MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-6394-TDI/610

PROPOSTA DE UM ESQUEMA DE CONTROLE DE ELEMENTOS DE SOFTWARE PARA A ABORDAGEM ORIENTADA A OBJETOS

Sergio Norio Itami

Dissertação de Mestrado em Computação Aplicada, orientada pelo Dr. Tatuo Nakanishi, aprovada em abril de 1997.

INPE
São José dos Campos
Abril de 1997

Publicado por:

Coordenação de Ensino, Documentação e
Programas Especiais - CEP

Instituto Nacional de Pesquisas Espaciais - INPE
Caixa Postal 515
12201-970 - São José dos Campos - SP - Brasil
Fone: (012) 345.6911
Fax: (012) 345.6919
E-Mail: marciana@sid.inpe.br

- Solicita-se intercâmbio
- We ask for exchange
- Si sollecita intercambio
- On demande l'échange
- Mann bittet un Austausch
- Pidese canje
- Просим обмену
- 歡迎寄作交換
- 出版物交換のお願い

Publicação Externa - É permitida sua reprodução para interessados.

INPE-6394-TDI/610

PROPOSTA DE UM ESQUEMA DE CONTROLE DE ELEMENTOS DE SOFTWARE PARA A ABORDAGEM ORIENTADA A OBJETOS

Sergio Norio Itami

Dissertação de Mestrado em Computação Aplicada, orientada pelo Dr. Tatuo Nakanishi, aprovada em abril de 1997.

INPE
São José dos Campos
Abril de 1997

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-6394-TDI/610

PROPOSTA DE UM ESQUEMA DE CONTROLE DE ELEMENTOS DE SOFTWARE PARA A ABORDAGEM ORIENTADA A OBJETOS

Sergio Norio Itami

Dissertação de Mestrado em Computação Aplicada, orientada pelo Dr. Tatuo Nakanishi, aprovada em abril de 1997.

INPE
São José dos Campos
Abril de 1997

681.3.06

ITAMI, S.N.
Proposta de um esquema de controle de elementos de software para a abordagem orientada a objetos / S. N. Itami. - São José dos Campos: INPE, 1997.
152 p. - (INPE-6394-TDI/610)

1. Engenharia de software. 2.Ferramentas de controle. 3. Programação orientada a objeto. 4. Gerenciamento de configuração. I. Título.

Aprovada pela Banca Examinadora em cumprimento a requisito exigido para a obtenção do Título de Mestre em Computação Aplicada

Dr. Solon Venâncio de Carvalho
Presidente

Dr. Tatuo Nakanishi
Orientador

Dr. Antônio Miguel Vieira Monteiro
Membro da Banca

Dra. Maria Carolina Monard
Membro da Banca
- Convidada -

Candidato: Sergio Norio Itami

São José dos Campos, 07 de abril de 1997

DEDICATÓRIA

À memória dos meus pais, Massako e Yoshifumi, fonte da presente vida.

À esposa, Célia Elmy, fonte de amor e compreensão.

Aos filhos, Mayumi, Matheus e Rafael, fonte de alegria e de esperança.

AGRADECIMENTOS

Aos Amigos Tatuo e João Bosco pela oportunidade do trabalho.

Ao Amigo Wilson pela motivação e incentivo.

RESUMO

Um dos problemas mais desafiadores no campo da Engenharia de Software é o de gerenciar e controlar as alterações de todos os elementos de software gerados e mantidos durante o ciclo de vida do software. Este é o objetivo do Gerenciamento de Configuração de Software - GCS. O progresso da tecnologia de GCS permitiu a ampliação da sua funcionalidade, tornando o GCS não somente um elemento de gerenciamento mas também um elemento de suporte aos trabalhos dos desenvolvedores. Procurando buscar soluções para a abordagem orientada a objeto, com relação às atividades do GCS, este trabalho propõe um Esquema de Controle de Elementos de Software cuja finalidade é gerenciar e controlar as alterações dos elementos de software, entre os quais as classes de objetos, gerados e mantidos durante o desenvolvimento de software orientado a objeto.

PROPOSAL OF A SOFTWARE ELEMENTS CONTROL SCHEME TO THE OBJECT-ORIENTED APPROACH

ABSTRACT

One of the challenging problems in the field of Software Engineering is the management and the control of the changes of all software elements generated and maintained during the software's life cycle. Solving this problem is the goal of Software Configuration Management (SCM). The progress of SCM's technology allowed the enlargement of its functionality, making it a managing element tool as well as a developer's task supporting one. Researching solutions for SCM's activities related to the object-oriented approach, this work proposes a Software Elements Control Scheme which has the capability of managing and controlling the changes of software elements, including classes of objects, generated and maintained during object-oriented software development.

SUMÁRIO

Pág.

LISTA DE FIGURAS.....xvii
LISTA DE TABELAS.....xix

CAPÍTULO 1 - INTRODUÇÃO..... 1

CAPÍTULO 2 - GERENCIAMENTO DE DESENVOLVIMENTO DE SOFTWARE..... 7

2.1 - CONSIDERAÇÕES INICIAIS..... 7
2.2 - METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE.....13
2.3 - CONTROLE DE DESENVOLVIMENTO DE SOFTWARE.....15
2.4 - ABORDAGENS DE GERENCIAMENTO DE SOFTWARE.....18
2.4.1 - GERENCIAMENTO DO CICLO DE VIDA.....20
2.4.2 - GERENCIAMENTO DO PRODUTO.....23
2.4.3 - GERENCIAMENTO DAS ATIVIDADES.....24
2.4.4 - GERENCIAMENTO DOS RECURSOS.....25
2.4.5 - GERENCIAMENTO DOS ELEMENTOS DE SUPORTE.....25
2.5 - O PAPEL DO GERENCIAMENTO DE CONFIGURAÇÃO DE SOFTWARE.....26
2.5.1 - FUNÇÃO DE GERENCIAMENTO.....27
2.5.2 - FUNÇÃO DE SUPORTE AO DESENVOLVEDOR.....28

CAPÍTULO 3 - GERENCIAMENTO DE CONFIGURAÇÃO DE SOFTWARE.31

3.1 - ORIGEM.....31
3.2 - EVOLUÇÃO.....34
3.3 - DEFINIÇÃO.....37
3.3.1 - ASPECTOS OPERACIONAIS CLÁSSICOS.....38
3.3.2 - ASPECTOS OPERACIONAIS COMPLEMENTARES.....41
3.4 - USUÁRIOS.....42

3.5 - REQUISITOS DE FUNCIONALIDADE.....45
3.6 - DIFICULDADES DE IMPLEMENTAÇÃO.....47
3.7 - AMBIENTES INTEGRADOS VERSUS FERRAMENTAS INTEGRADAS.....51
3.8 - PARADIGMAS DO SISTEMA GCS.....54
3.8.1 - PARADIGMA CHECKOUT/CHECKIN.....55
3.8.2 - PARADIGMA DA COMPOSIÇÃO.....56
3.8.3 - PARADIGMA DA TRANSAÇÃO LONGA.....58
3.8.4 - PARADIGMA DO CONJUNTO DE ALTERAÇÃO.....60
3.8.5 - OS PARADIGMAS E OS SISTEMAS GCS.....62
3.9 - ABORDAGENS DE CONFIGURAÇÃO.....62

CAPÍTULO 4 - PARADIGMA ORIENTADO A OBJETO.....67

4.1 - CONSIDERAÇÕES INICIAIS.....67
4.2 - CONCEITOS BÁSICOS.....68
4.3 - PROBLEMAS RELACIONADOS.....71
4.3.1 - PROBLEMA DO ENTENDIMENTO.....73
4.3.2 - PROBLEMA DA COMPLEXIDADE DA DEPENDÊNCIA.....74
4.3.3 - PROBLEMA DE TESTAR O COMPORTAMENTO DE ESTADO...78
4.3.4 - PROBLEMA DAS FERRAMENTAS DE SUPORTE.....80

CAPÍTULO 5 - SISTEMA DE CONTROLE DE ELEMENTOS DE SOFTWARE PARA A ABORDAGEM ESTRUTURADA.....83

5.1 - ASPECTOS GERAIS.....83
5.2 - ARQUITETURA.....84
5.3 - ELEMENTOS CONTROLADOS.....85
5.4 - RECURSOS COMPUTACIONAIS.....87
5.5 - PROCEDIMENTOS PARA CONTROLE DE ELEMENTOS.....89
5.6 - CONSIDERAÇÕES FINAIS.....90

CAPÍTULO 6 - ESQUEMA DE CONTROLE DE ELEMENTOS DE SOFTWARE PARA A ABORDAGEM ORIENTADA A OBJETOS.....91

6.1 - DESCRIÇÃO GERAL.....91
6.2 - METODOLOGIA DE COAD & YOURDON.....93
6.3 - AMBIENTES DE DESENVOLVIMENTO.....96
6.3.1 - AMBIENTE DE SISTEMA.....96
6.3.2 - AMBIENTE DE CONTROLE DE CONFIGURAÇÃO.....98
6.3.3 - AMBIENTE DE CONTROLE DE QUALIDADE.....99
6.3.4 - AMBIENTE DE SUBSISTEMAS.....100
6.3.5 - AMBIENTE DE INTEGRAÇÃO DE SUBSISTEMA.....100
6.3.6 - AMBIENTE DE INTEGRAÇÃO DE SISTEMA.....101
6.3.7 - AMBIENTE DE PROTOTIPAÇÃO.....101
6.4 - RECURSOS COMPUTACIONAIS.....102
6.5 - BANCO DE DADOS DE ELEMENTOS CONTROLADOS.....104
6.5.1 - ELEMENTOS DE SOFTWARE CONTROLADOS.....105
6.5.2 - RELACIONAMENTOS CONTROLADOS.....108
6.6 - SISTEMA DE SUPORTE AO CONTROLE DE ELEMENTOS.....113
6.6.1 - ENCAMINHAMENTO DE ELEMENTOS PARA CONTROLE.....115
6.6.2 - ALTERAÇÃO DE ELEMENTOS CONTROLADOS.....117

CAPÍTULO 7 - CONCLUSÕES.....121

REFERÊNCIAS BIBLIOGRÁFICAS.....125

LISTA DE FIGURAS

Pág.

2.1 - Sistema de Controle de Projeto.....17
2.2 - Estágios no desenvolvimento de software.....22
3.1 - Evolução da Tecnologia de Integração.....52
3.2 - Modelo Checkout/Checkin.....55
3.3 - Seleção de Versão do Componente.....57
3.4 - Histórico de Versão de uma Configuração.....60
3.5 - Conjunto de Alterações em uso.....61
3.6 - Processo de Configuração Estática.....63
3.7 - Processo de Configuração Dinâmica.....65
4.1 - Código Fonte da Caixa de Moeda.....79
6.1 - Modelo Cinco Camadas.....93
6.2 - Modelo Quatro Componentes.....94
6.3 - Arquitetura dos Ambientes de Desenvolvimento.....97
6.4 - Processo de Entrega de Elementos de Software....116
6.5 - Processo de Alteração de Elementos de Software..118

LISTA DE TABELAS

Pág.

3.1 - Os Paradigmas e os Sistemas GCS.....62
5.1 - Exemplo de Elementos Controlados na Abordagem Estruturada.....86
6.1 - Exemplo de Elementos Controlados na Abordagem Orientada a Objeto.....106

CAPÍTULO 1

INTRODUÇÃO

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

As técnicas e métodos, ferramentas e ambientes aplicados ao processo de desenvolvimento de software têm evoluído nestes últimos anos e têm contribuído para o aumento da qualidade e da produtividade de software. Entretanto, um dos problemas mais desafiadores durante o desenvolvimento de grandes sistemas de software, é o de gerenciar e controlar as alterações de todos os elementos e itens de software gerados e mantidos durante o ciclo de vida do software. Este é o objetivo do Gerenciamento de Configuração de Software (GCS) (Ferreira, 1996; Harter, 1991).

O GCS, apesar de ser considerado um elemento importante no processo de desenvolvimento de software, na prática, infelizmente, é freqüentemente ignorado e muitas vezes considerado como obstrutivo aos trabalhos da equipe de desenvolvimento. Os métodos e as ferramentas de GCS, sob o ponto de vista dos desenvolvedores, são considerados como elementos enfadonhos

e desnecessários, e que impõem trabalhos adicionais trazendo poucos benefícios às suas tarefas (Dart, 1992; Feiler, 1988; Bersoff et al., 1980).

Para diminuir esta resistência por parte dos desenvolvedores de software em relação às atividades do GCS, tem-se verificado uma ampliação cada vez maior das suas funcionalidades, fazendo com que o GCS seja considerado não somente como um elemento de gerenciamento, mas também como um elemento de suporte aos trabalhos dos desenvolvedores (Feiler, 1990).

O progresso da área de GCS nestes últimos anos, pode ser constatado em função do número de relatórios e conferências técnicas abordando este assunto, da quantidade de ferramentas comerciais disponíveis, da incorporação das facilidades do GCS em ambientes de desenvolvimento de software e da exigência de GCS estabelecida em normas e padrões de qualidade (Humprey, 1988).

A atual tecnologia de GCS, assim como a maioria dos processos, métodos, ferramentas e ambientes de desenvolvimento, estão totalmente voltados para a abordagem estruturada, em função da própria maturidade desta abordagem. Mas com a nova abordagem orientada a objeto, que trouxe novos conceitos e métodos para a Engenharia de Software, fez com que a tecnologia de desenvolvimento de software também se voltasse para esta nova metodologia (Kung et al., 1995; Lejter et al., 1992; Gibbs et al.,1990; Korson e McGregor, 1990).

A medida que a abordagem de orientação a objeto vai ganhando maturidade, a tecnologia de GCS também vai acompanhando este processo, assim como já acontece na área de banco de dados, nas ferramentas CASE e nos métodos

de análise e projeto de sistemas, onde já se pode encontrar aplicações voltadas para a orientação a objeto (Coad e Yourdon, 1992; Jacobson et al., 1992; Booch, 1991).

Procurando buscar soluções para a abordagem orientada a objeto, com relação às atividades do GCS, este trabalho propõe um Esquema de Controle de Elementos de Software, que tem por objetivo gerenciar e controlar as alterações dos elementos de software, entre os quais as classes/objetos, gerados e mantidos durante o ciclo de vida do software. Esta proposta certamente pode servir de base para uma futura implementação de um sistema GCS que atenda de forma completa ao gerenciamento de desenvolvimento de software orientado a objeto.

O Esquema de Controle proposto tomou por base uma aplicação prática utilizada para a abordagem estruturada, que foi o Sistema de Controle de Elementos de Software utilizado durante o desenvolvimento do Sistema de Controle de Satélites (SICS), um software de grande porte para controlar o 1° Satélite Brasileiro de Coleta de Dados (SCD1).

A utilização de tal Sistema de Controle permitiu alcançar excelentes níveis de qualidade e de produtividade, proporcionando a manipulação de cerca de 2700 módulos fontes (programas, funções, sub-rotinas, arquivos "includes") e outros milhares de itens de software (parâmetros de rotina, nomes lógicos, "events flags", fila de mensagens, área de dados globais etc.), produzidos por dezenas de desenvolvedores, e aplicados não somente durante o desenvolvimento do SICS mas também durante a sua manutenção. O controle de todos os produtos e itens de software incluiu o controle de versão, o controle de qualidade (correteza, consistência etc.) e o controle das

alterações (encaminhamento, análise, aprovação, implementação, verificação) (Cunha e Nakanishi, 1993a,b).

O Esquema de Controle proposto tem por características a utilização de um modelo de estrutura de dependência para descrever as interdependências entre os produtos e os itens de software de forma a permitir a identificação dos produtos/itens que serão afetados por uma dada alteração; de um sistema de controle de bibliotecas onde estarão armazenados os produtos e itens propriamente ditos; de uma organização de ambientes de desenvolvimento inclusive da organização da equipe de trabalho; e da definição de uma série de procedimentos estabelecendo normas e responsabilidades para garantir a correta execução do esquema de controle.

Para uma melhor compreensão da importância de um Sistema de Controle deste gênero, decidiu-se por apresentar primeiramente as considerações gerais a respeito do GCS como um todo bem como das suas principais características atualmente implantadas; e também por apresentar os conceitos principais da abordagem de orientação a objeto e as conseqüências advindas desta nova metodologia principalmente na área de gerenciamento e controle de desenvolvimento de software.

No Capítulo 2, tem-se a apresentação dos principais aspectos de gerenciamento de um projeto de software, mostrando a importância do papel do GCS na busca de melhor qualidade e produtividade de software.

A pesquisa dos avanços verificados na tecnologia de GCS, bem como seus principais conceitos e funcionalidades, são tratados no Capítulo 3.

O Capítulo 4 aborda os principais conceitos da abordagem de orientação a objeto, descrevendo as principais dificuldades no aspecto de gerenciamento do desenvolvimento de software ao se utilizar este tipo de abordagem.

No Capítulo 5 tem-se a apresentação do Sistema de Controle de Elementos de Software para a abordagem estruturada que foi implantado com sucesso.

E, finalmente, no Capítulo 6 tem-se a proposta de um Esquema de Controle Elementos de Software para a abordagem orientada a objeto.

CAPÍTULO 2

GERENCIAMENTO DE DESENVOLVIMENTO DE SOFTWARE

2.1 - Considerações Iniciais

O desenvolvimento de software, apesar da evolução das técnicas, das metodologias, das ferramentas de trabalho e dos próprios computadores, ainda não atingiu o mesmo nível de evolução em termos de produtividade e de confiabilidade que o desenvolvimento de projetos em outros ramos da tecnologia, como por exemplo, o desenvolvimento de hardware.

Freqüentemente o que se vê em relação ao desenvolvimento de software é que o produto final não atende totalmente aos requisitos do cliente, possui uma documentação insuficiente tanto em relação aos documentos de projeto quanto aos manuais, o custo final é maior do que aquele previsto, apresenta falhas de operação após sua utilização e, principalmente, é entregue bem mais tarde do que a data originalmente prevista (Cunha e Nakanishi, 1993a,b; Rook, 1988).

Estes problemas são comuns no desenvolvimento de sistemas em todas as áreas tecnológicas mas na área do software, principalmente no caso de grandes sistemas de software, estes problemas surgem com maior freqüência e com uma ordem de grandeza superior.

São duas as causas principais deste atraso tecnológico do desenvolvimento de software: a própria complexidade do desenvolvimento de software e o gerenciamento inadequado deste desenvolvimento.

O desenvolvimento de grandes sistemas de software é uma tarefa intrinsicamente complexa devido à sua natureza ampla e multidisciplinar envolvendo vários tópicos, tais como: metodologias e técnicas de desenvolvimento, equilíbrio entre custo-qualidade-benefício, características das aplicações, quantidade de informação a ser manipulada, processo iterativo de desenvolvimento, preparação da equipe etc.

Comparando-se com o desenvolvimento de hardware, que também possui sistemas complexos, a área do software apresenta algumas características peculiares que tornam o seu desenvolvimento ainda mais complexo (Coad e Yourdon, 1992; Booch, 1991; Bersoff et al., 1980).

O software não tem uma aparência física, não é um elemento palpável como, por exemplo, um equipamento, uma placa eletrônica, um "chi". Ele é um elemento abstrato e a sua existência é percebida através da sua execução em um computador. Isto faz com que haja uma maior dificuldade em integrar suas várias partes, acompanhar o seu progresso e fazer as correções necessárias durante seu desenvolvimento.

O software é fácil de ser alterado, mas esta propriedade que, a princípio, parece ser vantajosa, é, pelo contrário, um fator preocupante durante o desenvolvimento de software. Pela facilidade de se alterar um código fonte, há uma tendência em se fazer modificações a toda hora e de qualquer forma, até que o código fonte execute a função para o qual foi projetado. Como conseqüência há necessidade de maior controle no desenvolvimento de software.

No software, caso o desenvolvimento não seja realizado dentro da melhor linha de programação que se

baseia na forte coesão e baixo acoplamento entre módulos, haverá uma dependência muito grande entre módulos e isto fará com que a propagação dos efeitos de uma alteração seja explosiva, isto é, ao se alterar um único módulo de programa, pode haver a necessidade de se alterar outras dezenas de módulos. E caso não exista um adequado controle de dependência entre módulos, levar-se-á mais tempo para a correção dos módulos dependentes, provocando atrasos na conclusão do software.

O software, além de dados físicos, também trabalha com dados lógicos, o que torna mais complexo o seu gerenciamento e controle, pois uma simples alteração física de uma linha de código poderá alterar totalmente a lógica de um programa e a identificação deste problema, além de não poder ser feita automaticamente, poderá tomar muito tempo dos analistas e programadores.

Apesar de atualmente haver uma retomada em relação à reutilização de software, é muito pouco o reaproveitamento de softwares já produzidos. Geralmente o desenvolvimento de software é realizado sem se preocupar com a futura reutilização. Além disso outro fator que dificulta ou torna impossível a reutilização do software é a documentação de projeto que muitas vezes não existe, não é completa ou não é atualizada (Ferreira, 1996; Rook, 1988).

Portanto, estas características peculiares do software fazem com que seu desenvolvimento seja mais complexo que o de hardware.

A segunda causa do atraso tecnológico da área de software em relação a outras áreas tecnológicas está relacionada com o gerenciamento inadequado do desenvolvimento do software.

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

Independentemente da capacidade gerencial, existe outra dificuldade no gerenciamento de software que está relacionada com a abordagem de desenvolvimento utilizada nas primeiras produções de software e que ainda é utilizada no desenvolvimento de sistemas de pequeno porte. Trata-se da abordagem baseada no conceito de que a programação é uma arte e não uma ciência (Rook, 1988). Neste tipo de gerenciamento é permitido aos analistas e programadores a total liberdade para criar e alterar seus programas, quando e como quiserem de acordo com a sua criatividade. Ao se tentar restringir essa liberdade com a introdução de um controle de gerenciamento ou de uma padronização de projeto e de programação, alega-se que a criatividade será inibida e haverá prejuízo na produtividade pois os analistas e programadores consideram o controle e a padronização como perda de tempo ou como uma burocracia que nada tem a acrescentar.

Logo, além do problema relativo ao gerenciamento propriamente dito, existe esta questão de cultura arraigada na maioria dos analistas e programadores

em rejeitar, ou aceitar a contragosto, qualquer tipo de controle nos seus trabalhos. Neste caso o gerente deverá ter a habilidade adicional para conduzir uma equipe de "artistas" criativos, procurando guiar o individualismo de cada um na direção de um objetivo comum, procurando fazer com que haja um interfaceamento entre eles, e convencê-los a aceitar um mínimo de controle para que se possa concluir o software com sucesso.

Por causa de todas estas dificuldades é que fabricantes e pesquisadores de software têm procurado incutir ou até mesmo impor uma maior disciplina nas atividades técnicas de desenvolvimento, introduzindo formalismos na sua execução, com a finalidade de facilitar o controle de produção de software, de forma a se obter um produto final de maior qualidade (Bersoff et al., 1980).

Portanto, a ênfase nas abordagens modernas dentro da Engenharia de Software é de dar uma maior estruturação ao processo de desenvolvimento procurando facilitar o seu gerenciamento e diminuir a complexidade do desenvolvimento de software através da pesquisa de novas metodologias e técnicas de desenvolvimento. Há também o incentivo no desenvolvimento e aperfeiçoamento na utilização de ferramentas CASE e de ambientes de desenvolvimento automatizados. O objetivo principal é compatibilizar a disciplina com a criatividade de forma a aumentar a produtividade e a qualidade do software (Cunha e Nakanishi, 1993a,b; Rook, 1988).

Em termos de qualidade de software, o padrão que se está buscando tem como principais atributos (Meyer, 1988; Arthur, 1985):

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

- Eficiência - melhor uso da capacidade dos recursos de hardware e de software.

- Flexibilidade - quantidade mínima de esforço necessário para melhorar o software.

- Integridade - habilidade do software de proteger seus dados internos.

- Interoperabilidade - facilidade do software se acoplar em outros software.

- Manutenibilidade - facilidade para localizar e reparar erros no software.

- Portabilidade - facilidade do software para ser transferido a vários ambientes de hardware ou de software.

- Confiabilidade - grau de execução sem falhas em condições anormais.

- Reusabilidade - facilidade de reusar partes do software.

- Testabilidade - quantidade mínima de esforço para testar a estrutura e a correteza do código.

- Usabilidade - facilidade de operação do software por parte do usuário.

