar, acompanhar e controlar as atividades, os

produtos do projeto, os aspectos técnico-administrativo de um projeto. Para isso, propõe-se uma maneira simples e adequada para o ambiente em questão.

6.3.2.1 A Necessidade de um Roteiro

O primeiro passo para o planejamento é estabelecer os recursos, as atividades e os prazos (acordados com o usuário/cliente) de acordo com os objetivos do projeto. O planejamento é de responsabilidade da gerência e deve ser elaborado de forma sistemática e adequada com a adoção de formatos padronizados.

Em alto nível, vale o seguinte fluxo: o desenvolvimento recebe uma solicitação de projeto, acompanhada de um conjunto de documentos que contêm registros sobre necessidade de negócios, com apontamentos de urgências e criticidades.

Após estudos de alternativas e confirmação dos requisitos de projeto junto aos clientes, deve-se elaborar um particionamento do sistema proposto, também estabelecer graus de prioridades de desenvolvimento das partições, sempre tomando por base os requisitos de negócios (cliente). Cada partição deve ser planejada e executada seguindo um roteiro, de modo que, ao final, o sistema esteja completo com a liberação de todas as partições.

6.3.2.2 Estratégia de Projeto - Partições

Os requisitos de negócios devem orientar a criação de partições. Cada partição será implementada seguindo um roteiro, determinado pela gerência de acordo com a respectiva caracterização técnica. O particionamento é essencial ao planejamento e controle, pois atende a dois objetivos distintos:

◊ As partições devem ser coerentes do ponto de vista do negócio, para que o usuário visualize-as como sub-conjuntos de requisitos de negócios. Implementações de partições geram sistemas autônomos do ponto de vista de negócios, e podem entrar em operação assim que liberadas;

◊ As partições devem ser suficientemente pequenas para que o planejamento seja mais preciso, um acompanhamento mais rigoroso e um controle mais eficiente das atividades do projeto.

Assim, a realização do planejamento de um projeto deve seguir a dinâmica esquematizada na Fig.6.2. Para isso, o planejador deve realizar as seguintes atividades, sistematicamente:

1 - Com base nos registros de necessidades de negócios e com a modelagem de negócio adequada, organiza-se o particionamento do sistema. Cada partição se torna uma linha de cronograma, refletindo a estratégia de implementação por partes. Todas as informações podem ser documentadas no planejamento do projeto.

2 - As partições obtidas devem ser consolidadas segundo a Natureza, Classe. Com isso, seleciona-se o roteiro a ser adotado para a obtenção de cada partição;

3 - O sistema é integrado como combinação dos vários Sistemas i obtidos das partições. Novamente, o sistema integrado pode ser submetido aos processos de homologação e operação.

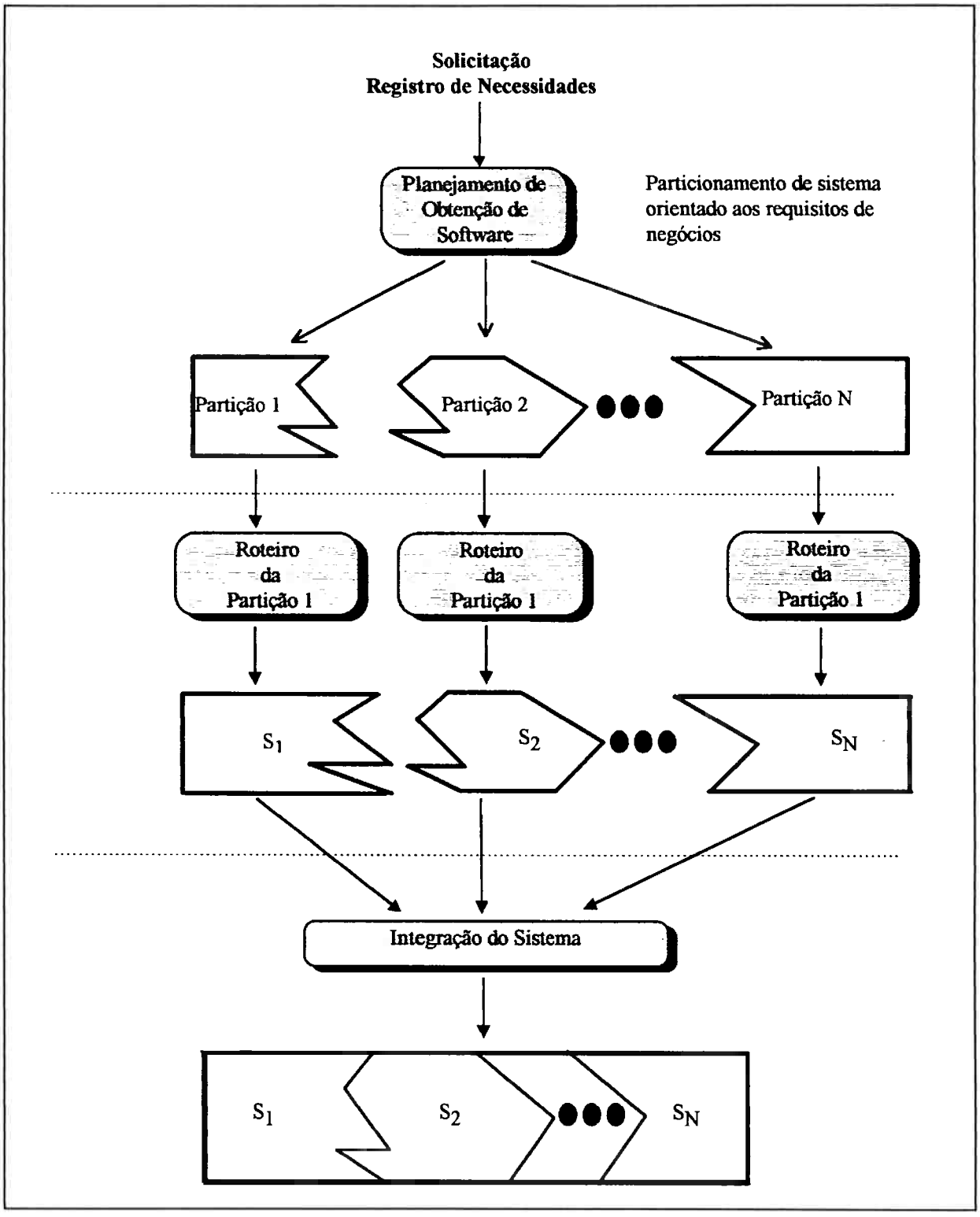

Fig.6.2 - Estratégia de projeto de um sistema organizada em N partições.

6.3.2.3 Roteiro de Planejamento e Controle

O planejamento e controle, conforme apresentado na Fig.6.3, pode ser executado seguindo-se os passos:

a - Particionamento do projeto: elabora-se o modelo de processos de negócios (com a eventual complementação de informações do registro de necessidades), usando a notação IDEF0 (PRESSMAN, 1988), produzindo daí o fluxo de informações e atividades do processo de negócio a ser automatizado.

O particionamento do projeto em segmentos de *fluxos de processos (workflow)* deve ser consolidado para a definição de sub-sistemas de implementação (partições), usando como referência inicial o registro de necessidades de negócio.

b - Elaborar um plano de obtenção de software: os sub-sistemas são organizados numa lista de priorização onde os mesmos são alocados em ordem de prioridade de implementação computacional, como parte de um plano de obtenção global do *software*, usando as informações necessárias para a elaboração deste plano.

c - Estimar o esforço de desenvolvimento da partição: para cada partição, o esforço de desenvolvimento é estimado de acordo com base histórica (pontos de função). Se não houver essa base, a equipe deve se apoiar em experiências de projetos anteriores, até montar uma consistente com as características do ambiente.

d - Elaborar plano de desenvolvimento da partição: para elaborar o plano de desenvolvimento, alguns parâmetros devem ser considerados como: natureza, classe e perfil do desenvolvimento, equipe, *time-box* e número de sub-partições.

e - Ajustar o cronograma de desenvolvimento: Após a elaboração do cronograma inicial, alguns ajustes do que foi planejado podem ser necessários. Deve ser elaborado um Sumário Gerencial, que resume o processo de planejamento para o controle dos níveis superiores de administração do projeto.

f - Replanejamento do projeto: O andamento do projeto deve ser aferido nos seus marcos de referência, conforme apontados no cronograma. Nos marcos de referência, uma revisão de projeto (*formal review*) deve ser feita através de Reuniões Técnicas Formais (RTF). O resultado da revisão de projeto pode indicar aceitação ou não aceitação do produto revisado. Caso o produto não seja aceito, e sejam necessárias modificações, um replanejamento do projeto pode ser necessário. O replanejamento, como uma atividade gerencial, deve ser feito independente das conclusões da RTF. A cada ciclo de replanejamento deve-se gerar o respectivo Sumário Gerencial.

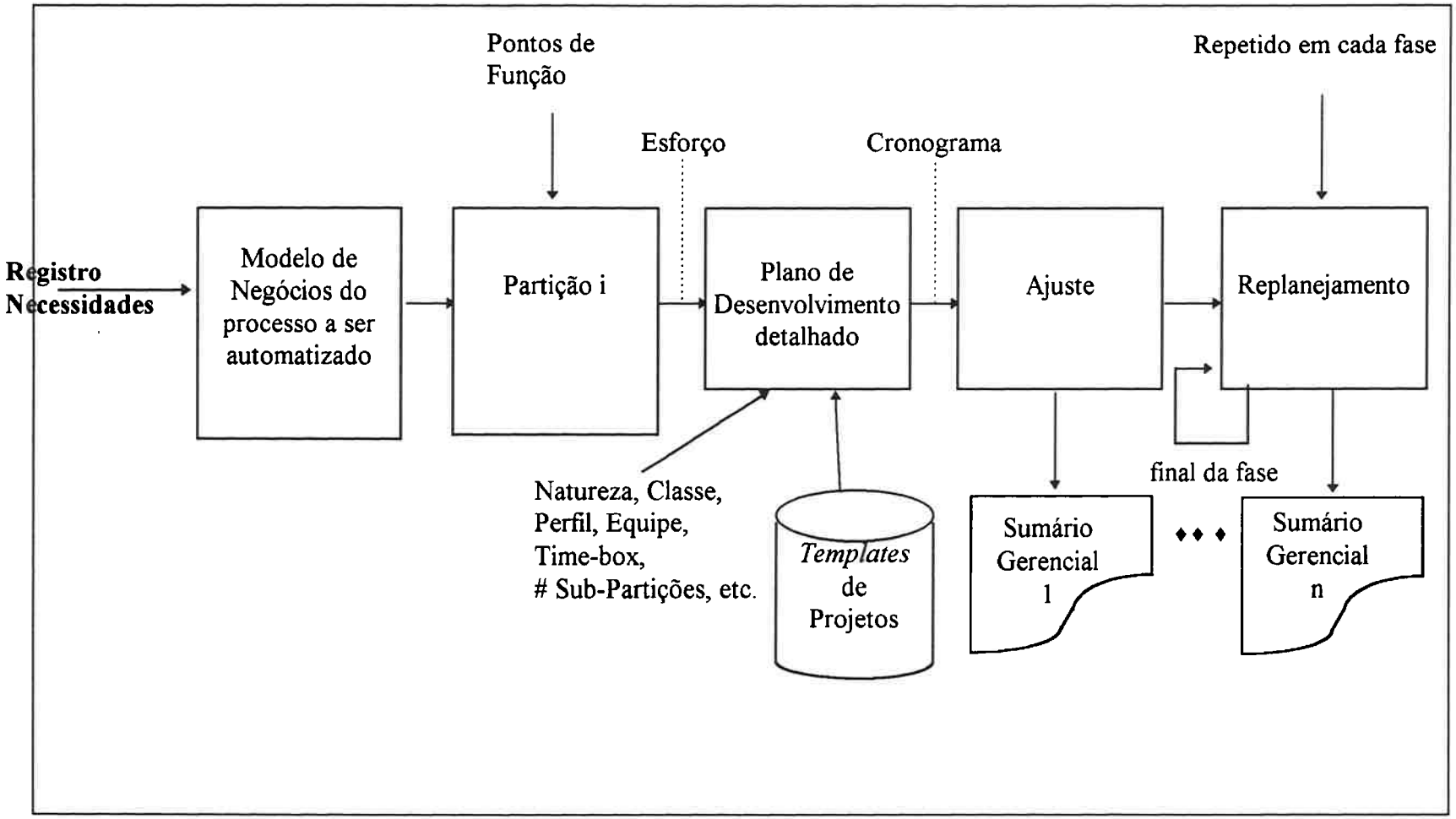

Fig.6.3 - Ciclo de Planejamento e Controle de Projetos.

6.3.3 Revisão técnica formal

A revisão técnica formal (RTF) é um instrumento de qualidade efetiva a ser aplicada na prática em todos as fases de um projeto. Uma maneira de implementá-la é realizar a avaliação dos produtos através de revisão feita por pessoas que não participam do processo (PRESSMAN, 1988). Esse mecanismo de revisão foi apresentado no Cap.2. e é inegável a sua importância como meio de detecção de problemas de projetos. Lembrando sempre que uma falha descoberta e corrigida em tempo de análise e projeto custa 100 (cem) vezes menos do que no período de implementação física (McCONNELL, 1996).

Existem duas maneiras de se aferir a qualidade de *software*, através de testes ou por revisões técnicas formais. Os testes permitem a detecção de erros através da execução dos programas. As revisões técnicas formais permitem a detecção precoce dos erros na especificação do *software* através de reuniões, durante o desenvolvimento dos trabalhos.

Os objetivos de uma RTF são descobrir erros em função, lógica ou implementação para qualquer representação de *software*, verificar se o que está sendo revisado atende aos seus requisitos, assegurar que o *software* está sendo representado de acordo com padrões pré-definidos, obter um *software* que seja desenvolvido uniformente e tornar os projetos mais gerenciáveis.

