UM SISTEMA DE GERÊNCIA COOPERATIVA DE
CONFIGURAÇÃO DE SOFTWARE

Ricardo Luiz Schneider

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

Aprovada por:

Prof Marcos Roberto da Silva Borges, Ph.D.

Profa. Cláudia Maria Lima Werner, D.Sc.

Profa. Vera Maria Benjamin Werneck, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

SETEMBRO DE 2001

SCHNEIDER, RICARDO LUIZ

Um Sistema de Gerência Cooperativa de Software [Rio de Janeiro] 2001

XIII, 117 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação. 2001)

Tese – Universidade Federal do Rio de Janeiro. COPPE

1. Gerência de Configuração de Software. 2. Engenharia de Software. 3. Engenharia de Sistemas

I. COPPE/UFRJ II. Título (série).

À LIGIA

AGRADECIMENTOS

Agradeço a todos os que contribuiram, direta ou indiretamente, para a realização deste trabalho. Minha mãe, meu pai de saudosa memória, meus filhos, irmãos, sogros, e toda a família, que foram privados de meu convívio mais intenso em função deste trabalho.

Gostaria de destacar a contribuição de algumas pessoas. O Prof. Marcos Borges pelo apoio e orientação e a paciência ao longo de todo este tempo que demorei para realizar este trabalho. Aos meus colegas do grupo CHORD, Renata, Flávia e todos os demais amigos que auxiliaram com o incentivo do contato constante.

Aos professores da COPPE que me ajudaram muito, como a Profa. Ana Regina e o Prof. Guilherme. Um agradecimento especial à Cláudia que sempre me apoiou desde o início e aceitou participar da banca. Também à Profa Vera que na última hora, se prontificou a ajudar, participando da banca. Agradeço, também, ao Prof José Carlos Maldonado pelas *dicas* do seminário de teses de João Pessoa e pelo interesse demonstrado em participar de alguma forma deste trabalho.

Um agradecimento muito especial ao Daniel, meu filho, e ao Cristiano Brêtas pela ajuda na implementação do protótipo da ferramenta, como parte do trabalho de final de curso deles.

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

UM SISTEMA DE GERÊNCIA COOPERATIVA DE CONFIGURAÇÃO DE SOFTWARE

Ricardo Luiz Schneider

Setembro/2001

Orientador: Marcos Roberto da Silva Borges

Programa: Engenharia de Sistemas e Computação

Este trabalho de tese apresenta a Gerência de Configuração de Software como parte fundamental do processo de desenvolvimento de sistemas com qualidade e produtividade. Ressalta os aspectos de colaboração e cooperação entre os participantes da equipe de trabalho. Propõe um método para gerenciar a configuração do software desenvolvido em harmonia com o administração de mudanças e de administração de projetos. Como forma de viabilizar a utilizacão do método, propõe uma ferramenta, e implementa um protótipo da mesma, com o objetivo de automatizar a gerência cooperativa de configuração de software, onde se destacam como suas principais características: o gerenciamento das revisões dos objetos de desenvolvimento, o gerenciamento de mudanças no processo de desenvolvimento e a obrigatoriedade de interação controlada entre os usuários da ferramenta.

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

A COOPERATIVE SOFTWARE CONFIGURATION MANAGEMENT SYSTEM

Ricardo Luiz Schneider

September/2001

Advisor: Marcos Roberto da Silva Borges

Department: System and Computation Engineering

This work of thesis presents the Configuration Management as a fundamental part of the system development process with quality and productivity. It highligths teamwork collaboration and cooperation aspects. It suggests a management method for software configuration developed in harmony with change request and project administration. In order to be able to use the method a tool is proposed, and a prototype is implemented with the purpose to automate the cooperative configuration software management, where the broad features are: management of software objects revisions, change management and the enforcement of controlled interaction among users of the tool.

LISTA DE SIGLAS

ABNT	- Associação Brasileira de Normas Técnicas
AC	- Agente de Comunicação
ANSI	- American Nacional Standards Intitute
API	- Application Programming Interface
CB-25	- Comitê Brasileiro de Qualidade
CMM	- Capability Mauturity Model
COPPE	- Coordenação dos Programas de Pós-graduação em Engenharia
CVS	- Concurrent Versions System
DoD	- Department of Defense
EIA	- Eletrical Industries Alliance
EUA	- Estados Unidos da América
GCCS	- Gerência Cooperativa de Configuração de Software
GCS	- Gerência de Configuração de Software
HTML	- Hyper Text Markup Language
IEEE	- Institute of Eletrical and Eletronic Engineers
IP	- Internet Protocol
ISO	- International Standards Institute
SCM	- Software Configuration Management
SEI	- Software Engineering Institute
SQA	- Software Quality Assurance
UCM	- Unified Change Management
UFRJ	- Universidade Federal do Rio de Janeiro
VOB	- Versioned Object Base
WfMC	- Workflow Management Coalition
WWW	- World Wide Web
XML	- eXtensible Markup Language

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

Um dos maiores problemas da Engenharia de Software é a dificuldade de lidar com as mudanças que ocorrem durante todo o ciclo de vida dos sistemas. A busca de solução tem gerado muitas pesquisas e modelos alternativos. O ciclo de vida clássico (modelo em cascata), herdado das engenharias mais convencionais, tem se mostrado pouco eficaz na administração das mudanças que ocorrem durante o desenvolvimento e manutenção de produtos de software. Vários outros modelos de ciclo de vida, como o desenvolvimento por protótipos, ciclo de vida em espiral, desenvolvimento incremental e linguagens de quarta geração, são propostos na literatura (PRESSMAN, 2001) com o objetivo de minimizar o impacto das mudanças no resultado final do processo de desenvolvimento de sistemas, tentando diminuir o risco de insucesso nesses projetos.

Neste contexto de incertezas e mudanças, conseguir a Garantia de Qualidade dos Produtos de Software passa a ser um dos objetivos permanentes e fundamentais da Engenharia de Software (PRESSMAN, 2001). Pressman, na referência citada, apresenta seis áreas de atividades diretamente ligadas à Qualidade e que devem merecer a atenção continuada do Engenheiro de Software:

- Métodos de Engenharia de Software;
- Padrões e Procedimentos;
- Revisões Técnicas Formais;
- Medições e Métricas de Software;
- Gerência de Configuração de Software e Garantia de Qualidade de Software; e,
- Testes.

Destas atividades, a Gerência de Configuração de Software – GCS - representa um papel central em relação às demais, agindo como uma atividade *guarda-chuva*.