2.2 - Metodologia de Desenvolvimento de Software

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

Os métodos têm por objetivo gerar um conjunto de modelos que descrevem, através de uma notação própria, os vários aspectos do software durante seu desenvolvimento. Existem, atualmente, muitos métodos de desenvolvimento de software que, basicamente, podem ser classificados em três categorias principais: métodos estruturados "top-down", métodos dirigido a dados e métodos orientados a objetos (Booch, 1991).

Quanto aos procedimentos, eles têm a finalidade de definir um conjunto de ações que devem ser executadas pelos membros da equipe de projeto na realização de suas tarefas de forma a padronizar o processo de desenvolvimento.

Ao se adotar uma determinada metodologia de desenvolvimento de software, tem-se por objetivo impor uma disciplina de trabalho, definir os produtos intermediários dos modelos que serão utilizados na comunicação entre as equipes de projeto, e estabelecer os marcos necessários para gerenciar o progresso do desenvolvimento.

Não existe uma metodologia universal que possa ser aplicada em todos os projetos de todas as empresas pois as estruturas organizacionais das empresas, os tipos de aplicações do software, os métodos de desenvolvimento fazem com que existam diferentes metodologias, cada qual sendo melhor aproveitada em determinados projetos ou por determinadas empresas e equipes.

Uma metodologia além de incluir os métodos para solucionar as dificuldades do desenvolvimento de software também deve incluir os procedimentos de gerenciamento para controlar o processo de desenvolvimento como um todo. Desta forma, os métodos e os procedimentos de gerenciamento se complementam pois os primeiros fornecem os produtos e os marcos necessários para o gerenciamento, e os procedimentos de gerenciamento executam o controle do desenvolvimento destes produtos e marcos (Rook, 1988).

A necessidade de se incluir os aspectos de gerenciamento dentro de uma metodologia de desenvolvimento se torna mais evidente quando o sistema de software a ser desenvolvido é de grande porte. Geralmente, os grandes sistemas de software possuem múltiplas linhas de desenvolvimento, vários níveis de visibilidade de progresso do desenvolvimento, grande quantidade de desenvolvedores e de tarefas.

Para facilitar o trabalho do desenvolvimento de software, criou-se o conceito de ambientes de desenvolvimento, onde se faz uma divisão em ambientes especializados, cada qual agrupando atividades afins e/ou partes do processo de desenvolvimento. Essa divisão leva em consideração a metodologia adotada e os recursos disponíveis (Cunha e Nakanishi, 1993a,b).

Dentro da concepção destes ambientes, os aspectos técnicos propriamente ditos ficaram isolados dos aspectos de gerenciamento e controle. A despreocupação com estes últimos aspectos levou a um desnível em termos de avanço tecnológico entre a produção e o gerenciamento de tal forma que os analistas e programadores possuem ferramentas automatizadas para facilitar suas atividades enquanto os procedimentos de gerenciamento e de controle têm que ser realizados de forma manual (Dart, 1992; Feiler, 1990).

Hoje com a crescente demanda por software de qualidade existe a necessidade urgente de uso de técnicas modernas em todas as fases de desenvolvimento de software, inclusive de gerenciamento de processos, apoiados por ferramentas ou ambientes automatizados, com suporte à descrição e à execução de processos auxiliando e controlando todas as atividades envolvidas na produção e manutenção de um software (Feiler e Smeaton, 1988).

2.3 - Controle de Desenvolvimento de Software

O processo de desenvolvimento de software pode ser visto como um controle de riscos (Boehm, 1991), riscos estes que poderão se transformar em falhas financeiras (prazo e orçamento ultrapassados) e/ou falhas técnicas (não atendimento aos requisitos do cliente). As fontes de riscos podem ser classificadas em três categorias:

- Perturbações: são as alterações dos requisitos do cliente e a detecção de problemas, erros ou falhas.

- Pessoal: é a falta de pessoal com capacitação tecnológica.

- Ambiente de projeto: são a indefinição da metodologia, falta de padrões de qualidade, detecção de erros muito tarde e inadequação do controle, da habilidade técnica, do suporte e da visibilidade.

Para que o desenvolvimento do software obtenha sucesso, deve-se procurar identificar, controlar e, se possível, eliminar os riscos que podem gerar prejuízos ao desenvolvimento. Geralmente, é utilizado um sistema de controle de projeto que tem por base o conceito de "feedback" ou realimentação, que irá assegurar um controle mais eficiente do sistema e irá orientar o desenvolvimento para atingir seu objetivo, que é obter um software com qualidade.

A Figura 2.1 ilustra, de forma resumida, um sistema de controle de projeto para desenvolvimento de software baseado no conceito da realimentação (Rook, 1988).

O módulo Desenvolvimento Técnico corresponde ao sistema que está sendo desenvolvido e que será controlado, e o módulo Gerenciamento de Projeto corresponde ao controlador do desenvolvimento. Neste tipo de controle, baseado na realimentação, tem-se como pré-requisito as estimativas do desenvolvimento, que serão tomadas como base inicial, a partir da qual serão feitas as realimentações e o acompanhamento do progresso do desenvolvimento.

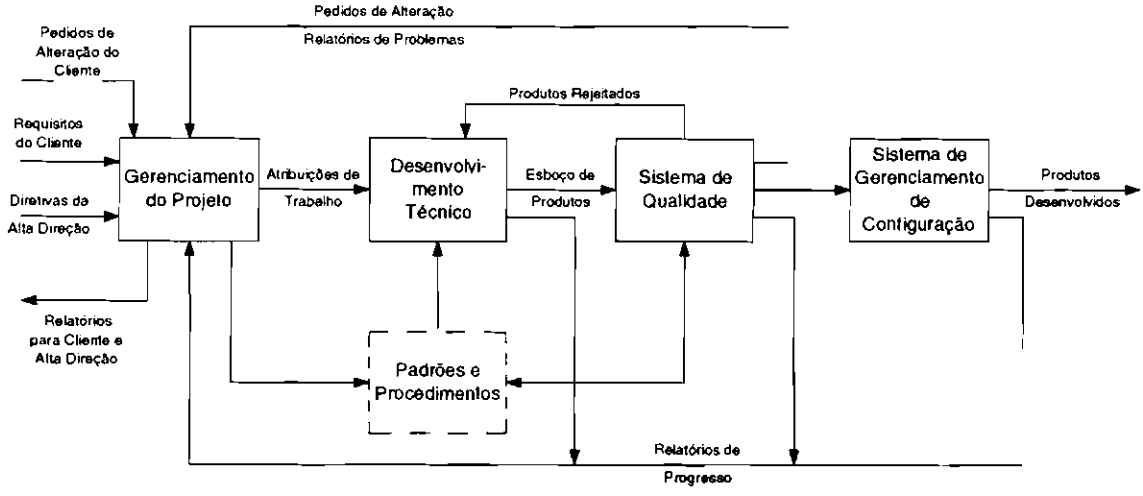

Fig. 2.1 - Sistema de Controle de Projeto.

FONTE: Rook (1988), p. 109.

Como se pode verificar na Figura 2.1, a realimentação age diretamente no Desenvolvimento Técnico, no Sistema de Qualidade e no Sistema de Gerenciamento de Configuração. Os Sistemas de Qualidade e de Gerenciamento de Configuração operam continuamente durante o processo de desenvolvimento e não somente na entrega dos produtos finais.

Enquanto o fluxo de Produtos Rejeitados representa o trabalho sobre o produto em si, o fluxo de realimentação Relatórios de Progresso representa a base para o controle. O controle consiste em obter as informações necessárias para tomar as decisões corretas e/ou corrigir erros, procurando controlar prazo e orçamento, minimizando os riscos técnicos.

O fluxo de Pedidos de Alteração e Relatórios de Problemas representa o caminho inevitável das alterações, que o gerente de projeto deve controlar através de procedimentos apropriados, tal como implantar um Comitê de Controle de Alteração por onde todas as alterações

deverão passar antes de serem implementadas (Cunha e Nakanishi, 1993a,b).

A operação de controle de projeto ilustrada na Figura 2.1 depende de uma organização onde a responsabilidade é claramente definida e respeitada para os quatro módulos apresentados: Gerenciamento de Projeto, Desenvolvimento Técnico, Sistema de Qualidade e Sistema de Gerenciamento de Configuração (Feiler e Smeaton, 1988; Rook, 1988).

2.4 - Abordagens de Gerenciamento de Software

O gerenciamento do desenvolvimento de software abrange várias atividades que vão desde a definição do ambiente de desenvolvimento, da formação da equipe de trabalho, passando pelo tratamento das perturbações que acontecerão ao longo do projeto, até a entrega final do produto ao cliente. Pode-se também incluir nestas atividades o ato de escolher a metodologia de desenvolvimento de software que será utilizada no projeto.

Ao longo dos últimos anos, houve um progresso muito grande das técnicas de desenvolvimento de software (Coad e Yourdon, 1992; Jacobson et al., 1992; Booch, 1991), tais como a utilização de especificações formais, a análise e o projeto estruturados, a programação estruturada, os métodos orientados a objetos, etc. Todas estas novas características que foram surgindo dentro da Engenharia de Software, as técnicas, métodos, padronizações de documentação, métricas, gerenciamento de configuração, controle de qualidade etc., constituem o conjunto de elementos com os quais se deve trabalhar dentro do processo de desenvolvimento de software.

Podem ser acrescentados ainda a este conjunto, as características do ambiente de desenvolvimento, os cronogramas e orçamentos sob os quais deve-se trabalhar, e outras considerações operacionais. E além disso, deve-se ainda, padronizar as atividades e os processos necessários tanto para a implementação quanto para o controle desses elementos durante todo o desenvolvimento do software.

Entretanto, esta seleção cuidadosa de todos os elementos que serão utilizados no desenvolvimento de software, não é suficiente para garantir o sucesso do produto. O sucesso ou a falha do desenvolvimento é determinado pelo seu gerenciamento, principalmente, para grandes projetos de software. Nenhuma metodologia, técnica de programação ou de projeto, ferramentas, computadores, por mais sofisticados que sejam, podem prescindir de uma abordagem sistemática no gerenciamento do projeto.

Portanto, o gerenciamento de projeto, que envolve as tarefas de planejar, definir, atribuir o trabalho para as equipes de desenvolvimento, monitorar o "status" e o progresso do desenvolvimento, tomar decisões necessárias, re-planejar e informar o cliente e o gerenciamento de alto nível sobre o progresso do desenvolvimento, é o elemento chave para se obter um software de qualidade (Feiler e Smeaton, 1988; Rook, 1988).

Com a finalidade de se organizar e tornar mais fácil a visualização dos elementos do gerenciamento, já que a quantidade de elementos necessários ao desenvolvimento de software é muito grande, pode-se dividi-los em subconjuntos de elementos e/ou de atividades, gerando as diferentes abordagens de gerenciamento.

Existem várias formas de se abordar este conjunto de elementos, e a descrição apresentada a seguir foi baseada nos trabalhos (Feiler, 1990; Rook, 1988). Pode-se dividir o gerenciamento em cinco grandes componentes: Gerenciamento do Ciclo de Vida, Gerenciamento do Produto, Gerenciamento das Atividades, Gerenciamento dos Recursos e Gerenciamento dos Elementos de Suporte.

2.4.1 - Gerenciamento do Ciclo de Vida

Com a finalidade de se estruturar o desenvolvimento de software há a necessidade de definir o seu processo de desenvolvimento estabelecendo os estágios pelos quais o software vai passar enquanto está sendo construído. Na Engenharia de Software adotou-se o termo *ciclo de vida* para identificar as várias fases deste desenvolvimento (Bersoff et al., 1980).

Atualmente existem vários modelos de ciclo de vida, cada qual com seu número e denominação de fases e respectiva interação entre elas. Há desde o tradicional modelo cascata, passando pelo modelo espiral e modelo de prototipação, e chegando-se aos mais modernos relacionados com os métodos de desenvolvimento orientados a objetos (Henderson-Sellers e Edwards, 1990).

Como a idéia básica do modelo ciclo de vida é proporcionar meios de gerenciar e de visualizar o progresso de desenvolvimento de software, em cada fase são definidos os produtos que deverão ser entregues e as atividades necessárias para se obter tais produtos. Estes produtos deverão atender aos requisitos estabelecidos na fase anterior e, se estiverem de acordo, serão aprovados para a próxima fase.

O final de cada fase representa um marco de progresso do desenvolvimento e é definido como uma linha de demarcação[1]. Os produtos da fase posterior a esta linha de demarcação serão medidos e verificados tomando como base os produtos gerados na fase anterior.

Este processo é apresentado na Figura 2.2 que utiliza o diagrama em V mostrando a simetria entre a decomposição sucessiva do projeto e a construção do produto final por estágios sucessivos de integração e teste. Cada fase do processo é verificado e validado de acordo com as linhas de demarcação anteriores (Rook, 1988).

Uma das dificuldades do gerenciamento do ciclo de vida é que não é possível estabelecer as linhas de demarcação com rigidez, no sentido de não poder retornar às fases anteriores uma vez entregue os produtos daquela fase ou no sentido de não poder avançar para as fases seguintes antes do término da fase onde se encontra. Atualmente, a iteração entre as fases do ciclo de vida é muito dinâmica fazendo com que as atividades de uma fase posterior necessitem de informações da fase anterior antes de seu término, que os problemas encontrados em uma fase posterior necessite da revisão dos produtos das fases anteriores, e que a implementação das alterações dos requisitos do cliente, necessariamente tenha que passar por todas as fases já encerradas.

Este dinamismo dentro do ciclo de vida deve ser muito bem gerenciado a fim de se ter um visão correta e um total controle do software que está sendo produzido. Deve-se observar neste Gerenciamento se o modelo do ciclo

[1] Em inglês: baselines

de vida adotado retrata fielmente o desenvolvimento que está sendo realizado.

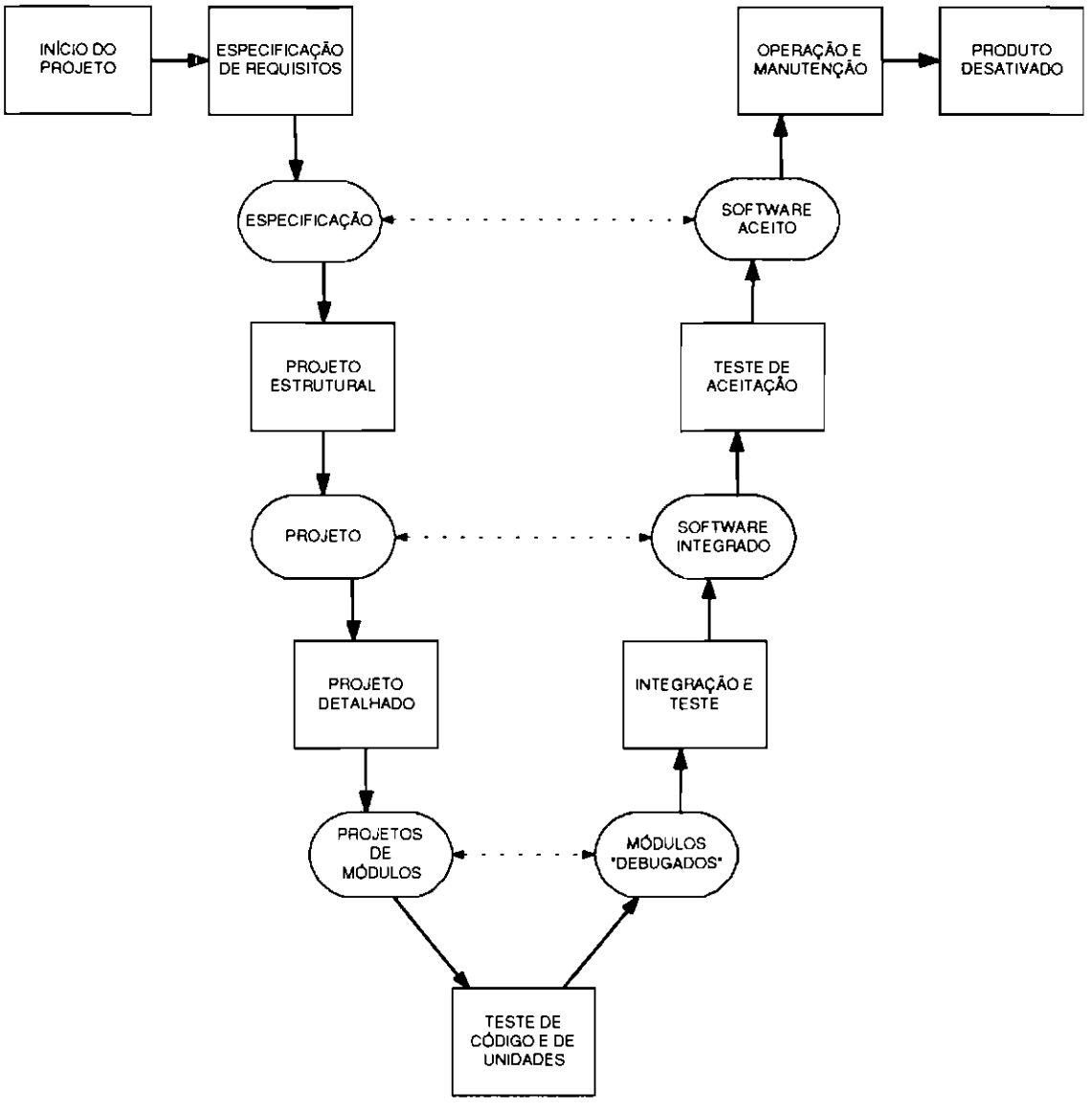

Fig. 2.2 - Estágios no desenvolvimento de software

FONTE: Rook (1988), p. 112.

2.4.2 - Gerenciamento do Produto

No Gerenciamento do Produto estão incluídos os controles relacionados aos produtos, intermediários e finais, propriamente ditos que estão sendo gerados ao longo do desenvolvimento do projeto. Dentre estes controles podem ser destacados o Controle de Versão, o Controle de Configuração, o Controle de Atualização, e Controle de Dependência.

O Controle de Versão refere-se ao controle e à manutenção das versões sucessivas, paralelas e derivadas dos produtos. As versões sucessivas são aquelas versões pertencentes à linha normal de desenvolvimento que surgem em decorrência das alterações que naturalmente ocorrem durante o desenvolvimento. As versões paralelas são as versões que pertencem às linhas diferentes de desenvolvimento, ou seja, o mesmo software sendo desenvolvido para diferentes sistemas operacionais ou para diferentes computadores. E as versões derivadas são as versões dentro da mesma linha de desenvolvimento mas que implementam novas características, como por exemplo, a utilização de novas interfaces gráficas, de outros métodos de desenvolvimento, ou de outras linguagens de programação. O Controle de Versão está relacionado com as operações de seleção e recuperação de versões bem como da coordenação das modificações das versões dos produtos.

O Controle de Configuração, atividade específica dentro do processo geral de Gerenciamento de Configuração, se refere à descrição, construção e reconstrução de todos os componentes do sistema, e à construção de sistemas híbridos compostos por um misto de versões diferentes de cada componente.

O Controle de Atualização do produto representa as facilidades para atualizar automaticamente os produtos que foram alterados bem como os produtos que possuem um relacionamento de dependência com eles.

O Controle de Dependência refere-se à facilidades de implementação para definir e consultar as relações de dependência entre os produtos de forma que cada um deles possa ser rastreado em função do seu "status", de onde se encontra a sua especificação e com quais produtos ele está relacionado.

2.4.3 - Gerenciamento das Atividades

O Gerenciamento das Atividades inclui a definição dos padrões, procedimentos, e protocolos relativos aos processos que devem ser executados durante o desenvolvimento de software com o objetivo de se ter um controle ordenado e consistente destas atividades. Neste tipo de controle inclui-se o Controle de Alteração, o Controle de Qualidade e o Controle de Tarefas.

O Controle de Alteração está relacionado com as alterações dos produtos estabelecendo que toda alteração somente poderá ser feita se obedecer a um determinado procedimento. Com este tipo de controle permite-se identificar quem fez a alteração, quando, porquê, quem solicitou e em que versão foi implementada.

Em relação ao Controle de Qualidade, há a especificação dos critérios de qualidade na forma de padronização, aplicação de procedimentos de validação e verificação, e testes de aceitação, que deverão ser satisfeitos ao longo do desenvolvimento do software.

Já o Controle de Tarefas se refere às facilidades para controlar as atribuições das tarefas, para rastrear estas atribuições, registrar e relatar a finalização das tarefas, e para ativar tarefas pendentes.

2.4.4 - Gerenciamento dos Recursos

O Gerenciamento dos Recursos está relacionado com as atividades de planejamento e controle da parte financeira do projeto. Dentre as atividades de planejamento, tem-se: o desenvolvimento da estrutura de divisão de trabalho, a estimativa de custo, a distribuição de recursos e estabelecimento de um cronograma. Quanto às atividades de controle existem aquelas atividades relacionadas com o cronograma, com o consumo real e estimado de recursos, com as alterações de pessoal e alterações da estrutura organizacional.

Quanto ao suporte de desenvolvimento, o Gerenciamento dos Recursos está relacionado com a capacidade de fornecer relatórios sobre o produto, a equipe, a estrutura de divisão de trabalho e as revisões dos planejamentos.

2.4.5 - Gerenciamento dos Elementos de Suporte

Todo desenvolvimento de software inclui a utilização do hardware, do sistema operacional, dos compiladores, de ferramentas CASE, etc., que formam então o ambiente de desenvolvimento. O Gerenciamento dos Elementos de Suporte está relacionado exatamente com as alterações deste ambiente de desenvolvimento que ocorrem durante a

execução de um projeto, tanto a nível de hardware quanto de software.

Sabe-se que durante o tempo de vida de um projeto, especialmente de grandes projetos, é comum que o ambiente de desenvolvimento seja alterado, podendo ocorrer a introdução de novas ferramentas, atualização das versões das ferramentas utilizadas, instalação de novos computadores, atualização do banco de dados, do sistema operacional etc.

Neste caso este Gerenciamento é responsável pela configuração e pelo controle de todos os elementos do ambiente que dão suporte ao desenvolvimento de software.

2.5 - O Papel do Gerenciamento de Configuração de Software

Em função desta amplitude para gerenciar um projeto de software, o Gerenciamento de Configuração de Software (GCS) passa a ter um papel importante uma vez que fornece o controle e a visibilidade do desenvolvimento de software. Em termos práticos, o GCS está relacionado com as atividades das abordagens de gerenciamento descritas anteriormente exceto com o Gerenciamento de Recursos cujas atividades são controladas por ferramentas específicas (Rook, 1988).

Infelizmente, na prática, os métodos e ferramentas de GCS são vistos como desnecessários pelos desenvolvedores pois estes acham que um GCS somente impõe mais trabalho com muito pouco benefício às suas tarefas (Dart, 1992; Feiler e Smeaton, 1988; Bersoff et al., 1980).

Para sanar esta visão errônea, a tecnologia do GCS vem evoluindo no sentido de fornecer não somente a

função de gerenciamento mas oferecendo também a função de suporte ao desenvolvedor na medida que vem facilitando o uso de tais sistemas por parte dos desenvolvedores (Feiler, 1990).

Como função básica, o GCS fornece técnicas, métodos e procedimentos para manter a história dos elementos de software, identificando-os e localizando-os de forma única, e para iniciar, avaliar e controlar as alterações dos componentes ao longo do desenvolvimento. Desta forma o GCS aperfeiçoa a produtividade e a qualidade do software, retirando a confusão e a redundância de trabalho (Bersoff et al., 1980).

Em termos da função de suporte ao desenvolvedor, o GCS pode minimizar a quantidade de elementos a serem alterados e o tipo de reprocessamento das alterações, podendo gerenciar e integrar a área de trabalho dos desenvolvedores, e pode dar suporte à migração do software para outras plataformas (Dart, 1992).

2.5.1 - Função de Gerenciamento

O GCS na função de gerenciamento é essencial para manter a integridade do produto, fornecer controle das alterações e visibilidade ao projeto.

A integridade do produto requer que os elementos relevantes do software sejam devidamente identificados e colocados sob controle de configuração. Este controle de configuração significa que os elementos não podem ser alterados de forma aleatória a bel prazer do desenvolvedor. Qualquer alteração terá que passar por um processo de análise, aprovação, implantação e verificação. Desta forma os elementos de software deverão ter

identificações únicas, distinguindo-se entre versões, e armazenados de forma que possa ser controlado o acesso a eles.

Uma vez sob controle de configuração, as alterações nos elementos de software devem ser controladas e gerenciadas a fim de manter a integridade do software. Desta forma os pedidos de alteração devem ser identificados e classificados de acordo com a sua prioridade: não atendimento aos requisitos do cliente, erros de desenvolvimento, fora do padrão de qualidade, aperfeiçoamentos de performance, adaptação ao novo ambiente de desenvolvimento, alteração dos requisitos do cliente etc. (Dart, 1992).