Participam desta revisão, o Produtor (analistas de sistemas e programadores) que irá submeter o seu produto a uma revisão, o(s) Revisor(es) (pessoas com perfil técnico que, idealmente, não participaram do desenvolvimento) que irá (ão) formular questões sobre o produto e um Líder da Revisão que é o responsável para dar um bom andamento à RTF.

O produto, objeto da reunião, deve ser uma parte bem específica do *software* global, para que a RTF seja produtiva. Podemos ter várias sessões de RTF. Como resultado da RTF pode ocorrer a aceitação do produto sem qualquer alteração, a rejeição do produto devido a erros graves ou a aceitação provisória devido a pequenos erros que deverão ser sanados.

As decisões tomadas na revisão devem ser anotadas num Relatório de Revisão Técnica Resumido. Opcionalmente, deve-se gerar uma Lista de Questões que poderá ser utilizada para implementar as modificações necessárias no produto.

Essa revisão formal é concretizada no planejamento através de marcos programados nos cronogramas, a cada vez que houver um evento significativo como a liberação de um produto intermediário, ou término de atividades.

6.3.4 Planejamento e validação de métricas

As métricas a serem consideradas na fase de planejamento e controle relacionadas a seguir devem ser coletadas na fase inicial da reorganização dos processos com a

finalidade básica de montar uma base de dados de referência para estimativas de produtividade, complexidade, e esforços necessários para execução de atividades.

As métricas listadas a seguir podem ser coletadas, uma vez adotada a sistemática de planejamento e controle, conforme descrito:

- ◊ medidas indicadoras dos requisitos alocados ao sistema, como as mudanças que levam ao replanejamento;
- ◊ indicadores de atividades de planejamento como quantidade de atividades, prazos, complexidades, número de pontos de avaliação de resultados; e
- ◊ de acompanhamento, com indicadores da quantidade de revisões técnicas formais realizadas em relação ao planejado, quantidade de replanejamentos, com alteração de complexidades, custos, recursos e atividades alocadas, com medidas de sub-contratações de fornecedores terceirizados.

6.4. Roteiros Técnicos de Engenharia

Os roteiros técnicos de engenharia são básicos para a operacionalização da Metodologia de Desenvolvimento, Aquisição e Manutenção de *Software*. Apresenta-se, a nível gerencial e técnico, um modo sistemático de atendimento de uma solicitação de desenvolvimento na forma de um projeto de *software*. Para tanto, utiliza-se dos roteiros possíveis, de acordo com a caracterização dos projetos, as fases que os compõem e os produtos de cada uma delas.

Apresenta-se, ainda, as alternativas que a alta gerência técnica/administrativa e os usuários dispõem para promover a execução de um projeto de *software* desde a montagem do planejamento inicial de um projeto, o seu particionamento, a caracterização de cada módulo resultante até a adoção dos roteiros correspondentes para a sua completa execução.

Os roteiros de obtenção de *software* estão classificados primariamente de acordo com a natureza do projeto: desenvolvimento, aquisição ou manutenção. As classificações (normal ou emergencial) e os perfis (GUI, banco de dados ou rotinas de negócios) aparecem influindo nos produtos obtidos nas fases previstas dentro dos roteiros.

Cada um dos roteiros apresenta partes constituídas de fases, atividades e produtos a serem elaborados e que marcam a realização dos trabalhos.

A seguir, é apresentado cada um dos roteiros e suas respectivas tabelas que detalham fases, atividades e produtos de cada roteiro.

6.4.1 Roteiro de aquisição

A aquisição de módulos de *software* pode ser realizada através de dois roteiros alternativos (Fig.6.4). O primeiro ocorre quando a aquisição é emergencial, ou seja, o pacote de *software* deve ser adquirido num prazo muito curto. O outro

roteiro é o caso normal, por ter prazos não emergenciais, contempla toda as fases de aquisição.

Se um *software* exigir trabalho de customização, a sua aquisição não pode ser emergencial. Os produtos gerados e as responsabilidades em cada fase podem ser vistas na Tab.6.1.

6.4.1.1 Fluxo de Aquisição de *Software*

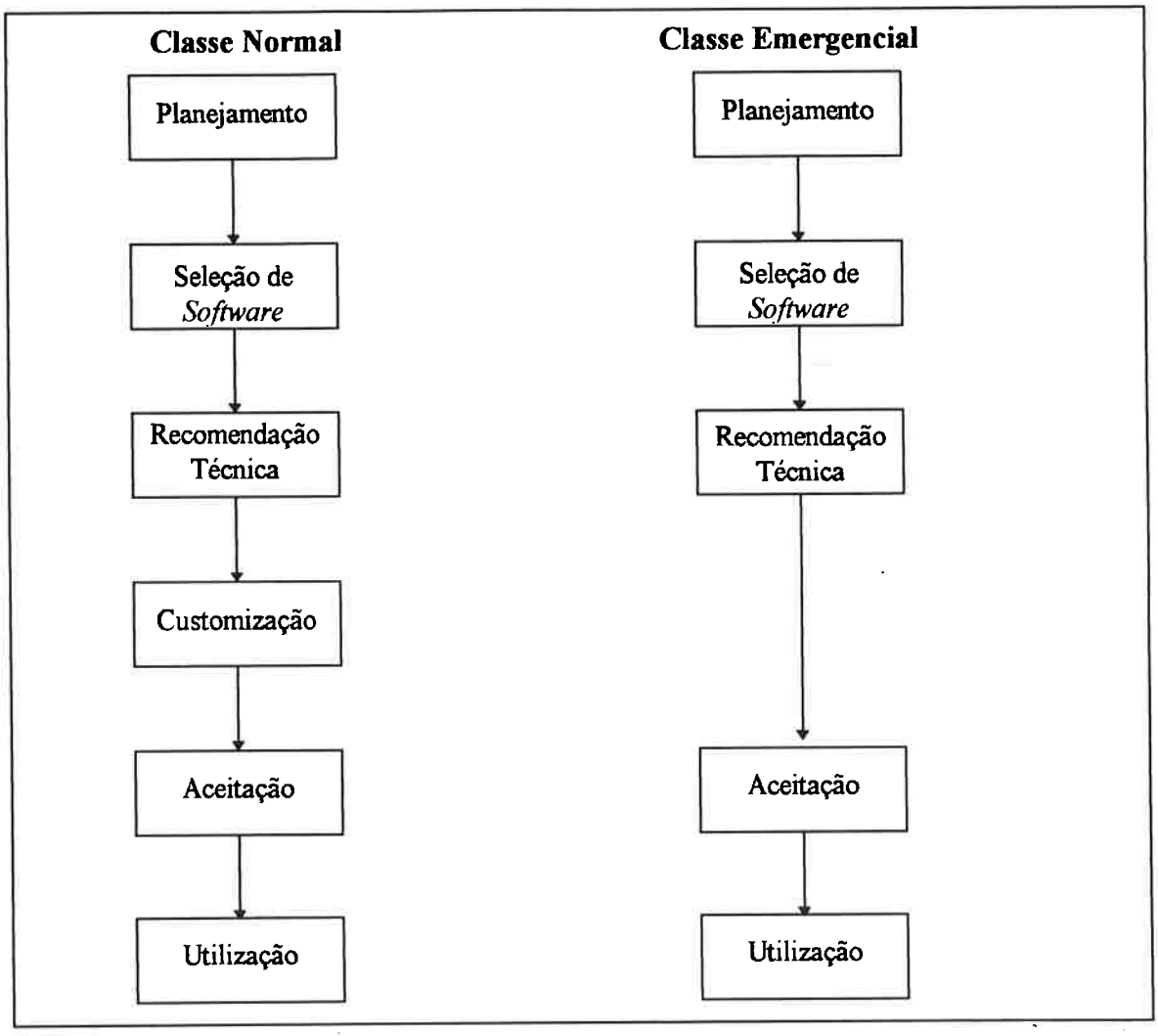

Fig.6.4. Roteiros de Aquisição de Software Normal e Emergencial.

6.4.1.2 Tabela do Roteiro de Aquisição de Software

N.°	Fase	Atividades	Documentos	Classe	
				Normal	Emerg.
1	Planejamento	• Levantar lista de requisitos preliminar	• Lista de Requisitos Preliminar	▨	
		• Avaliar as alternativas em termos técnicos e custos	• Estudo de Alternativas de *Software* .	▨	▨
		• Fazer pesquisa de fornecedores • Elaborar lista de requisitos definitiva	• Lista de Requisitos Definitiva	▨	▨
2	Seleção de *Software*	• Verificar atendimento aos requisitos • Realizar *Benchmarks*	• Relatório de *Benchmarks*	▨	
		• Elaborar plano de aceitação • Elaborar especificação de testes de aceitação	• Plano de Aceitação	▨	▨
3	Recomendação Técnica	• Elaborar recomendação técnica para aquisição de *software*	• Recomendação Técnica	▨	▨
4	Customização	• Realizar configuração do *software* para o seu uso	• Relatório de Configuração do *Software*	▨	
5	Aceitação	• Realizar teste de aceitação conforme plano de aceitação • Elaborar relatório de aceitação	• Relatório de Testes de Aceitação	▨	▨
6	Utilização	• Entregar *software* para o usuário • Entregar manuais	• Entrega de *Software*	▨	▨
		• Realizar treinamento do usuário	• Relatório de Treinamento	▨	

Tab.6.1 - Roteiro de Aquisição de Software.

Obs. 1: Na coluna Classe, o hachurado indica que os produtos e as respectivas atividades valem para aquela classe. Caso esteja em branco, os produtos correspondentes são dispensáveis.

Obs. 2: A coluna documentos indica o conjunto de documentos mínimo e, portanto, obrigatório a serem produzidos pela respectiva atividade, de acordo com a classe: normal ou emergencial (crítico).

6.4.2 Roteiros de desenvolvimento

O desenvolvimento de módulos de *software* pode ser realizado seguindo-se o roteiro da Fig.6.5. Podemos seguir este roteiro para as duas classes de projetos definidos anteriormente (normal e emergencial). Essa classificação e a definição de atividades correspondentes são de responsabilidade da gerência do projeto.

Deve ser ressaltado que sistemas desenvolvidos emergencialmente, sem uma definição de atividades, correm o risco de se tornarem muito fracos em termos de documentação de projeto e análise. Se o sistema assim obtido, mais tarde exigir manutenção de grande porte, um trabalho intenso de levantamento da documentação a partir da sua codificação (engenharia reversa) será necessário.

O roteiro emergencial objetiva definir formas de desenvolvimento acelerado. No entanto, exige-se que itens mínimos e importantes de documentação sejam elaborados numa fase posterior.

6.4.2.1 Fluxo de Desenvolvimento de *Software*

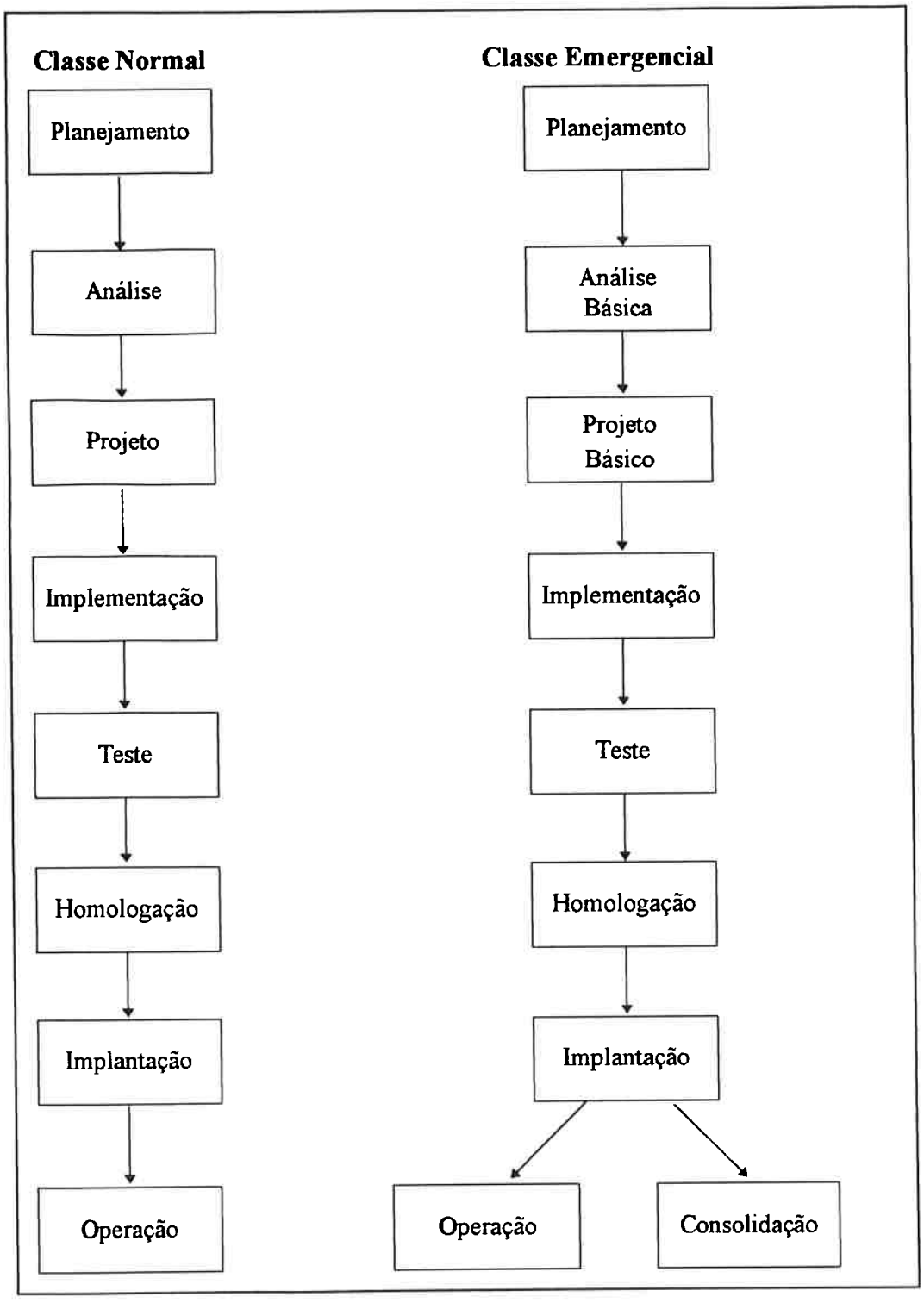

Fig.6.5 - Roteiros de Desenvolvimento de Software Normal e Emergencial.