Revisões de Software e Testes podem estar vinculadas diretamente ao processo de Gerência de Configuração. Os Padrões e Procedimentos serão automaticamente reforçados em presença dos controles previstos na Gerência de Configuração. Medições e Métricas, principalmente as relativas aos componentes de tamanho e complexidade, podem ser incorporadas facilmente à Gerência de Configuração. E as demais atividades podem se beneficiar de um efetivo controle do processo de Gerência de Configuração.

O modelo CMM – Capability Maturity Model, do SEI – Software Engineering Institute da Carnegie Mellon University, de avaliação da capacitação das organizações em desenvolver software com qualidade, coloca já no nível 2 - Repetível, dos cinco níveis de maturidade previstos, a exigência da aplicação de técnicas de GCS. Isto demonstra a importância que este modelo atribui ao gerenciamento de configuração de software.

Por outro lado, a tendência universal de horizontalizar e diminuir o tamanho das organizações administrativas, em geral, e das equipes de projeto de software, em particular, têm levado à formação de equipes de desenvolvimento com maior poder de decisão (*empowered teams*) (KHOSHAFIAN, BUCKIEWICZ, 1995). Estas equipes se colocam cada vez mais dispersas geograficamente, formando equipes virtuais em que a palavra chave de atuação é Cooperação.

Pela análise das principais ferramentas existentes de GCS, constatamos uma carência de características de cooperação ou colaboração na dinâmica operacional das mesmas, geralmente centradas na atuação coordenadora da gerência de projeto. O *Code Co-op* (RELIABLE, 2000) é um exemplo de software em que estão incluídos alguns aspectos de cooperação, mas basicamente, em relação à distribuição dos documentos. Já experiências mais avançadas de colaboração entre os desenvolvedores, foram propostas e utilizadas em grupos distribuídos de desenvolvimento, como no projeto Apache (FIELDING, 1999). Neste caso, no entanto, o nível adequado de cooperação foi obtido por mecanismos organizacionais convencionais, sem o apoio de ferramentas de compulsão.

Algumas poucas ferramentas, como o Clear Case (WHITE, 2000) e o StarTeam, possuem mecanismos de cooperação embutidos em seus procedimentos. Mas isto é a exceção, não a regra.

1.1 Os Problemas a resolver

Neste contexto, detectamos alguns problemas que merecem o esforço de um trabalho de dissertação de mestrado, para alcançar uma solução razoável para os mesmos.

Consideramos os seguintes problemas básicos a serem analisados, equacionados e, tentativamente, resolvidos neste trabalho:

- A abordagem demasiadamente gerencial do processo de Gerência de Configuração convencional;
- A resistência a controles administrativos em geral, dos profissionais de sistemas;
- A falta de *amigabilidade* das ferramentas de trabalho dos analistas e programadores;
- A baixa conexão semântica entre os produtos gerados pelo processo de desenvolvimento, gerando documentos e componentes pouco relacionados entre si;
- O baixo nível de cooperação nos trabalhos de desenvolvimento de sistemas, gerando propostas isoladas e de baixa aderência às necessidades dos usuários e clientes.

Na Gerência de Software mais convencional o papel chave é exercido pela gerência de projeto. As próprias ferramentas de GCS são enfáticas em seus mecanismos de controle, em procedimentos de *check in* ou *check out*, por exemplo, na prescrição da execução de determinadas tarefas, não dando muita margem a um trabalho mais aberto e

colaborativo. Em equipes mais amplas, de pesquisa em projetos distribuídos e inovadores, há a necessidade de um grau maior de paralelismo nas decisões de projeto, mesmo que ao custo de um maior esforço posterior de coordenação e sincronização. O desenvolvimento de sistemas abertos para a Internet é cada vez mais freqüente.

A resistência a planejamento e controles é uma constante, em particular no Brasil, com sua cultura tipicamente latina. Ao contrário das culturas anglo-saxônicas, ou até mesmo de origem asiática, a índole do brasileiro é bastante refratária a controles, o que torna difícil a introdução de métodos ou processos altamente controlados e centralizados.

Em geral, as ferramentas de Gerência de Configuração apresentam interface bastante "*dura*" com os usuários, pois suas origens remontam aos tempos das interfaces a caracter. Modernamente, têm sido feitos esforços no sentido de adaptá-las a uma interatividade maior, principalmente no lado cliente da interação. Na interação com o CVS (CEDERQVIST, 1993), por exemplo várias ferramentas clientes foram construídas, seja usando Java, C++, ou outras plataformas similares (MERANT, 2000, MICROSOFT, 1996). No entanto, ainda há espaço para melhorar diversos aspectos em relação a este ponto. Em especial, as ferramentas de Gerência de Configuração devem incorporar mecanismos de cooperação nas suas interfaces, como por exemplo, informações sobre a situação dos colegas de trabalho, no tempo e no espaço (*awareness*).

Outro problema crítico das ferramentas de Gerência de Configuração se refere aos tipos de objetos a serem guardados nos seus bancos de dados. Como a diversidade é muito grande, as ferramentas deixam por conta do usuário a definição do *quê* será armazenado. Em geral, é feita uma distinção simplista entre módulos de textos ou de scripts e módulos binários. Ficando por conta dos usuários identificar e administrar a semântica mais complexa do inter-relacionamento entre os objetos armazenados.

Acreditamos que a inclusão de um modelo de definição semântica mais aprimorada sobre os objetos e seus relacionamentos seja fundamental para auxiliar os

usuários na recuperação e administração destes objetos. Mesmo correndo o risco de perder um pouco nas características de generalidade dos assuntos a serem armazenados nos bancos de dados da ferramenta.

Finalmente, o problema central a ser resolvido, de acordo com a nossa visão da Gerência de Configuração, é a falta de mecanismos adequados para garantir ou, no mínimo, permitir a cooperação adequada entre os membros das equipes de trabalho. A cooperação, na grande maioria das ferramentas atuais, é deixada totalmente por conta da iniciativa dos membros das equipes de trabalho. Existem relatos de experimentos bem sucedidos de cooperação no uso de ferramentas de Gerência de Configuração, como o Projeto Apache, relatado por FIELDING (1999), mas os mecanismos administrativos para garantir a cooperação foram *manuais*, ou seja, as pessoas foram compelidas a cooperar sem nenhuma obrigatoriedade (*enforcement*) garantida pela ferramenta.

A seguir apresentaremos algumas idéias preliminares sobre as soluções propostas, como forma de consolidar os requisitos maiores do trabalho a ser desenvolvido.