Os pedidos de alteração são avaliados por um Comitê de Controle de Alteração que emitirá uma autorização para implementar a alteração. Como os pedidos de alteração são registrados e mantidos, há possibilidade de se fazer os relatórios para a alta gerência e para o cliente sobre o "status" do desenvolvimento do produto (Cunha e Nakanishi, 1993b; Dart, 1990).

2.5.2 - Função de Suporte ao Desenvolvedor

Esta função do GCS pode ser vista de três perspectivas diferentes de acordo com o ponto de vista do desenvolvedor.

O desenvolvedor pode ver o GCS como um gerenciador do software e de sua integridade, gerenciador de execução das alterações e como fornecedor de uma visibilidade dos elementos (Feiler, 1990).

Geralmente os elementos de configuração são armazenados em um repositório do GCS, que é logicamente separado da área de trabalho dos desenvolvedores e para recuperarem determinado elemento, têm que fazer uma solicitação ao repositório. O GCS pode dar suporte ao desenvolvedor neste tipo de tarefa fazendo uma integração entre o repositório e as áreas de trabalho, de modo que o acesso ao repositório seja transparente ao desenvolvedor.

O gerenciamento do processo de alteração também pode se tornar uma função de suporte ao desenvolvedor na medida que o controle sobre as alterações e as dependências entre elementos, fornecem as informações necessárias para que o desenvolvedor não perca o seu tempo procurando o conjunto de alterações já realizadas. A área de trabalho do desenvolvedor pode ser gerenciada e integrada pelo GCS de tal forma que a qualquer momento o desenvolvedor pode ter acesso ou ser informado sobre qualquer alteração que foi implementada em outros módulos de software desenvolvidos por outra equipe. Pode-se definir também que a área de trabalho seja atualizada em determinados períodos de tempo.

E a migração de alterações também pode ser uma atividade disponível no GCS para dar suporte ao desenvolvedor pois as alterações realizadas por um desenvolvedor serão integradas automaticamente com as alterações dos outros desenvolvedores, tornando-se disponíveis para integração e teste. Além disso as alterações migram para diferentes ramificações de uma família de sistema e a visibilidade da história das alterações será útil aos desenvolvedores já que haverá, para cada alteração, registro sobre quem, quando, e porquê foi feita a alteração.

Portanto o GCS pode exercer tanto a função de gerenciamento como também a função de suporte ao desenvolvedor, entretanto, isto requer que o GCS se torne parte integrante do ambiente de desenvolvimento de software.

CAPÍTULO 3

GERENCIAMENTO DE CONFIGURAÇÃO DE SOFTWARE

3.1 - Origem

Os primeiros princípios, procedimentos e métodos de gerenciamento e de controle do desenvolvimento de sistemas, desde a sua especificação até a sua produção final, foram aplicados à área militar dos Estados Unidos da América na fabricação de seus equipamentos. Dentre estes princípios, procedimentos e métodos se destacavam: a documentação de todas as informações do produto e do seu processo de fabricação; a utilização da identificação única dos componentes do sistema através de "part numbers"; e o controle do desenvolvimento através do mapeamento das alterações e da verificação da qualidade de cada um de seus componentes (Dart, 1992).

A medida que foi crescendo a complexidade dos sistemas a serem desenvolvidos, e, conseqüentemente, foi aumentando o número de equipes de trabalho envolvidas no projeto, inclusive com a necessidade de contratação de empresas especializadas externas à organização, houve a necessidade de se padronizar tais princípios, procedimentos e métodos de gerenciamento. Esta padronização permitia a definição clara da interface de trabalho entre as equipes de desenvolvimento e dos termos contratuais entre a organização e as empresas contratadas, estabelecendo as responsabilidades de cada parte.

Surgiu então a disciplina Gerenciamento de Configuração (GC) que, fundamentalmente, agia como uma função de gerenciamento propiciando o controle e a

visibilidade do projeto como um todo, aperfeiçoando a sua produtividade e o seu controle de qualidade e, principalmente, evitando a confusão e a redundância de trabalhos (Bersoff et al., 1980).

Desta forma, o GC aplicado ao desenvolvimento de hardware fornecia os procedimentos e métodos para construir o histórico do produto e de seus componentes; para criar a identificação única de cada componente, inclusive de cada versão; e para autorizar, avaliar e controlar as alterações do produto e de seus componentes ao longo do seu desenvolvimento e durante a sua manutenção.

Com o surgimento do computador e, principalmente, do software, elemento que veio aumentar ainda mais a complexidade dos sistemas, percebeu-se a necessidade de também se aplicar os princípios, procedimentos e métodos do GC à produção de software, resultando então na nova disciplina: o Gerenciamento de Configuração de Software (GCS).

Ao longo da sua evolução, o GCS foi consolidando os seus conceitos básicos e suas aplicações, de tal forma que adquiriu uma caracterização distinta do GC, disciplina da qual se originou. Das diferenças entre estas duas disciplinas, as três principais são descritas a seguir (Feiler, 1991; Rook, 1988; Bersoff et al., 1980).

Uma dessas diferenças se refere aos sistemas que serão gerenciados e às suas formas de produção. No GC os procedimentos e métodos de gerenciamento podem ou não estar disponíveis em computadores na forma de ferramentas CASE, e o sistema a ser desenvolvido junto com seus processos de produção estão relacionados com hardware, ou seja, equipamentos, máquinas ou instrumentos.

No GCS todos esses elementos, procedimentos e métodos de gerenciamento, o próprio sistema a ser desenvolvido e os seus processos de produção, estão disponíveis no computador na forma de softwares. Neste caso então, o ambiente de trabalho onde se localiza o GCS além de fornecer os meios para se manter as informações do sistema em desenvolvimento, do seu "status" e do seu histórico, fornece também os próprios meios onde serão feitos o armazenamento, a recuperação e a produção do sistema.

A segunda diferença entre o GC e o GCS se refere ao maior potencial deste último para automação e para a integração direta entre o sistema e seu desenvolvimento, já que se encontram no mesmo ambiente de trabalho. Desta forma o GCS, além de fornecer a função primária de gerenciamento e controle, pode fornecer também uma função de suporte ao desenvolvedor.

E a última diferença básica entre estas disciplinas, que faz com que o GCS seja mais difícil de ser implementado, se refere à diferença propriamente dita entre hardware e software. Geralmente o software é visto como um elemento que pode ser facilmente alterado, seja através da inserção de um comando, de uma linha de programação, de uma função ou procedimento. Esta percepção faz com que haja uma maior freqüência de alterações no software que está sendo desenvolvido. Desta forma, o GCS deverá implementar diferentes graus de controle, muitas vezes tendo que chegar ao nível de controlar o conteúdo dos programas, para atender às suas funções de gerenciamento e de suporte ao desenvolvedor. Esta facilidade de alterar software implica não somente em diferenças de procedimentos entre GC e GCS, mas diferença na implementação do GCS pois, neste caso, há a necessidade de se estabelecer uma nova disciplina de

trabalho entre os desenvolvedores de software, uma vez que estes sempre procuram fugir das regras de controle impostas pelo GCS.

Desta forma, apesar das dificuldades em se reconhecer a necessidade de se usar um GCS no desenvolvimento do software, esta disciplina vem evoluindo e ganhando seu espaço dentro da Engenharia de Software, principalmente, por causa da busca de software de qualidade.

3.2 - Evolução

Inicialmente, quando surgiram as primeiras preocupações com relação ao GCS, cada organização solucionava seus problemas individualmente. Algumas organizações não adotavam os conceitos do GCS pois, para elas, os problemas de gerenciamento de desenvolvimento de software não eram evidentes. Outras, que percebiam a necessidade de um GCS, não implementavam pois além de não existir na época nenhuma ferramenta disponível que oferecesse tais facilidades, o custo para que a própria organização desenvolvesse uma ferramenta era muito caro (Dart, 1992).

As primeiras soluções para o GCS foram implementadas através de ferramentas e ficaram restritas às facilidades de controle de versão ou de atualização automática (Dart, 1992; Tichy, 1985). Este era o caso do Source Code Control System (SCCS) (Rockind, 1975), Code Management System (CMS) (Christian e Zucker, 1986), Project Automatic Librarian (PAL) (Prager, 1983), que ofereciam o controle de versão de códigos fonte e do Programa Make (Feldman, 1979) que automatizava a atualização de arquivos.

Mas estas primeiras soluções para o GCS eram dependentes dos sistemas operacionais, e as organizações, para implementar uma ferramenta, teriam que adquirir aquela que correspondesse ao seu sistema operacional ou adquirir as ferramentas e os computadores onde elas seriam instaladas.

Além disso, junto com as poucas facilidades automatizadas que as ferramentas ofereciam, as organizações eram obrigadas ainda a desenvolver outras facilidades, tais como: definir formas de solicitação de alteração, controlar e rastrear tais solicitações. Geralmente, essas facilidades implementadas não eram integradas às outras ferramentas, nem eram automatizadas. Assim, os desenvolvedores teriam que executar manualmente atividades enfadonhas como preencher formulários de encaminhamento ou de autorização de alteração, enfrentar uma burocracia de controle (solicitar, aguardar aprovação, executar, encaminhar), ou consultar pastas de arquivo para buscar o rastreamento das suas alterações.

De qualquer forma esta primeira geração de ferramentas GCS fez com que a indústria e os pesquisadores adquirissem um bom entendimento sobre os princípios e procedimentos do GCS, levando-os à conclusão de que um simples controle de versão não era a solução para as necessidades do GCS.

Nestes últimos anos, a preocupação com os princípios e procedimentos do GCS cresceu de tal forma que, atualmente, quando se faz referência ao termo GCS, já não se fala em *disciplina GCS*, mas já se fala em termos mais abrangentes como *Tecnologia GCS* ou *Sistemas GCS* (Dart, 1990).

A indústria e os centros de pesquisa têm proporcionado uma evolução acentuada com a disponibilização de produtos voltados para esta tecnologia do GCS. A indústria ao incorporar as idéias dos pesquisadores, tem fornecido um laboratório prático de experimentação das próprias idéias dos pesquisadores fazendo com que haja um aperfeiçoamento de tais produtos.

Hoje em dia, a tecnologia do GCS está voltada para as facilidades de automatização dos procedimentos do GCS e o suporte para implementar tais facilidades, que pode ser através de um ambiente de desenvolvimento dedicado ou utilizando o próprio sistema operacional. Dentre as possibilidades disponíveis no mercado, podem ser encontrados os ambientes de desenvolvimento com facilidades de GCS, os sistemas de GCS e as ferramentas de GCS (Feiler, 1991). No caso de ferramentas elas podem ser fornecidas como uma ferramenta básica de GCS permitindo ao usuário manipular as primitivas da ferramenta para montar um GCS customizado às suas necessidades, ou podem ser fornecidas como um sistema GCS fechado, onde a política, o manuseio pelo usuário e os controles já estão preestabelecidos de acordo com os requisitos do usuário.

Como exemplos dos produtos disponíveis desta atual tecnologia GCS tem-se (Feiler, 1991): ambientes de desenvolvimento com capacidades GCS, tais como Rational, ISTAR e National Software Environment (NSE); ferramentas de GCS tais como IDE's Software Through Pictures e Procase's Smart System; e sistemas de GCS, tais como Aide-De-Camp (ADC) e Change Configuration Control (CCC).

Esta gama de possibilidades de implementação de um GCS, fez com que novos conceitos e

procedimentos fossem acrescentados àquelas atividades clássicas originalmente criadas com a disciplina GCS. Estas novas atividades são apresentadas no Item 3.3.2.

Em termos futuros, a preocupação geral está voltada para os seguintes assuntos (Dart, 1992): capacidade para chavear as facilidades do GCS de forma estática ou dinâmica para customizar o sistema GCS; interoperabilidade entre sistemas GCS; e GCS distribuído.

3.3 - Definição

Geralmente a definição de conceitos é uma questão pessoal onde cada autor possui a sua, ou escolhe a que ele ache melhor dentre aquelas que se encontram no seu campo de ação. Não existiria uma definição mais correta que a outra, existiria sim aquela mais ou menos concisa, com palavras mais apropriadas ou aquela mais aceita pela comunidade. De qualquer forma todas elas conteriam o núcleo principal da questão complementado com características relevantes de cada autor.

No caso do GCS o núcleo central da definição seria: GCS é uma disciplina para controlar a evolução de sistemas de software (Dart, 1992). E, para uma definição formal optou-se por aquela contida nos manuais do IEEE: "**GCS é o processo de identificar e definir componentes em um sistema, controlando a atualização e alteração através do seu ciclo de vida, registrando e relatando o "status" dos componentes, das solicitações de alteração, e verificando a completeza e correteza dos componentes do sistema.**[2]"

[2] "SCM is the process of identifying and defining components in a system, controlling the release and change throughout the live cycle, recording and reporting the status of components and change requests, and verifying the completeness and correctness of system components."(IEEE, 1987, p. 9)

3.3.1 - Aspectos Operacionais Clássicos

A definição acima reflete os quatro aspectos operacionais clássicos do GCS definidos por Bersoff et al. (1980): identificar, controlar, acompanhar[3] e examinar[4]. Resumidamente pode-se assim definir cada elemento:

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

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

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

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

---

[3] Em inglês: accounting
[4] Em inglês: audit

Esses elementos são essenciais para manter a integridade do software, para fornecer o gerenciamento e o controle das alterações e permitir a visibilidade do software durante o seu desenvolvimento.

Para manter a integridade do software durante seu desenvolvimento, há a necessidade de se identificar e colocar sob controle de configuração todos os elementos relevantes que compõem o software. Estes elementos são denominados por Elementos de Software (ES), ou Itens de Software ou, ainda, Itens de Configuração. Pode-se incluir entre eles: documentos de especificações, documentos de projeto, planos de verificação e validação, código fonte, planos de teste, relatórios de teste, manuais do usuário e de manutenção, cronogramas, documentos de procedimentos etc. O nível de granularidade desses elementos, ou seja, até que nível serão controlados, dependerá do tamanho e complexidade do software a ser desenvolvido.

Estes ES's, a partir do momento que ficam sob controle de configuração, passam a receber uma identificação única, inclusive cada versão, e começam a ser controlados quanto às suas alterações.

Para um melhor controle dos ES's ao longo do ciclo de vida do software, são estabelecidas as linhas de demarcação ao final de cada uma das fases. Desta forma em cada fase existirá uma coleção de ES's referenciado com uma única versão. Os ES's sob estas condições somente poderão ser alterados com uma solicitação formal. As linhas de demarcação representariam então os pontos de entrega de ES's para o prosseguimento do seu desenvolvimento, ou seja, os ES's, a medida que atendessem às especificações, seriam promovidos de uma fase para outra.

Além do controle dos ES's, as suas alterações também deverão ser gerenciadas e controladas a fim de manter a integridade do produto. Desta forma as solicitações de alteração devem ser identificadas e classificadas. Há basicamente três categorias de alterações (Dart, 1990):

- Alterações corretivas: são aquelas realizadas por discrepância com as especificações, implementação incorreta ou desrespeito aos padrões.

- Alterações adaptativas: são aquelas realizadas pela introdução de novos requisitos do cliente, acréscimo ou troca do hardware.

- Alterações perfectivas: são aquelas que visam o aperfeiçoamento do software, principalmente da sua performance.

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

A execução das alterações também deverá ser de forma controlada. O primeiro aspecto deste controle é impor que a alteração real implementada seja realmente a

alteração solicitada, pois muitas vezes aproveita-se uma autorização de alteração para se implementar alterações não autorizadas. O segundo aspecto do controle de alteração é a coordenação de atualização para que todos os ES's dependentes dos ES's alterados, sejam atualizados. Portanto, o "status" das solicitações de alteração é registrado e mantido, relacionando às versões que refletem a alteração, para que se possa dispor tais informações ao gerenciamento e aos clientes.

Estes dois aspectos, relatar "status" dos ES's e de suas alterações, e revisar a correteza e a consistência do software e do seu desenvolvimento, possibilita estabelecer o "status" de um projeto nas várias fases do ciclo de vida do projeto.

3.3.2 - Aspectos Operacionais Complementares

Atualmente a tecnologia do GCS apresenta alguns aspectos operacionais que foram acrescentados àqueles clássicos e que neste trabalho está se denominando de aspectos operacionais complementares (Brown et al., 1991; Dart, 1990). Resumidamente, tais elementos podem ser assim definidos:

- Produção: este aspecto operacional está relacionado com o gerenciamento da produção ou da construção do software de forma ótima; com a facilidade de se obter um instantâneo ou um congelamento do "status" do produto a qualquer hora; com os mecanismos para otimizar esforços ao se construir sistemas, reduzindo a necessidade da recompilação de componentes que não foram alterados nem que foram afetados por alterações em outros componentes; com

a facilidade para se fazer a análise do impacto de uma alteração de tal forma que forneça todas as ramificações possíveis ao se fazer determinada alteração; e com a facilidade de regerar qualquer fase ou parte do produto a partir de qualquer fase do desenvolvimento.

- Processo: este aspecto tem relação direta com a facilidade de fornecer suporte para a política e para o modelo do ciclo de vida adotado pela organização; com a facilidade de identificar tarefas a serem realizadas e como e quando elas serão completadas; com a facilidade para divulgar informações para pessoas certas sobre eventos relevantes; e com a facilidade para documentar o conhecimento sobre o produto.

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

3.4 - Usuários

Cada um dos participantes da equipe de desenvolvimento de software possui as suas tarefas e, principalmente, as suas responsabilidades, desempenhando uma função, ou um papel, dentro do projeto. De acordo com esta função, as pessoas irão interagir com o GCS, fornecendo ou buscando informações, de formas distintas. Assim, surgem os diversos tipos de usuários do GCS que têm diferentes privilégios, restrições de acesso, facilidades

etc. Deve-se observar que o termo usuário não se refere à pessoa em si, mas ao papel ou função desempenhada pela mesma, podendo-se ter uma mesma pessoa desempenhando, ou fazendo o papel de dois ou mais tipos de usuários dentro do projeto (Dart, 1990).

Geralmente têm-se os seguintes tipos de usuários:

- o gerente de projeto que é o responsável pelo desenvolvimento do software como um todo;
- o gerente de configuração que é o responsável pela aplicação das técnicas e procedimentos do GCS;
- os analistas e programadores que são os responsáveis pelo desenvolvimento e manutenção do software;
- os aplicadores de testes que validam a correteza do software;
- o gerente do controle de qualidade que assegura a qualidade do software;
- o cliente que usa o software.

O gerente de projeto, para assegurar que o software será desenvolvido atendendo a todos os requisitos do cliente, deverá monitorar o progresso do desenvolvimento detectando e resolvendo os problemas que surgirem. Essas informações poderão ser fornecidas pelo GCS através, por exemplo, dos relatórios sobre o "status" do projeto e das revisões realizadas.

O gerente de configuração deverá assegurar que todas as técnicas e os procedimentos do GCS estejam definidos e implementados de forma que o GCS realmente cumpra com seu papel de controlar o desenvolvimento de software. Este gerente deverá disseminar tais mecanismos

entre todos os usuários e verificar se estão sendo seguidos.

Quanto aos analistas e programadores, o objetivo principal deles é trabalhar efetivamente na construção do software. Muitas vezes executam seus trabalhos (criar, alterar, testar e integrar) de forma independente, cada qual na sua área de trabalho ou no seu equipamento. Mas como o projeto é único, deverão se comunicar e se coordenar entre si para troca de informações a respeito da criação de novos ES's ou mesmo das suas alterações. É através do GCS que eles poderão obter e fornecer tais informações.

O objetivo dos aplicadores de testes é conferir se os ES's estão de acordo com as especificações. Neste caso também deverá ser feito o controle e o registro dos testes e de seus resultados, de forma a conferir se um determinado ES foi testado, quais testes foram aplicados e a qual versão. Todas estas informações poderão estar disponíveis no GCS.

Quanto ao gerente do controle de qualidade, cujo objetivo é assegurar a qualidade do software, deverá estabelecer técnicas e procedimentos a serem respeitados quanto à qualidade do produto, verificando se todas as especificações do cliente estão sendo atendidas. O GCS é que poderá conter todas estas informações.

O cliente, cujo objetivo é obter o software de acordo com as suas especificações, poderá obter informações sobre o andamento do projeto, testar uma versão do software ou de uma parte dele e solicitar modificações às partes já desenvolvidas, seja por critério de alteração de especificação ou por não atendimento às especificações já definidas. Para que a atuação do cliente no

desenvolvimento do software seja feita de forma controlada, o cliente também poderá ser um usuário do GCS.

Como cada um destes diferentes tipos de usuários tem uma função específica a desempenhar e uma visão diferente do GCS, haverá requisitos diferentes a serem exigidos para o GCS a ser adotado.

3.5 - Requisitos de Funcionalidade

Em função da diversidade de usuários de um GCS, onde cada usuário tem a sua visão do GCS, pode-se estabelecer os requisitos que um GCS deverá atender em relação à necessidade operacional de cada um destes usuários (Dart, 1990).

A seguir apresentar-se-ão alguns dos requisitos básicos classificados de acordo com os aspectos operacionais do GCS.

Em relação aos Elementos de Software (ES), os usuários necessitam de funções que lhes permitam: registrar versões dos ES; identificar um grupo de ES que forma uma configuração, inclusive com as respectivas versões de cada ES; mostrar as linhas de demarcação do ciclo de vida do desenvolvimento para os ES; identificar o contexto do projeto como uma coleção de ES; e, além disso, fornecer meios de manipular repositórios ou bibliotecas onde serão armazenados os ES e as informações do GCS. Deverá ser permitido trabalhar com todos os tipos de ES, desde o código fonte, objeto ou executável, até diagramas e documentos de projeto.

Em relação à Estrutura de Relacionamentos, há a necessidade de funções que permitam aos usuários:

modelar a estrutura do produto através de um modelo de sistema contendo todos os ES daquele produto; especificar as interfaces entre ES, ou entre versões e configurações, procurando torná-los reusáveis; identificar e manter os relacionamentos entre ES; e selecionar ES compatíveis para se obter uma versão válida e consistente do produto.

Quanto às facilidades de construção, os usuários necessitam de funções para: facilitar a construção do produto; habilidade para, a qualquer momento, tirar um "flash" ou congelar o desenvolvimento do produto; otimizar os esforços na construção de sistemas diminuindo-se a necessidade de recompilar os ES; permitir a análise do impacto das alterações previstas, inclusive em todas as ramificações de versões; e facilidade para regerar qualquer fase ou parte do produto a qualquer momento.

Em termos de auditoria, as funções devem permitir aos usuários: registrar a história das alterações; rastrear os ES e suas evoluções; gerar arquivos "log" contendo todos os detalhes das operações executadas.

Quanto ao aspecto operacional acompanhar, deve-se ter as funções que permita aos usuários: instalar o acesso controlado aos ES do produto para não permitir alteração não autorizada; detectar e tratar conflitos de alterações; dar suporte "on-line" à geração de formulários de solicitação de alteração e de relatórios de problemas; rastrear os "bugs" identificando como, quando, e por quem eles foram tratados; acompanhar a propagação das alterações, de forma controlada; meios de particionar o produto para limitar os efeitos das alterações.

Em relação ao processo as funções devem permitir: dar suporte ao modelo de ciclo de vida e à política de trabalho da organização; dar facilidades para

identificar as tarefas a serem realizadas e também como e quando elas serão completadas; dar facilidades para transmitir informações para as pessoas envolvidas sobre os eventos relevantes; facilitar a documentação gerada durante o desenvolvimento.

Quanto ao aspecto de equipe, aos usuários deve ser permitido: definir o espaço de trabalho individual ou de grupos de trabalho; facilitar a resolução de conflitos quando mesclando versões; dar suporte à criação, controle e manutenção de um família de produtos.

Portanto, um GCS deveria idealmente atender a todos os requisitos dos usuários citados acima, mas, atualmente, não existe nenhuma ferramenta ou ambiente de desenvolvimento que forneça todas estas funcionalidades citadas.

3.6 - Dificuldades de Implementação

A diversidade de sistemas de GCS disponíveis atualmente, cada qual implementando uma funcionalidade diferente, faz com que a implementação de tal sistema não seja uma tarefa simples.

Há de se considerar vários aspectos, desde o preço até o porte do sistema, antes de se escolher qual sistema deverá ser implementado (Bounds e Dart, 1993).