6.4.2.2 Sub-Particionamento

O roteiro de desenvolvimento normal pode ser executado através da técnica de sub-particionamento (veja a Fig.6.6).

Uma atividade importantíssima, referente a esta técnica, está na fase de Análise. Aqui, o software a ser elaborado é planejado em sub-partições que serão incrementalmente evoluídas até a obtenção completa dos requisitos previstos para o sistema, de comum acordo com os usuários. Lembrar que nessa situação, cada sub-partição é liberada apenas para avaliação pelo usuário, nunca para operação em produção. Somente no final o sistema estará com todos os requisitos especificados para se submeter ao processo de validação completa.

6.4.3 *Roteiro de desenvolvimento usando Sub-Particionamento*

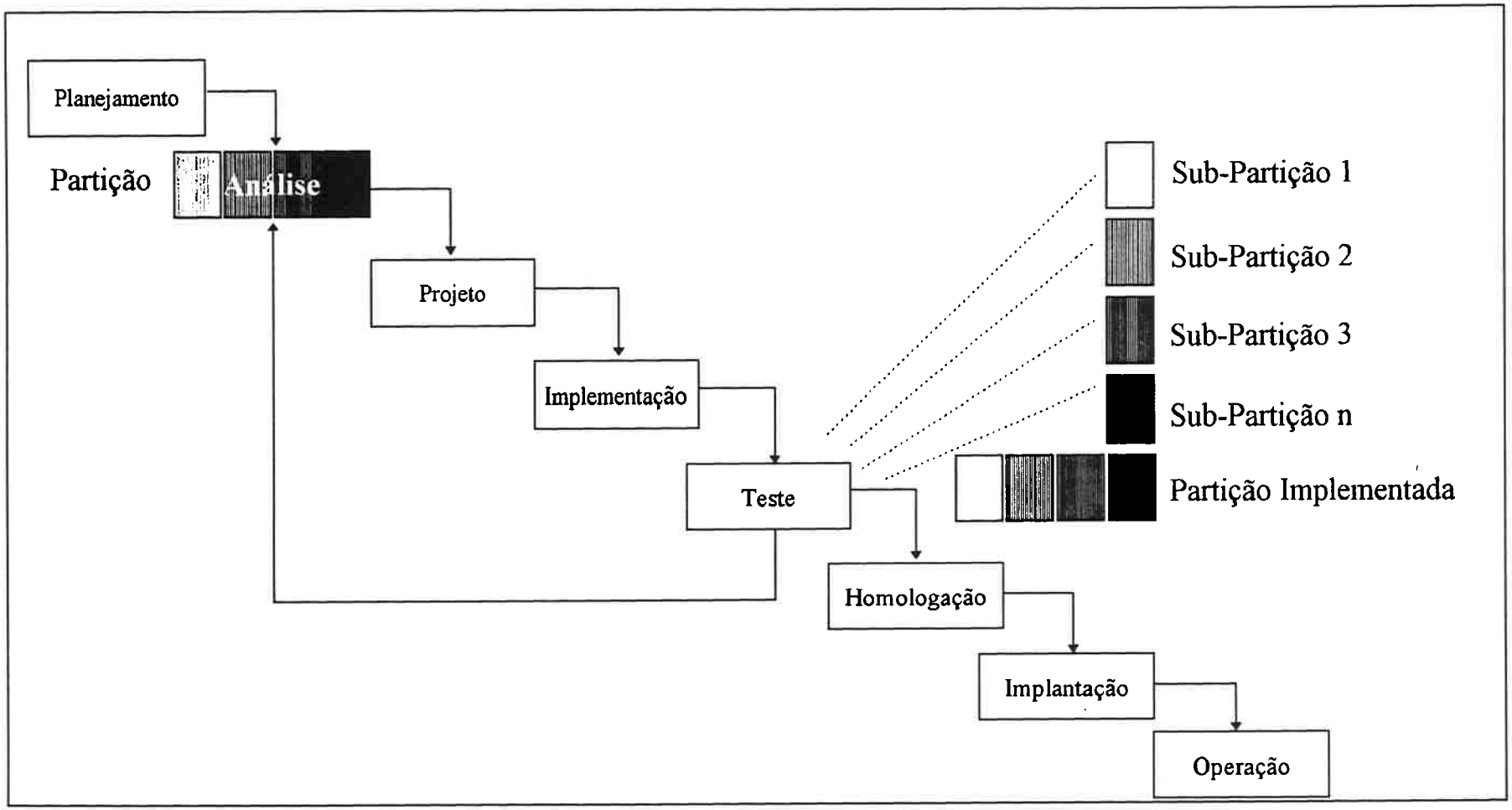

Fig.6.6 - Dinâmica de Aplicação da Técnica de Sub-Particionamento.

6.4.3.1 Tabela do Roteiro de Desenvolvimento de *Software*

N.º	Fase	Atividades	Documentos Finais	Classe	
				Normal	Emerg.
1	Planejamento	• Definir alternativas de *software* • Avaliar as alternativas em termos técnicos e de custos • Levantamento de Informações	• Estudo de Alternativas de *Software*		
		• Elaborar Protótipo de Especificação	• Protótipo de Especificação		
		• Elaborar modelos de processos de negócio • Elaborar modelo objeto preliminar • Particionar o sistema • Priorizar o desenvolvimento das partições • Elaborar cronograma de desenvolvimento	• Plano de Obtenção de *Software*		
2	Análise	• Elaborar Modelo Objeto • Elaborar Modelo Dinâmico • Elaborar Modelo Funcional • Elaborar Especificação de Classes	• Especificação de Requisitos de *Software*		
		• Elaborar plano de validação do aplicativo	• Plano de Validação do Aplicativo		
3	Análise Básica	• Elaborar Modelo Objeto • Elaborar Modelo Dinâmico • Elaborar Especificação de Classes	• Especificação de Requisitos de *Software* Preliminar		
		• Elaborar plano de validação do aplicativo	• Plano de Validação do Aplicativo		
4	Projeto	• Especificar arquitetura do aplicativo, composto de componentes de mercado e de negócio	• Especificação de Projeto		
		• Elaborar especificação de componentes	• Especificação de Componentes		
		• Elaborar plano de testes de validação de componentes • Elaborar plano de testes de integração de componentes ao aplicativo	• Plano de Testes de Validação e Integração de Componentes		

(continuação)

N.º	Fase	Atividades	Documentos Finais	Classe	
				Normal	Emerg.
5	Projeto Básico	• Especificar arquitetura preliminar do aplicativo, composta de componentes de mercado e de negócio	• Especificação de Projeto Preliminar		
		• Elaborar especificação de componentes	• Especificação de Componentes		
		• Elaborar plano de teste de validação de componentes • Elaborar plano de testes de integração de componentes ao aplicativo	• Plano de Testes de Validação e Integração de Componentes		
6	Implementação	• Gerar código-fonte • Integrar componentes ao aplicativo	• Listagem de Programa		
		• Elaborar manual de geração de código	• Manual de Geração de Código		
7	Teste	• Realizar testes de integração dos componentes e de validação do aplicativo	• Registro de Testes		
8	Homologação	• Realizar testes de homologação • Elaborar relatório de testes de homologação	• Termo de Homologação		
		• Elaborar manuais	• Manual de Operação • Manual de Sistema • Manual de Manutenção • Manual de Treinamento		
		• Elaborar Guia de Referência para Implantação e Produção do Aplicativo	• Guia de Referência para Implantação e Produção do Aplicativo		
9	Implantação	• Realizar treinamento do usuário	• Relatório de Treinamento		
		• Instalar o aplicativo • Realizar operação assistida	• Relatório de Implantação		

(continua)

(continuação)

N.°	Fase	Atividades	Documentos Finais	Classe	
				Normal	Emerg.
10	Consolidação	• Elaborar documentação de sistema	• Especificação de Requisitos de *Software* • Plano de Validação do Aplicativo • Especificação de Projeto • Especificação de Componentes • Plano de Testes Validação e Integração de Componentes • Manual de Geração de Código • Manual de Operação • Manual de Sistema		
11	Operação	• Entregar *software* para o usuário • Entregar manuais	• Entrega de *Software*		
		• Operar o sistema	• Relatório de Operação		

Tab.6.2 - Roteiro de Desenvolvimento de Software.

Obs: Na coluna Classe, o hachurado indica que os produtos e as respectivas atividades valem para aquela classe. Caso esteja em branco, os produtos correspondentes são dispensáveis.

6.4.4 Roteiro de manutenção

Os tipos de manutenção de software possíveis são corretiva e evolutiva. Entende-se como corretiva a manutenção realizada para corrigir divergência de funcionalidade em relação ao especificado para o sistema (funcionamento incorreto). A manutenção evolutiva inclui adaptação do sistema em novos ambientes ou melhoria da funcionalidade existente.

Existem para a Manutenção de Software, dois roteiros possíveis. O primeiro está relacionado com as manutenções emergenciais, que se caracterizam por ter um prazo muito curto, e o segundo é voltado aos sistemas normais, conforme a Fig.6.7.

Como a manutenção emergencial concentra-se na Implementação (geração de código-fonte), ela perde pela precariedade dos instrumentos utilizados no seu roteiro. Neste caso, pode ser necessário um trabalho intenso de consolidação de documentos. Esta consolidação, ao ser realizada, evita a necessidade de atividades de engenharia reversa para levantamento da documentação.

A decisão e risco na definição do roteiro a ser adotado são de responsabilidade dos gerentes.

6.4.4.1 Fluxo de Manutenção de *Software*

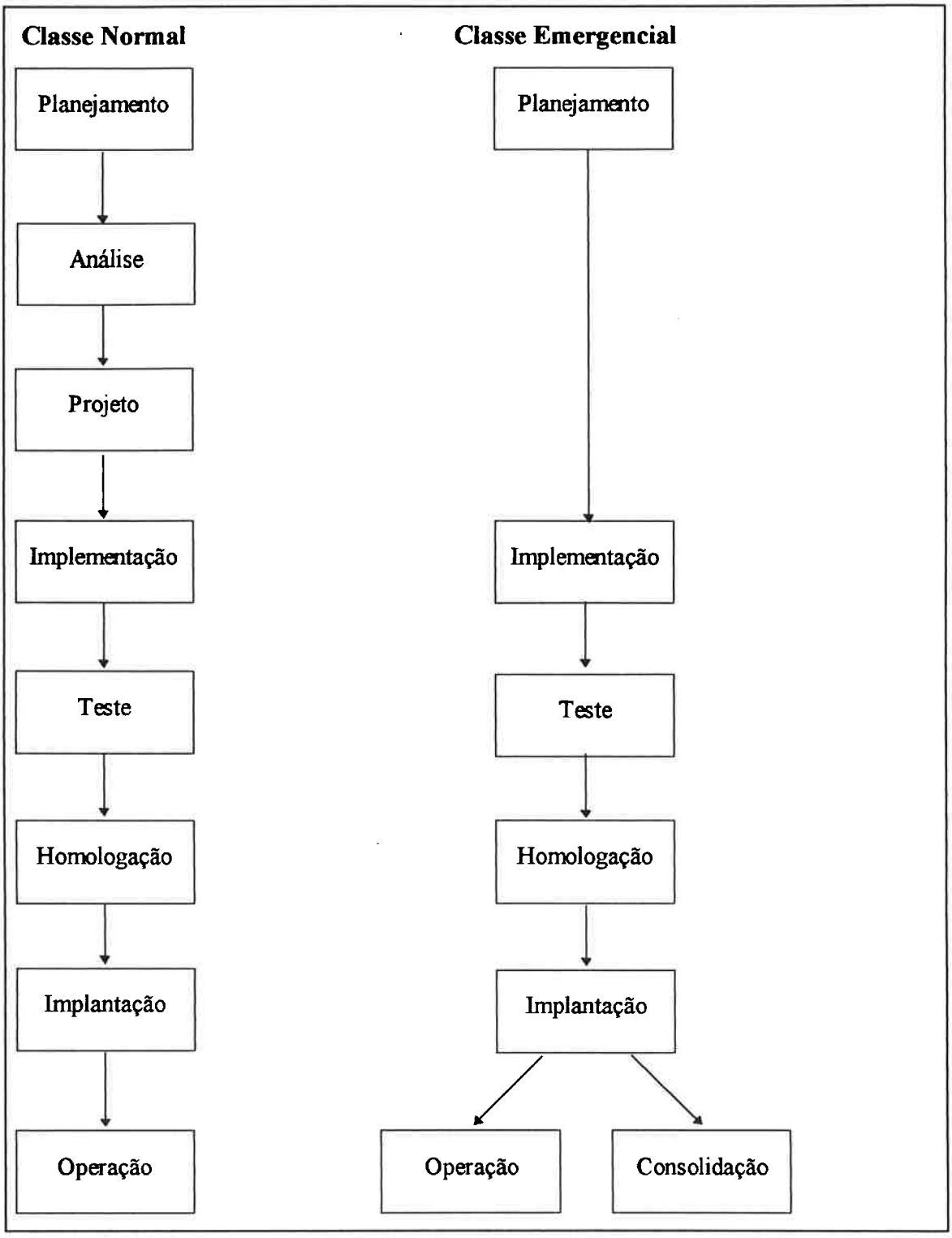

Fig.6.7 - Roteiros de Manutenção de Software Normal e Emergencial.