1.2 Encaminhamento de uma solução

Para tentar solucionar alguns dos problemas levantados, foi feita uma análise dos principais produtos de Gerência de Configuração, avaliando seus pontos positivos e negativos em relação aos problemas detectados.

Um denominador comum nos problemas examinados, é o baixo nível de cooperação "*embutido*" nos sistemas e ferramentas de GCS. Evidentemente podemos alegar que a cooperação, ou a colaboração num sentido mais amplo, sempre pode ser obtida por mecanismos administrativos ou gerenciais, colocados à parte do sistema de gerenciamento de configuração em si. No entanto, nossa proposta básica é a de embutir estes mecanismos como uma obrigatoriedade (*enforcement*) da ferramenta. Tendo o

nível gerencial, ou de coordenação dos trabalhos das equipes, um papel de dosar o grau de *cooperação* a ser aplicado em cada projeto específico.

A ênfase exagerada em mecanismos gerenciais, nas ferramentas tradicionais, pode ser minimizada pela introdução de mecanismos de cooperação e colaboração, que reduziriam esta ênfase. Por exemplo, se ao invés do gerente do projeto receber, ou ter acesso fácil, às informações sobre o andamento das tarefas alocadas a cada membro da equipe, tornarmos estas mesmas informações disponíveis para todos os membros da equipe sob o enfoque de "ciência" das ações (*awareness*), teremos uma solução melhor de colaboração substituindo controle.

O simples conhecimento do que cada colega está fazendo, em relação ao objetivo do projeto, exerce um poder de coerção muito forte. E, simetricamente, o que fizermos irá motivar os nossos colegas. Portanto, mostrar o andamento dos trabalhos ativará o espírito de cooperação. A menos, é claro, que o projeto esteja com problemas de qualquer natureza que o torne inviável. Neste caso a falta de motivação se tornará tão aparente que o encerramento do projeto será considerado de forma totalmente natural. Ou, serão consideradas alterações estruturais que tornem o projeto novamente viável e motivador.

Para melhorar o nível de *amigabilidade* da interação dos membros da equipe, entre si e com as ferramentas de desenvolvimento, propomos uma integração forte entre as ferramentas, obrigando o seu uso de forma natural. Este objetivo talvez seja o mais difícil de ser obtido devido à diversidade de ferramentas utilizadas pelos analistas e programadores em seus trabalhos de desenvolvimento de sistemas. A proposta básica é de que da ferramenta de controle possamos passar às ferramentas de edição e retornar a ela, de forma automática. O processo de trabalho seria, assim, orientado ou conduzido pela ferramenta de GCS.

Um passo seguinte, na elaboração da proposta de solução, é a formulação de um modelo de integração entre os diversos documentos que compõem um projeto de desenvolvimento de sistemas. Uma rede semântica, formada pela conexão destes

documentos, permitirá identificar ligações que facilitarão o gerenciamento das mudanças ocorridas ao longo das fases e transversalmente às mesmas.

1.3 Objetivos a atingir

O objetivo maior deste trabalho é o de propor um método para a Gerência Cooperativa de Configuração de Software entre os membros das equipes de projeto de sistemas.

Para atingir este objetivo se faz necessário rever os métodos e as ferramentas existentes, comparar as soluções propostas, selecionar as melhores alternativas, agregar novas idéias e, finalmente, desenvolver uma nova proposta.

Detectamos, por uma análise inicial, e discussão das idéias desta proposta de trabalho em diversas ocasiões, como, por exemplo, no V WTES – V Workshop de Teses em Engenharia de Software (SCHNEIDER, BORGES, 2000), que a ênfase deverá ser no uso da metodologia pelos analistas e programadores, ou seja, no componente *cliente* da ferramenta de implementação, que é onde vamos encontrar as melhores oportunidades de melhoria em relação à cooperação entre os membros das equipes de desenvolvimento.

O método de trabalho em Gerência de Configuração, bem como a ferramenta desenvolvida para dar assistência ao mesmo, deve atender a diversos ambientes e processos de desenvolvimento de sistemas. Prevemos seu uso para ambientes mais convencionais, como desenvolvimento usando C, C++, PASCAL, ambientes de desenvolvimento RAD, usando DELPHI, Visual Basic, e, mesmo, ambientes dinâmicos como desenvolvimento para WEB, Java, etc.

As características básicas das equipes a serem atendidas pelo método incluem dispersão geográfica dos seus membros e um baixo grau de hierarquização na

estrutura de comando. Tipicamente, equipes de projeto aberto de software. No entanto, deve ser possível utilizar a metodologia, com ajustes em seus parâmetros, em situações mais convencionais, como desenvolvimento por equipes concentradas e com comando central de sua atuação.

1.4 Organização do trabalho de Tese

No capítulo 2, o trabalho de tese começa apresentando os princípios e o funcionamento da Gerência de Configuração. Um breve histórico da área serve para encaminhar as principais características desta importante área da Engenharia de Software. São apresentadas as funções operacionais, as técnicas e as necessidades de ferramentas para uma boa execução.

A seguir, no capítulo 3, é realizado um levantamento das principais ferramentas existentes de Gerência de Configuração, e, ao mesmo tempo, é feita uma análise comparativa das suas características. As ferramentas analisadas foram escolhidas em função da importância no mercado e das melhores características técnicas. Foram pesquisadas as ferramentas com ênfase especial nos atributos de cooperação embutidos.

No capítulo 4 são apresentados os conceitos importantes da área de Cooperação, Workflow e Groupware. Em especial são analisados os aspectos relevantes destes conceitos para a área de Gerência de Configuração.

No capítulo 5 é apresentada a parte central do trabalho de tese. É discutida e apresentada uma solução para os problemas levantados. Cada detalhe, considerado relevante, será examinado e, depois, sugerida uma abordagem que resolva ou melhore o desempenho das equipes de trabalho.

No capítulo seguinte a ferramenta será especificada e projetada, definindo uma solução genérica que permita diversas implementações, em nível crescente de completeza.

No Capítulo 7 serão colocadas as conclusões do trabalho, ressaltando os pontos positivos alcançados e, eventualmente, as partes ainda incompletas, com recomendações para futuros trabalhos de complementação.

Complementando o trabalho, serão apresentados a Bibliografia, no Capítulo 8, e no Apêndice A, um exemplo de utilização do protótipo da ferramenta a ser desenvolvida.