Em primeiro lugar há de se definir que um sistema completo de GCS, idealmente falando, é aquele que implementa todas as características apresentadas no item anterior. Em termos práticos sabe-se que nenhum sistema possui todas estas característica. Para generalizar o termo sistema GCS, qualquer sistema que forneça alguma forma de

controle de versão, identificação de configuração, uma estrutura ou modelagem do sistema a ser desenvolvido, que tenha por objetivo realizar as funções de um GCS em algum nível, será considerado como sendo um sistema de GCS. Existem atualmente cerca de 40 sistemas de GCS disponíveis no mercado (Dart, 1990).

Na discussão a seguir serão apresentados somente alguns temas pois não é objetivo esgotar o assunto neste trabalho. O objetivo é apenas mostrar o nível de dificuldade que se tem para implementar um sistema GCS.

O porte do sistema GCS é um aspecto que deve ser considerado ao se escolher a forma de se trabalhar ou de acordo com as ferramentas que se tem à disposição. Nestes termos pode-se escolher um sistema de GCS que faça parte de um ambiente de desenvolvimento onde as funções do GCS são partes integrantes deste ambiente. Um exemplo deste tipo é o ambiente Relational (Feiler, 1990). Outra escolha se refere a um GCS separado do ambiente de desenvolvimento sendo que as funções são fornecidas através de uma ferramenta separada que coexiste com outras ferramentas. Um exemplo deste tipo é o Revision Control System (RCS) (Tichy, 1985).

Quanto ao nível de integração da ferramenta GCS ao ambiente de desenvolvimento, pode-se ter a integração a nível de processo, onde o padrão de uso da ferramenta GCS será totalmente integrado ao padrão de uso do ambiente; pode-se ter a integração da ferramenta a nível de sistema de forma que o sistema GCS coexista com outras ferramentas do ambiente; ou pode-se ter ainda a integração de banco de dados, onde a ferramenta GCS possui o seu próprio banco de dados ou faz uso das informações contidas em um banco de dados separado.

Sabe-se que teoricamente o GCS é aplicável por toda a vida do produto desde a criação até a manutenção, mas nem todos os sistemas GCS atuais oferecem facilidades para que o sistema de GCS seja aplicado desde o início do projeto com todas as vantagens.

A definição da fase a partir da qual será aplicado o sistema de GCS também pode influir na implementação do mesmo. Pode-se escolher entre fazer o controle a partir do final do ciclo de desenvolvimento quando o produto já está pronto para ser entregue ao cliente e o controle será feito somente para a etapa de manutenção, ou, por outro lado, aplicar o GCS desde o início do projeto até a fase de manutenção. Cada uma destas escolhas têm suas vantagens e desvantagens.

Em função de quão enfadonha é a tarefa de realizar uma alteração, ou seja, se há um número muito grande de procedimentos manuais, tais como preencher formulários de solicitação de alteração, esperar pela resposta do comitê, receber a aprovação etc., pode-se deixar para o mais tarde possível o início da aplicação do GCS, fazendo com que cada membro da equipe de desenvolvimento se responsabilize pelo controle da sua própria produção (Bounds e Dart, 1993).

Se, pelo contrário, a maioria dos procedimentos podem ser realizados de forma "on-line" e são muitos rápidos de serem realizados, então o GCS pode ser aplicado desde o início do ciclo de vida do desenvolvimento.

Há vários graus de controle sobre os usuários quanto à evolução do produto. Pode-se requerer uma solicitação formal e escrita de uma alteração; esta

solicitação passa por um comitê de análise que vai aprovar ou não a alteração; e caso seja aprovada, o responsável recupera do repositório o componente que deve ser modificado e faz a alteração autorizada; após esta alteração, o componente alterado é enviado novamente para ser analisado, onde será verificado se foi feita a alteração conforme autorizado; e, finalmente o componente é aprovado e colocado no repositório novamente.

Este processo pode ser feito com uma automatização completa deste processo, sendo tudo feito de forma automática, e ao final somente ter-se-ia um arquivo "log" para checar se todos os procedimentos foram executados corretamente.

Na primeira forma de se trabalhar, fazer uma alteração é desencorajador, enquanto na segunda forma há um aumento de produtividade.

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

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

Outro cenário de discussão está relacionado com a quantidade de automação que o GCS deverá fornecer. É

possível realizar GCS sem qualquer tipo de facilidades "on-line", mas, hoje em dia isto é ineficiente. O objetivo é automatizar tanto quanto possível as partes não-criativas do GCS. Por exemplo, formulários de solicitação de alteração por escrito, protocolo para a resposta, podem ser implementados através de serviços "on-line", onde o GCS fornece os formulários automaticamente. Os sistemas GCS automatizam uma certa quantidade de funções e os usuários deverão implementar outros procedimentos de forma manual a fim de atender aos requisitos de se ter todas as funcionalidades de um GCS completo (Feiler, 1991).

3.7 - Ambientes Integrados versus Ferramentas Integradas

Em termos de evolução das tecnologias de desenvolvimento de software, incluindo o caso particular do desenvolvimento da tecnologia do GCS, há dois paradigmas de integração em voga atualmente (Brown et al., 1992). Existe o paradigma do Ambiente Integrado de Suporte ao Projeto (Integrated Project Support Environment - IPSE) e o paradigma da Integração de Ferramentas CASE (Integrated CASE - ICASE).

Estes dois paradigmas de integração surgiram de tecnologias independentes que evoluíram de forma paralela ao longo dos anos 80. A Figura 3.1 apresenta estas duas linhas de desenvolvimento.

O paradigma IPSE surgiu da pesquisa em torno dos ambientes de desenvolvimento e sua principal característica é o suporte que dá aos modelos de arquitetura e modelo conceitual do software a ser desenvolvido, fornecendo uma integração de ferramentas e um repositório particular de serviços.

Já o paradigma ICASE é o resultado do desenvolvimento da tecnologia das ferramentas CASE, e a sua característica principal é que o ambiente do software é um conjunto de ferramentas que estão fortemente acopladas, mas não representa um modelo único de arquitetura ou modelo conceitual. Na verdade este paradigma surgiu da aliança estratégica entre os fabricantes de ferramentas CASE.

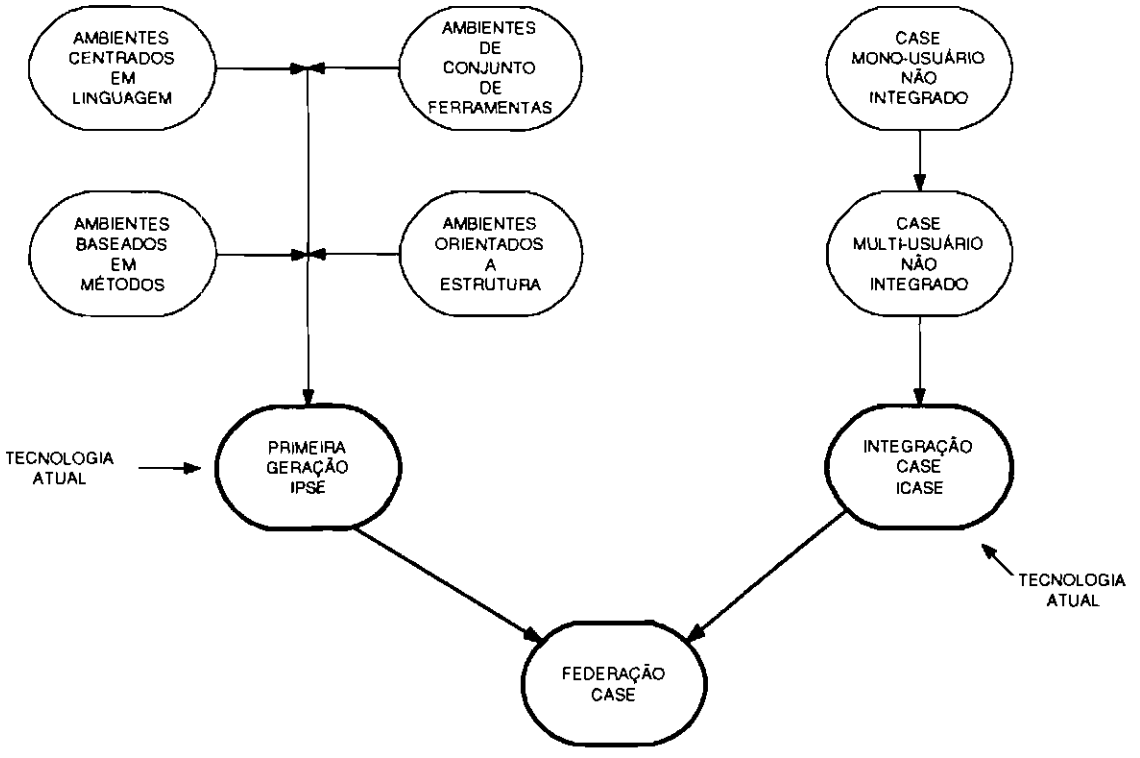

Fig. 3.1 - Evolução da Tecnologia de Integração
FONTE: Brown et al. (1992), p. 36.

Muitos desenvolvedores confundem o IPSE com os tradicionais ambientes ICASE. As diferenças básicas são:

- Os IPSEs são ambientes abertos e extensíveis e são feitos para dar suporte a vários métodos, enquanto os ambientes ICASE trabalham com um método simples e não são extensíveis.

- OS IPSEs têm como objetivo permitir que as equipes de desenvolvimento trabalhem juntos em vários projetos, enquanto os ambientes ICASE são projetados originalmente para as equipes trabalharem em um projeto único.

- Os IPSEs têm evoluído a partir do desenvolvimento de aplicações científicas e de engenharia, enquanto os ambientes ICASE são mais comuns para desenvolvimento de sistemas de informação.

O paradigma IPSE está limitado pela própria complexidade da sua tecnologia e pela falta de fabricantes interessados no seu desenvolvimento e comercialização. Mas são válidas as suas premissas básicas da necessidade de unificar os princípios de arquitetura para gerenciar o desenvolvimento de grandes sistemas de software e para a evolução dos ambientes de desenvolvimento.

No caso do paradigma ICASE, este não fornece o princípio da unificação do modelo de arquitetura, mas, na verdade promove uma integração limitada das várias facilidades, funcionais e comerciais, das ferramentas CASE.

Atualmente existe o processo de união destas duas linhas de desenvolvimento na forma denominada *Federação CASE*, que irá unir as vantagens e descartar as desvantagens destes dois paradigmas de integração (Brown, 1992).

3.8 - Paradigmas do Sistema GCS

Ao se examinar os vários sistemas GCS disponíveis no mercado, tem-se observado certos conceitos padrões no uso do repositório. Estes padrões podem ser descritos como paradigmas de repositório do GCS (Brown et al., 1991): o paradigma checkout/checkin, o paradigma da composição, o paradigma da transação longa e o paradigma do conjunto de alteração.

Estes quatro paradigmas caracterizam as capacidades individuais dos sistemas GCS e cada qual suporta certos aspectos do processo de GCS. O paradigma checkout/checkin é o mais conhecido e é o que fornece a versão individual de cada componente e controla a concorrência através de um chaveamento, não permitindo que um componente seja acessado por mais de um usuário.

Os outros três paradigmas refletem o avanço das funcionalidades de GCS sobre este paradigma básico. O paradigma da composição enfoca sobre a criação de configurações em um processo de dois passos - a descrição da estrutura de composição de um sistema em termos de seus componentes, e a seleção de uma versão apropriada de cada um destes componentes. O paradigma da transação longa enfoca sobre a evolução de um sistema através do controle de uma série de versões de configurações. O paradigma do conjunto de alteração enfoca sobre a evolução de um sistema em termos de alterações lógicas.

3.8.1 - Paradigma Checkout/Checkin

O paradigma checkout/checkin incorpora a facilidade do repositório para criar versões de componentes, incluindo a ramificação e a mesclagem, e o controle concorrente através do chaveamento de seqüências de versões. A Figura 3.2 apresenta o esquema de funcionamento deste paradigma em relação ao usuário.

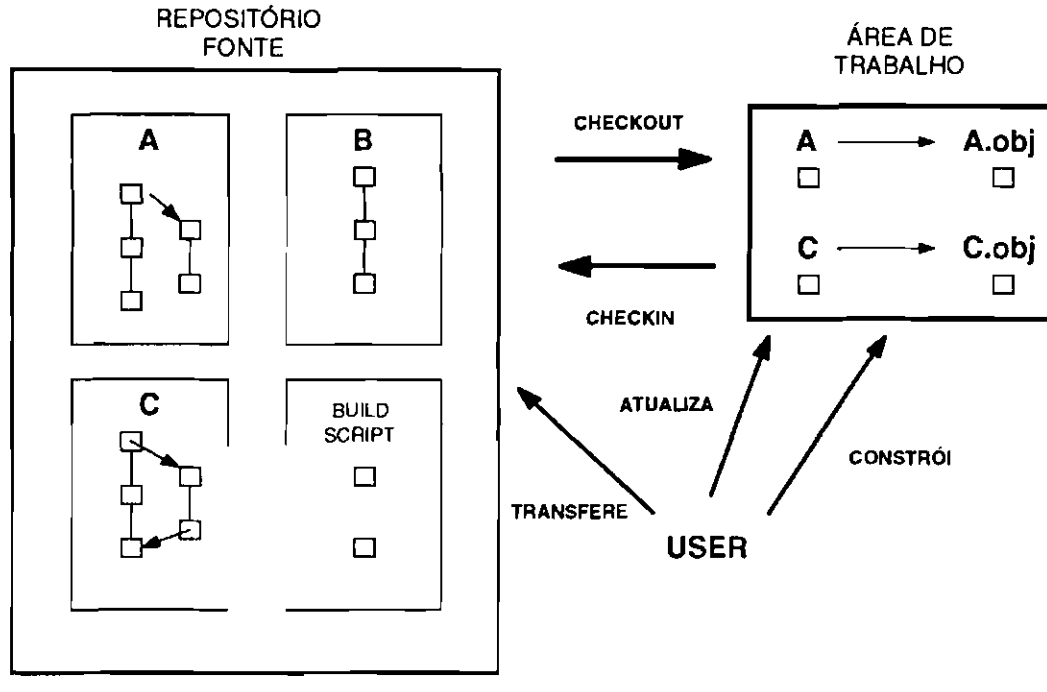

Fig. 3.2 - Modelo Checkout/Checkin
FONTE: Brown et al.(1991), p. 22.

Conforme pode ser verificado na figura, o usuário trabalha com o repositório fonte e com o sistema de arquivo que é a sua área de trabalho. Os arquivos contêm várias versões que são armazenadas no repositório e este faz o controle da criação de novas versões. Os arquivos, entretanto, não são diretamente acessíveis no repositório, um usuário têm de recuperá-los fazendo-se um "check out" de

uma versão do arquivo. Uma cópia desta versão do arquivo será transferida para a área de trabalho do usuário.

Uma vez recuperados, os arquivos podem ser utilizados para leitura ou para modificação, sendo que neste último caso um mecanismo de chaveamento no repositório garante que nenhum outro usuário terá acesso ao arquivo para modificá-lo também, ou seja, não podem ser feitas alterações no mesmo arquivo ao mesmo tempo.

Os arquivos modificados serão armazenados de volta no repositório quando o usuário fizer um "check in" no arquivo, quando então haverá uma nova versão do arquivo no repositório.

O sistema GCS pode ter controle de acesso ao repositório de tal forma que pode ser definido se um determinado usuário tem ou não permissão para fazer um "check out" de um arquivo para modificação.

O paradigma checkout/checkin fornece as seguintes primitivas conceituais para versão de arquivos: evolução de um história seqüencial de versão (referenciada como revisões); criação de ramos de versões, isto é, seqüências de versões que têm como ponto inicial uma determinada versão em um ramo existente mas que evolui independentemente; e mesclagem de duas versões de diferentes ramos em uma nova versão de um dos ramos.

3.8.2 - Paradigma da Composição

O paradigma da composição trata do aperfeiçoamento para criar configurações, para gerenciar sua história, e para usá-las dentro do contexto de trabalho de um sistema GCS.

Uma configuração neste paradigma consiste de um modelo do sistema e de regras de seleção de versão. Um modelo de sistema lista todos os componentes que formam um sistema e as regras de seleção de versão indicam quais versões devem ser escolhidas para cada componente, de forma a montar uma configuração. As regras de seleção são aplicadas para um modelo de sistema, selecionando uma versão de cada componente.

Os desenvolvedores definem o sistema em termos de seus componentes e, em um passo separado, selecionam uma versão apropriada para cada componente.

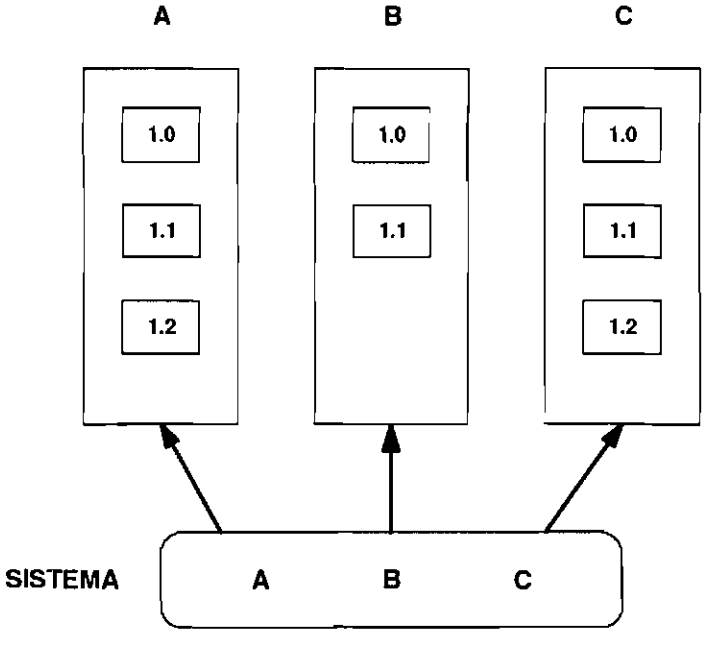

Fig. 3.3 - Seleção de Versão do Componente
FONTE: Brown et al.(1991), p. 24.

Este modo de operação para tratar com versões é apresentado na Figura 3.3. Na figura apresentada os componentes são mostrados como caixas quadradas enquanto

as configurações são mostradas como caixas com cantos arredondados.

3.8.3 - Paradigma da Transação Longa

O paradigma da transação longa trabalha com a evolução de sistemas de software como uma série de alterações das configurações e dá suporte à equipe de desenvolvimento com a coordenação de alterações concorrentes.

Os desenvolvedores operam primariamente com as versões das configurações. Os desenvolvedores primeiro selecionam a versão da configuração de sistema, e depois trabalham sobre a estrutura do sistema. A versão dos componentes é implicitamente determinado pela configuração, ao contrário do paradigma da composição, onde o desenvolvedor está primeiramente preocupado com a estrutura do sistema e então faz a seleção das versões dos componentes.

Uma transação longa tem dois elementos: um espaço de trabalho, e um esquema de controle de concorrência. O espaço de trabalho representa o contexto onde o desenvolvedor trabalha e fornece a memória local onde serão armazenados os programas, arquivos, que somente são acessados dentro da área de trabalho. O sistema GCS se incumbe de transferir o conteúdo das áreas de trabalho para o repositório. A área de trabalho substitui o tradicional de sistema de arquivo. Isto permite ao sistema GCS não somente gerenciar o repositório como também dar suporte aos desenvolvedores enquanto estão alterando o sistema na sua área de trabalho. Desta forma têm-se o acesso transparente ao repositório eliminando a tradicional transferência de componentes entre repositório e sistema de arquivos.

Como a área de trabalho de um desenvolvedor é isolada, as alterações de outras áreas de trabalho e do repositório não interferem na sua área de trabalho. A importação/exportação das alterações entre as áreas de trabalho e repositório são realizadas sob controle do desenvolvedor. O desenvolvedor é quem decide se a sua área de trabalho deve ou não ser atualizada, e se for, cria-se então uma nova configuração. Se ao criar esta nova configuração, houver conflitos com as alterações de outras áreas de trabalho, pode-se recuperar a versão original da configuração até estes conflitos serem resolvidos.

Uma alteração de configuração é realizada em uma transação, que pode durar horas, dias ou meses, e as modificações dos componentes não estão disponíveis até o término da transação. As transações múltiplas são coordenadas através dos esquemas de controle de concorrência a fim de garantir que nenhuma alteração seja perdida.

Uma configuração particular é escolhida como ponto inicial para as alterações e o resultado de uma transação é uma nova versão de configuração. O resultado de uma série de alterações seqüenciais é uma seqüência de versões de configurações, referenciadas como caminho de desenvolvimento. As versões de configurações podem ramificar de um caminho de desenvolvimento existente para um novo caminho independente. Esta interpretação do grafo de versão de uma configuração é mostrado na Figura 3.4.

Nesta figura tem-se o caminho de desenvolvimento onde cada caixa com cantos arredondados representa uma configuração e as quadrados de cor preta representam os componentes da configuração. Pode-se ter uma derivação deste caminho de desenvolvimento representado na

figura por uma seta, e então ter-se-á uma nova configuração de componentes. A figura também apresenta a fase de transação durante a qual estão sendo realizadas as alterações nos componentes da configuração.

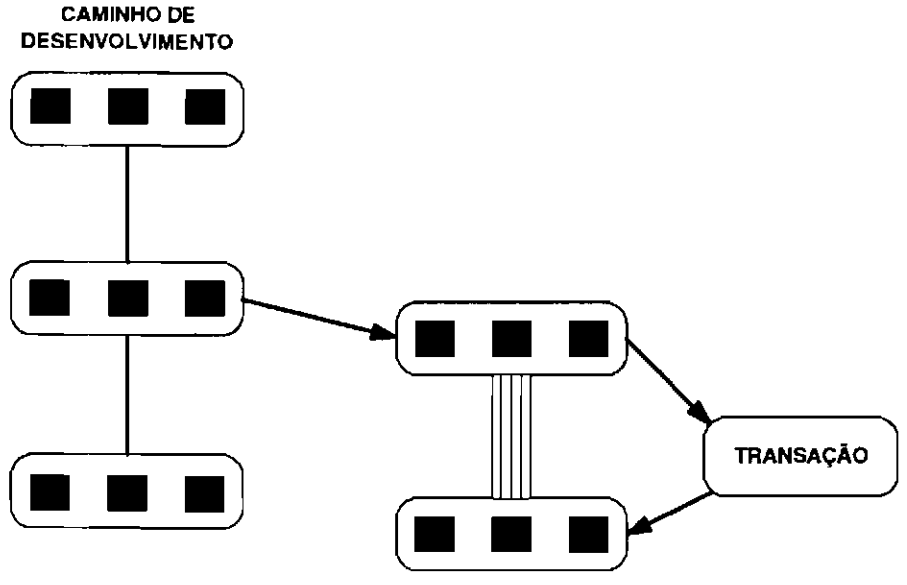

Fig. 3.4 - Histórico de Versão de uma Configuração
FONTE: Brown et al. (1991), p. 26.

3.8.4 - Paradigma do Conjunto de Alteração

O paradigma do conjunto de alteração fornece o suporte ao gerenciamento de alterações lógicas para configurações de sistema. O conceito de conjunto de alteração representa o conjunto de modificações dos diferentes componentes e que produzem uma alteração lógica. Os usuários de um sistema deste tipo podem operar diretamente no conjunto de alterações.

Este paradigma é orientado a alteração e trabalha de forma natural com as alterações lógicas, diferentemente da visão orientada a versão contida nos

outros três paradigmas, que trabalham com versões de componentes e configurações.

Os conjuntos de alterações permitem ver as configurações em termos de coleções de alterações lógicas. A Figura 3.5 mostra como uma versão de uma configuração é montada a partir de uma determinada versão base mais certas alterações. As alterações lógicas são propagadas ao se adicionar um conjunto de alterações para as configurações apropriadas. As novas configurações são criadas ao se fazer uma combinação de conjuntos de alterações em uma configuração básica.

Fig. 3.5 - Conjunto de Alterações em uso
FONTE: Brown et al.(1991), p. 28.

As restrições deste paradigma surgem quando um conjunto de alteração é dependente da presença de outros conjuntos de alteração, ou quando há conflitos entre conjuntos. Desta forma, as alterações quando são propagadas para outras configurações, devem, muitas vezes, incluir o respectivo conjunto de alteração.

Embora os conjuntos de alteração estejam relacionados a transações, eles não fornecem controle de concorrência. Há necessidade, portanto, de se complementar

este sistema com o paradigma checkout/checkin para controlar a concorrência das alterações dos componentes.