6.4.4.2 Tabela do Roteiro de Manutenção de *Software*

N.º	Fase	Atividades	Documentos	Classe	
				Normal	Emerg.
1	Planejamento	• Avaliar o porte da manutenção • Definir alternativas de manutenção de *software*	• Estudo de Alternativas de Manutenção		
		• Elaborar plano de manutenção	• Plano de Manutenção		
2	Análise	• Atualizar documentação	• Relatório de Atualização de Documentação		
3	Projeto	• Atualizar documentação	• Relatório de Atualização de Documentação		
4	Implementação	• Gerar código-fonte	• Listagem de Programa		
		• Atualizar manual de geração de código	• Manual de Geração de Código		
5	Teste	• Realizar teste de regressão ou de sistema	• Relatório de Testes		
6	Homologação	• Realizar testes de homologação • Elaborar relatório de testes de homologação	• Termo de Homologação		
		• Atualizar manuais	• Manual de Operação • Manual de Sistema • Manual de Manutenção • Manual de Treinamento		
		• Atualizar configuração de sistema	• Configuração de Sistema		
		• Elaborar Guia de Referência para Implantação e Produção do Aplicativo	• Guia de Referência para Implantação e Produção do Aplicativo		
7	Implantação	• Realizar treinamento do usuário	• Relatório de Treinamento		
		• Instalar o *software* • Realizar operação assistida	• Relatório de Implantação		

(continua)

(continuação)

N.º	Fase	Atividades	Documentos	Classe	
				Normal	Emerg.
8	Consolidação	• Atualizar documentação de sistema	• Especificação de Requisitos de *Software* • Plano de Validação do Aplicativo • Especificação de Projeto • Especificação de Componentes • Plano de Testes de Validação e Integração de Componentes • Manual de Geração de Código • Manual de Operação • Manual de Sistema		
9	Operação	• Entregar *software* para o usuário • Entregar manuais	• Entrega de *Software*		
		• Operar o sistema	• Relatório de Operação		

Tab.6.3 - Roteiro de Manutenção de Software.

Obs. 1: Na coluna Classe, o hachurado indica que os produtos e as respectivas atividades valem para aquela classe. Caso esteja em branco, os produtos correspondentes são dispensáveis.

Obs. 2: As atividades de atualização de documentação estão centradas no uso de ferramentas de controle de versões.

6.4.5 *Infraestrutura de apoio aos processos*

Mostra as interações entre o desenvolvedor de aplicativo e os demais envolvidos no processo de desenvolvimento de *software*: administrador de base de dados, administrador de componentes e fornecedor de componentes.

O objetivo é mostrar as responsabilidades de cada personagem envolvido, destacando a importância de interfaces bem definidas para que não haja retrabalho e conseqüente perda de produtividade.

6.4.6 *Processo de desenvolvimento* usando componentes

A Fig.6.8 mostra o fluxo do processo de desenvolvimento de *software*, destacando os personagens envolvidos e os documentos a serem trocados.

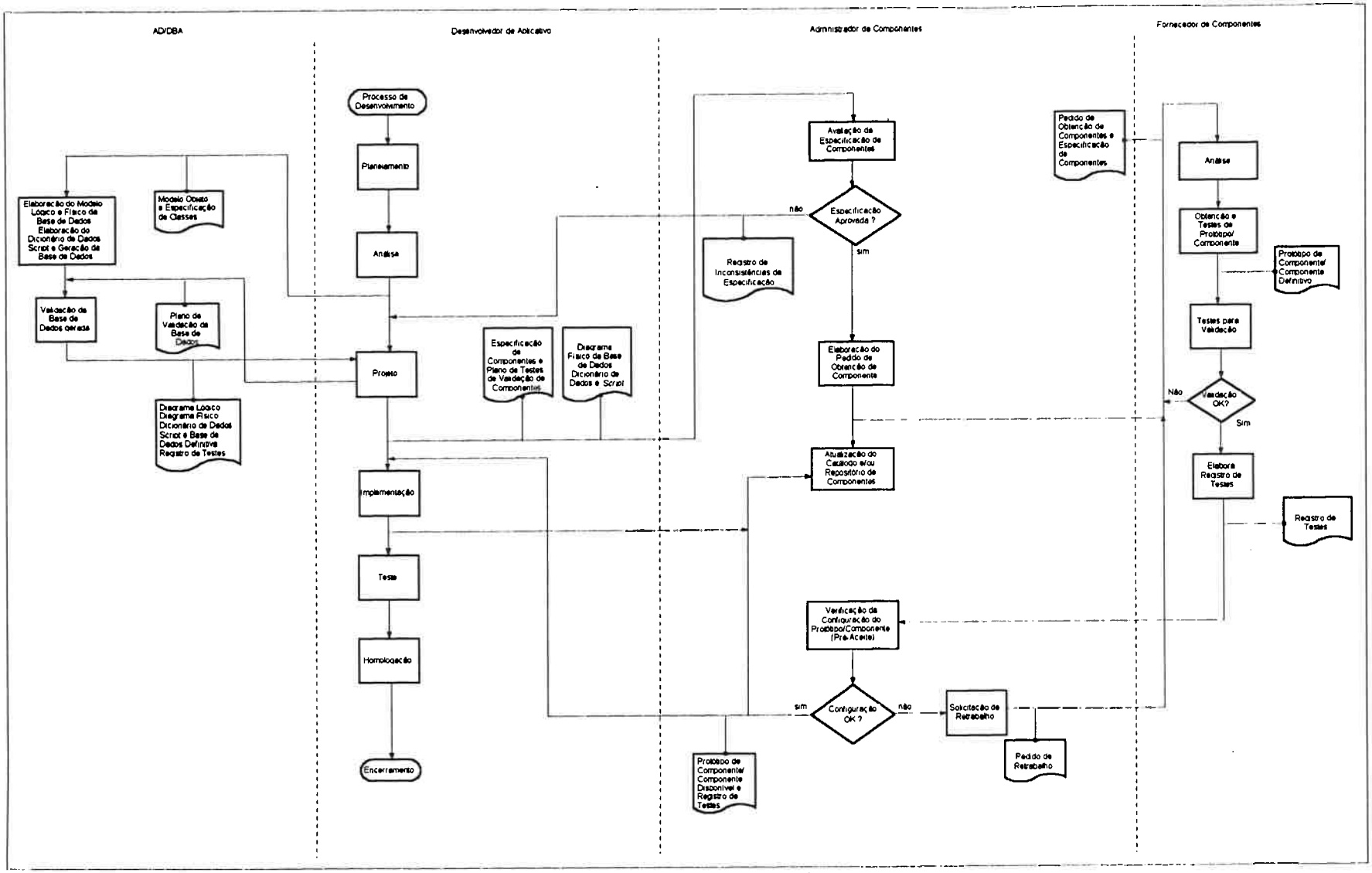

Fig.6.8 - Fluxo de processo de desenvolvimento de *software* e interações entre personagens.

Além do processo de desenvolvimento a ser aplicado por desenvolvedores de aplicativos propriamente dito, o mecanismo de componentização de aplicações na arquitetura cliente servidor exige ainda um apoio intensivo e direto nos aspectos de administração da base de dados e da administração dos componentes. A administração da base de dados garante a integridade e a coerência das informações e dos dados que servirão de apoio aos componentes de negócios das aplicações. A administração de componentes deve buscar o uso racional de componentes existentes quer estimulando o reuso e evitando a construção de componentes similares, quer organizando o conjunto de fornecedores e implementadores de componentes homologados.

6.4.6.1 Responsabilidades

A Tab.6.5 apresenta, para cada atividade do processo de desenvolvimento de *software*, seus responsáveis e participantes.

Fase	Atividade	Desenv. Aplicativo	Fornec. Comp.	Analista Negócio	Adm. Comp.	AD/DBA
Planejamento	Modelamento do Processo de Negócio	R		P		
	Elaboração do Modelo Objeto Preliminar	R				
	Elaboração do Protótipo de Especificação	R				
	Particionamento do Negócio	R		P		
	Elaboração do Plano de Obtenção de Software	R				

(continua)

(continuação)

Fase	Atividade	Desenv. Aplicativo	Fornec. Comp.	Analista Negócio	Adm. Comp.	AD/DBA
Análise	Modelamento Objeto, Dinâmico e Funcional	R		P	P	P
	Elaboração do Plano de Validação do Aplicativo	R				
Projeto	Elaboração do Modelo Lógico e Físico da Base de Dados	P				R
	Validação dos Modelos Lógico e Físico da Base de Dados	R				P
	Elaboração do Plano de Validação da Base de Dados	R				
	Validação da Base de Dados	R				P
	Identificação de Componentes e Interação com o catálogo	R			P	
	Elaboração da Arquitetura do Aplicativo	R				
	Elaboração da Especificação de Componentes	R			P	
	Elaboração do Plano de Testes de Validação e Integração de Componentes	R				
	Avaliação da Especificação de Componentes				R	
	Elaboração do Pedido de Obtenção de Componente	P			R	

(continua)
(continuação)

Fase	Atividade	Desenv. Aplicativo	Fornec. Comp.	Analista Negócio	Adm. Comp.	AD/DBA
	Manutenção do Catálogo e Repositório de Componentes				R	
Implementação	Desenvolvimento do Aplicativo	R				
	Integração do Protótipo de Componente / Componente Definitivo	R				
Teste	Teste de Validação do Componente	R	P		P	
	Teste de Integração do Componente	R				
Homologação	Homologação de Negócio			R		
	Homologação Técnica do Ambiente	P			R	R

Onde:

R - Responsável pela atividade

P - Participante da atividade

AD/DBA - Administrador de Dados /Administrador de Base de Dados

Tab.6.5 - Responsáveis e Participantes do Processo de Desenvolvimento de Software.

6.4.7 Administração de componentes

A incorporação da tecnologia orientada a objetos neste case exemplo está diretamente relacionada com o processo de componentização. Esse direcionamento foi dado em função de uma necessidade básica de promover a reutilização considerando as características das aplicações neste ambiente. Dada a arquitetura de hardware, de software e da própria estruturação de negócios, evidenciou-se a potencialidade muito grande de componentes de uso comum entre nas várias aplicações.

6.4.7.1 Terminologia

Quando se fala em Administração de Componentes de *Software*, surgem alguns conceitos importantes que devem ser detalhados.

Classificação de Componentes

Um componente de *software* pode ser de diversos tipos:

- ◊ **Ativo:** componente que é utilizado por um ou mais aplicativos em produção e deve ser utilizado por aplicativos em desenvolvimento;
- ◊ **Legado:** componente que é utilizado por um ou mais aplicativos em produção mas que não deve ser utilizado por aplicativos em desenvolvimento; e
- ◊ **Inativo:** componente que já não é utilizado em nenhum aplicativo, sendo mantido apenas para histórico.

Ciclo de vida de um componente

A Fig.6.9 mostra o ciclo de vida de um componente de *software*, desde a sua especificação até a sua desativação.

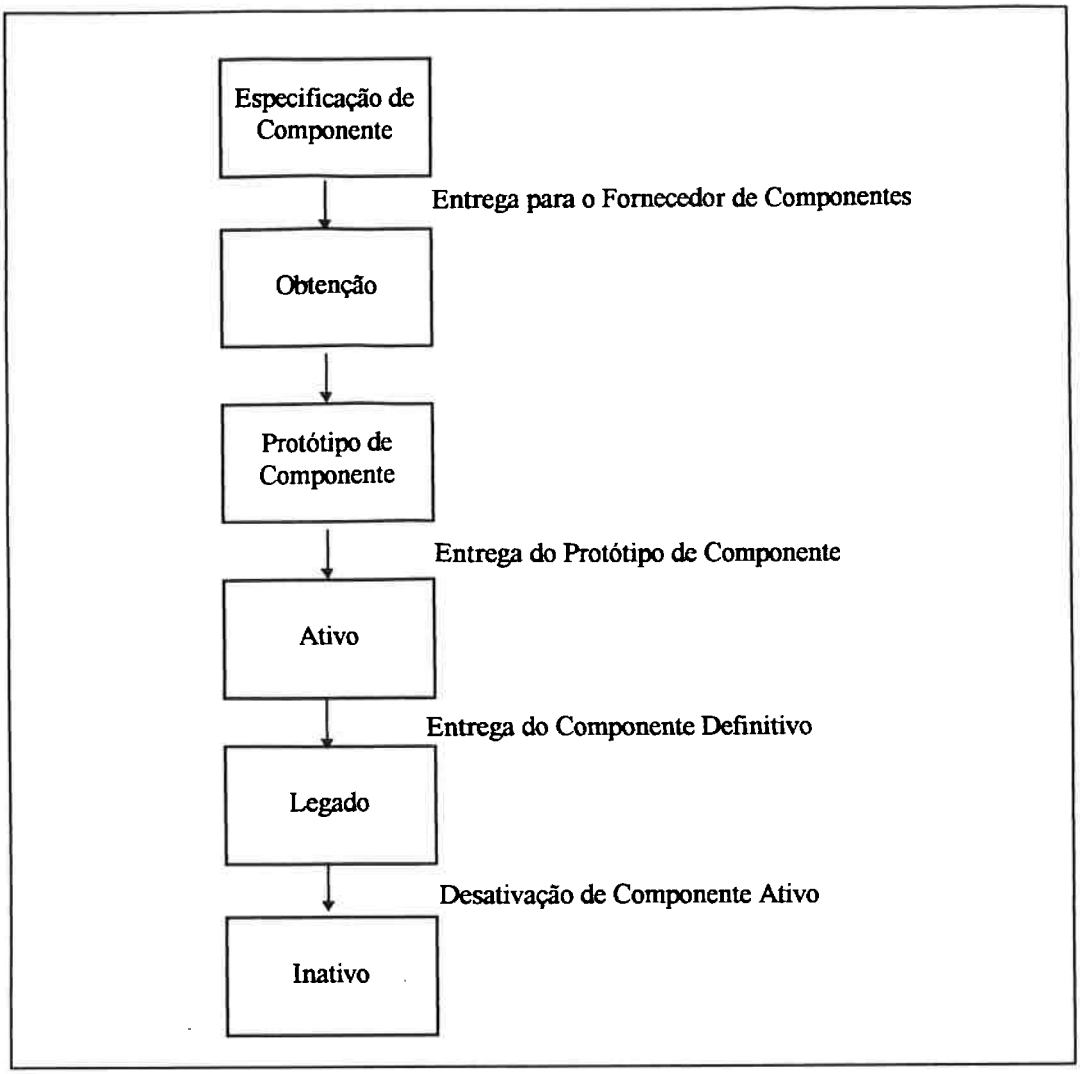

Fig.6.9 - Ciclo de vida de um componente de software.