Capítulo 2

O Processo de Gerência de Configuração

A Gerência de Configuração de Software é uma técnica bastante antiga na área de Software. BUCKLE (1982) reporta que em 1973 Barry Bohem, na TRW, já publicava manuais de *Desenvolvimento e Gerência de Configuração*. Em grandes projetos de sistemas da área de defesa dos EUA sempre foi comum o uso de Gerência de Configuração de Software, conforme relata o próprio Buckle.

A origem destas técnicas vem das engenharias mais convencionais, como a Engenharia Mecânica, a Industrial e, mais recentemente, a Engenharia Elétrica e Eletrônica, onde o controle dos componentes e de suas *versões* é fundamental para garantir qualidade mínima aos equipamentos e sistemas construídos. Não se pode imaginar um novo projeto de um parafuso para um equipamento, sem que a porca correspondente também não tenha sido redesenhada. Ou, em um equipamento eletrônico, haja a mudança do sinal de saída de um circuito, sem que o circuito que recebe e processa esta saída, também não tenha sido ajustado à nova interface.

Mais recentemente, na área de Gerência de Configuração de Software, tem surgido necessidades especiais no gerenciamento de sistemas de acesso à Internet. Scripts de html, imagens, sons, vídeos e outros objetos são colocados em servidores de Web e são alterados com freqüência sem precedentes na história dos sistemas de software.

A palavra chave neste contexto é *mudança*. Segundo BABICH (1986):

"*... Gerência de Configuração é a arte de identificar, organizar e controlar as modificações do software em construção por uma equipe de programação. O objetivo é maximizar produtividade pela redução das falhas*".

Gerência de Configuração não deve ser confundida com a fase de Manutenção de Sistemas. As mudanças que se pretende controlar ocorrem durante todo o ciclo de vida de um sistema, inclusive, durante a manutenção. São mudanças de diversas origens, que oportunamente serão explicitadas, mas que ocorrem desde a especificação inicial até a fase de manutenção dos produtos já prontos.

No decorrer deste capítulo serão formalizados os conceitos que caracterizam o que é Gerência de Configuração de Software.

2.1 O que é Gerência de Configuração

Gerência de Configuração, segundo BUCKLE (1982), é um termo usado para designar um conjunto de técnicas que, quando aplicadas ao desenvolvimento e manutenção de software, melhorará a qualidade do produto de software, reduzirá os custos do ciclo de vida e melhorará a função gerencial no processo de desenvolvimento e produção. Gerência de Configuração está fortemente ligado, mas é diferente de técnicas de Garantia de Qualidade e Controle de Qualidade. Podemos visualizar na Figura 2.1, usando Pressman (2001), o relacionamento macro entre as técnicas envolvidas com qualidade.

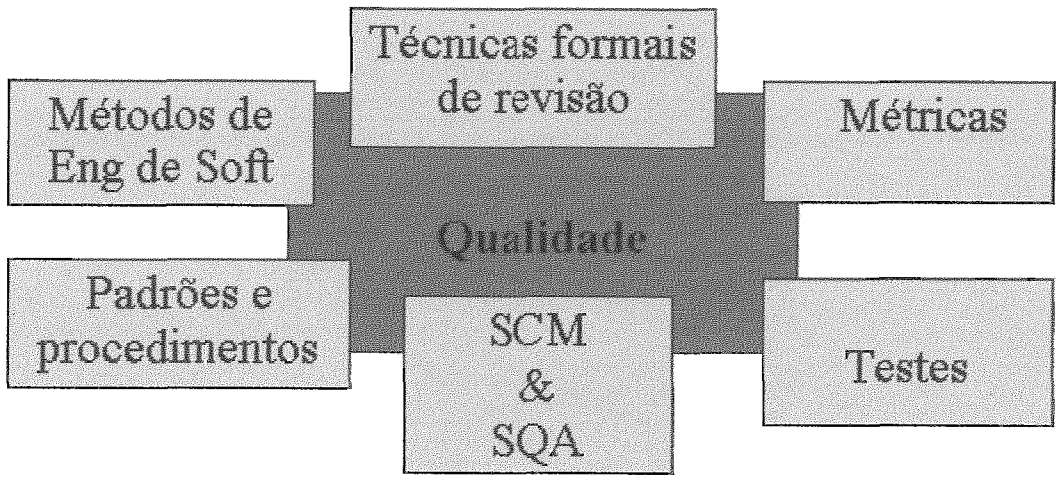

Figura 2.1 - Obtendo Qualidade de Software

Segundo este modelo, a qualidade será conseguida pela ação simultânea das seguintes técnicas:

- Métodos de Engenharia de Software
- Padrões e Procedimentos
- Revisões Técnicas Formais
- Medições e Métricas de Software
- Gerência de Configuração de Software e Garantia de Qualidade de Software
- Testes

Os Métodos de Engenharia de Software provêm as fundações sobre as quais a qualidade é construída. Métodos de análise, projeto e programação atuam na melhoria da qualidade, providenciando técnicas uniformes e resultados previsíveis. Técnicas de revisão formal, como *walkthroughs*, ajudam a garantir a qualidade de cada produto produzido como resultados dos passos da engenharia de software. Padrões e procedimentos ajudam a garantir uniformidade e a o processo de garantia de qualidade de software reforça uma filosofia de qualidade total. Testar o software é o último baluarte no qual a qualidade pode ser avaliada e os erros podem ser descobertos.

A Gerência de Configuração, conforme já foi dito anteriormente, desempenha um papel central neste processo todo, porque esta gerência permitirá que cada item de configuração de software seja especificado, projetado, construído, testado, avaliado, medido e controlado, garantindo a efetiva aplicação das demais técnicas.

Uma plataforma de Gerência de Configuração assenta-se em quatro bases bem definidas (BUCLE, 1982):

- **Identificação**: todos os itens componentes de um produto de software devem ser adequadamente especificados e identificados. A forma destes itens pode variar ao longo das fases e da vida do projeto, desde uma ata de reunião, ou outro documento bastante informal de especificação, até o código fonte implementado e formal;

- **Controle**: a habilidade de obter consenso sobre os objetos de software e "congelar" o estado dos mesmos, só fazendo alterações, a partir daí, com a aquiescência de uma autoridade competente; este controle de mudanças permite que todos os fatores relevantes e os efeitos possíveis da mudança sejam considerados antes de autorizá-la;

- **Acompanhamento do Status:** o registro e relato dos dados atuais e históricos de um item de configuração;

- **Verificação**: a série de revisões e auditorias para garantir que existe conformidade entre um produto e sua apropriada identificação.