3.8.5 - Os Paradigmas e os Sistemas GCS

A Tabela 3.1 apresenta um resumo de alguns sistemas GCS e os respectivos paradigmas que são implementados, procurando mostrar que todos estes paradigmas estão presentes nos sistemas comerciais.

TABELA 3.1 - OS PARADIGMAS E OS SISTEMAS GCS

SISTEMA GCS	PARADIGMAS
Unix SCCS	checkout/checkin (co/ci)
Unix RCS	co/ci
SMDS Aide-De-Camp	conjunto de alteração + co/ci
Softool CCC	transação + co/ci
CaseWare Amplify	composição + co/ci
Apollo DSEE	composição + co/ci
Sun NSE	transação
Relational Environment	transação + co/ci
Procase Smartsystem	transação
IDE Software Through Pictures	composição + co/ci

FONTE: Brown et al. (1991), p. 29,30.

3.9 - Abordagens de Configuração

Há basicamente duas abordagens de configuração que podem ser aplicadas ao desenvolvimento de software conforme Kramer e Magee (1985): a configuração estática e a configuração dinâmica.

A abordagem da configuração estática para a construção de um sistema a partir de uma especificação de configuração é ilustrada na Figura 3.6.

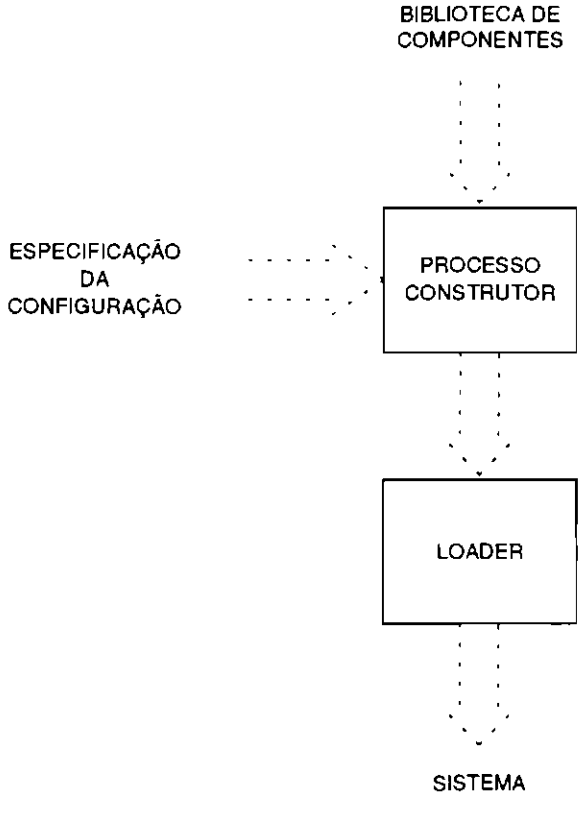

Fig. 3.6 - Processo de Configuração Estática
FONTE: Kramer e Magee (1985), p. 425.

O Processo Construtor é coordenado pela especificação de configuração para que ele possa buscar os componentes especificados na Biblioteca de Componentes, entregando-os ao "Loader" para montar o sistema. Este esquema representa então a abordagem de configuração estática, pois todos os componentes do sistema são configurados ao mesmo tempo, antes de entrarem em operação.

Para o caso de alteração, o sistema deverá parar a sua operação, ser reconstruído de acordo com a nova configuração e depois é que o sistema entraria em operação novamente.

Esta abordagem é equivalente ao processo editar/compilar/"link" usado na programação seqüencial. Esta abordagem não oferece possibilidade de se fazer alterações incrementais à configuração do sistema e conseqüentemente não é possível alterar o sistema durante a sua execução.

A abordagem de configuração dinâmica já permite que as alterações incrementais sejam realizadas sem reconstruir o sistema inteiro. Estas alterações incrementais ao sistema seriam feitas sem parar as partes não afetadas do sistema. Este processo dinâmico pode ser ilustrado pela Figura 3.7.

A Especificação de Configuração descreve a estrutura lógica e física do sistema e as alterações do sistema são submetidas na forma de especificações de alteração, como por exemplo, introduzir novos componentes, modificar componentes, fornecer interconexões diferentes etc.

As alterações deverão ser validadas para assegurar que as mesmas sejam compatíveis com o sistema existente. Como resultado tem-se uma nova especificação.

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

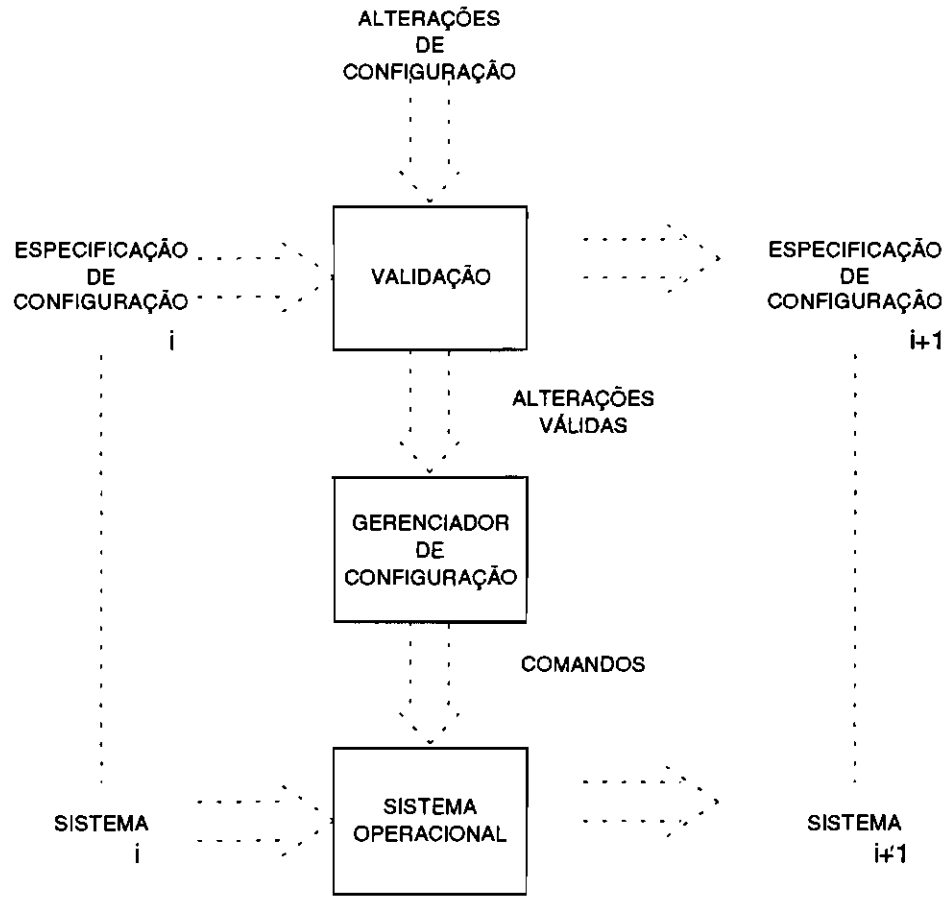

Fig. 3.7 - Processo de Configuração Dinâmica
FONTE: Kramer e Magee (1985), p. 425.

Para implementar uma configuração dinâmica há necessidade de que tanto as linguagens de programação quanto os sistemas operacionais, tenham determinadas características; e que exista uma linguagem própria de configuração. Dentre estas características citadas por Kramer e Magee (1985), podem ser destacar as seguintes:

- Linguagem de programação: tenham independência de contexto, independência de interconexão e transparência de comunicação. Exemplos deste tipo

de linguagens são: PRONET (Leblanc e Maccabe, 1982) e CONIC (Kramer et al., 1983).

- Linguagem de Configuração: tenha definição de contexto, instanciação, interconexão, especificação de alteração, separação de função, modularidade de especificação. Alguns exemplos deste tipo de linguagem são: Mesa (Mitchell et al., 1979), NETSLA (Leblanc e Maccabe, 1982) e PCL (Liskov e Sheifler, 1983).

- Sistema Operacional: tenha gerenciamento de módulo, gerenciamento de conexão, suporte de comunicação. Em termos de sistemas operacionais tem os seguintes exemplos: Medusa (Ousterhout et al., 1980), Amoeba (Tanenbaum e Mullender, 1981) e Accent (Rashid e Robertson, 1981).

CAPÍTULO 4

O PARADIGMA ORIENTADO A OBJETO

4.1 - Considerações Iniciais

A perspectiva de dar uma maior estabilidade ao desenvolvimento de software, que passou a ser mais complexo e muito mais volátil, fez com que a abordagem orientada a objeto fosse ganhando seu espaço dentro da Engenharia de Software principalmente por causa da demanda por software de qualidade.

A abordagem orientada a objeto oferece um modo mais natural de se trabalhar com os sistemas que têm maior interatividade com os usuários, mesmo porque são os sistemas "on-line" e interativos que predominam hoje em dia e que dão uma maior atenção à interface com o usuário em comparação com os antigos sistemas de processamento (Coad e Yourdon, 1992; Jacobson et al., 1992; Nerson, 1992; Booch, 1991).

Atualmente, o paradigma orientado a objeto está rapidamente ganhando aceitação na indústria do software. A maioria das organizações de desenvolvimento de software estão em processo de observação e/ou fazendo uma transição para o paradigma orientado a objeto (Lejter et al., 1992).

O que se tem verificado, entretanto, é que as características vantajosas deste novo paradigma introduziu também um novo conjunto de problemas com relação a teste, configuração e manutenção de sistemas (Wilde e Huitt, 1992).

Antes de se apresentar estes novos problemas e também para um melhor entendimento dos mesmos, serão apresentados, resumidamente, alguns conceitos básicos do paradigma orientado a objeto.

4.2 - Conceitos Básicos

O objeto é o primeiro e mais importante conceito na construção de um sistema orientado a objeto. A palavra objeto em si tem vários significados dependendo do contexto onde ele é aplicado. Dentro da abordagem orientada a objeto, objeto é uma entidade capaz de guardar informações sobre o seu estado e capaz de fornecer operações que informam ou afetam este estado (Coad e Yourdon 1992; Jacobson et al. 1992; Booch, 1991; Ambiola e Bendix, 1989). Da perspectiva do projeto de um sistema de software, objetos modelam as entidades no domínio da aplicação. As características básicas de um objeto são descritas a seguir (Kung et al., 1995; Korson e McGregor, 1990).

- Um objeto modela uma entidade ou coisa no domínio da aplicação. Por exemplo, um livro ou um empregado no mundo real podem ser modelados como objetos.

- Um objeto tem um conjunto de valores de atributos que definem o seu estado. Por exemplo, o atributo "status" de um livro na biblioteca pode ter os valores 'disponível', 'emprestado', 'reservado', 'desaparecido' ou 'alocado'. Estes valores serão utilizados para determinar o estado de um objeto livro.

- Um objeto tem um conjunto de operações que é capaz de realizar uma alteração em seus valores de atributos ou em valores de atributos de outros objetos. Por exemplo, ao se preencher uma ordem de emissão de mercadorias em uma fábrica, tem-se as seguintes alterações: 1) a ordem de emissão altera seu estado de 'ordem nova' para 'ordem emitida'; 2) o balanço do cliente é alterado para refletir a quantidade que recebeu; e 3) a quantidade disponível é atualizada para refletir a quantidade vendida ao cliente.

- Um objeto tem uma identidade que pode ser usada para sua identificação, que deve ser única. Cada objeto tem sua própria identidade, e caso dois objetos tenham os mesmos valores de atributos, eles podem ser distinguidos através de suas identidades.

Os objetos a serem utilizados no projeto de software não são somente os objetos do mundo real, podendo ser um item, uma unidade ou uma entidade abstrata que tem um papel bem definido no domínio do problema (Booch, 1991). Dentro da abordagem orientada a objeto há outros tipos de objetos que são criados durante o desenvolvimento de software e que têm atributos e operações específicas dentro do software. Por exemplo: janelas, menu, telas, gráficos, desenhos que são utilizados para interface gráfica com o usuário.

A modelagem básica de um sistema utilizando a abordagem orientada a objeto consiste de um conjunto de objetos que se comunicam entre si. Alguns destes objetos podem ter características comuns e podem ser agrupados de acordo com estas características. Este agrupamento representa uma classe. Portanto uma classe representa um

molde para vários objetos que descreve como é a estrutura interna padrão destes objetos. Em sistemas orientados a objeto cada objeto pertence a uma classe e o objeto que pertence a uma determinada classe é denominado de instância daquela classe (Gibbs et al., 1990).

Outro conceito básico da abordagem orientada a objeto é o encapsulamento que significa a capacidade de esconder informações, ou seja, quem for utilizar um objeto não precisa necessariamente saber como o objeto foi modelado, como foram implementados os atributos e as operações que o objeto é capaz de realizar (Kung et al., 1995).

Dentro do paradigma estruturado, a modelagem dos atributos e das operações é feita separadamente. Por exemplo, as operações que podem ser realizadas sobre os objetos livros são modeladas usando diagramas de fluxo de dados e implementadas por funções/procedimentos, enquanto os atributos dos objetos livros são especificados em um dicionário de dados e implementados por estruturas de dados. No paradigma orientado a objeto, a ligação interna das estruturas de dados e os detalhes de implementação dos procedimentos são implementados juntos, dentro dos objetos.

A propriedade da herança, outro conceito básico dentro da abordagem orientada a objeto, é uma relação entre classes que permite que a definição e implementação de uma classe sejam baseadas em uma ou mais classes existentes. Quando uma classe Y herda de uma classe X, diz-se que Y é a classe derivada, ou subclasse, e X é a classe básica, ou superclasse. Neste caso a classe Y tem duas partes, uma parte derivada e uma parte que foi incrementada. A parte derivada é a parte herdada da classe

X e a parte incrementada é o novo código escrito especificamente para a classe Y (Wilde e Huitt, 1992).

Uma classe objeto pode ter mais do que uma superclasse. Por exemplo, assistentes de professores que fazem pós-graduação são uma subclasse tanto da classe estudantes como da classe empregados. Geralmente isto é conhecido como herança múltipla (Kung et al., 1995).

O conceito de polimorfismo significa a habilidade para tomar mais de uma forma, ou seja, um atributo pode ter mais do que um conjunto de valores, e uma operação pode ser implementada por mais de um método. Um exemplo comum, geralmente utilizado, é o de gráficos por computador, em que diferentes métodos de desenhar podem ser implementados para a operação desenhar. Este método apropriado, por exemplo, desenhar_círculo e desenhar_retângulo, será ativado em tempo de execução de acordo com o objeto, círculo ou retângulo (Lejter et al., 1992).

E o último conceito a ser citado é o da ligação dinâmica significando que o método que implementa a operação é desconhecido até o tempo de execução. É o mecanismo utilizado para implementar o polimorfismo. A escolha de qual implementação usar quando uma operação é invocada, somente é determinada em tempo de execução de acordo com os tipos, o número de argumentos e/ou a função apontada (Korson e McGregor, 1990).

4.3 - Problemas Relacionados

Quando se está trabalhando na área de software, existe um consenso de que não há nenhuma solução mágica para os problemas originalmente difíceis associados

ao desenvolvimento de software. Não há nenhuma abordagem, método, técnica ou ferramenta que irá aumentar automaticamente a produtividade por um fator de dez, por exemplo, e eliminar, simultaneamente, todos os erros e defeitos de um software (Gibbs et al., 1990).

As abordagens estruturadas e as orientadas a objeto, bem como as ferramentas CASE e os ambientes de desenvolvimento podem e ajudam no desenvolvimento de software, mas o seu uso não é uma garantia de sucesso do projeto de software.

Apesar de haver uma divulgação de que a abordagem orientada a objeto traz benefícios e vantagens para a produção de software (Coad e Yourdon, 1992; Korson e McGregor, 1990), não há nenhuma garantia de que as técnicas de modelagem orientadas a objeto possam evitar um projeto desastroso (Kung et al., 1995). Sabe-se que ao não se fazer uma análise adequada das entidades relevantes do sistema que serão os objetos do projeto, ao não se definir com cuidado os atributos e métodos dos objetos, ao se fazer uso excessivo de heranças múltiplas e de polimorfismo, pode-se ter como conseqüência o insucesso do produto. Sem esquecer daqueles problemas que independem de método ou de ferramenta, tais como as incoerentes decisões políticas da alta gerência, o mau gerenciamento do projeto ou o envolvimento de pessoas pouco competentes para o desenvolvimento do projeto.

No caso da abordagem orientada a objeto, a sua maior dificuldade reside exatamente na transição da metodologia na qual está-se trabalhando para esta nova abordagem, pois o enfoque orientado a objeto é substancialmente diferente, por exemplo, da decomposição funcional, da análise estruturada e dos métodos típicos de modelagem de informações (Coad e Yourdon, 1992).

Além deste problema principal, decorrência natural quando se vai começar a trabalhar com quaisquer novas tecnologias ou novos métodos que necessitam de uma nova forma de abordar o desenvolvimento de software, há outros problemas decorrentes das características inerentes da abordagem orientada a objeto. A seguir serão apresentados alguns dos problemas que podem surgir em função destas características e que mais tenham a ver com as atividades de gerenciamento de desenvolvimento, de teste e de manutenção de software.

Deve-se ressaltar que estes problemas abaixo descritos também podem surgir em outros tipos de abordagem ou de metodologia, caso os conceitos preconizados pela metodologia e pela boa programação não sejam corretamente aplicados. O que se tem verificado é que na abordagem orientada a objeto, a falta de uma aplicação correta tanto da metodologia quanto da programação orientada a objeto, trazem uma maior complexidade na hora de se implementar as alterações corretivas, adaptativas e/ou perfectivas, e, como agravante, podem até causar um colapso no sistema em construção de tal forma que não se consegue obter uma solução para o problema, sendo necessário, por vezes, ter que recomeçar o projeto.

4.3.1 - Problema do Entendimento

O problema do entendimento pode ser gerado pelo uso exagerado e sem controle das características do encapsulamento. As informações escondidas pode fazer com que várias funções membro de, possivelmente, várias classes sejam invocadas para realizar uma determinada funcionalidade. Geralmente, uma função membro de uma classe

invoca outras funções membro tendo como resultado uma cadeia de invocação de funções membro.

Como conseqüência da existência destas cadeias de invocação, quem for fazer o teste ou a manutenção do produto tem que conhecer as seqüências de chamada das funções membro e as semânticas das classes, antes de preparar quaisquer casos de teste e/ou antes de modificar uma determinada funcionalidade do sistema. Desde que há necessidade de se conhecer suficientemente todas as partes em detalhe antes de se fazer testes/modificações, isto pode trazer uma maior complexidade para testar e manter sistemas orientados a objeto, principalmente, quando houver uma utilização exagerada de heranças múltiplas e/ou de polimorfismo.

Caso se queira construir um diagrama que retrate a estrutura de dependência das classes, a estrutura de representação será um grafo e não uma árvore como na abordagem funcional. No caso de grandes sistemas e caso se utilize em demasia a herança múltipla e o polimorfismo, pode ser difícil a compreensão do grafo devido ao seu tamanho e complexidade.

4.3.2 - Problema da Complexidade da Dependência

O problema da complexidade da dependência é gerado pelos novos tipos de relacionamentos que existem na programação orientada a objeto, tais como: herança, agregação, associação, instanciação da classe, classes aninhadas, criação dinâmica de objeto, invocação de função membro, polimorfismo e ligação dinâmica. Existindo um destes tipos de relacionamentos entre classes, implicará no estabelecimento de um relacionamento de dependência entre estas classes.

Muitas ferramentas de configuração e de manutenção dependem do rastreamento de dependências dentro dos programas. As linguagens orientadas a objetos, tais como Smaltalk, complicam consideravelmente este processo de rastreamento por permitir uma ligação dinâmica de mensagens para métodos específicos. Uma dada mensagem pode se referir a um objeto de qualquer classe usando o polimorfismo e o método a ser executado quando uma dada mensagem é enviada para ele, será determinado em tempo de execução pela classe do objeto que ele representa naquele instante (Korson e McGregor, 1990).

De forma semelhante, em C++, ponteiros e referências de objetos podem se referir não somente aos objetos declarados pela sua classe, mas também para as suas subclasses. Quando uma operação é realizada sobre um objeto, uma função declarada como virtual se ligará dinamicamente à implementação da função que é apropriada à classe inferior do objeto em tempo de execução (Stroustrup, 1986).

Enquanto a ligação dinâmica fornece muita flexibilidade nas linguagens orientadas a objetos, ela pode complicar o rastreamento de dependências. Por exemplo, pode-se constatar que quando uma mensagem é enviada para uma variável interagir com um objeto, a implementação real do método que será chamado depende da classe do objeto. Desde que diferentes implementações podem estabelecer diferentes dependências, uma estrutura estática não será sempre capaz de identificar precisamente as dependências no programa.

Uma dependência de forma geral em um sistema de software é, informalmente, uma relação direta

entre duas entidades do sistema X → Y tal que quando houver uma alteração em X, deve-se verificar os possíveis efeitos em Y.

No desenvolvimento tradicional podem existir relacionamentos entre variáveis, módulos de processamento e tipos de dados. Estes relacionamentos de dependência podem ser do tipo: dependência de dados entre duas variáveis; dependência de chamada entre dois módulos; dependência funcional entre um módulo e a variável com o qual ele trabalha; e dependências de definição entre as variáveis e seus tipos.

Ao se tratar com linguagens orientadas a objeto, há necessidade de se trabalhar com os seguintes tipos de entidades adicionais: classes, métodos e mensagens. Além disso os atributos podem agora representar instâncias de uma classe ao invés de valores convencionais de dados.

O uso exagerado do polimorfismo e da herança podem criar tipos de dependências em maior quantidade e com maior complexidade, e que precisam ser considerados durante as fases de teste, de manutenção e para as atividades do GCS.

A seguir são apresentados alguns tipos de relacionamentos que podem ocorrer na programação orientada a objeto (Wilde e Huitt, 1992):

- Dependências entre classes:
  - C1 é uma super classe de C2
  - C1 é uma sub classe de C2
  - C1 herda de C2

- C1 usa C2 (que pode ser uso de interface ou uso de implementação)

- Classe e Método:
  - método M retorna objeto da classe C
  - C implementa método M
  - C herda método M

- Classe e Mensagem:
  - C entende a mensagem

- Método e Variável:
  - V é um parâmetro para método M
  - V é uma variável local do método M
  - V é importado por M
  - V é definido por M
  - M se refere a V

- Método e Mensagem:
  - mensagem M' é nome do método M
  - método M envia mensagem M'

- Método e Método:
  - Método M1 invoca método M2
  - Método M1 sobrepõe-se a M2

Os tipos de relacionamentos que passam a existir na abordagem orientada a objeto, podem tornar a integração e teste, a manutenção e as atividades de GCS extremamente difíceis, principalmente, quando não for seguida corretamente a aplicação dos conceitos da abordagem e da boa programação orientadas a objeto, e/ou quando se fizer uso exagerado de herança múltipla e de polimorfismo. Nestes casos haverá, basicamente, uma maior dificuldade no entendimento de uma dada classe caso ela tenha muitos e

complexos relacionamentos de dependência com várias outras classes; na definição da inicialização dos testes de uma biblioteca de classes caso não se tenha os dados suficientes; na definição e gerenciamento dos casos de teste e na criação e inicialização apropriada de determinados objetos; na previsão e teste de todos os possíveis uso de um molde de classe.

4.3.3 - Problema de Testar o Comportamento de Estado

Um outro tipo de problema que até então não existia nas abordagens tradicionais, surge devido ao fato de que os objetos têm estados e comportamentos dependentes destes estados, isto é, o efeito de uma operação sobre um objeto depende do estado em que se encontra tal objeto, que poderá, ou não, mudar de estado. Logo, deve ser testado o efeito combinado das operações. Kung et al. (1995) apresenta o seguinte exemplo da caixa de moeda.