6.4.7.2 Catálogo de Componentes

Documento que relaciona todos os componentes de *software* existentes no ambiente de desenvolvimento, com informações como funcionalidades, versões e datas. Este documento é utilizado para facilitar as consultas na identificação de possíveis componentes a serem utilizados no projeto.

Estrutura:

1. Identificação do Componente
1.1. Nome
1.2. Versão
1.3. Fornecedor
1.4. Data Prevista para Disponibilização
1.5. Data de Disponibilização
1.6. Unidade de Negócio de Origem
1.7. Grupo Responsável
2. Características
2.1. Objetivos do componente
2.2. Descrição funcional
2.3. Categoria/Classificação
2.4. Status
(Em aquisição, protótipo, disponível, legado, inativo)
2.5. Palavras-chave para consulta
3. Aspectos Tecnológicos
3.1 Tipo de Componente
(DLL, VBX, OCX, Servidor OLE)
3.2. Requisitos de Hardware e Software
4. Referências
4.1 Códigos de Repositórios
(Repositórios que contêm o componente)
4.2. Código da Especificação de Componentes
(Facilita a consulta de maiores detalhes sobre o componente)
4.3. Sistemas que utilizam o componente

6.4.7.3 Repositório de Componentes

Local físico em que são armazenados os componentes de *software*. Como repositório, pode-se ter um armário, uma estação de trabalho ou um servidor. Cada repositório deve possuir um código que o identifica unicamente dentro da organização. Um repositório pode ser:

- **Ativo:** composto por componentes que são utilizados pelos aplicativos existentes e em desenvolvimento, ou seja, composto por componentes ativos;

◊ **Legado:** composto por componentes que são utilizados por aplicativos existentes mas não deve ser utilizado por aplicativos em desenvolvimento, ou seja, composto por componentes legados; e

◊ **Inativo:** composto por componentes que não são mais utilizados pelos aplicativos existentes e em desenvolvimento, ou seja, composto por componentes inativos.

6.4.7.4 O Administrador de componentes

Basicamente, o administrador de componentes desenvolve duas atividades:

◊ Cadastramento e Manutenção de Componentes: consiste, basicamente na manutenção do catálogo e do repositório de componentes, através do controle de inclusões, alterações, exclusões e consultas ao catálogo e ao repositório; e

◊ Suporte ao Desenvolvimento: suporte ao desenvolvedores de aplicativo na consulta ao catálogo de componentes para identificação de possíveis componentes a serem utilizados, apoio à elaboração da Especificação de Componentes e aprovação da Especificação.

6.4.7.5 Cadastramento e Manutenção de Componentes

Operações sobre o Catálogo

A Tab.6.6 apresenta as operações de manutenção possíveis sobre um Catálogo de Componentes.

Operação	Descrição
Inclusão	Permite incluir um novo registro no catálogo de componentes. Esta inserção deve ocorrer após a encomenda de um novo componente ou de uma nova versão.
Atualização	Permite atualizar o status de um componente já cadastrado no catálogo, passando-o, por exemplo, do status de "em obtenção" para "disponível". Permite também a atualização da data de disponibilização do componente e dos repositórios que o contêm.
Desativação	Permite alterar o status de um componente, tornando-o legado, em virtude da existência de novas versões otimizadas. Isto evita a utilização desse componente no desenvolvimento de novos aplicativos. Caso o componente não seja utilizado por mais nenhum aplicativo, ele torna-se inativo.

Tab.6.6 - Operações de Manutenção sobre o Catálogo de Componentes.

Operações sobre o Repositório

A Tab.6.7 apresenta as operações de manutenção possíveis sobre um Repositório de Componentes.

Operação	*Descrição*	*Quem Realiza*
Inclusão	Permite incluir um novo componente no repositório, em virtude de nova obtenção.	Administrador de Componentes
Desativação	Permite movimentar para o repositório adequado, ou seja, movimenta os componentes legados para o repositório legado e os componentes inativos para o repositório inativo.	Administrador de Componentes

Tab.6.7 - Operações de Manutenção sobre o Repositório de Componentes.

6.4.7.6 Utilização de Catálogo e Repositório de Componentes

Operação sobre o Catálogo

A Tab.6.8 apresenta a operação de utilização possível sobre o Catálogo de Componentes.

Operação	*Descrição*	*Quem Realiza*
Consulta	Permite a recuperação de determinados componentes que possuem características desejadas, possibilitando a identificação de componentes a serem reutilizados. Estas consultas podem ser realizadas através de atributos do catálogo que permitem a busca, tais como, palavras-chave, classificação, código do documento de Especificação de Componente, etc. Para agilizar a pesquisa esta busca deve, sempre que possível, ser automatizada.	Administrador de Componentes e Desenvolvedor de Aplicativo

Tab.6.8 Operação de utilização sobre o Catálogo de Componentes.

Operações sobre o Repositório

A Tab.6.9 apresenta a operação de utilização possível sobre o Repositório de Componentes.

Operação	*Descrição*	*Quem Realiza*
Execução	*Permite a utilização/execução do componente no aplicativo em desenvolvimento.*	*Desenvolvedor de Aplicativo*

Tab.6.9 - Operação de utilização sobre o Repositório de Componentes.

6.4.7.7 Roteiro para Administração de Componentes

O fluxograma apresentado na Fig.6.10 mostra o roteiro completo para Administração de Componentes, destacando a forte interação com o catálogo e o repositório de componentes. Este fluxo detalha as atividades do Administrador de Componentes apresentadas em alto nível na Fig.6.8

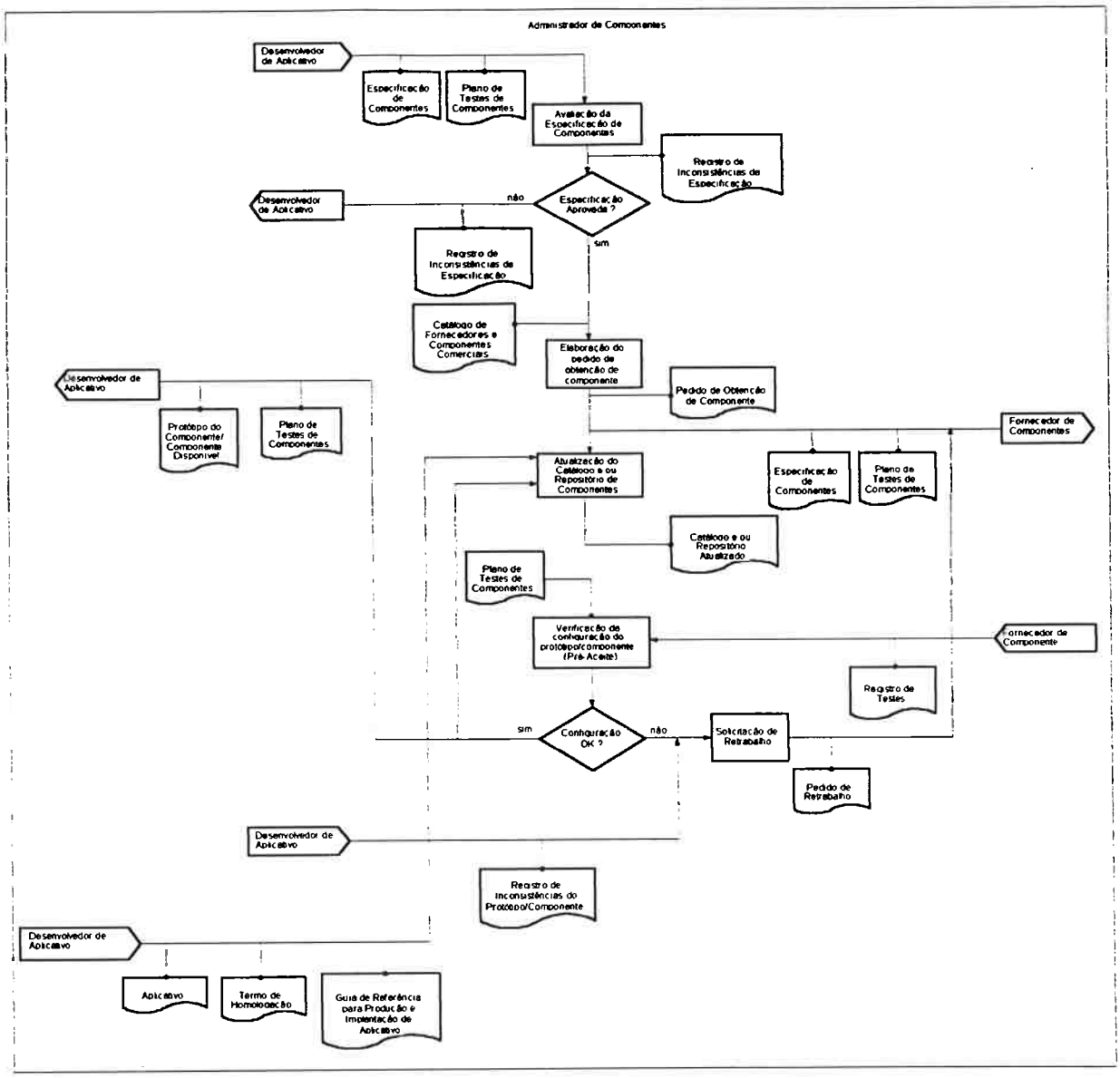

Fig.6.10 - Fluxo de Administração de Componentes.

6.5. Elementos Estratégicos de Implantação

A implantação dos processos, reorganizados segundo descrições apresentadas nos itens anteriores deste capítulo, não é imediata. A implantação desta reorganização está associada a 4 (quatro) elementos estratégicos, vinculados a 4 (quatro) questões: Como a tecnologia pode contribuir ? (tecnologia); Qual a diferença do processo em relação a situação original ? (processo); Por que os produtos são melhores agora? (produtos); Estes elementos são discutidos a seguir.

6.5.1 Tecnologia

A tecnologia orientada a objetos foi a principal elemento estratégico de impacto na reorganização do ambiente exemplo. A justificativa para tal fato estão apresentadas no Cap.4. Os seguintes pontos contribuíram diretamente para adoção desta técnica:

- ◊ PARTICIONAMENTO - Facilita a implementação da estratégia "dividir para conquistar" por técnica e procedimentos estabelecidos pela tecnologia;
- ◊ ESFORÇO EM ANÁLISE - Por técnica, desloca esforços na fase de elaboração do modelo de análise, facilitando a produção de modelos de processos de negócios adequados à expectativa do usuário;
- ◊ COMPONENTIZAÇÃO - Facilita a implementação da reutilização efetiva, desde a análise até a implementação física.

6.5.2 Processo

Quanto ao componente processo, eis alguns pontos estratégicos que nortearam a reorganização:

- ◊ AVALIAÇÃO DA QUALIDADE - Todas as atividades propostas nos roteiros estão prontos para serem submetidos a revisões técnicas formais, buscando minimizar erros em fases anteriores à codificação. O planejamento baseia-se fortemente no particionamento de modelos de negócios, o que garante uma conformidade do sistema a ser implementado com o conjunto de requisitos estabelecidos pelo usuário;
- ◊ PARTICIONAMENTO - Modelar o processo de negócio, usando a técnicas IDEF0 facilita a visualização do fluxo de atividades e informações do processo a ser automatizado. Quebrar este modelo em partições corresponde a decompor um grande sistema em partes menores fáceis de planejar e controlar;
- ◊ PLANEJAMENTO E CONTROLE - Uma vez particionado, a maior dificuldade está na estimação de esforços e tempos necessários. Fica facilitado o fato do planejamento estar baseado no modelo particionado. O cronograma global acaba sendo uma composição de pequenos cronogramas referentes a cada uma das partições. O controle efetivo do projeto está em acompanhar adequadamente as atividades, os produtos liberados, as revisões técnicas formais que mostram a evolução do

projeto, com eventuais necessidades de planejamento detectados a tempo para negocição com todos os envolvidos;

◊ ANÁLISE ORIENTADA A OBJETOS - Incorporar a análise orientada a objetos nas atividades de modelagem tem como estratégia implantar a tática do desenvolvimento rápido, conforme discutido no Cap.2. Nesse caso a orientação a objetos é usado como estratégia de obter modelos de análise de requisitos em conformidade com o usuário (cliente);

◊ PROJETOS POR COMPONENTES - Incorporar recursos de componentes visa facilitar a reutilização de software e também melhor estruturar os projetos dos sistemas para facilitar a correção e evolução de software;

◊ PLANEJAMENTO DE TESTES - Garantir que um sistema tenha sido devidamente testado, validado corresponde à realização de uma boa cobertura de testes, com planejamento prévio. Um sistema organizado por componentes, que sofrem processos intensos de testes faz com que o sistema final apresente uma maturidade de testes superior.

6.5.3 Produtos

A estratégia adotada nesta reorganização possibilita produtos com qualidade superior em itens com:

◊ QUALIDADE DE TESTES - A qualidade do produto aumenta quanto maior for a bateria de testes a que se submete. Além disso, sendo

produto uma integração de componentes, previamente validados, então o sistema deve apresentar uma maior robustez;

◊ CONFIABILIDADE e INTEGRIDADE - A alocação de grandes esforços na consolidação de modelos de análise junto aos clientes (solicitantes do desenvolvimento) faz com que as funções implementadas no sistema seja de maior qualidade no que se refere a falhas operacionais e integridade das operações;

◊ MANUTENÇÃO CORRETIVA e EVOLUTIVA - Sistemas melhor estruturados (Cap.2) facilita a detecção, localização e correção de erros. O mesmo vale para a melhoria (evolução do sistema) das funcionalidades do sistema. Orientação a objetos melhora a estruturação à medida que organiza o sistema como um conjunto de elementos autônomos com responsabilidades bem definidas e específicas, que são os objetos.