As técnicas básicas são aplicáveis a projetos de todos os tamanhos, e em todos os estágios do desenvolvimento e da produção de software. Entretanto, a forma na qual são aplicadas pode diferir bastante com o tipo do projeto e o tamanho da equipe e com o estágio do ciclo de vida do produto.

Um programador profissional irá normalmente aplicar estas técnicas ao seu trabalho de uma maneira quase automática. O que queremos é conseguir uma maneira de aplicar disciplina semelhante em projetos envolvendo mais de uma pessoa.

2.2 Linhas Base de Configuração de Software

O conceito de linha base (*baseline*) ajuda a resolver um dos problemas mais críticos do gerenciamento do desenvolvimento e manutenção de produtos de software. Devido à baixa *visibilidade* (PRESSMAN, 2001) do software, conseqüência da falta de

características físicas como cor, peso, comprimento, largura, etc existe muita dificuldade de caracterizar o término do produto e a qualidade embutida no mesmo. A melhor forma que temos para contornar este problema é criar linhas base ou marcos de referência (*milestones*) que permitem a fixação de pontos de revisão para validação e verificação dos produtos gerados até então. Nestes momentos, serão realizadas comparações com os objetivos definidos e, então, avaliado o grau de obtenção dos mesmos.

Uma linha base é um ponto de referência no plano de trabalho, cuja obtenção pode ser demonstrada satisfatoriamente. Possui as seguintes funções interligadas:

- Um ponto de progresso mensurável;
- Uma base para o desenvolvimento e controle subseqüente;
- Um ponto de medida para avaliar a qualidade e a obtenção dos objetivos, antes de passar para a fase seguinte.

De acordo com um modelo de ciclo de vida clássico, de referência para o software a ser desenvolvido, podemos propor bases bem definidas em função do progresso a ser obtido desde a especificação até o sistema implementado. No início, temos quase somente documentos sendo gerados e, à medida que nos aproximamos das fases finais, o produto vai se materializando em módulos executáveis. Uma lista preliminar pode ser construída, então:

- Enunciado dos requisitos do sistema
- Especificação total do sistema
- Especificação detalhada do software

} Documentos

- Esboço do projeto de software
- Projeto detalhado do software
- Versão preliminar do sistema
- Versão da 1ª. liberação do sistema
- Versão da 2ª. liberação do sistema etc.

} Sistema atual

Para garantir que cada linha base sirva como uma plataforma sobre a qual vamos construir o sistema o final de uma fase e o início da próxima é marcado pela *aceitação* da linha base. A forma da aceitação pode variar ao longo das fases, mas o critério de aceitação será sempre o mesmo: *a nova linha base comporta as restrições impostas pela linha base anterior, e as linhas base subsequentes e o sistema final?* Em cada fase de especificação ou projeto, a aceitação da linha base será feita por uma revisão da documentação de projeto e, quando o projeto e implementação já se manifestarem em produtos, serão feitos testes de validação. É desta forma que existe uma ligação forte entre as técnicas de revisões e testes com a gerência de configuração.

A aceitação de uma linha base significa que o documento ou o código gerado deva ser *"congelado"* para formar uma fundação sólida para o próximo estágio de desenvolvimento ou operação, conforme apresentado esquematicamente na Figura 2.2. No entanto, como as atividades humanas não são perfeitas, existe sempre a possibilidade de mudar alguma coisa na especificação, projeto ou implementação. A sistemática de gerência de configuração deve permitir estas modificações, usando os mesmos cuidados e sob a mesma autoridade de aprovação, através do controle de mudanças, que será examinado mais adiante.

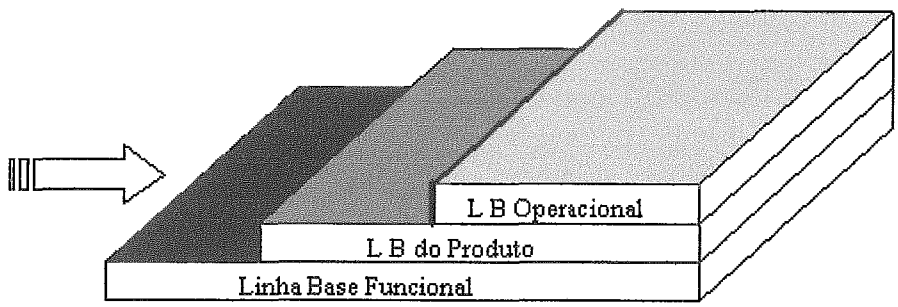

Figura 2.2 - Linhas base de GCS

A definição precisa e completa das linhas base vai depender de uma série de fatores, como a metodologia, o modelo de ciclo de vida utilizado e, em última instância,

de cada projeto em particular. Será uma decisão a ser definida dentro do processo de planejamento de projeto.

Um aspecto a ser ressaltado é a visão do processo de desenvolvimento de sistemas como uma aquisição de conhecimentos sobre o mesmo. Desde as etapas mais iniciais, em que buscamos definir o escopo e especificar as características relevantes do sistema em desenvolvimento, através de diversos modelos, até as fases posteriores em que estamos integrando módulos já construídos e testados para gerar o sistema final, estamos sempre incorporando conhecimento sobre o sistema, seu ambiente de funcionamento, os usuários, etc. Dentro desta visão, podemos imaginar as linhas base como plataformas que representem os níveis de conhecimento adquirido, do sistema em desenvolvimento, em relação ao grau de detalhe compatível com a fase em que o projeto se encontra. Imaginando um desenvolvimento de cima para baixo (*top-down*), quanto mais cedo for a fase, mais geral e abrangente será o conhecimento adquirido, quanto mais posterior, mais detalhado e profundo será este conhecimento.

Outra forma bastante interessante, apresentada por ARMOUR (2000), é ver o desenvolvimento de sistemas como a redução da ordem de ignorância que temos sobre o sistema e seu uso. Ele menciona cinco ordens, sucessivas e crescentes, de ignorância: *Falta de ignorância* – conheço alguma coisa e posso demonstrar que conheço; *Falta de conhecimento* – não conheço alguma coisa e posso identificar este fato; *Falta de consciência* (*awareness*) – não sei que não conheço alguma coisa; *Falta de processo* – eu não sei uma maneira eficiente de descobrir que eu não sei que eu não conheço alguma coisa; *Meta ignorância* – eu não sei que existem as cinco ordens de ignorância (agora ninguém mais, dentre eu e os leitores deste texto, possui a meta ignorância!). Neste caso, o processo pode ser visto como a construção de plataformas sucessivas de ignorância cada vez menor, e consequentemente conhecimento cada vez maior. Ou seja, supondo que eu já não tenha a meta ignorância, o caminho é encontrar um processo para adquirir consciência do que eu não conheço e, finalmente, adquirir este conhecimento.