Considere uma classe caixa de moeda, implementada em C++, de uma máquina de venda de bebidas. Por simplicidade, assume-se que a caixa de moeda tem uma funcionalidade muito simples e que o código para controlar o dispositivo físico foi omitido. A caixa aceita somente moedas de 25 centavos e permite a venda quando são colocadas duas moedas. Ela mantém o controle do total de moedas recebidas (totalQtrs), das moedas correntes recebidas (curQtrs), e se a venda é permitida ou não (allowVend). Suas funções incluem adicionar uma moeda, retornar as moedas correntes, dar um "reset" na caixa de moeda para seu estado inicial, e permitir a entrega do produto. O código fonte C++ para esta caixa de moeda simples é mostrado na Figura 4.1.

```
class CoinBox
{
      unsigned totalQtrs;// total quarters collected
      unsigned curQtrs; //current quarters collected
      unsigned allowVen; // 1 = vending is allowed
pblic:
      CoinBox() { Reset (); }
      void AddQtr (); // add a quarter
      void ReturnQtrs() { curQtrs = 0;} // return current quarters
      unsigned isAllowedVend () { return allow Vend; }
      void Reset() { totalQtrs = 0; allowVend = 0; curQtrs = 0 }
      void Vend(); // if vending allowed, update totalQtrs and curQtrs
};
void CoinBox::AddQtr()
{
   curQtrs = curQtrs + 1; // add a quarter
   if (curQtrs > 1) // if more than one quarter is collected
      allowVend = 1; // then set allowVend
}
void CoinBox::Vend()
{
   if (isallowedVend() ) // if allow  Vend
   {
      totalQtrs = totalQtrs + curQtrs; // update totalQtrs,
      curQtrs = 0;                     // curQTRS, and
      allowVend = 0;                   // allowVend
   }                                   // else no action
}
```

Fig. 4.1 - Código Fonte da Caixa de Moeda
FONTE: Kung et al. (1995), p. 79.

Ao examinar o código apresentado, pode-se verificar que há um erro de implementação, pois a seqüência de operação AddQtr(); AddQtr(); ReturnQtrs(); Vend() pode ser sucessivamente realizada. Isto significa que um cliente pode obter uma bebida grátis, pois basta inserir duas moedas, requisitar a devolução das duas moedas e pressionar o botão para obter uma bebida. Pode-se perceber que este erro não foi descoberto até ser aplicado o teste do comportamento de estado do objeto. Este tipo de erro não pode ser facilmente detectado pelo teste funcional e/ou teste da estrutura da unidade das funções membros, nem pode ser facilmente detectado por teste de integração das funções membro, isto porque: cada uma das funções membro

implementa uma determinada funcionalidade; ao se seguir corretamente cada uma das cadeias de chamadas de métodos não iria ser detectado o erro, a menos que o objeto seja posto propriamente em um determinado estado; o teste de integração detecta erros somente nas interfaces das funções membro.

Este tipo de erro também pode acontecer utilizando-se a linguagem C, mas o exemplo apresentado tem somente a finalidade de mostrar que dentro da abordagem orientada a objeto há a necessidade de se testar o comportamento de estado de cada objeto do sistema. Geralmente para realizar este teste de comportamento de estado há a necessidade de se derivar, a partir do código fonte, os comportamentos dependentes de estado dos objetos e representar estas informações em um diagrama de estado do objeto. Dependendo de como são aplicados os conceitos da abordagem orientada a objeto e de como foi desenvolvido o projeto, pode existir uma maior dificuldade em se montar os casos de teste para este tipo de comportamento. De qualquer forma as técnicas e métodos do GCS podem dar um excelente suporte a esta atividade.

4.3.4 - Problema das Ferramentas de Suporte

Os desenvolvedores que trabalham em grandes sistemas de software têm necessidade de conhecer a estrutura global do sistema, que pode ser obtida, no caso da abordagem estruturada, com a identificação dos relacionamentos entre funções, procedimentos, definições e uso de variáveis etc. Sabe-se que o GCS pode fornecer todas estas informações na medida em que tem cadastrado os relacionamentos de dependência entre estes elementos. Quanto maior e mais complexo o sistema, maiores são os números de elementos e de relacionamentos.

Em linguagens tradicionais, a estratégia para identificar os relacionamentos de dependência entre os elementos de software se faz através do uso do nome do elemento em uma busca léxica dentro dos arquivos onde os mesmos estão armazenados. No uso destas linguagens tem-se a prática de se quebrar o sistema de software em arquivos por funcionalidade. Desta forma a definição da função e das variáveis, geralmente, se encontram no mesmo arquivo (Lejter et al., 1992).

Nas linguagens orientadas a objeto, tal como C++, que suporta sobreposição de função e de operador, herança, ligação estática e dinâmica, e possui grandes facilidades para abstração de dados, faz com que a abordagem léxica para a busca de símbolos seja ineficiente. Por exemplo, em C++, uma referência à variável X dentro de uma função membro de uma classe pode somente ser identificada após examinar o escopo da função e da classe nas quais ela aparece e em todas as superclasses da classe (Wilde e Huitt, 1992).

As ferramentas CASE estão evoluindo para que suporte integralmente a abordagem orientada a objeto, sendo que algumas ferramentas existentes que ainda utilizam técnicas e métodos das abordagens convencionais, podem, com algumas adaptações, ser utilizadas para se trabalhar com a abordagem orientada a objeto (Kung et al., 1995).

Como citado anteriormente, as alterações em um programa orientado a objeto podem se espalhar através de todo o software e a identificação manual do impacto de uma alteração é difícil e toma muito tempo dos desenvolvedores, além de se correr o risco de omitir certas partes realmente afetadas.

Portanto, as ferramentas CASE podem, com maior rapidez, executar a identificação das alterações e de seus impactos durante as fases de teste e de manutenção bem como durante a geração e/ou reutilização de software. Estas ferramentas devem atender a dois requisitos básicos: capacidade de descrever a estrutura que está sendo desenvolvida e fornecer uma interface amigável ao usuário de forma eficiente.

CAPÍTULO 5

SISTEMA DE CONTROLE DE ELEMENTOS DE SOFTWARE PARA A ABORDAGEM ESTRUTURADA

5.1 - Aspectos Gerais

Um Sistema de Controle de Elementos de Software foi implementado e utilizado durante o desenvolvimento do Sistema de Controle de Satélites (SICS), da Missão Espacial Completa Brasileira (MECB). A descrição resumida deste Sistema, a ser apresentada a seguir, foi baseada nos trabalhos de Cunha e Nakanishi (1993a,b) que contém os conceitos básicos da implementação.

Este Sistema de Controle produziu muitos bons resultados não somente durante o desenvolvimento do SICS, mas também durante a sua manutenção e durante a sua migração para máquinas mais modernas.

O mecanismo de reserva de produtos e a verificação preliminar automática de consistência dos elementos de software incrementaram a qualidade e a produtividade de software. A geração automática de versões de sistema e o controle de consistência dos elementos de interface contribuíram para que o software fosse concluído no tempo previsto. O acesso automático dos registros de elementos simplificou a análise do impacto das modificações. A organização de ambientes de desenvolvimento contribuiu para a distribuição correta de trabalho e de responsabilidades, bem como para disciplinar as atividades das equipes envolvidas durante a vida útil do SICS.

Na implementação deste Sistema de Controle foram utilizados basicamente quatro pacotes de software: dois pacotes da DEC, o Code Management System (CMS), para controle de versões e o Module Management System (MMS), para geração de sistema; e outros dois pacotes, o Banco de Dados de Elementos Controlados (BDEC), e o Sistema de Suporte ao Controle de Elementos (SSCE), que foram desenvolvidos pela equipe do Instituto Nacional de Pesquisas Espaciais (INPE) responsável pelo gerenciamento e controle do desenvolvimento do SICS.

Originalmente, este Sistema de Controle foi implementado para o desenvolvimento de software utilizando a abordagem estruturada, mas, como será visto ao longo deste capítulo, a sua arquitetura e seus processos de controle também poderão ser utilizados, com algumas modificações, para a abordagem orientada a objeto.

A seguir serão apresentadas as principais características deste Sistema.

5.2 - Arquitetura

Para que as diferentes atividades do processo de desenvolvimento e manutenção de software fossem executadas de forma controlada e disciplinada, a arquitetura do Sistema de Controle de Elementos de Software implementou os Ambientes de Desenvolvimento. Em cada um destes ambientes foram agrupadas as atividades relacionadas aos mesmos objetivos ou que pertenciam a uma mesma parte do processo de desenvolvimento e/ou de manutenção. Neste agrupamento levou-se em conta ainda, as características do software que foi desenvolvido, da metodologia adotada, da qualidade requisitada e dos recursos computacionais que

estavam disponíveis. Foram definidos os seguintes ambientes:

- AMBIENTE DE SISTEMA
- AMBIENTE DE CONTROLE DE CONFIGURAÇÃO
- AMBIENTE DE CONTROLE DE QUALIDADE
- AMBIENTE DE IMPLEMENTAÇÃO E PROJETO
- AMBIENTE DE INTEGRAÇÃO DE SUBSISTEMA
- AMBIENTE DE INTEGRAÇÃO DE SISTEMA
- AMBIENTE DE PROTOTIPAÇÃO

5.3 - Elementos Controlados

No processo de controle, os elementos produzidos durante o desenvolvimento de sistemas eram checados em relação à sua qualidade antes de se tornarem um elemento controlado. Na condição de elementos controlados, eles já estavam prontos para serem lidos e consultados, mas não poderiam ser modificados sem uma prévia autorização.

Os elementos referenciados para controle eram produtos (unidades de software e documentos) e não parte deles. Neste tipo de Sistema de Controle, os Itens e os Relacionamentos também foram considerados como elementos controlados. Um exemplo de um conjunto de elementos controlados de um grande sistema com tarefas concorrentes e utilizando a abordagem estruturada é apresentado na Tabela 5.1.

Deve-se observar que para a finalidade de controle, além dos programas, rotinas e documentos tradicionais, o tipo de produto incluía arquivos de dados, "command procedures" e arquivos "include". Os Itens também foram considerados como partes elementares dos produtos, ou

como grupos de elementos. O Processo de Diagrama de Fluxo de Dados e as Variáveis Globais são exemplos de Itens, e neste caso, são, respectivamente, partes de documentos e de programas.

TABELA 5.1 - EXEMPLO DE ELEMENTOS CONTROLADOS NA ABORDAGEM ESTRUTURADA

Produtos	Itens	Relacionamentos
A - Arquivos de Dados	DF - Data Flows	CAL - Call
F - Funções	EE - External entity	DEL - Delete
I - Arquivos Include	DP - DFP process	WRI - Write
O - Command Procedures	DS - Data store	WAI - Wait
P - Programas	ER - Entity (ER)	USE - Use
S - Subrotinas	RR - Relationships (ER)	CRI - Create
SSD - Specification and Analysis	AR - Attribute (ER)	SOF - Switch off
SDD - System Design	SU - Subsystem	GEN - Generate
ITP - Integration and Test Plans	TA - Task	REA - Read
UMD - User Manual	MQ - Message Queues	SON - Switch on
IMD - Installation Manual	SA - Shared Area	MAP - Map
ITD - Integration and Test Report	SI - Signal	CMP - Compose
	MS - Synchronism Message	SIM - Simulate
	MD - Module	TST - Test
	CB - Common Blocks	
	EF - Event Flags	
	CE - Cluster of event flags	
	GS - Global sections	
	JB - Jobs	
	CS - Chain Subsystem	
	KB - Keyboards	
	LN - Logican Names	
	RP - Routine Parameters	
	GV - Global Variables	

FONTE: Cunha e Nakanishi (1993b), p. 485.

Os Relacionamentos são aqueles elementos que associam Produtos a Produtos, Produtos a Itens e Itens a Itens. Alguns exemplos de relacionamentos são: um programa chama (CHA) uma sub-rotina, ou um programa escreve (ESC) em uma Mailbox.

Enquanto os Itens eram importantes para manter a coerência entre os Produtos, os Relacionamentos foram importantes para manter a coerência entre todos os Itens e Produtos. A definição de quais elementos devem ser

controlados depende das características do software e do processo que será usado para seu desenvolvimento.

Os Produtos, após serem verificados e aprovados, eram guardados em Bibliotecas Controladas. Os registros de todos estes elementos, Produtos, Itens e Relacionamentos eram mantidos no BDEC. Deste modo, o controle do BDEC e das Bibliotecas Controladas eram a base do processo de controle dos elementos.

5.4 - Recursos Computacionais

As atividades de desenvolvimento de software sempre envolvem a utilização de diferentes recursos computacionais tais como: arquivo de memória secundária, mecanismos de comunicação e de interrupção, recursos de controle de processo, utilidades etc. O uso destes recursos deve ser disciplinado e deve obedecer determinadas normas a fim de permitir o correto funcionamento dos esquemas de controle.

Além desses, os sistemas de controle necessitaram de outros recursos, tais como: pacotes de software para controle de versões e para geração de sistema, banco de dados para armazenar os elementos controlados e sistema de suporte para controle de elementos. O maior ou menor nível de automatização das atividades de controle depende da análise de custo-benefício, das características do software a ser desenvolvido e das ferramentas de software e do hardware disponíveis.

O pacote de software utilizado para o controle de versões de produtos, CMS, tinha as seguintes características principais: permitia um arquivo ser

modificado somente se ele fosse previamente reservado; informava quais usuários estavam utilizando os arquivos da biblioteca; mantinha um arquivo "log" transações de uso da biblioteca; mantinha versões de arquivos de biblioteca.

O pacote de software utilizado para a geração do sistema, MMS, tinha as seguintes características principais: mantinha uma relação da versão dos produtos que compunham cada versão do sistema, definia e gerava as versões do sistema, garantia que qualquer alteração posterior do produto seria refletida na versão do sistema.

O BDEC tinha como principal objetivo armazenar os registros dos elementos controlados contendo informações dos Produtos, Itens e Relacionamentos. Além disso, o BDEC era capaz de fornecer diferentes tipos de relatórios e realizar o cruzamento de informações armazenadas sobre estes elementos. As facilidades do BDEC suportavam várias atividades tais como: verificação da qualidade de elementos controlados; verificação de consistência entre Produtos, Itens e Relacionamentos; geração de versões de sistema; avaliação do impacto sobre o sistema quando uma modificação de um elemento controlado era requerida.

O SSCE, pacote de software que foi desenvolvido pela própria equipe do INPE, fornecia facilidades tais como: geração e transmissão de formulários padrões e outras mensagens, registro de fluxo de formulários e seus "status", transferência controlada de arquivos e Produtos, verificação parcial do Produto levando-se em conta sua completeza e consistência. Tais processos eram realizados e controlados de forma manual.

5.5 - Procedimentos para Controle de Elementos

Havia basicamente duas categorias de procedimentos para o controle de produtos: procedimentos para entrega de novos produtos e procedimentos para modificação de produtos controlados.

Quanto à entrega de novos produtos havia os seguintes procedimentos básicos:

- ENTREGA DE PRODUTOS
- COLEÇÃO DE PRODUTOS
- VERIFICAÇÃO DOS PRODUTOS
- MANIPULAÇÃO DE PRODUTOS ANALISADOS
- AÇÕES DO DESENVOLVEDOR

Quanto à modificação de elementos controlados havia os seguintes procedimentos básicos:

- COMUNICAÇÃO DE PROBLEMA/MODIFICAÇÃO
- AVALIAÇÃO DO FORMULÁRIO DE COMUNICAÇÃO DE PROBLEMA/MODIFICAÇÃO
- GERENCIAMENTO DAS AUTORIZAÇÕES DE MODIFICAÇÃO DE PRODUTOS
- MODIFICAÇÃO DE PRODUTOS
- CONTROLE DE PRODUTOS MODIFICADOS

A maior parte destes procedimentos eram feitos de forma manual, o que tornava o processo de controle e gerenciamento muito tedioso, tomando muito tempo das equipes de desenvolvedores, principalmente da equipe de controle de configuração. O tempo gasto não era somente por causa das análises dos formulários e dos produtos encaminhados, mas também por causa da busca e solução de

erros dentro dos processos de controle e cadastramento, uma vez que os desenvolvedores não estavam acostumados a qualquer tipo de controle de configuração.

5.6 - Considerações Finais

O principal objetivo da apresentação do Sistema de Controle de Elementos de Software utilizado para desenvolvimento do SICS, foi de destacar a aplicação prática sobre a importância de controlar os elementos de um grande sistema de software quando se busca qualidade e produtividade. Além disso deve ser observado que somente controlar os produtos não é suficiente, também é necessário controlar os vários tipos de itens e de relacionamentos para que se possa ter controle sobre as alterações nos elementos de software que ocorrem durante a vida útil do sistema.

Portanto, ao se implementar um esquema de controle é necessário tomar um conjunto de precauções, de forma que o controle se torne uma parte essencial do processo de desenvolvimento, e que ele seja aplicado durante todo o ciclo de vida do sistema.

CAPÍTULO 6

ESQUEMA DE CONTROLE DE ELEMENTOS DE SOFTWARE PARA A ABORDAGEM ORIENTADA A OBJETOS

6.1 - Descrição Geral

Esta proposta de um Esquema de Suporte ao Controle de Elementos de Software Orientado a Objeto foi baseada no Sistema descrito no Capítulo 5 e tem como finalidade definir os aspectos fundamentais de um Esquema que permita dar suporte ao controle dos elementos de software para a abordagem orientada a objeto, e servir de base para uma futura implementação de um Sistema de Controle de Elementos de Software Orientado a Objeto.

A descrição deste Esquema de Controle procura abstrair de quaisquer metodologias de desenvolvimento de software orientado a objeto, permitindo a aplicação o Esquema, com adaptações, à maioria das metodologias orientadas a objeto. Quando for escolhida a metodologia com a qual irá desenvolver o software, haverá a necessidade da definição adicional de outros elementos de software - produtos, itens e relacionamentos - que serão específicos à metodologia adotada.

Quando se utiliza a abordagem orientada a objeto, basicamente se trabalha com classes de objetos que serão os principais elementos de software a serem gerenciados e controlados. Este Esquema de Controle utiliza um banco de dados como repositório, onde estarão armazenadas todas as informações das classes e de outros elementos de software tais como componentes, documentos e

diagramas, definidos de acordo com a metodologia orientada a objeto que for adotada.

As classes possuem um grande potencial de serem reutilizadas em função da sua própria concepção e da utilização dos mecanismos de encapsulamento, abstração de dados, instanciação, herança e generalidade. Esta reutilização das classes poderá ser feita não somente pelos próprios desenvolvedores que implementaram as classes, mas também por desenvolvedores de outras aplicações, de outros projetos e até de outras organizações (Coad e Yourdon, 1992, 1993). Logo, este Esquema de Controle pode servir não somente como uma ferramenta de suporte ao gerenciamento e controle de classes para o desenvolvimento de software, mas também como um repositório de uma coleção de classes do qual poder-se-á recuperar classes com finalidades de reutilização em outras aplicações (Gibbs et al., 1990).

Nos itens a seguir serão apresentados os principais aspectos do Esquema de Controle proposto, sendo que a parte de controle de classes tem o aspecto de generalidade, procurando fazer com que seja válida para a abordagem orientada a objeto como um todo, e a parte de controle específica está relacionada com os elementos de software que serão definidos de acordo com a metodologia orientada a objeto que for adotada.

Antes de se passar à descrição propriamente dita, será feita a apresentação dos conceitos básicos da metodologia de Coad & Yourdon (1992, 1993) que servirá apenas para exemplificar alguns tipos de elementos de software que poderiam ser definidos ao se adotar tal metodologia. Deve-se ressaltar que a apresentação destes conceitos será feita de forma sucinta, apenas para identificar alguns elementos de software que podem ser controlados por este tipo de Esquema.

6.2 - Metodologia de Coad & Yourdon

A metodologia de Coad & Yourdon (1992, 1993) tem por base a Análise Orientada a Objetos (Object-Oriented Analysis - OOA) e o Projeto Orientado a Objetos (Object-Oriented Design - OOD).

O modelo da OOA consiste em cinco camadas conforme pode ser visto na Figura 6.1. Estas camadas podem ser consideradas como cortes horizontais do modelo geral e, dentro da abordagem global, estas cinco camadas correspondem às cinco atividades principais da OOA: Determinar Classes-&-Objetos, Identificar Estruturas, Identificar Assuntos, Definir Atributos e Definir Serviços.

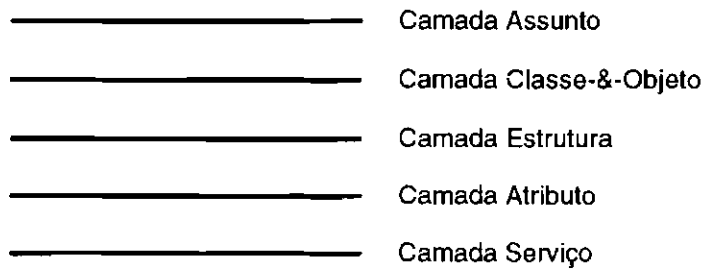

Fig. 6.1 - Modelo Cinco Camadas
FONTE: Coad e Yourdon (1992), p. 149.

Estas atividades objetivam guiar o trabalho dos desenvolvedores de forma que possam partir dos níveis mais altos de abstração, como por exemplo, as Classes-&-Objetos dentro do domínio do problema, até os níveis mais baixos de abstração, no caso, Estruturas, Atributos e Serviços. Entre estes níveis mais baixos não existe qualquer ordem seqüencial para serem desenvolvidos.

O modelo do OOD consiste em quatro componentes conforme é apresentado na Figura 6.2. Estes componentes podem ser considerados como cortes verticais do modelo geral, e dentro da abordagem global, estes quatro componentes correspondem às quatro atividades principais do OOD: Projetar o Componente Domínio do Problema, Projetar o Componente Interação Humana, Projetar o Componente Gerenciamento de Tarefas e Projetar o Componente Gerenciamento de Dados. Estas atividades também não precisam ser necessariamente executadas em passos seqüenciais.

Fig. 6.2 - Modelo Quatro Componentes
FONTE: Coad e Yourdon (1992), p. 174

As atividades da OOA e do OOD tentam incorporar estratégias distintas de forma a se obter de modo eficiente a análise e o projeto do sistema. Na OOA procura-se identificar e definir todas as Classes e Objetos contidos no domínio do problema e suas respectivas responsabilidades. Enquanto no OOD, após incorporar o resultado da OOA, procura-se identificar Classes e Objetos adicionais de forma a implementar completamente as especificações dos requisitos do sistema.

O resultado da OOA passa diretamente para o Componente Domínio do Problema (CDP), e dentro deste componente as partes das Classes, Estruturas, Atributos e Serviços sofrerão modificações de acordo com determinados critérios técnicos dentre os quais podem ser destacados: o critério para reutilização do projeto e da codificação de

classes que estão sendo desenvolvidos; o critério de agrupamento de classes dentro do esquema de generalização-especialização ou todo-parte; o critério para melhorar a performance do sistema; o critério para acrescentar detalhes de baixo nível.

O Componente Interação Humana (CIH), inclui todas as interfaces necessárias para a efetiva interação dos usuários com o sistema. A forma de se apresentar as interfaces irão variar de acordo com as interfaces gráficas existentes no mercado - Macintosh, Windows, Xwindows e Motif. Deve-se observar que as classes definidas para este componente podem ser reutilizadas pelos subsistemas e por diferentes aplicações.

Dentro do Componente Gerenciamento de Tarefas (CGT), existe a definição das tarefas, da comunicação entre tarefas e da coordenação/sincronização das tarefas, bem como as definições sobre a alocação de hardware, sobre os protocolos para sistemas e dispositivos externos.

E, finalmente, o Componente Gerenciamento de Dados (CGD), que fornece o acesso e o gerenciamento dos dados permanentes. Este componente possui a natureza de interface entre o gerenciamento dos dados e o meio onde estão armazenados estes dados, podendo atuar sobre várias formas de armazenamento - arquivos simples ou banco de dados.

Deve-se ainda observar que esta metodologia propõe como vantagens, a utilização de uma representação uniforme e de uma estrutura de maior estabilidade ao longo de todo o desenvolvimento do software. Isto facilita o entendimento do projeto como um todo, a extensão do software para acréscimo de novas funções, e,

principalmente, a reutilização da análise, do projeto e da codificação de classes.

Neste aspecto, acredita-se que estas vantagens citadas somente poderão ser obtidas a partir do momento que se tenha um sistema de controle dos elementos produzidos onde é possível o armazenamento, o controle e a recuperação de todos os elementos de software e as informações pertinentes a cada camada e a cada componente do modelo, bem como de todos os relacionamentos existentes entre eles.

Portanto, o Esquema que está sendo proposto pode fornecer a base necessária para o correto gerenciamento dos elementos de software orientado a objeto durante seu desenvolvimento e também para a reutilização de software.

6.3 - Ambientes de Desenvolvimento

A arquitetura adotada para o Esquema de Controle para abordagem orientada a objeto é semelhante àquela apresentada no Item 5.2 e tem por base a implementação de Ambientes de Desenvolvimento. A Figura 6.3 contém uma representação desta arquitetura e a seguir será apresentada uma descrição sucinta de cada ambiente.