6.5.4 Pessoas

As pessoas envolvidas podem apresentar uma sinergia maior no processo de desenvolvimento. Para isso as seguintes estratégias podem ser conduzidas:

◊ CLIENTE - O foco da modelagem no processo a ser automatizado, através do uso de objetos associados aos negócios (objetos de negócios - Cap.2 e 4.), a uniformidade de notações e modelos entre as fases de análise e projeto facilita o envolvimento do cliente ao projeto, vindo contribuir duplamente: de um lado auxiliando convergência de requisitos

entre o que se espera para o sistema (clientes) e o que foi interpretado pelos desenvolvedores; De outro ajustando as expectativas tanto do cliente como dos desenvolvedores;

◊ COORDENADORES DE EQUIPES DE DESENVOLVIMENTO - A coordenação dos projetos, dentro do novo processo, tem uma atuação bastante estratégica e ativa: ela deve registrar os planejamentos e os replanejamentos de esforços, recursos, prazo em banco de dados; acompanhar as atividades e produtos gerados; coordenar as revisões técnicas e, por fim, interpretar os indícios de projeto para saber da necessidade de replanejamento e renegociação perante ao cliente dentro de um prazo admissível. Deve manter a motivação da equipe, a qualidade dos trabalhos de acordo com

◊ DESENVOLVEDORES - As atividades descritas nos roteiros, os produtos intermediários a serem elaborados durante o projeto aumentam a qualidade geral, pois definem procedimentos sistemáticos a serem utilizados por todos os desenvolvedores e uniformizam o nível técnico dos produtos intermediários e finais. Além disso, as atividades de revisões técnicas formais são um complemento forte para garantir a qualidade mínima aos projetos;

◊ TREINAMENTO "ON-THE-JOB" - Aproveitando o conhecimento dos desenvolvedores nos processos de negócio, objetos da automação, a montagem de treinamento direto no projeto (*on-the-job training*) é uma estratégia de assimilação pela equipe técnica eficaz, melhor do que

treinamentos genéricos. Isto porque os roteiros e respectivas atividades, e os modelos de produtos a serem gerados são a referência para este tipo de treinamento. Assim, as equipes de desenvolvimento tendo o conhecimento do processo de negócio, e alguns especialistas (orientadores) conhecedores da tecnologia orientada a objetos e do processo de obtenção de software, formam a combinação para a execução das atividades de projeto com treinamento.

6.6. Resultados Obtidos

Os resultados obtidos da aplicação desse processo reorganizado estão descritos de comentados no Cap.7. Naquele capítulo são mostrados resultados bastantes significativos, especialmente nos aspectos de qualidade de produto e processo, com discussões adicionais sobre a assimilação por parte da equipe técnica.

7. CONSIDERAÇÕES FINAIS

Objetivo do Capítulo:

Apresentar considerações sobre os resultados obtidos na aplicação da metodologia para inserir mecanismos de melhoria e avaliação da qualidade de software, a aplicabilidade do trabalho e os novos passos de pesquisas.

Tópicos do Capítulo:

7.1. Resultados

O resultado, na aplicação efetiva e prática desta metodologia (descrita no Cap. 5), assunto desta tese, foi a reorganização de um processo de software que apresentava baixa maturidade (padrões SEI/CMM), com a conseqüente melhoria no processo de desenvolvimento, em seu todo e em suas partes, tanto em aspectos de qualidade como em aspectos de produtividade, possibilitando a implantação de mecanismos de avaliação de qualidade/produtividade. Como se poderá observar no desenrolar deste capítulo, as conseqüências positivas desta reorganização foram praticamente imediatas, naturais do processo metodológico.

7.1.1 Abrangência dos experimentos

A aplicação da metodologia no processo de desenvolvimento de software atingiu as expectativas quanto a abrangência: o processo foi reorganizado globalmente, isto é, a reorganização foi feita em todas as etapas do processo, abarcando desde planejamento e controle até atividades de validação da implementação final.

O uso de métricas no processo também foi implantado de forma abrangente e já é procedimento padrão nas etapas consideradas fundamentais, sendo, assim, coletadas especialmente nas fases de planejamento, controle, análise, projeto e implementação.

No que se refere a recursos humanos, conseguiu-se resultados positivos também globais, com envolvimento, motivação e engajamento de todas as equipes participantes. Este resultado não foi imediato, o que já era esperado, pois a estratégia adotada era a de, uma vez definido o processo de trabalho

(metodologia de trabalho), implantar seu uso de forma incremental, isto é, por partes, de forma crescente até atingir todo o processo de desenvolvimento, refinando (dando qualidade) e ajustando (adequando) as atividades e os produtos dos roteiros. A operacionalização do processo de trabalho foi trabalhosa, e as maiores dificuldades estavam associadas à formação das equipes, mas com a adoção da estratégia de implantação incremental, e com o impacto dos resultados positivos apresentados nas equipes que foram submetidas ao novo processo de trabalho, a motivação, o envolvimento e o engajamento das demais equipes foram espontâneos e naturais.

A aplicação da metodologia cobriu os seguintes tópicos:

- ◊ Diagnóstico global - Para identificação e classificação dos tipos de projetos de obtenção de software;
- ◊ Estabelecimento dos fatores para padrões de qualidade - Para identificar fatores de maior impacto, maior relevância, nos sistemas, considerando o contexto da indústria financeira. Esses fatores são mapeados nas atividades dos roteiros do processo de software (Cap.2);
- ◊ Definição dos roteiros de planejamento e definição dos procedimentos para acompanhamento, controle e execução - Para garantir melhoria do planejamento adotou-se as técnicas de particionamento. Para o garantir o acompanhamento e controle da evolução dos trabalhos foram estabelecidas e realizadas revisões técnicas de atividades e produtos;

◊ Definição de produtos com base na técnica de orientação a objetos - O uso desta técnica possibilita o uso de componentização no processo e a incorporação da prática de reutilização. Aplicando-se os conceitos e os procedimentos da tecnologia orientada a objetos é possível realizar a concepção, a arquitetura e a implementação do sistema de forma a visualizar o processo como um conjunto de componentes (componentização). Estes componentes, baseando-se no conceito de objetos, podem ser reutilizados e sua atuação e/ou seu uso dentro do processo serão retratados pelas interligações indicadas entre os componentes. Desta forma, os componentes serão definidos apenas uma única vez, resultando em otimização do tempo e do esforço de desenvolvimento, acrescentando-se o resultado, extremamente importante, de visão global e transparente do processo (visualização clara e objetiva do processo em seu todo, e em suas partes), com conseqüente poder de acompanhamento de sua evolução e otimização de qualidade e produtividade;

◊ Treinamento da equipe técnica - Adoção de treinamento do tipo tipo *on-the-job training* (foi feito um planejamento e está sendo executado). Neste tipo de treinamento, as equipes são treinadas durante o desenvolvimento dos projetos, sendo direcionadas para a aplicação dos métodos estabelecidos para o novo processo. Dessa maneira estão aprendendo a: a) utilizar os roteiros do processo e os templates de produtos; b) aplicar a tecnologia orientada a objetos na análise, no projeto e na implementação; e c) montar planejamento e

controle de obtenção de software, utilizando técnicas de modelagem de negócios e particionamento;

7.1.2 Qualidade dos produtos e dos processos

No ambiente deste exemplo prático, a fase de implantação do processo reorganizado atinge, neste momento, a ordem de 12 projetos, num total de 60 desenvolvedores, dos quais 7(sete) são líderes. O período envolvido é de 6 meses, durante os quais, já se obteve melhora considerável na qualidade do processo, uma vez que a evolução do projeto apresenta-se visível (graças às revisões de produtos intermediários). Houve também aprimoramento nos produtos, verificado nos processos de validação e na aceitação por parte dos usuários. Veja alguns itens de processo que retrataram também o aumento de qualidade e de produtividade::

- ◊ Qualidade do produto no planejamento - A qualidade do produto foi aprimorada com as melhorias feitas no planejamento do mesmo, que passou a ser elaborado seguindo o método de particionamento[1], que resultou em melhorias no detalhamento das atividades, dos recursos e dos prazos;
- ◊ Qualidade do processo no planejamento - O planejamento do processo foi aprimorado com a inclusão de um roteiro (baseado no particionamento do modelo de processo de negócio[2]), refletindo-se na melhoria do processo;

[1] Dados estatísticos 1: Houveram melhorias no acerto de estimativas de prazos em 60% (ARAKAKI, registros técnicos de acompanhamento *on-the-job training*, 1997).
[2] Dados estatísticos 2: O roteiro aumentou em duas vezes (200%) em itens de revisão técnica para a montagem do planejamento.

◊ Qualidade do produto na análise - A melhoria da qualidade do produto, advinda da melhoria no processo de análise do mesmo, é verificada pela adequação de expectativas das pessoas envolvidas (de um lado os usuários/cliente, de outro os desenvolvedores). Esta adequação tem aumentado, como demonstraram estatísticas, e ocorre devido à adoção, no processo de análise do produto, da técnica de orientação a objetos, que possibilita a elaboração de um modelo de análise de requisitos bastante adequado às expectativas dos usuários, face ao processo de negócios. Esta elaboração demanda mais tempo e esforço que os processos tradicionais de análise, no entanto, apresenta resultados que justificam sua utilização (aumenta a possibilidade de se atingir o ideal: fazer bem, uma única vez, na primeira vez).;

◊ Qualidade do processo na análise - A melhoria na qualidade do processo pode ser verificada pelo aumento da facilidade de validação do processo junto ao usuário. A análise de requisitos, com a incorporação de técnica orientada a objetos, aumentou a possibilidade de validação, junto ao usuário, da modelagem de objetos e da modelagem dinâmica[3];

◊ Qualidade do produto no projeto - O produto é estruturado como uma organização de componentes, facilitando a manutenção corretiva e evolutiva;

[3] Dados estatísticos 3: O entendimento das notações utilizadas nos modelos objeto e dinâmico pelos clientes, especialistas de negócio, permitiram revisões de todas as definições de objetos de negócios e de todos os cenários - diagramas de eventos. Estima-se uma melhoria de 100 % na análise de requisitos.

◊ Qualidade do processo no projeto - O conceito de componentização de objetos foi aplicado no processo e a especificação detalhada e abrangente dos componentes otimiza o processo de subcontratação de terceiros[4];

◊ Qualidade do produto na implementação - O produto implementado, segundo os modelos elaborados na análise e detalhados no projeto, apresenta uma qualidade superior, especialmente durante as atividades de manutenção corretiva e evolutiva;

◊ Qualidade do processo na implementação - A principal melhoria na implementação é o estabelecimento do uso de integração de códigos, ou seja, os componentes são tratados separadamente, isto é, são especificados, implementados e testados de forma isolada, garantido robustez por componentes. A integração do sistema final, consiste em acionamentos dos componentes, que são integrados facilmente devido a sua maturidade, que neste instante do processo estão testados e validados;

7.1.3 Influência do contexto na elaboração desta metodologia

A indústria financeira, ambiente de aplicação desta metodologia, apresenta características marcantes e peculiares quanto ao perfil da equipe técnica e quanto a tipos de projetos, já citados nos Cap.1, 2, e 4. Porém, cabe ressaltar algumas características, detectadas durante o inter-relacionamento com as pessoas envolvidas no processo:

[4] Dados: é possível terceirizar a implementação de componentes sem abrir mão do processo de negócio - são os casos de processos de negócio sigilosos.

◊ Com gerência da equipes - O posicionamento do corpo administrativo denota a grande necessidade de organizar, de amadurecer um processo de software. Na indústria financeira, a pressão sobre prazos, qualidade e produtividade faz com que os gerentes busquem uma excelência de qualidade nos processos. Essa busca de excelência de qualidade nos processos é determinada pela consciência de que produtos e/ou ferramentas da moda (milagreiros) são enganosos, são ilusões, oferecendo pequenos ganhos em codificação. Tal conscientização conduz ao entendimento de que desenvolver sistemas rapidamente, mas com qualidade, implica em procedimentos e posicionamento do tipo "fazer certo da primeira vez" (Cap.2);

◊ Os desenvolvedores - Necessitam de uma ferramenta que lhes dê condições de cumprir suas tarefas de acordo com as exigências do contexto. Os desenvolvedores são pressionados pela demanda de trabalhos, necessitando de apoio, referências e infra-estrutura de desenvolvimento para a execução das atividades de obtenção de software, de uma maneira mais eficaz, sem perdas de tempo com retrabalhos.

7.2. Algumas Conclusões

Expõe-se, a seguir, algumas conclusões tiradas a partir da experiência na implantação desta metodologia:

A implantação de mecanismos de avaliação da qualidade só se justifica se obter ao longo do processo a organização do processo de desenvolvimento de software.