2.3 Itens de Configuração de Software

O controle de qualquer sistema de software exige que o sistema, suas versões e partes componentes sejam unicamente identificados. Isto não é uma atividade burocrática ou administrativa. Considere o problema de encontrar um erro em um sistema se existem várias versões e não é conhecido qual apresentou o erro. Este é um problema eminentemente técnico.

A identificação da configuração propõe um meio de isolar os componentes do sistema como uma base para o controle. Existem três dimensões para isto: primeiro, o sistema deve ser dividido em um número conhecido de partes gerenciáveis; segundo, as partes devem nomeadas de forma única; por último, à medida que estas partes mudem com o tempo, as várias versões que aparecem também devem ser univocamente identificadas. A primeira dimensão está ligada diretamente ao processo de desenvolvimento: especificação, análise e projeto. E as outras duas requerem um rigoroso cumprimento de padrões.

Evidentemente, o sistema como um todo deve ter uma identificação e deve ser possível distinguir entre as diversas versões que podem existir. Estas versões podem ser passos para uma versão final, ou alternativas adaptadas para ambientes ou usuários diferentes. Ou seja, podemos ter duas versões diferentes porque a mais nova corrigiu um erro da mais antiga, ou duas versões referentes à mesma especificação funcional, só que uma delas em Português e a outra em Inglês, por exemplo. Será muito útil que, além da identificação geral do sistema como um todo, cada parte também seja identificada. Em especial quando os módulos pertencem a *bibliotecas* de uso geral, compartilhadas por outros sistemas.

Um problema menos óbvio, mas também importante, se refere à forma ou meio no qual cada item pode ser encontrado. Uma especificação pode estar em papel ou em arquivo de disco, num formato de texto simples ou texto formatado (*Rich Text*). Um módulo pode estar no formato de programa fonte ou executável. Devemos ser capazes

de saber se estas cópias são do mesmo objeto ou são versões diferentes e identificá-las corretamente.

As próprias ferramentas que são usadas para editar ou modelar os objetos de configuração podem mudar ao longo do tempo. Talvez tenhamos que identificar corretamente estas ferramentas e as versões das mesmas que são utilizadas. Compiladores, editores de diagramas, editores de texto, gerenciadores de bancos de dados, etc., são ferramentas que sofrem revisões ao longo do tempo. E, estando *embutidos* em nosso sistema, devem também sofrer o controle de configuração.

A identificação e o controle dos itens de configuração irão exigir, devido à complexidade inerente, que tenhamos um registro adequado dos itens e das inter-relações entre eles, através de um banco de dados dos mesmos.

2.4 Controle de Versões

Existem diversas formas de identificar e controlar as versões de sistemas e objetos, propostos na literatura.

A Free Software Foundation (CEDERQVIST, 1993), responsável pelo desenvolvimento da ferramenta de gerência de configuração CVS, propõe uma forma de identificar e controlar versões de acordo com os conceitos apresentados a seguir.

Um arquivo, parte integrante de um produto de software, pode apresentar várias versões. Neste sentido, a versão é chamada de **revisão**.

Um produto de software completo, por exemplo, o 'Windows 98 – 4.10.2222', é chamado de um lançamento ou **release**, em inglês. A palavra genérica **versão** não é formalmente usada, exatamente para não confundir com nenhum dos dois conceitos acima enunciados. A identificação do release é livre e utilizar um esquema simples tipo

'*<Versão_Principal>.<Secundária>.<Montagem>*' tem sido bastante utilizado. Esta é a estrutura usada pela Microsoft e por quase todas as *Software Houses*: três níveis hierárquicos de versões, numerados seqüencialmente e separados por pontos. Por exemplo: 'Windows 98 versão 4.10.2222', onde 'Windows 98' é o nome do produto, no caso, o sistema operacional; '4' é a versão Principal (*major*); '10' a versão Secundária (*minor*) e '2222' a Montagem (*build*) do produto de software. Toda vez que houver uma mudança significativa, que implique num processo de conversão de uma versão do sistema para a outra, o número Principal da versão será aumentado de uma unidade e os demais serão zerados. No caso do exemplo acima, passaria de 4.10.2222 para 5.0.0. Quando a mudança for de porte médio haverá mudança no segundo número (*minor*), 4.10.2222 para 4.11.0 no exemplo acima, ou quando forem pequenas no terceiro número, 4.10.2222 para 4.10.2223. Este número da liberação de um produto é controlado manualmente. Muitas vezes, pequenos acertos de um lançamento de um produto de software são qualificados por letras, como '5.0.5a'.

Cada revisão de um arquivo possui um **número de revisão**, que é constituído por um número par de inteiros separados por pontos ('.'), como por exemplo: '1.1', '1.3', '1.2.4.8' ou mesmo '1.3.2.5.3.7'. Por padrão, o primeiro número de revisão é '1.1'. A Figura 2.3 mostra uma seqüência de revisões de um arquivo. O último número da cadeia cresce seqüencialmente. Este número de revisão normalmente é assinalado de forma automática pelo sistema de controle de versões.

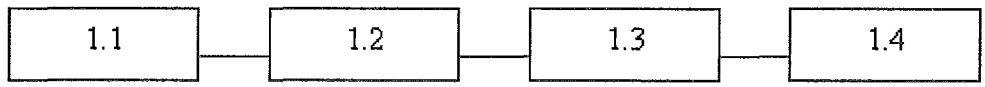

Figura 2.3 - Revisões de um arquivo

Quando um novo arquivo é incluído no diretório ele terá o prefixo igual ao maior prefixo existente e o sufixo será igual a um. Por exemplo, se os maiores números forem '1.7.3.1'e '4.12', o número de revisão de um arquivo a ser adicionado será '4.1'.

Em geral, a numeração das revisões é feita automaticamente pelo software (CVS ou outro similar). Mas se o usuário desejar identificar explicitamente uma determinada versão poderá colocar uma **etiqueta** (*tag*) em uma configuração de arquivos.

Para fazer o desenvolvimento em separado de algum arquivo, por exemplo, para corrigir um erro de um módulo sem interromper a linha principal, podemos abrir um **ramo** (*branch*) paralelo, conforme mostrado na Figura 2.4.