6.3.1 - Ambiente de Sistema

Neste ambiente são realizadas as atividades relacionadas com a Especificação dos Requisitos, Definição do Processo de Desenvolvimento, Criação de Normas e Padrões, Especificação Funcional e de Dados, Divisão em Subsistemas, Especificação de Interfaces entre Subsistemas,

Aprovação de Especificações de Subsistemas, Definição dos Testes de Aceitação do Sistema, Aprovação de Planos de Testes, Aprovação da Execução da Modificação de Elementos Controlados.

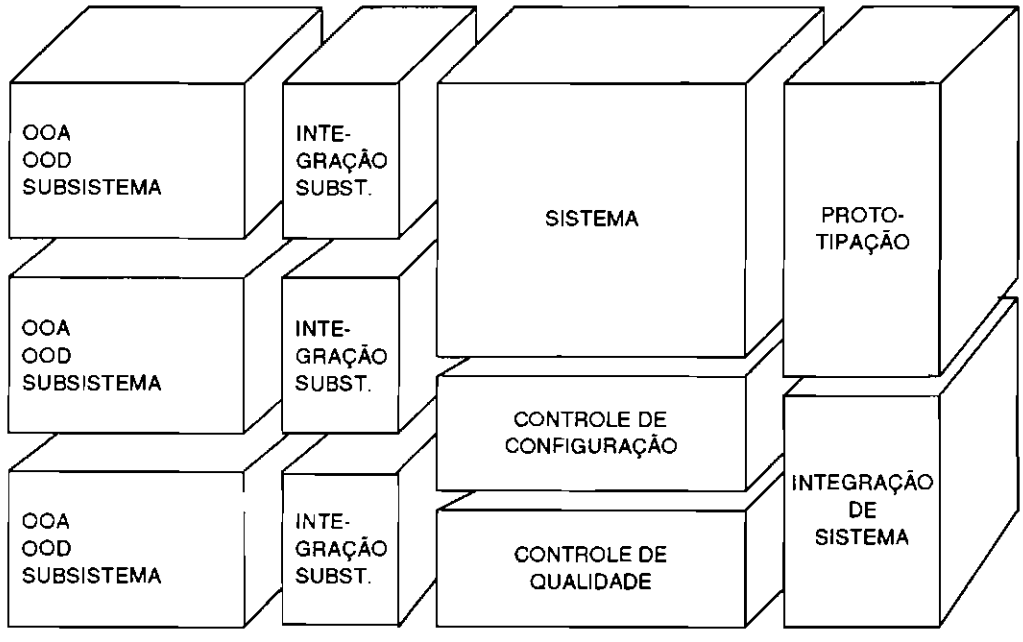

Fig. 6.3 - Arquitetura dos Ambientes de Desenvolvimento
FONTE: Cunha e Nakanishi (1993)

A Equipe de Sistemas é composta pelos líderes de equipes, pelos representantes dos usuários e é liderada pelo Gerente de Sistema de Software. Esta equipe deve manter o controle da visão global do sistema e de suas interfaces durante todo o desenvolvimento.

No estabelecimento de normas e padrões devem ser definidos os formatos dos nomes dos elementos de software relativos à metodologia orientada a objeto adotada; dos nomes de bibliotecas; formato da documentação interna dos programas, dos documentos e manuais etc.

As especificações de requisitos deverão ser feitas neste Ambiente de forma centralizada para se obter uma definição clara dos subsistemas, suas interfaces e suas respectivas especificações. E deverá ainda realizar a aprovação dos planos de integração do sistema, plano de testes dos subsistemas, planos de testes das camadas e dos componentes tomando por base as especificações dos requisitos.

A Equipe de Sistema ainda tem a função de aprovar as alterações dos elementos de software antes das mesmas serem implementadas, sendo que as alterações são analisadas sob os aspectos de custo/benefício e de alocação de recursos materiais e humanos.

6.3.2 - Ambiente de Controle de Configuração

As atividades deste Ambiente estão relacionadas com a Geração e Manutenção das Versões do Sistema, Manutenção das Bibliotecas Controladas, Transferência de Elementos entre Ambientes, Controle do Banco de Dados de Elementos Controlados (BDEC). Estas atividades são realizadas pela Equipe de Controle de Configuração.

Neste Ambiente a Equipe de Controle de Configuração ficará a cargo de realizar a geração e a manutenção das versões do sistema, definindo não somente os elementos que compõem uma versão do sistema, mas também o conjunto de regras de dependências e de ações para a construção de uma determinada versão do sistema.

Fica a cargo desta Equipe também o gerenciamento das Bibliotecas Controladas que armazenam os produtos aprovados pela Equipe de Sistema ou pela Equipe de

Qualidade, tais como, documentos de especificação, planos de teste, arquivos de dados, classes etc. Como o controle das bibliotecas faz parte deste Ambiente, as transferências de elementos de software entre os ambientes de desenvolvimento são feitos pela Equipe de Configuração de Software, devendo manter inclusive um registro de todas estas operações de transferências.

6.3.3 - Ambiente de Controle de Qualidade

Aqui as atividades são relativas à verificação dos elementos de software quanto às especificações, às normas e aos padrões adotados. A responsabilidade por tais atividades é da Equipe de Controle de Qualidade.

As atividades desenvolvidas neste Ambiente consistem em checar se os elementos produzidos estão de acordo com as especificações e com as normas e padrões de qualidade estabelecidos, encaminhando as solicitações de alterações caso necessário. Também é de responsabilidade da Equipe de Controle de Qualidade a verificação da consistência e a completeza dos registros cadastrados no BDEC.

Após os elementos de software serem aprovados por esta Equipe, eles passam a ser elementos controlados por este Ambiente, ou seja, tais elementos e respectivos registros somente poderão ser alterados após uma prévia autorização fornecida pela Equipe de Sistema.

Complementando as atividades desta Equipe, têm-se as atividades de assessoramento e de geração de propostas à Equipe de Sistema para o estabelecimento das

técnicas, métodos, normas e padrões de forma a garantir o nível de qualidade desejada para o Sistema.

6.3.4 - Ambiente de Subsistemas

As atividades neste ambiente são realizadas de forma distribuída em mais de um Ambiente de Subsistemas. Em cada um destes ambientes é desenvolvido um subsistema a fim de facilitar o controle de elementos. Para a Metodologia de Coad & Yourdon (1992,1993), por exemplo, poder-se-ia ter as atividades da OOA e da OOD para cada subsistema. As atividades deste Ambiente estão relacionadas com as atividades de Especificação de Subsistemas, Análise e Projeto de Subsistemas, Planejamento de Testes de Subsistemas, Elaboração de Documentos, Modificação de Software e de Documentos durante as fases de integração e de manutenção. A responsabilidade destas tarefas fica a cargo de cada Equipe de Subsistema.

6.3.5 - Ambiente de Integração de Subsistema

Com a finalidade de testar e integrar um subsistema, os outros subsistemas e entidades externas são substituídos por simuladores que são essencialmente unidades de software produzidas para esta finalidade e que têm interfaces similares àquelas utilizadas pelos subsistemas. Neste Ambiente cada subsistema é testado sob circunstâncias similares às condições reais de operação.

Estas atividades são da responsabilidade de cada Equipe de Subsistema que desenvolverá a integração de acordo com um plano previamente aprovado. Quanto aos testes realizados neste Ambiente, eles são monitorados pela Equipe

de Integração do Sistema já que estes testes fazem parte da integração do sistema como um todo.

Neste Ambiente os subsistemas são integrados e testados isoladamente em condições próximas da realidade através de simuladores, pois nem todos os subsistemas estarão prontos ao mesmo tempo.

6.3.6 - Ambiente de Integração de Sistema

A integração e teste de sistema são realizados neste Ambiente e são feitos com base nos subsistemas e entidades externas, que já deverão estar individualmente testados.

Os passos de integração e testes são realizados pela Equipe de Integração de Sistema de acordo com um plano previamente aprovado pela Equipe de Sistema que fiscalizará tais atividades. Uma versão do sistema é criada pela Equipe de Controle de Configuração para cada passo da integração.

6.3.7 - Ambiente de Prototipação

Os protótipos de software podem ser desenvolvidos para diferentes finalidades. Uma delas é completar a especificação da interface entre o usuário e o sistema. Uma outra é a viabilidade de verificação dos requisitos operacionais e de performance. A vantagem da prototipação em um ambiente fechado é não permitir interferências indesejadas de outras atividades com aquelas relacionadas com projeto e implementação.

Os protótipos são desenvolvidos pelas Equipes de Subsistemas e não há necessidade que sejam definidos como produtos controlados.

A utilização deste Ambiente dentro da abordagem orientada a objeto é intensa uma vez que haverá prototipação de classes/objetos para um melhor conhecimento do domínio do problema e das responsabilidades do sistema; para se conhecer quais objetos e interações serão essenciais ao sistema; para verificar a existência de multitarefas ou de tarefas de tempo real (Coad e Yourdon, 1992,1993).

6.4 - Recursos Computacionais

Assim como na abordagem estruturada, basicamente o Esquema de Controle para a abordagem orientada a objeto também deverá utilizar pacotes de software que ofereçam facilidades para controle de versões, geração de configuração de sistema, para armazenar os registros de dependência entre elementos e para dar suporte a todas estas atividades.

Para o controle de versões de elementos, há necessidade que o pacote de software utilizado seja capaz de executar as seguintes atividades básicas: definir bibliotecas de controle, armazenar as versões dos elementos e permitir recuperação de qualquer destas versões, permitir que os elementos sejam alterados somente após uma operação de reserva realizada por usuários autorizados, manter registro histórico das alterações e dos processos de reserva/alteração/reposição dos elementos, e informar quais usuários utilizaram ou estão utilizando um determinado elemento. Em termos da arquitetura adotada na forma de ambientes de desenvolvimento, este pacote de software

deverá ser capaz também de controlar a distribuição das áreas de trabalho em termos de diretórios no caso de se trabalhar na estrutura de "mainframe" ou em termos de microcomputadores e "workstations" no caso de se trabalhar na estrutura de redes.

Para a geração de configuração de sistema, ou seja, para gerar as versões adequadas dos elementos que compõem o sistema, há necessidade que o pacote de software seja capaz de realizar as seguintes atividades básicas: manter a relação das versões dos elementos que compõem o sistema para a montagem de sua versão sempre da mesma maneira, prover facilidades para gerar e manter mais de uma versão do sistema, refletir automaticamente na montagem do sistema as alterações que forem implementadas em qualquer elemento controlado do sistema.

O banco de dados a ser utilizado é o do tipo relacional onde poderão ser armazenados todos os elementos de software bem como os relacionamentos entre eles. Desta forma as informações armazenadas neste banco de dados poderão ser compartilhadas com outras ferramentas dentro do ambiente de desenvolvimento, permitindo que estas ferramentas obtenham e manipulem estas informações sobre os elementos de software cadastrados, principalmente informações das classes com finalidades de controle e de reutilização. Quanto ao uso de outros tipos de banco de dados, como por exemplo, o banco de dados orientado a objeto, também há necessidade de se verificar as implicações, as vantagens e desvantagens da aplicação deste novo tipo de banco de dados em relação ao banco de dados relacional para este Esquema de Controle.

Dentro do Esquema de Controle proposto, o banco de dados para cadastro e para repositório das informações das classes e de outros produtos, itens e

relacionamentos, deverá possuir o seguinte conjunto de facilidades: produzir listagens diversas, como por exemplo, a relação de todos os produtos e itens relacionados com um dado produto; realizar o cruzamento de informações armazenadas, como por exemplo, para identificar todos os subsistemas/componentes que utilizam uma determinada classe; manter o "status" dos produtos e itens etc.

Para dar suporte a todas estas atividades também há necessidade do Sistema de Suporte ao Controle de Elementos (SSCE). Neste caso há necessidade de que o pacote de software seja capaz de fornecer a automatização dos procedimentos de controle possibilitando facilidades do tipo: troca de mensagens entre ambientes especializados, manutenção do registro das trocas de mensagens para fins de controle, transferência de produtos entre os ambientes.

6.5 - Banco de Dados de Elementos Controlados

A estrutura do Banco de Dados de Elementos Controlados (BDEC) é formada basicamente pela definição, para cada sistema a ser desenvolvido, dos Elementos de Software e dos relacionamentos entre eles que deverão ser controlados de forma que todos os registros armazenados neste repositório possam fornecer informações tanto para o gerenciamento e controle do desenvolvimento quanto para a reutilização dos elementos em outras aplicações ou outros projetos (Cunha, 1997).

As informações de relacionamentos e dependências entre Elementos de Software são úteis, principalmente, entre classes, pois, apesar de uma classe formar uma unidade de código, normalmente uma classe tem relações de dependência com outras classes, de forma que

geralmente não é uma única classe que é reutilizada, mas sim um grupo de classes (Gibbs et al., 1990).

Além disso, as informações de relacionamentos entre classes podem ajudar na detecção de inconsistência e de incompleteza, e ajudar na avaliação do impacto das alterações em outras classes.

6.5.1 - Elementos de Software Controlados

No seu aspecto de generalidade, o Esquema de Controle deverá gerenciar e controlar as classes que são os elementos de software básicos da abordagem orientada a objeto. Para isso haverá um repositório contendo as classes que serão desenvolvidas dentro de um projeto.

O sistema de banco de dados utilizado como repositório de classes permitirá que as informações de uma determinada classe possam ser armazenadas, consultadas, recuperadas, excluídas e, principalmente, possam ser compartilhadas por outras ferramentas do ambiente de desenvolvimento.

As informações relevantes que podem ser armazenadas sobre uma determinada classe são: métodos implementados (públicos ou privados, se a linguagem de programação permitir), autor da classe, versão ou atualização, aplicação da classe etc. Ao se consultar uma determinada classe pode se obter informações a respeito de quais métodos ela implementa, quem é o proprietário, quem está utilizando, qual aplicação etc.

TABELA 6.1 - EXEMPLO DE ELEMENTOS CONTROLADOS NA ABORDAGEM ORIENTADA A OBJETO

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

A Tabela 6.1 apresenta, além das classes, alguns exemplos de elementos de software que poderiam ser controlados dentro da metodologia OOA/OOD de Coad & Yourdon. Não é objetivo deste trabalho definir todos os tipos de produtos e de itens, nem todos os tipos de relacionamentos entre eles, mas sim, mostrar que este Esquema de Controle de Elementos de Software pode ser utilizado com sucesso para a abordagem orientada a objeto.

Em termos de tipos de Produtos pode-se ter:

- Classes: é o produto básico dentro da abordagem orientada a objeto. Deve-se ressaltar que aqui está se referindo à definição da classe que conterá atributos e métodos. O controle não abrangerá a entidade objeto devido à sua complexidade, pois os objetos são instâncias da classe que são criados durante a execução do software.

- CDP: este produto indica a função do software que está sendo desenvolvido, seja em termos de aplicativos (folha de pagamento, controle de estoque etc.) seja em termos de subsistemas que fazem parte de um sistema (conjunto de telemetria, conjunto de telecomando etc. que podem fazer parte de um sistema de controle de satélites).

- CIH, CGT e CGD: estes produtos são os componentes gerados durante o OOD, como já vistos anteriormente.

- CAS, CCO, CET, CAT e CSV: estes produtos correspondem às várias camadas definidas durante a OOA, e são basicamente documentos contendo os diagramas correspondentes às camadas já definidas anteriormente.

- DER, PIT, MUS, MIN e RLT: são basicamente documentos, cada qual com seu conteúdo correspondente.

Em termos de Itens, que são os elementos que fazem parte dos Produtos, também serão cadastrados no

BDEC a fim de manter o controle de seus desenvolvimentos. Os exemplos de Itens são:

- Métodos: são os métodos definidos dentro de cada classe.

- Dispositivo de Armazenamento: indica o nome do dispositivo onde está armazenada uma determinada classe, se num arquivo simples, num banco de dados relacional ou num banco de dados orientado a objeto.

- Tarefas: são as tarefas definidas dentro do CGT.

- Telas: são as interfaces com os usuários definidas dentro do CIH.

6.5.2 - Relacionamentos Controlados

Com relação aos relacionamentos entre classes é útil distinguir dois tipos de categorias. A primeira categoria contém os relacionamentos estruturais, deriváveis a partir do código fonte, como por exemplo, *Subclasse_de* ou *Herda_de*, *Instancia_de*, e *Depende_de*. A segunda categoria inclui os relacionamentos decorrentes da utilização de uma determinada metodologia orientada a objeto de desenvolvimento de software, como por exemplo, na abordagem de Coad & Yourdon, podem existir, entre o Componente Gerenciamento de Tarefas e uma classe, relacionamentos do tipo *usa*, *implementa* ou *define*.

O relacionamento *Subclasse_de* ou *Herda_de* é uma das características padrões das linguagens orientadas a objeto (Gibbs et al., 1990). Logo, este tipo de

relacionamento deve ser cadastrado dentro do BDEC de forma a se obter informações sobre quais classes são subclasses de outras. A herança simples é um relacionamento 1-n enquanto a herança múltipla seria um relacionamento n-n. Deve-se observar que dentro das linguagens orientadas a objeto a característica de herança não são iguais para todas elas, há variedades (Snyder, 1986). Portanto, durante a implementação do BDEC, há a necessidade de se verificar se a modelagem das variedades de herança está ou não definida para o BDEC.

O papel do relacionamento *Instancia_de* seria aplicado para algumas linguagens orientadas a objeto que contém metaclasses e neste caso as classes poderiam ser vistas como instâncias daquelas. Além disso este relacionamento pode ser utilizado na especificação de classes polimórficas que seriam modeladas como metaclasses, e as classes derivadas seriam instâncias destas metaclasses (Gibbs et al., 1990).

Quanto ao relacionamento *Depende_de* haveria vários modos de dependência entre classes. Uma classe pode ser um *Cliente_de*, ou seja, invoca os métodos de uma outra classe; uma classe pode ser *Parte_de* uma outra, quando uma classe tem instâncias de outras classes entre suas variáveis instanciadas. Em linguagens orientadas a objeto fortemente tipadas, uma classe pode depender de outra por declará-la como tipo de um parâmetro de método (Gibbs et al., 1990).

Estas dependências descritas acima são exemplos de aspectos gerais dentro do relacionamento *Depende_de*, identificando para um determinada classe, quais classes dependem dela e por sua vez, de quais ela depende. Estes relacionamentos são comuns a muitas linguagens orientadas a objetos, porém há outros relacionamentos que

são dependentes da linguagem. Por exemplo, o relacionamento "friend" em C++ (Stroustrup, 1986). Se uma classe declara uma segunda classe como sua "friend", então os métodos privados e variáveis instanciadas da primeira classe estão disponíveis para a segunda. Outros exemplos resultam da agregação de classes tais como as "features" (Kaiser e Garlan, 1987) ou como as "frameworks" (Wirfs-Brock e Johnson, 1990). Neste caso uma classe pode estar relacionada com outra através da participação na mesma "feature" ou "framework". Em geral as linguagens determinam alguns tipos específicos de relacionamentos de dependência que deverão estar previstos no BDEC.

Além destes relacionamentos decorrentes do código fonte, existem os relacionamentos específicos para a metodologia orientada a objeto que estiver sendo adotada, e que associam as classes com as documentações e/ou outras informações de projeto. No caso da Metodologia de Coad & Yourdon haveria os Componentes Domínio do Problema, Interface Humana, Gerenciamento de Tarefas, Gerenciamento de Dados que teriam relacionamentos com as classes e com outros elementos de software. E em termos de documentação, haveria os diagramas correspondentes às Camadas Assunto, Classe-&-Objeto, Estrutura, Atributo e Serviço que também teriam relacionamentos com classes e com outros elementos de software.

A seguir são apresentados alguns tipos de relacionamentos envolvendo alguns dos elementos apresentados na Tabela 6.1. Estes relacionamentos são definidos em termos de tabelas onde se tem o Agente Sujeito, dono da ação; o Agente Objeto, que sofre a ação; a ação; o papel do Sujeito e o papel do Objeto. Por exemplo: **Classe** (Agente Sujeito) **DEF** (ação) **Método** (Agente Objeto); papel do Sujeito: **define**; papel do Objeto: **é definido**.

Desta forma um Produto pode ser Agente Sujeito em um relacionamento, e em outro pode ser Agente Objeto.

Alguns tipos de relacionamentos são apresentados a seguir.

- Relacionamentos entre classes

  - {CLS} **SUB** {CLS} : Uma Classe é *subclasse_de* outra classe.

  - {CLS} **INS** {CLS} : Uma Classe é *instância_de* outra classe.

  - {CLS} **PAR** {CLS} : Uma Classe é *parte_de* outra classe.

  - {CLS} **CLI** {CLS} : Uma Classe é *cliente_de* outra classe.

- Relacionamentos com CDP

  - {DER} **GER** {CDP} : O Documento de Especificação gera o Componente Domínio do Problema, ou o CDP é gerado pelo DER.

  - {CDP} **CON** {CIH, CGD, CGT} : O CDP contém CIH, contém CGD, contém CGT; ou o CIH, CGD, CGT está contido no CDP.

- Relacionamentos com Classes

  - {TLA, TRF} **USA** {CLA} : A Tela usa a Classe, a Tarefa usa a Classe; ou a Classe é usada pela Tela, ou pela Tarefa.

  - {CLA} **IMP** {ATR, MTD} : A Classe implementa o atributo, implementa o método; ou o Atributo é implementado pela Classe, o Método é implementado pela Classe.

- Relacionamentos com Componentes

  - {CGT} **DEF** {TRF} : O CGT define a Tarefa; ou a Tarefa é definida pelo CGT.

  - {CGT} **USA** {TRF} : O CGT usa a Tarefa; ou a Tarefa é usada pelo CGT.

  - {CIH} **DEF** {TLA} : O CIH define a Tela; ou a Tela é definida pelo CIH.

  - {CGD] **USA** {DPA} : O CGD usa o Dispositivo de Armazenamento; ou o DPA é usado pelo CGD.

- Relacionamentos com Documentos

  - {PIT} **TST** {CDP} : O Plano de Integração e Teste testa o CDP; ou o CDP é testado pelo PIT.

  - {MUS, MIN} **PER** {CDP} : O Manual do Usuário pertence ao CDP, o Manual de Instalação pertence ao CDP; ou o CDP contém MUS, contém MIN.

Uma vez definida a tabela de relacionamentos, ela deverá ser respeitada durante o cadastramento de elementos e de relacionamentos.

Com o cadastramento dos relacionamentos fica definida a regra de dependência entre a maioria dos elementos do sistema, possibilitando fazer o levantamento de quais elementos deverão ser analisados, modificados ou recompilados caso seja feita uma alteração em um determinado elemento de software.

Além disso, estas informações armazenadas poderão ser úteis na reutilização de software, pois muitos elementos, como por exemplo, classes, telas e tarefas, poderão ser reutilizados para outras aplicações (Cunha, 1997; Souza, 1997; Ferreira, 1996).

Mais uma vez cabe observar que existem relacionamentos de dependência que são específicos às linguagens de programação, e que também deverão estar previstas no BDEC para que um Sistema de Controle deste tipo possa atender completamente a abordagem orientada a objeto.

6.6 - Sistema de Suporte ao Controle de Elementos

Basicamente o Sistema de Suporte ao Controle de Elementos (SSCE) deverá fornecer a capacidade de execução automática dos procedimentos para controle de elementos de software. Neste aspecto esta automatização levará em conta a arquitetura adotada em termos de ambientes de desenvolvimento, seja para uma arquitetura em "mainframe" ou para uma arquitetura em rede.

Assim como no Sistema de Controle para a abordagem estruturada, este sistema terá basicamente duas categorias de procedimentos: os utilizados para entrega de elementos e aqueles para alteração de elementos controlados.

Para haver um melhor controle sobre as alterações nos elementos controlados, principalmente, nas classes, está sendo implementada a figura do "proprietário da classe" conforme apresentado por Thomas e Johnson (1988), que descreve um tipo de organização da equipe de projeto para a abordagem orientada a objeto. Neste tipo de organização os analistas/programadores são divididos em duas categorias, desenvolvendo diferentes papéis: os programadores de classe e os programadores de aplicação. Eles assumem os papéis de criadores e consumidores de classes, respectivamente.

Os programadores de classe são responsáveis pela produção de classes que tenham um grande potencial de serem reutilizados, possibilitando a construção de uma biblioteca de classes para a organização na qual esta equipe trabalha.

Os programadores de aplicação procuram sempre reutilizar, tanto quanto possível, os elementos existentes na biblioteca de classes. O objetivo é configurar as classes existentes acrescentando código para sua aplicação específica, e ao mesmo tempo sugerir aperfeiçoamentos às classes existentes.