Implantar sistemas (ou mecanismos) de avaliação da qualidade não garante qualidade de processo de desenvolvimento de software, muito menos de seu produto. Avaliar qualidade ou implantar mecanismos para tal num ambiente de desenvolvimento desorganizado, quasi-caótico (nível 1), resultará em simplesmente retratar, com padrões de medição, a situação caótica em que se encontra. A meta almejada, e que justifica (valida) a implantação de tais mecanismos, é melhorar a qualidade, isto é, eliminar as deficiências, aumentar a produtividade, enfim, é buscar a otimização do processo e de seus resultados como um todo. Esta meta só será atingida através da adoção de uma metodologia de trabalho que, adotando mecanismos de avaliação da qualidade e produtividade, consiga organizar todo processo de desenvolvimento, tirando-o da situação caótica e colocando-o numa situação organizada. Isto implica em deixá-lo transparente (passível de visualização) no todo e nas partes, de forma a se ter condições de analisá-lo, acompanhar suas etapas, avaliar cada etapa, fazer adequações (realimentação). Somente com uma organização, obtida através de uma metodologia que lhe garanta tais características, seria produtiva a implantação de mecanismos de avaliação da qualidade, pois, nestas condições, os resultados da avaliação teriam utilidade, ou seja, seriam empregados para aprimorar o processo, uma vez que as deficiências seriam identificadas , localizadas, diagnosticadas e os ajustes seriam feitos, podendo ser novamente

avaliados, já que se tem a visão (e o conseqüente controle) do processo no todo e em suas partes. Resumindo, implantar mecanismos de avaliação ou avaliar a qualidade de um processo caótico, sem reorganizá-lo, serviria apenas para retratar a situação sem condições de sanar problemas, donde se depreende que, antes de se avaliar qualidade ou implantar mecanismos de avaliação de qualidade, é necessário primeiro buscar organizar o processo de desenvolvimento de software, adotando metodologias que contenham mecanismos de avaliação da qualidade que propiciem a de organização, que resultem na organização do processo. A adoção deste princípio traz benefícios e resultados positivos muito rapidamente, comprovados durante a aplicação da metodologia proposta nesta tese nos processos de desenvolvimento de software, dentro da indústria financeira.

A organização de um processo deve focalizar aspectos de qualidade

A metodologia proposta organiza os processos de obtenção de software, deixando-o preparado para se enquadrar no nível 2 do SEI/CMM (Cap.2 e 3). No entanto, seu princípio básico é a busca da qualidade, organizar visando o aprimoramento da qualidade e, conseqüentemente, da produtividade, obtendo resultados otimizados; é organizar focalizando aspectos de qualidade, de forma a obter resultados que melhorem a qualidade do processo e do produto.

A evolução destas pesquisas caminhará em direção ao aprimoramento desta metodologia, que expandirá o assunto e o uso de métricas.

A evolução deste trabalho (pós-doutorado) estará direcionada para a montagem de bases de dados (históricas e específicas para cada ambiente, registrando dados de planejamento, de métricas de aplicações e de componentes - Cap.3) e também para a implantação e análise de métricas, a partir de dados obtidos no período de 1997-1998.

O processo quasi-caótico pode ser melhorado.

A relevância de se melhorar ou não o processo quasi-caótico depende do grau de importância atribuído aos sistemas de software dentro da instituição, podendo ser prioritário ou não. No entanto, é possível tirá-lo do estado quasi-caótico, é possível melhorá-lo. Segundo a metodologia proposta nesta tese, incorpora-se aspectos de qualidade (usando mecanismos como tecnologia orientada a objetos, componentização e métricas) e reorganiza-se o ambiente, segundo parâmetros propostos pela metodologia , que garantirão a qualidade do processo em seu todo e em suas partes. Atualmente, em sua maioria, a indústria financeira considera prioritário, senão vital, o processo de desenvolvimento de software, ou seja, os processos constituem infra-estrutura, são apoios para suas atividades. Constatou-se, neste ambiente, na prática, que processos quasi-caóticos podem ser melhorados em qualidade e produtividade.

No aspecto aplicabilidade dos resultados, esta tese deve contribuir para a formação de novas linhas de pesquisa em Engenharia de Software, com produtos em graduação, pós-graduação e extensão.

Estas pesquisas poderão centrar-se em:

- ◊ Formação de recursos humanos na definição e otimização de processos de software, utilizando a tecnologia orientada a objetos;
- ◊ Organização de sistemas de software com alto grau de reutilização, por componentização; e
- ◊ Organização de sistemas compatíveis com redes como a Internet (CHAPPELL, 1996).

Indústrias de software, voltadas para produção de software genéricos ou específicos, são exemplos de áreas que podem beneficiar-se com a aplicação da metodologia proposta.

7.3. Conclusões Finais

A metodologia proposta nesta tese é parte minúscula da Engenharia de Software, porém a sua aplicabilidade é grande tanto na linha de pesquisa acadêmica, como também na linha da indústria. As estatísticas demonstram carências fortes em processos de software e, a aplicação desta metodologia garante rápida melhoria no processo de produção de software, através do uso pragmático e simples da tecnologia orientada a objetos, resultando em processos com maturidade suficiente para serem avaliados do ponto de vista de qualidade e produtividade.

Os benefícios resultantes da aplicação desta metodologia serão certamente a organização do processo de desenvolvimento do software; a melhoria da qualidade do processo; a melhoria da qualidade do produto; aumento da

produtividade; implantação de procedimentos de avaliação da qualidade, do processo e do produto (no todo e em partes); facilidade de planejamento, análise, acompanhamento e implementação do processo e do produto (no todo e em seus componentes); visibilidade do processo (transparência), facilidade de diagnósticos e ajustes; motivação, envolvimento e engajamento das pessoas envolvidas no processo; treinamento do pessoal em técnicas atualizadas que poderão ser extrapoladas para outros desenvolvimentos; busca de padrões de medida que permitirão medir quesitos de qualidade e produtividade julgados não mensuráveis; aumento do grau de confiabilidade do processo como um todo e em suas partes; melhora no relacionamento e nas negociações cliente - desenvolvedor; cronogramas realísticos; diminuição de pressões; cumprimento de metas propostas; melhora no processo de validação; enfim há uma melhora de qualidade no processo como um todo que se reflete em todas suas etapas e componentes.

8. BIBLIOGRAFIA

Objetivo do Capítulo:

Apresentar a relação de publicações utilizadas nos estudos e pesquisas que levaram a elaboração desta tese.

Tópicos do Capítulo:

8. Bibliografia

A pesquisa bibliografica foi realizada usando-se a busca de artigos e livros especializados em bibliotecas, como também na Internet. Ela é composta de uma parte da pesquisa com textos básicos, que apoiaram a elaboração da tese e uma parte com bibliografia de apoio.

As referências classificam-se basicamente em periódicos, trabalhos e palestras ministradas, livros e relatórios técnicos internos produzidos nos trabalhos de pesquisa.

8.1 Referências Bibliograficas

1. Albrecht, A.. J. ***Measuring application development productivity.*** In Proc. IBM Applications Development Joint SHARE/GUIDE Symposium, p.83-93, October, 1979.

2. Anacleto, A.. L.; Grahi, E. A..; Hugo, M. ***Resultados de mensuração do processo de software no modelo CMM/SEI***, Anais do X SBES Simpósio Brasileiro de Engenharia de Software, p.41-50, São Carlos, 1996.

3. ARAKAKI, R. ***Modelagem de Processos de Negócios usando Orientação a Objetos.*** In: Developers & Object Forum, Anais. São Paulo, 1996.

4. ARAKAKI, R.; HIRAMA, K.; FREGNI, E. ***Classificação de Roteiros de Desenvolvimento de Software Comerciais.*** Escritório de Engenharia de Software, São Paulo, 1996. (Estudo Técnico).

5. ARAKAKI, R; GONDO, M. ***Estimativa de Esforços e Prazos usando Objetos de Negócios.*** São Paulo, 1996 (Estudo Técnico).

6. ARAKAKI, R.; MELNINKOFF, S. S. ***Introdução em Engenharia de Software.*** Escola Politécnica, Spectrum Engenharia, 1996.

7. ARAKAKI, R. ***Uma Metodologia de Desenvolvimento e Implantação de Sistemas CAE/CAD/CAM.*** São Paulo, 1991. 177p. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo.

8. ARNOLD, K. L. **The manager's guide to Iso 9000.** New York, the Free Press, 1994.

9. ATCHISON, L. **Object-Oriented Test & Measurement Software Development in C++.** New Jersey, Prentice Hall, 1997.

10. Baker, A. L. ; Zweben, S. H. ***A comparison of measures of control flow complexity.*** IEEE Transactions on Software Engineering, v.6, n.6, p. 506-512, 1980.

11. Baker, A. L.; Zweben, S. H. ***A comparison of measures of control flow complexity.*** IEEE Transactions on Software Engineering, v.6, n.6, p.506-512, 1980.

12. Baker, A. L.; Bieman, J. M.; Fenton, N.; Gustafson, D. A.; Melton, A.; Whitty, R. ***A philosophy for software measurement.*** Journal of Systems and Software, v.12, p.277-281, 1990.

13. Basili, V. R.; Zelkowitz, M. V. ***Analyzing medium scale software development.*** Proc. 3rd International Conference on Software Engineering, p.116-123, IEEE, 1978.

14. Basili, V.R.; Weiss, D. M. ***A methodology for collecting valid software engineering data.*** IEEE Transactions on Software Engineering, v.10, n.6, p. 728-738, 1984.

15. Basili, V. R. ; Freburger, K. ***Programming measurement and estimation in the Software Engineering Laboratory.*** Journal os Systems and Software, p.47-57, 1981.

16. Basili, V. R.; Rombach, H. D. ***The TAME project: Towards improvement-oriented software environments.*** IEEE Transactions on Software Engineering, v.14, n.6, p. 758-773, 1988.

17. BERRY, J. **The Waite Group's C++ Programming Language.** Indiana, Howard. W. SAMS & Company, 1989.

18. Bersoff, E. H. Elements of Software Configuration Management. IEEE Transaction on Software Engineering, v.10, n.1, p79-87, 1984.

19. Bieman, J. M. ***Deriving measures of software reuse in object-oriented systems.*** Formal Aspects of Measurement, p.79-82, Spring-Verlag, London, 1992.

20.Boehm B. W. Software Engineering Economics, Englewood Cliffs, NJ, Prentice Hall, 1981.

21.Boehm, B. W. ***Improving Software Productivity***. IEEE Computer, v.17, n.6, p.30-44, 1987.

22.Boehm, B. W. ***Software Risk Management.*** Washington, DC, IEEE Computer Society Press, 1989.

23.BOOCH, G. **Object Solutions: managing the object-oriented project.** CA, Addison-Wesley, 1996.

24.Bulut, N.; Halstead, M. H. ***Impurities found in algorithm implementations.*** ACM SIGPLAN Notices p.9-12, 1974a.

25.Bulut, N.; Halstead, M. H.; Bayer, R. ***Experimental validation of a strctural property of FORTRAN algorithms.*** ACM National Conference Proc., p.207-211, 1974b.

26.Card, D.N.; Glass, R. L. ***Measuring Software Design Quality.*** Prentice Hall, NJ, 1990.

27.CHAPPELL, D. **Undestanding Active X and OLE**. Redmond, Washington, Microsoft Press, 1996.

28.Chidamber, S. R.; Kemerer, C. F. ***A metrics suite for object oriented design.*** IEEE Transactions on Software Engineering, v.20, n.6, p.476-493, 1992.

29.Carnegie Mellon University/Software Engineering Institute. ***The Capability Maturity Model: Guidelines for Improving the Software Process.*** Reading, Massachussets, Addison-Wesley, 1995.

30.COAD, P.; YOURDON, E. **Object-Oriented Analisys.** 2ed. New Jersey, Prentice-Hall, 1991.

31.COAD P. ; NICOLA, J. **Object-Oriented Programming.** New Jersey, Prentice-Hall, 1993.

32.Comer, D.; Halstead, M. H. A simple experiment in top-down design. IEEE Transactions on Software Engineering, v.5, n.2, p.105-109, 1979

33.CONNER, D. B. et. al. **Object Oriented Programming in Pascal: a graphical approach.** Massachusetts, Addison-Wesley, 1995.

34.COOK, S.; DANIELS, J. **Designing Object Systems: object-oriented modelling with syntropy.** Cambridge, Prentice Hall, 1994.

35.COOPER, A. **About Face: The essentials of user interface design.** CA, IDG Books, 1995.

36.Cornell L.M.; Ottestein. ***Further investigations into a software science relatioship.*** Proc. Computer Measurement Group VII Conference, p.195-198, 1976.

37.CORRIVEAU, J. P. **Traceability Process for Large OO Project.** Computer, v. 29, n.9, p.63-68, Set, 1996.

38.Cox, B. ***There is a Silver Bullet***. Byte Magazine, p.209-226, October, 1990.

39.DAVIS, S. R. **C++ for Dummies** . IDG Books, 1994.

40.DAVIS, T. **Adopting a Policy of Reuse.** IEEE Spectrum, v.8, n.12, Jun, 1994.

41.DeMarco, T.; Lister T. Peopleware: Productive Projects and Teams. New York, Dorset House, 1987.

42.De Millo, R. A.; Lipton, R.J. ***Software project forecasting.*** In Alan Perlis, Frederick G. Sayward, Mary Shaw, Software metrics, p.77-89, MIT Press, Cambridge, 1981.

43.DEPARTMENT OF DEFENSE. **Military Stardand: Defense System Software Development DOD-STD-2167A.** DoD, June 1995

44.DOUGLAS. N. **The Metrics Conundrum: How Do We Chose the Best Metric?** American Programmer, v.8, n.12, p. 20-24, Dec. 1995.

45.Elshoff, J. L. ***An analisys of some comercial PL/I programs.*** IEEE Transactions on Software Engineering, v.2, n.2, p. 113-120, 1976a.

46.Fagan, M. E. ***Design and Code Inspections to Reduce Errors in Program Development.*** IBM Systems Journal, v.15, n.3, p.182-211, 1976.

47.Fenton, N. E. ***Software Metrics: A Rigourous Approach.*** Chapman and Hall, London, 1991.

48.FREGNI, E. ; ARAKAKI, R. ; TAKATA, K. **Conceituação de Orientação Primária de Interfaces: A documento ou a serviços.** São Paulo, 1995. (Estudo Tecnico).

49.Funami, Y; Halstead, M. H. ***A software phisics analisys of Akima's debugging data.*** Microwave Research Institute XXIV International Symposium: Computer Software Engineering. Polythecnic Press, New York, 1976.

50. Gannon, J. D.; Katz, E. E.; Basili, V.R. ***Metrics for ADA packages: An Initial study.*** Communications of ACM, v.229, n.7, p.616-623, 1986.