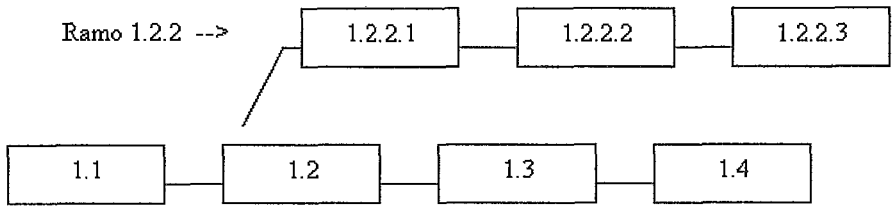

Figura 2.4 - Ramo de desenvolvimento

O **número do ramo** é formado pelo número da revisão do arquivo de onde foi derivado, concatenado com o inteiro par posterior ao último usado, iniciando em 2. Desta forma, podemos obter mais de um ramo de um mesmo arquivo inicial sem haver duplicações. No exemplo acima, o número do ramo será '1.2.2' e a primeira revisão '1.2.2.1'. Se houvesse mais um ramo, iniciando também em '1.2', sua primeira revisão seria '1.2.4.1'.

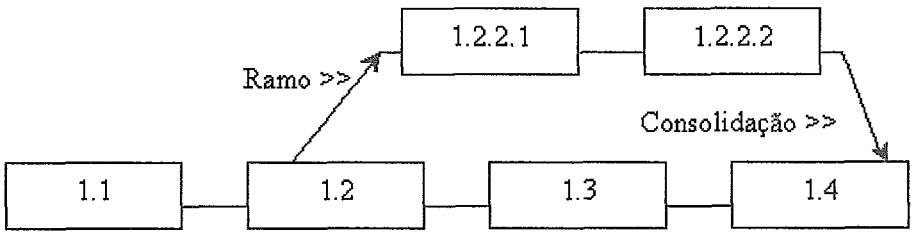

Figura 2.5 - Ramificação e Consolidação

Um ramo pode, após algumas revisões, voltar a se juntar ao tronco principal através de uma **consolidação** (*merge*), conforme ilustrado na Figura 2.5. As alterações deverão ser sincronizadas adequadamente.

Mecanismos simples podem ser criados para garantir o controle sobre as versões de arquivos em desenvolvimento. Quando um módulo é retirado do controle de versões para ser editado e modificado (***check out***) pode ser marcado com um cadeado (*lock*) que inibe alterações por outros programadores. Quando for retornado ao sistema com as alterações, será realizado o **check in**, retirando o cadeado. Este tipo de procedimento é bastante burocrático e torna o desenvolvimento enfileirado.

Existem alternativas que permitem as alterações em paralelo, por vários programadores, mas que vão exigir controles maiores de sincronização. O desenvolvimento de cada programador pode ser conduzido como um ramo separado e no final será realizada a consolidação destes diversos ramos. Nesta hora surgirão conflitos, como linhas alteradas com textos diferentes por cada programador, que deverão ser resolvidos explicitamente para permitir a inclusão no histórico do arquivo em edição.

2.5 O Controle de Mudanças

O controle, ou gerenciamento, de mudanças é o registro, acompanhamento e a reportagem das solicitações de mudanças de um sistema de software (WHITE, 2000). Inclui o processo de tomada de decisão sobre que mudanças realmente devem ser realizadas e o processo para executá-las. É essencial para a administração suave de um projeto. Envolve outras disciplinas do desenvolvimento de sistemas, como: gerência de requisitos, testes, gerenciamento da liberação de software, suporte ao usuário e gerência de projeto.

O registro de uma solicitação de mudança deve incluir informação sobre a origem e o impacto do problema, a solução proposta, e o seu custo e efeito sobre os prazos do projeto. O processo de solução depende de uma série de fatores. No entanto, as solicitações são tratadas, em geral, de acordo com duas principais categorias: solicitações de melhoria e correção de defeitos.

Uma **solicitação de melhoria** especifica uma nova característica ou uma mudança no comportamento projetado do sistema. **Defeitos** são anomalias ou falhas em um produto entregue. Enquanto grande parte dos dados mantidos para o controle de melhorias ou de defeitos é similar, estes dois tipos de solicitações são tratados de forma bem diferente no processo de controle de mudanças.

O processo de gerenciamento das solicitações de mudanças pode variar bastante com a abrangência do problema a ser tratado. Uma mudança significativa na especificação do sistema pode representar até a interrupção ou redefinição por completo do projeto. Para mudanças "*rotineiras*", no entanto, pode ser definido um modelo de transição de estados das solicitações de mudanças, definindo eventos e ações correspondentes a cada evento. Os estágios básicos deste processo são:

1. **Submissão**. Na submissão é feito o registro da solicitação de mudança. Defeitos e melhorias normalmente diferem bastante na origem da requisição e no tipo de informação coletada. Solicitações de melhoria chegam dos usuários, diretamente ou através do suporte ao usuário ou da comercialização. Os dados relevantes se referem à importância da requisição para o usuário, detalhes sobre a requisição e a identidade do solicitante original, para que se possa buscar mais detalhe, se for necessário. A maioria dos defeitos é encontrada, registrada e resolvida internamente. Os dados importantes são como o defeito foi encontrado, como reproduzir o defeito, a severidade, quem descobriu o defeito e a versão que o apresentou.

2. **Avaliação**. Durante a avaliação alguém deve olhar para todas as novas solicitações de mudança e definir o caráter de cada uma. É realmente um

defeito ou pode ser definido como uma melhoria? O nível de severidade assinalado é apropriado? Os defeitos podem ser reproduzidos? Qual a prioridade comparada com outras solicitações? É uma duplicata de outras solicitações? As solicitações de mudança devem ser priorizadas com base na severidade, importância, número de usuários afetados, importância dos usuários que fizeram o pedido, impacto no mercado, no faturamento e na equipe de suporte, etc.

3. **Decisão**. Deve ser decidido se a solicitação de mudança deve ser implementada, adiada ou descartada. Quase sempre, defeitos e melhorias são tratados diferentemente. A decisão em relação às melhorias leva em conta aspectos de mercado e competição, já os defeitos são atendidos com base na fase do ciclo de vida em que são descobertos e o esforço necessário para implementá-los. Bem no início do desenvolvimento, os defeitos podem ser tratados de maneira informal, porque o custo de se fazer alguma alteração é bem pequeno (PRESSMAN, 2000). Em fases posteriores o processo de decisão passa a ser bem mais formal incluindo mecanismos mais rígidos, como comitês de aprovação, por exemplo.