Para que esse tipo de organização funcione, há necessidade da definição de quem é responsável por determinada classe, para que somente ele tenha permissão de alterá-la.

Nesta proposta de Sistema de Controle pode-se definir como classes fundamentais, passíveis de serem reutilizadas, as do componentes CIH, CGD e CGT.

Portanto, dentro destes procedimentos de controle também haveria a necessidade também de definir e verificar as prioridades de cada usuário, estabelecendo quem é o responsável por cada classe.

6.6.1 - Encaminhamento de Elementos para Controle

Estes procedimentos se aplicam ao processo de entrega de elementos de software ao Ambiente de Controle de Configuração para que possam ser analisados quanto à qualidade e padronização, e, se aprovados, possam ser incluídos na categoria de elementos controlados. A Figura 6.4 apresenta um diagrama com o fluxo de encaminhamento dos elementos de software e de formulários padrões pelos diversos ambientes de desenvolvimento.

- ENTREGA DE ELEMENTOS

Assim que termina o desenvolvimento de um elemento de software, ele é entregue para o Controle de Configuração, utilizando o Formulário para Entrega de Produto (FEP), cujo conteúdo é uma lista de elementos de software, entre os quais classes, componentes, arquivos ou documentos.

Antes de se fazer a entrega, o elemento deverá estar inserido na respectiva biblioteca do desenvolvedor, as classes na Biblioteca de Classes, arquivos na Biblioteca de Arquivos e documentos na

Biblioteca de Documentos. Além disso todos os registros, inclusive de relacionamentos, relativos aos arquivos, aos componentes, às classes, e aos documentos que são usados, deverão estar cadastrados no BDEC.

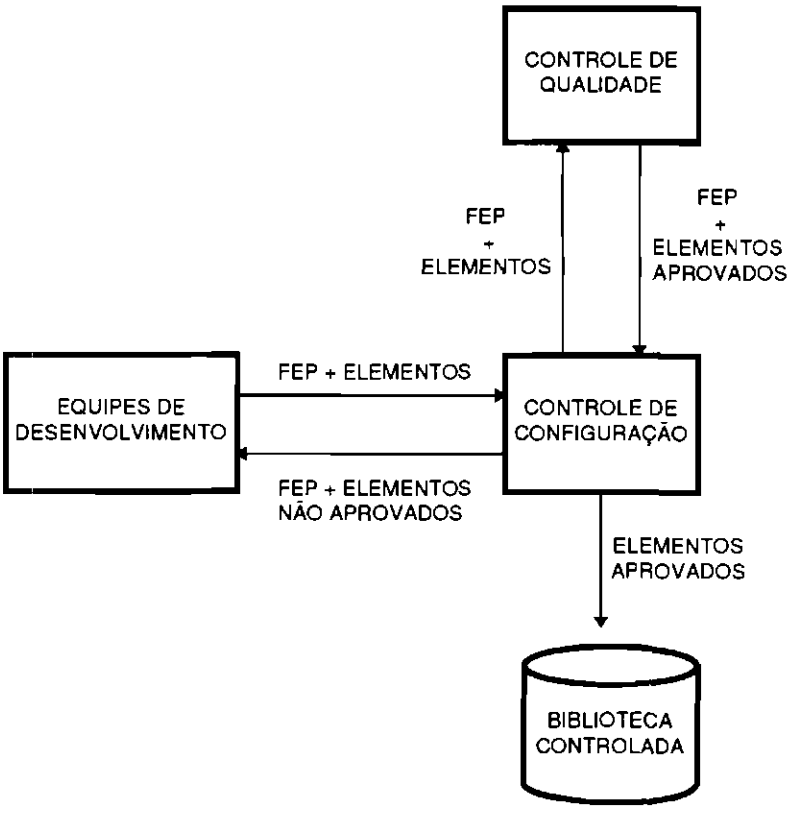

Fig. 6.4 - Processo de Entrega de Elementos de Software

- RECEBIMENTO DO FEP

O FEP é recebido e registrado no Mapa de Controle de FEP a fim de se iniciar o controle sobre o encaminhamento dos elementos de software. Estes elementos são reservados na biblioteca do desenvolvedor e transferidos para a área de trabalho do Ambiente de Controle de Configuração. Enquanto reservados, estes elementos encaminhados não poderão ser alterados pelos desenvolvedores.

- ANÁLISE DOS PRODUTOS

No Ambiente de Controle de Qualidade, os elementos encaminhados são analisados relativamente à consistência em relação à especificação de requisitos, completeza interna e externa, coerência dos relacionamentos cadastrados, obediência às normas e padrões.

Neste caso, para que seja possível esta análise, serão necessárias consultas ao BDEC de forma a obter as informações necessárias e inserir os registros da análise realizada.

- MANIPULAÇÃO DOS PRODUTOS ANALISADOS

Após a análise do FEP, os elementos aprovados serão transferidos para as respectivas bibliotecas controladas centrais de acordo com seus tipos. Os elementos não aprovados serão devolvidos ao seu local de origem.

Aprovados ou não os elementos são liberados nas bibliotecas controladas dos desenvolvedores com uma mensagem contendo o resultado da análise.

6.6.2 - Alteração de Elementos Controlados

Os procedimentos descritos a seguir são utilizados para a alteração de elementos controlados. A Figura 6.5 apresenta um diagrama com o fluxo de

encaminhamento dos elementos de software e dos formulários padrões pelos diversos ambientes de desenvolvimento.

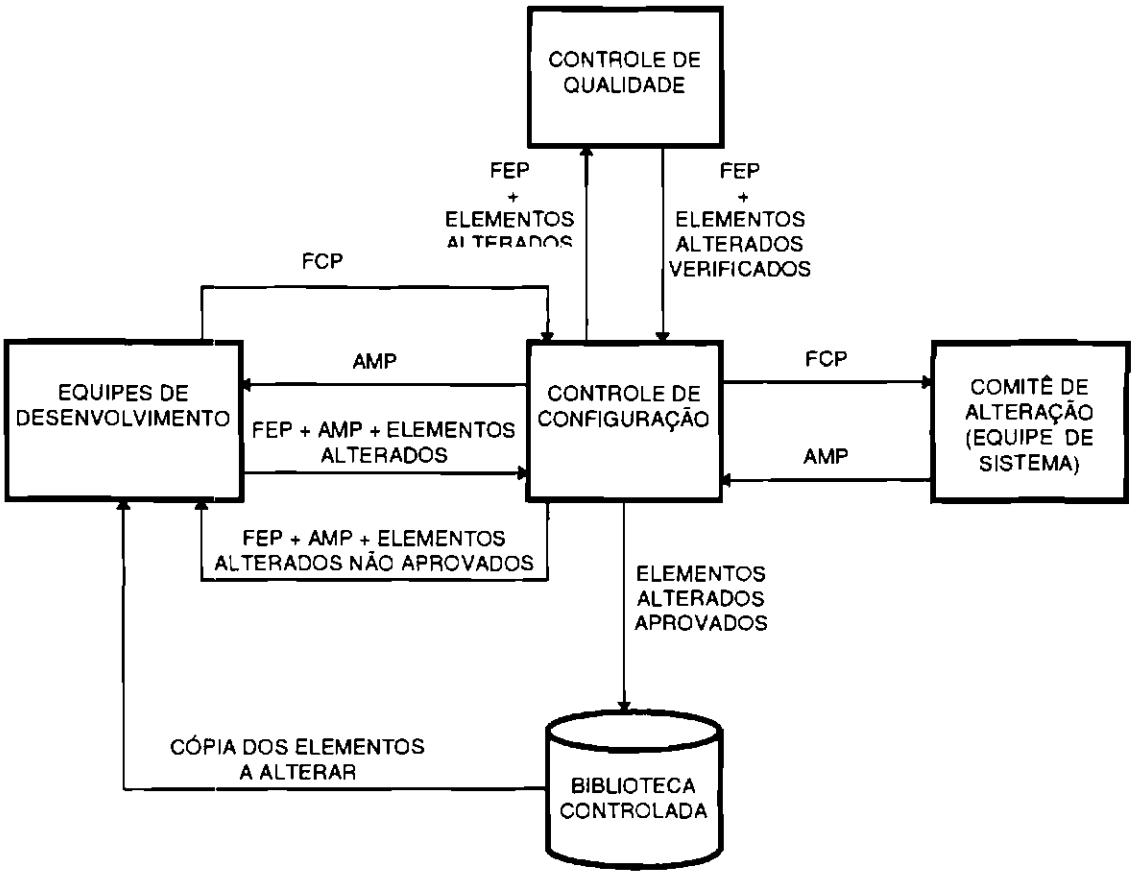

Fig. 6.5 - Processo de Alteração de Elementos de Software

- COMUNICAÇÃO DE PROBLEMA/MODIFICAÇÃO

As Equipes de Subsistemas devem comunicar suas necessidades de modificação seja em termos correção ou aperfeiçoamentos através do Formulário de Comunicação de Problemas (FCP) encaminhado para o Controle de Configuração, que irá mapear e gerenciar estes FCPs.

- AVALIAÇÃO DE FCP

Uma vez cadastrado o FCP, ele deve ser encaminhado para a Equipe de Sistema, onde um Comitê irá analisar o problema/alteração, verificando o contexto e a extensão do problema/alteração, dos elementos envolvidos, do custo e do impacto no desenvolvimento do produto como um todo. Após a análise, o problema/modificação receberá uma das seguintes condições: autorizado, postergado ou rejeitado. No caso de ser autorizado, será emitido o formulário denominado Autorização para Modificação de Produto (AMP) contendo a autorização e o que deverá ser alterado. A AMP também será cadastrada e gerenciada pela Equipe de Configuração.

- GERENCIAMENTO DAS AMPs

Uma vez cadastradas, as AMPs serão encaminhadas para as Equipes de Subsistemas que deverão executar a alteração autorizada e, após a execução desta tarefa, deverão encaminhar novamente os elementos alterados através de um FEP.

- ALTERAÇÃO DE ELEMENTOS

O responsável pela alteração dos elementos deverá fazer uma cópia dos mesmos a partir da Biblioteca Controlada Central, onde estão armazenados todos os elementos controlados, para a sua área de trabalho, obtendo assim a versão mais atual do elemento a ser alterado. Assim que os elementos forem alterados, serão encaminhados novamente para o Controle de Configuração.

- CONTROLE DE ELEMENTOS ALTERADOS

Os elementos alterados são analisados de forma similar aos elementos novos, com algumas modificações. Neste caso são verificadas as AMPs e os elementos correspondentes encaminhados, e se as alterações descritas nas AMPs foram realmente implementadas. Encerrada esta operação, os mapas de controle de AMPs e de FEPs deverão ser atualizados.

CAPÍTULO 7

CONCLUSÕES

Neste trabalho foram discutidos os principais aspectos relacionados com o gerenciamento de desenvolvimento de software, incluindo as suas diferentes abordagens e as dificuldades da sua execução. Discutiu-se também sobre o papel desempenhado pelo Gerenciamento de Configuração de Software (GCS), mostrando sua versatilidade não somente como uma ferramenta de gerenciamento mas também como uma ferramenta de suporte ao desenvolvedor.

Quanto ao GCS, foi feita uma discussão sobre sua evolução, desde a sua origem, até chegar nas suas mais modernas concepções caracterizando suas novas funcionalidades que foram implementadas em função do progresso das inúmeras ferramentas e ambientes disponíveis comercialmente. A implantação de um GCS em um processo de desenvolvimento de software, apesar de suas dificuldades, é vital para se obter uma maior produtividade e melhor qualidade de software, principalmente, em se tratando de grandes projetos.

A tecnologia pesquisada permitiu a identificação de uma ampla variedade de sistemas GCS, uma vasta teoria sobre o assunto e uma integração cada vez maior entre estes tipos de sistemas, permitindo que se tenha disponível os instrumentos mais adequados para gerenciar os grandes projetos de software. Cabe à cada organização, à cada equipe de trabalho pôr em prática o embasamento teórico que a área de GCS oferece, de forma a produzir software de melhor qualidade, pois somente a aquisição de um sistema GCS não é suficiente para se

alcançar a melhor qualidade e maior produtividade de software. Há necessidade de incutir no trabalho de cada participante do projeto o respeito à disciplina que um sistema de controle, tal como um sistema GCS, impõe ao ritmo de trabalho de cada desenvolvedor.

Atualmente os sistemas GCS oferecem uma interface muito amigável aos seus usuários, tornando o trabalho de controle e de gerenciamento muito mais rápido e mais eficiente, não tendo os desenvolvedores mais desculpas para reclamar que o sistema de configuração toma tempo e obstrui os seus trabalhos.

Procurou-se neste trabalho evidenciar que nenhum sistema GCS, ferramenta, ambiente ou metodologia de desenvolvimento garante o sucesso do software, mas certamente, sem a aplicação de quaisquer destes conceitos, ficará mais difícil de se obter um software de qualidade.

Neste trabalho também foi abordado o paradigma orientado a objeto, onde procurou-se discutir as conseqüências de seus conceitos nas atividades de controle de desenvolvimento, de teste e integração e de manutenção de software, e ao mesmo tempo buscando soluções para o gerenciamento de software dentro deste novo paradigma.

A apresentação neste trabalho do Sistema de Controle de Elementos de Software utilizado para o Sistema de Controle de Satélites (SICS), teve como objetivo não somente de servir de base para a proposta apresentada, mas também teve por finalidade resgatar a importância que o Sistema representou para o sucesso do desenvolvimento do SICS em termos de produtividade e de qualidade.

A proposta apresentada de um Esquema de Controle para a abordagem orientada a objeto é uma busca de

solução dentro deste paradigma, e tomou-se por base o sistema que foi implementado com sucesso para a abordagem estruturada. Apesar de concepções diferentes dentro de cada abordagem, o Esquema de Controle apresentado pode proporcionar meios para gerenciar e controlar os elementos de software orientados a objeto.

O Esquema proposto tem suas limitações, principalmente no aspecto de gerenciar as instâncias das classes, as ligações dinâmicas, as heranças múltiplas, enfim, características da própria metodologia orientada a objeto. De qualquer forma, em termos de controle da produção de classes, de programas fonte, de arquivos e de documentação, este esquema pode atender aos requisitos mínimos de um sistema GCS, que é controlar a configuração do sistema, os elementos produzidos ao longo do projeto, o processo de alteração dos elementos, e, conseqüentemente, controlar a qualidade e produtividade de software.

Uma avaliação completa deste Esquema apresentado somente será possível após sua implementação e sua utilização prática em algum desenvolvimento de software. A implementação de um protótipo deste Esquema é uma sugestão para trabalhos futuros, bem como seu aperfeiçoamento com a implementação de ferramentas, ambientes e banco de dados orientados a objeto.

De qualquer forma as idéias apresentadas, juntamente com os avanços que a tecnologia orientada a objeto vem alcançando nas diversas áreas da Engenharia de Software, permitirão que um esquema de controle deste gênero seja capaz de atender plenamente o gerenciamento e o controle de desenvolvimento de software orientado a objeto.

REFERÊNCIAS BIBLIOGRÁFICAS

Ambiola, V.; Bendix, L. Object-Oriented Configuration Control. **ACM SIGSOFT Software Engineering Notes**, 14(7):133-136, Nov. 1989.

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

Bersoff, E.H.; Henderson, V.D.; Siegel, S.G. **Software Configuration Management**. New Jersey, Prentice-Hall, 1980.

Boehm, B.W. Software risk management: Principles and practices. **IEEE Software**, 8(1):32-41, Jan. 1991.

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

Bounds, N.M.; Dart S.A. **Configuration Management (CM) Plans: The Beginning to Your CM Solution.** Pittsburgh, Software Engineering Institute, July 1993. 48 p. (Technical Report CMU/SEI-93-TR).

Brown, A.W.; Dart, S. A.; Feiler, P.H.; Wallnau, K.C. **The State of Automated Configuration Management**. Pittsburgh, Software Engineering Institute, Sept. 1991. 52 p. (Annual Technical Review).

Brown, A.W.; Feiler, P.H.; Wallnau, K.C. **Understanding Integration in a Software Development Environment**. Pittsburgh, Software Engineering Institute, Jan. 1992. 28 p. (Technical Report CMU/SEI-91-TR-31).

Christian, K.B.; Zucker, S.H. Automated Configuration Management on a DOD Satellite Ground System. **IEEE AES Magazine**, 10(11):10-15, Nov. 1986.

Coad, P.; Yourdon, E. **Análise Baseada em Objetos**. 2. ed. Rio de Janeiro, Campus, 1992.

_____ **Projeto Baseado em Objetos**. Rio de Janeiro, Campus, 1993.

Cunha, J.B.S. **Uma Abordagem de Desenvolvimento de Software para Melhoria da Qualidade e da Produtividade.** (Tese de Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1997.

Cunha, J.B.S.; Nakanishi, T. O Controle do Desenvolvimento e Manutenção de Sistemas de Software. In: Congresso Ibero Latino-Americano de Métodos Computacionais em Engenharia, 14., São Paulo, 1993. **Anais**. São Paulo, Instituto de Pesquisas Tecnológicas, 1993a, v. 2.

_____ Quality and Productivity: The Control of Software System Elements. In: Structural Optimization 93, Rio de Janeiro, 1993. **Proceedings of The World Congress on Optimal Design of Structural Systems**. Rio de Janeiro, 1993b, v. 2, p. 483-490.

Dart, S.A. **Spectrum of Functionality in Configuration Management Systems**. Pittsburgh, Software Engineering Institute, Dec. 1990. 37 p. (Technical Report CMU/SEI-90-TR-11).

_____ **The Past, Present, and Future of Configuration Management**. Pittsburgh, Software Engineering Institute, July 1992. 27 p. (Technical Report CMU/SEI-92-TR-8).

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

_____ **Software Configuration Management: Advances in Software Development Environments**. Pittsburgh, Carnegie Mellon University, Mar. 1990. 12 p. (Academic Report).

Feiler, P.H.; Smeaton, R. **Managing Development of Very Large Systems: Implications for Integrated Environment Architectures.** Pittsburgh, Software Engineering Institute, May 1988. 22 p. (Technical Report CMU/SEI-88-TR-11).

Feldman, S.I. MAKE - a program for maintaining computer programs. **Software - Practice Experience**, 9(3):225-265, Mar. 1979.

Ferreira, M.G.V. **Componente Gerenciador de Dados Configurável**. (Dissertação de Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1996. 182 p.

Gibbs, S.; Tsichritzis, D.; Casais, E. ; Niertrasz, O; Pintado, X. Class Management for Software Communities. **Communications of the ACM**, 33(9):91-102, Sept. 1990.

Gustavsson, A. Maintaining the Evolution of Software Objects in an Integrated Environment. **Software Engineering Notes**, 14(7):114-117, Nov. 1989.

Harter, R. Object-Oriented Software Configuration Management. **Dr. Dobb's Journal**, 16(10):36-46, Oct. 1991.

Henderson-Sellers, B.; Edwards, J.M. The Object-Oriented Systems Life Cycle. **Communication of the ACM**, 33(9):144-159, Sept. 1990.

Humprey, W. Characterizing the Software Process: A Maturity Framework. **IEEE Software**, 3(5):73-79, Mar. 1988.

Institute of Electrical and Electronics Engineers (IEEE). **Guide to Software Configuration Management**. New York, Sept. 1987. (IEEE/ANSI Standard 1042-1987).

Jacobson, I.; Christerson, M.; Jonsson, P.; Overdaard, G. **Object-Oriented Software Engineering.** Wokingham, Addison-Wesley Publishing Co., 1992.

Kaiser, G.E.; Garlan, D. MELDing data flow and object-oriented programming. In: Conference on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA'87, Orlando, Florida, 1987. **SIGPLAN Notices** 22(12):254-267, Dec. 1987.

Korson, T.; McGregor, J.D. Understanding Object-Oriented: A Unifying Paradigm. **Communications of the ACM.** 33(9): 40-60, Sept. 1990.

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

Kramer, J.; Magee, J.; Lister A. CONIC: An integrated approach to distributed computer control system, **Proc. Inst. Elec. Eng.**, 30(1), Jan. 1983.

Kung, D. et al. Developing an Object-Oriented Software Testing and Maintenance Environment. **Communications of The ACM**, 38(10):75-87, Oct. 1995.

Leblanc, R.J.; Maccabe, A.B. The design of a programming language based on connectivity networks. **Proceedings International Conference Distributed Computing Systems**, Florida, 1982.

Lejter, M.; Meyers, S.; Reiss, S.P. Support for Maintaining Object-Oriented Programs. **IEEE Transactions of Software Engineering**, 18(12):1045-1052, Dec. 1992.

Liskov, B.; Sheifler, R. Guardians and actions: Linguistic support for robust distributed programs. **ACM Transactions Programming Language System**, 5(3): 381-404, July 1983.

Madhavji, N.H. Environment Evolution: The Prism Model of Changes. **IEEE Transaction on Software Engineering**, 18(5):380-392, May 1992.

Mahler, A.; Lampen, A. Integrating Configuration Management into a Generic Environment. **SIGSOFT - Software Engineering Notes**, 15(6):229-237, Dec. 1990.

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

Mitchell, G.J.; Maybury, W.; Sweet R. **Mesa Language Manual Version 5.0.** Xerox Palo Alto Res. Center, Rep. CSL-79-3, Apr. 1979.

Monarchi, D.E.; Puhr, G. I. A Research Typology for Object-Oriented Analysis and Design. **Communication of ACM**, 35(9):35-47, Sept. 1992.

Narayanaswamy, K.; Scacchi, W. Maintain Configurations of Evolving Software Systems. **IEEE Transaction on Software Engineering**, 13(3):324-334, Mar. 1987.

Nerson, J. Applying Object-Oriented Analysis and Design. **Communications of the ACM**, 35(9):63-74, Sept. 1992.

Ousterhout, J.K.; Scelza, D.A.; Sindhu, P.S. Medusa: An experiment in distributed operating structure, **Communications ACM**, 23(2):92-104, Feb. 1980.

Prager, J.M. The project automated librarian. **IBM Systems Journal**, 22(3):214-228, Mar. 1983.

Rashid, R.; Robertson, G. Accent: A communication oriented network operating system kernel. In: **ACM Sigops Proceedings 8th Symposium Operating Systems Principles**, CA, Dec. 1981, p. 64-75.

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

Rook, P. Controlling software projects. In: Thayer, R.H., ed. **Tutorial: Software Engineering Project Management**. Los Alamitos, CA, IEEE, 1988, p. 108-117.

Rubin, K.S.; Goldberg, A. Object Behavior Analysis. **Communication of The ACM**, 35(9):63-74, Sept. 1992.

Sametinger, J. Schiffer, S. Design and Implementation Aspects of an Experimental C++ Programming Environment. **Software - Practice and Experience**, 25(2):111-128, Feb. 1995.

Schmerl, B. Configuration Management Support for Dynamically Bound Systems. In: **Proceedings JRCASE - Flinders University Collaborative Research Workshop on Software Engineering**, Victor Hargor, July 1994

Simpson, H.R.; Jackson K. Process synchronization in MASCOT. **The Computer Journal**, 22(4):332-345, Nov. 1979.

Snyder, A. Encapsulation and inheritance in object-oriented programming languages. In: Conference on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA'86, Portland, Oregon, 1986. **SIGPLAN Notices** 21(11):38-45, Nov. 1986.

Souza, G.J. **Interface Configurável**. (Dissertação de Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1997. 132 p.

Stroustrup, B. **The C++ Programming Language**. Wokingham, Addison-Wesley Publishing Co., 1986.

Tanenbaum, A.S; Mullender, S.J. An overview of the Amoeba distributed operating system. **ACM Operating System Revue**, 15(3):51-64, Mar. 1981.

Thomas, D.; Johnson K. Orwell - A Configuration Management System For Team Programming. **SIGPLAN Notices**, 23(11):135-141, Nov. 1988.

Tichy, W.F. RCS: A System for Version Control. **Software: Practice and Experience**, 15(7):637-654, July 1985.

Wegner, P. Dimensions of object-based language design. In: Conference on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA'87, Orlando, Florida, 1987. **SIGPLAN Notices** 22(12): 168-182, Dec. 1987.

Wilde, N.; Huitt, R. Maintenance Support for Object-Oriented Programs. **IEEE Transactions of Software Engineering**, 18(12):1038-1044, Dec. 1992.

Wilson, W.; Olsson R.A. An Approach to Genuine Dynamic Linking. **Software - Practice and Experience,** 21(4):375-390, Apr. 1991.

Wirfs-Brock, R.J.; Johnson, R.E. A survey of current research in object-oriented design. **Communications of the ACM**, 33(9):104-124, Sept. 1990.