51. GARMUS, D.; HERRON, D. **Effective Early Estimation.** Software Development , v.4 , n.7, p. 57-65, Jul, 1996.

52. Gilb, T. ***Principles of Software Engineer Management.*** Wokingham, England: Addison-Wesley, 1988.

53. GILB, T. **Software Metrics and The Results Method.** American Programmer, v.8, n.12, p. 33- 40, Dec. 1995.

54. Gordon, R. D.; Halstead, M. H. ***An Experiment Comparing FORTRAN programming time with the software physics hyphothesis.*** NCC proc. P.935-937, 1976.

55. Gordon, R. D. ***Measuring improvements in program clarity.*** IEEE Transactions on Software Engineering, v.5, n.2, p. 79-90, 1979a.

56. Gordon, R. D. ***A qualitative justification for a measure of program clarity.*** IEEE Transactions on Software Engineering, v.5, n.2, p. 121-128, 1979b.

57. Grady, R. **Successfully Applying Software Metrics.** Computer, v.27, n.9, p.18-25, September, 1994.

58. Gurovitz, H. **Delete-Se. Fato: Bilhões de dólares são torrados em investimentos inúteis. Questão: dá para se defender?.** Exame, ano 30, n.12, Ed. Abril, Junho de 1997.

59. Halstead, M. H. ***Language Level, a missing concept in information theory.*** ACM SIGMETRICS Performance Evaluation Review, v.2, n.1, p.10-15, 1973b.

60. Halstead, M. H. ***An Experimental determination of `purity` of a trivial algorithm.*** ACM SIGMETRICS Performance Evaluation Review, v.2, n.1, p.7-9, 1973a.

61. Halstead, M. H. ***Using the methodology of natural science to understand software.*** AFIPS Conference Proc., AFIPS Press, N.J., 1976a.

62. Halstead, M. H. ***The essential design criterion of computer languages: Software Science.*** AFIPS Conference Proc., AFIPS Press, N.J., 1976b.

63. Halstead, M. H. ***Elements of Software Science.*** Elsevier, North Holand, 1977.

64.Halstead, M. H. ***Management Prediction - can software Science help?*** COMPSAC 78, p.126-128, 1978.

65.Halstead, M. H. ***Advances in Software Science.*** Advances in Computers, p.119-172, 1979.

66.HARMLEN, M. V. **Melding User Interface Design with Object** Modelling. v.2, p. 30-37, Nov/Dec, 1996.

67.HEDERSON-SELLERS, B. **Object-Oriented Metrics: Measures of Complexity.** New Jersey, Prentice Hall, 1995.

68.HENDERSON-SELLERS, B. **A Book of Object-Oriented Knowkedge: An Introduction to Object-Oriented Software Engineering.** 2ed. New Jersey, Prentice Hall , 1997.

69.HIGHSMITH, J.; NIX L. **Mission Possible: Feasibility Analisys.** Software Development , v.4 , n.7, p. 40-45, Jul, 1996.

70.HORGAN, J. R. et. al. **Achieving Software Quality with Testing Coverage Measures.** IEEE Computer, v.27, n.9, p.60-69, Set, 1994.

71.Horgan, J. R.; London, S.; Lyu, M. R. **Achieving Software Quality with Testing Coverage Measures**. Computer, v.27, n.9, p.60-70, September, 1994.

72.Humpherey, W. S. ***Characterizing the software process: A maturity framework.*** IEEE software, v.5, n.2, 73-79, 1988.

73.HUMPHREY, W. S. **Managing the Software Process.** Reading, MA, Addison-Wesley, 1989.

74.JACOBSON, I. et. al. **Object-Oriented Software Engineering. A Use Case Driven Approach.** 3ed. Greenwich, Addison-Wesley, 1995.

75.Jones, C. ***Assessment and Control Software Risks.*** Englewood Cliffs, N.J., Yourdon Press, 1994.

76.JONES C. **Why Software Fails.** Software Development , v.4 , n.7, p. 49-54, Jul, 1996.

77.JURAN, J. M. **A Qualidade desde o Projeto: novos passos para o planejamento da qualidade em produtos e serviços.** São Paulo, Pioneira, 1992.

78.Kafura, D.; Henry, S. ***Software quality metrics based on interconnectivity.*** Journal of Systems and Software, p.121-131, 1981.

79.Karunanithi, S.; Bieman, J.M. Candidate reuse metrics for object oriented and ADA software. In IEEE-CS International Software Metrics Symposium, 1993

80.Kearney, J. K. ; Sedlmeyer, R. L. ; Thompson, W. B. ; Gray, M. A.; Adler, M. A. ***Software complexity measurement.*** Communications of the ACM, v.29, n.11, p.1044-1050, 1986.

81.Keufell, W. **CORBA Masterminds Object Management. DBMS**. Tools and Strategies for is Professionals, V.10, n.3, March, 1997.

82.KHOSHAFIAN, S. **Object-Oriented Databases.** New York, Wiley, 1993.

83.Khoshgoftaar, T., M.; Oman, P. **Software Metrics: Charting the Course**. Computer, v.27, n.9, p.13-15, September, 1994.

84.Krantz, D. H.; Luce, R. D.; Suppes, P.; Tversky, A. ***Foundations of Measurements.*** New York, Vol.1, Academic Press, 1971.

85.KUAE, L. K. N. **Diretrizes para Apresentação de Dissertações e Teses.** São Paulo, USP, Escola Politécnica da USP, Serviço de Bibliotecas, 1991.

86.Kyburg, H. E. ***Theory and Measurement.*** Cambridge University Press, Cambridge, 1984.

87.Lake, A..; Cook, C. ***A software complexity metric for C++.*** Technical Report 92-60-03, Computer Science Department, Oregon State University, Corvallis, 1992.

88.LANNING, D. L. **Modeling the Relatioship Between Source Code Complexity and Maintenance Difficulty.** IEEE Computer, v.27, n.9, p.35-41, Set, 1994.

89.Lanning, D. L.; Khoshgoftaar, T. M. **Modeling the Relationship Between Source Code Complexity and Maintenance Difficulty**. Computer, v.27, n.9, p.35-40, September, 1994.

90.Li, W.; Henry, S. ***Maintenance metrics for the object oriented paradigm.*** In IEEE-CS International Software Metrics Symposium, 1993.

91.MARTIN. J ; ODELL J. J. **Object-Oriented Analisys and Design.** New Jersey, Prentice-Hall, 1992.

92.MARTIN, R. C. **Designing Object Oriented C++ applications using Booch method.** New Jersey, Prentice Hall, 1995.

93.McCabe, T. J. ***A complexity measure.*** IEEE Transactions on Software Engineering, v.4, n.4, p. 308-320, 1976.

94.McCall, J.; Richards, P.; Walters, G. ***Factors in Software Quality.*** NTIS AD-A049-014, 015, 055, November, 1977.

95.MCCONNEL, S. **Rapid Development: taming wild software schedules.** Washington, Microsoft Press, 1996.

96.MELTON, A. **Software Measurement.** London, ITC Press, 1995.

97.Microsoft Education and Cerification. ***Designing Components Solutions, Students Workbook***, USA, Microsoft Press, 1995.

98.MÖELLER K. H.; PAULISH, D. J. **Software Metrics: A Practitioner's Guide to Improved Product Development.** Piscataway, IEEE Press, 1993.

99.Mowbray, T; Zahavi, R. **The Essential CORBA - System Integration Using Distributed Objects**. USA, Wiley & Sons, 1995.

100.NORTON, P. ; HOLZNER S. **C++ Programming.** New York, Brady, 1991.

101.Oldehoeft, R. ***A contrast between language level measures.*** IEEE Transactions on Software Engineering, v.3, n.6, p. 476-478, 1977.

102.Oldehoeft, R. R.; Bass, L. J. ***Dynamic software science with applications.*** IEEE Transactions on Software Engineering, v.5, n.5, p. 497-504, 1979.

103.Orfali, R.; Harkey, D.; Edwards, J. **Distributed Object Systems CORBA, Java, and the Object Web**. Software Development, v.5, n.4, p.30-36, April, 1997.

104.Ott, L. M.; Bieman, J. M.; Kang, B. K.; Mehra, B. ***Developing measures of class cohesion for object-oriented software.*** Proc. Annual Oregon Workshop on Software metrics, 1995.

105.Ottenstein, L. M. ***Quantitative estimates of debugging requirements.*** IEEE Transactions on Software Engineering, v.5, n.5, p. 504-514, 1979.

106.Paulish, D. J.; Carleton, A. D. **Case Studies of Software Process-Improvement Measurement**. Computer, v.27, n.9, p.50-57, September, 1994.

107.Paulk, M.; Curtiss, B.; Chrissis, M. B.; Weber, C: V. ***Capability maturiy model for software.*** Technical report SEI-CMU-1-TR-24, Software Engineering Institute, Pittisburgh, PA, 1991.

108.PAULK, M. C. **Capability Maturity Model for Software, Versão 1.1.** Pittsburgh, Pennsylvania, Software Engineering Institute, Carnegie Mellon University, 1993.

109.Pfleeger S. L.; McGowan, C. ***Software metrics in the process maturity framework.*** Journal of Systems and Software, v.122, p.255-261, 1990.

110.Pfleeger, S. L.; Fitzerald, J. C. Rippy, D. A.. ***Using multiple metrics for analisys improvement.*** Software Quality Journal, 1992.

111.PFLEEGER, S. L.; FENTON, N. **Evaluating Software Engineering Standards.** IEEE Computer, v.27, n.9, p.71-98, Set, 1994.

112.Pfleeger, S. L.; Fenton, N.; Page, **Evaluating Software Engineering Standards**. Computer, v.27, n.9, p.71-98 September, 1994.

113.Porter A.. A..; Selby, R. W. ***Empirically guided software development using metric-based classification trees.*** IEEE Software, v.7, n.2, p.46-54, 1990.

114.PRESSMAN, R. S. **Sofware Engineering: a practitioner's Approach**. USA, McGraw-Hill, 1988.

115.Putnam, L. H; Ware M. ***Measures for Excellence: Reliable Software on Time, Within Budget.*** Englewood Cliffs, N.J, Yourdon Press, 1992.

116.QUATRANI, T.; CHONOLES, M. J. **Succeding with Booch and OMT methods: a practical approach.** CA, Addison-Wesley, 1996.

117.Rifkin, S.; Cox, C. ***Measurement in practice. Technical Report*** SEI-CMU-91-TR-16, Software Engineering Institute, Pittisburg, Cambridge, PA, 1991.

118.Rumbaugh, J; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. ***Object-Oriented Modeling and Design***, Englewood Cliffs, N.J., Prentice-Hall, 1991.

119.SADR. B.; DOUSETTE, P. J. **An OO Project Management Strategy.** Computer, v. 29, n.9, p.33-43, Set, 1996.

120.Schneider V. ***Prediction of software effort and project duration - four new formulas.*** ACM SIGPLAN Notices, v.13, n.6, p.49-59, 1978

121.SELIC, B. et. al. **Real-Time Object-Oriented Modeling.** New York, John Wiley & Sons, 1994.

122.Sheetz, S. D.; Tegarden, D. P.; Monarchi, D. E. ***Measuring object-oriented system complexity.*** Proc. 1st Workshop on Information Tecnologies and Systems, December, 1991.

123.Shepherd, G.; Wingo, S. **Mastering DCOM - Windows and Distributed Components**. Software Development, V.5, n.4, p.39-45, April, 1997.

124.Shooman, M. L. ***Software Engineering - Design, Reliability and Management.*** NY, McGraw Hill, 1983.

125.SOMMERVILLE, I. **Software Engineering.** 3ed. Avon, Addison-Wesley, 1989.

126.SPARKS, S. et. al. **Managing Object-Oriented Framework Reuse.** Computer, v. 29, n.9, p.52-61, Set, 1996.

127.Stanley, J. C; Hopkingx, K. D. ***Educational and Psychological Measurement.*** Prentice-hall, N.J. 1972.

128.Stark, G.; Durst, R. C. **Using Metrics in Management Decision Making**. Computer, v.27, n.9, p.42-48, September, 1994.

129.STEUDEL, H. J. **Como escrever as rotinas de qualidade: rotinas e abordagem.** Trad. de Flavio E. F. Morgado, Infobook SA, 1993.

130.STROUSTRUP, B. **The C++ Programming Language.** New Jersey, Addison-Wesley, 1987.

131.Tegarden, D. P.; Sheetz, S.D. ***A Software Complexity model of Object-Oriented Systems.*** Decision Support Systems: The International Jornal, p.241-262, 1995.

132.THOMSETT, R. **Added-Value Metrics: Adding Meanig to Measures.** American Programmer, v.8, n.12, p. 2-13, Dec. 1995.

133.Tingey, M. O. Comparing ISO 9000, Macolm Baldrige and SEI CMM for Software. NJ, Prentice Hall, 1997.

134.Walston C.; Felix C.P. ***On lines of code and programming productivity.*** IBM Systems Journal, v.16, n.4, p.422-423, 1977b.

135.WELLER, E. F. **Using Metrics To Manage Software Projects.** IEEE Computer, v.27, n.9, p.27-34, Set, 1994.

136.Weller, F. E. **Using Metrics to Manage Software Projects**. Computer, v.27, n.9, p.27-33, September, 1994.

137.Woodfield, S. N. ***An experiment on unit increase in problem complexity.*** IEEE Transactions on Software Engineering, v.5, n.2, p.76-79, 1979.

138.Zuse, H. ***Software Complexity Measures and Methods.*** W. de Gruiter, Berlin, 1991.

8.2 Bibliografia de Apoio

1. DEPARTMENT OF DEFENSE. **Military Stardand: Defense System Software Development DOD-STD-2167A.** DoD, June 1995

2. Software Engineering Standards Subcommitttee of the Technical Committee on Software Engineering. **IEEE Standard for a Software Quality Metrics Methodology**, IEEE Std 1061-1992, New York, 1993.

3. Microsoft Education and Cerification. ***Designing Components Solutions, Students Workbook***, USA, Microsoft Press, 1995.