4. **Implementação**. Na implementação são criados e modificados módulos do sistema para satisfazer a solicitação de mudança. Tipicamente, as solicitações de melhorias vão exigir um esforço maior de análise e projeto e as correções de defeitos mais depuração, programação e testes. A documentação deve ser modificada correspondentemente e, quando for correção de defeitos, deve ser tomada a decisão de corrigir a documentação de forma integral ou emitir notas de liberação (*release notes*) do acerto executado.

5. **Verificação**. Testes finais e acertos na documentação são realizados na verificação. Quando se tratar de mudanças de melhoria, verificamos se atende às especificações da solicitação. No caso de defeitos, os testes devem garantir que os defeitos detectados tenham sido sanados e novos defeitos não foram introduzidos. Isto usualmente é feito tentando reproduzir os erros

originais numa nova liberação do software.

6. **Finalização**. É o fechamento da solicitação de mudança. Pode ser executado quando a mudança foi implementada ou quando foi descartada, por algum motivo. Uma boa prática é sempre fechar o ciclo com o solicitante original da mudança, principalmente se for um usuário externo.

As atividades componentes do processo de mudanças de software devem ser incorporadas ao ciclo de vida como um todo. Em MAIDANTCHICK (1999) é apresentada uma abordagem geral, baseada no modelo CMM, para o gerenciamento de processos de software para equipes geograficamente dispersas.

2.6 Verificação e Auditoria de Configuração

O processo de verificação e auditoria de configuração inclui:

- A verificação da configuração inicial dos itens de configuração e a incorporação das mudanças de engenharia aprovadas, para garantir a performance e os requisitos documentados;
- A auditoria da configuração dos registros de verificação de configuração e dos produtos físicos para validar que o programa de desenvolvimento obteve os seus requisitos de performance.

O término bem sucedido das atividades de verificação e auditoria resulta em um sistema ou item de configuração e um conjunto de documentação que pode ser, com confiança, considerado uma *linha base* do produto.

A verificação de configuração é um processo comum à gerência de configuração, engenharia de sistemas, engenharia de projeto, fabricação, e garantia de qualidade. É a forma como o construtor verifica a sua solução de projeto. Aspectos funcionais da verificação de configuração incluem todos os testes e demonstrações realizadas pra

garantir a qualidade da implementação das especificações. Os aspectos físicos verificam se a configuração "como construída" (*as-built*) está de acordo com a configuração projetada (*as-designed*).

Uma vez que a configuração inicial foi verificada, as mudanças aprovadas da configuração também devem ser verificadas. A verificação da mudança pode envolver uma auditoria detalhada, uma série de testes, uma validação de operações, manutenção, instalação, ou modificações de instruções, ou uma simples inspeção. A escolha do método adequado depende da natureza do item de configuração, da complexidade da mudança, e do conjunto de itens que a mudança afeta.

A auditoria de configuração é um processo complementar à verificação que garante de forma mais formal a qualidade dos produtos gerados. Enquanto a verificação normalmente é conduzida pela própria equipe de projeto e construção, a auditoria deve ser conduzida por pessoas externas ao projeto, sob os auspícios do contratante. Também a auditoria é dividida em auditoria funcional e auditoria física, de forma semelhante à verificação.

A auditoria funcional visa garantir que a performance atual dos itens de configuração atende os requisitos especificados. A auditoria física verifica se a configuração atual dos itens de configuração é representativa da configuração do produto. Valida também os processos usados pelo construtor no desenvolvimento dos itens.

O processo de auditoria, ao contrário da validação, não é utilizado ao longo de todo o desenvolvimento, mas é conduzido quando os produtos já estejam totalmente projetados, prontos ou iniciando a produção.

2.7 Padrões de Gerência de Configuração de Software

O Departamento de Defesa Americano (DoD) tem tomado iniciativas para definir padrões militares e governamentais para a área de gerência de configuração. A preocupação abrange equipamentos, software de uso geral, mas, principalmente, softwares embarcados em sistemas de armas, aeronaves, veículos militares, etc., por motivos óbvios.

A partir destas iniciativas, a indústria em geral tem se mobilizado para definir normas sobre a gerência de configuração. No Brasil, a ABNT emitiu algumas normas, em especial a NBRISO 10007, relacionadas ao assunto. A Tabela 2.1 consolida informações importantes sobre a normalização mais relevante.

A norma MIL STD 973 – Gerência de Configuração Militar, padrão militar emitido pelo Departamento de Defesa dos Estados Unidos da América (DoD), consolidou e tornou mais clara a informação contida em diversas normas mais antigas, como a MIL STD 480 e a MIL STD 483, na área de identificação de configuração e controle de mudanças. Apresenta padrões e recomendações para a execução das quatro atividades básicas da gerência de configuração: 1) Identificação e documentação das características físicas e funcionais dos itens de configuração; 2) Controle de mudanças dos itens de configuração e a documentação relacionada; 3) Registro e relato das informações necessárias para gerenciar os itens de configuração de forma efetiva; e, 4) Auditoria dos itens de configuração para verificar a conformidade com as especificações, projetos, documentos de controle de interface e outros requisitos de contrato.

Norma	Emissor	Assunto	Observações
MIL-STD-973 17/04/1992	DoD USA	Military Configuration Management	Cancelada em 30/09/2000
NBRISO10007 30/12/1996	ABNT/ISO	Gestão de qualidade – Diretrizes para a gestão de configuração	Tradução brasileira da ISO 10007 de 1995
MIL-STD-2549 30/07/1997	DoD USA	Military Configuration Management Data Interface	Cancelada em 30/09/2000

EIA-649 1998	ANSI/EIA	National Consensus Standard For Configuration Management	Adotada como padrão no DoD em 01/02/1999
IEEE 828-1998	IEEE	Standard for Software Configuration Management Plans	
DI-CMAN-81588 03/11/2000	DoD USA	Configuration Management Data Interface Transactions Data Information Packets	

Tabela 2.1 – Normas de Gerência de Configuração

A MIL STD 2549 – Gerência de Configuração de Interface de Dados, complementa e substitui a norma MIL STD 973, no que for conflitante. Define o padrão de gerência de configuração para os dados a serem manipulados, conforme será visto na seção 2.8.

Ambas as normas, MIL STD 973 e MIL STD 2549, foram canceladas em 30/09/2000 e substituídas pela EIA 649 – Padrão Nacional Consensual de Gerência de Configuração. Como o próprio nome diz, esta norma surgiu, no âmbito da EIA – Eletrical Industries Alliance com o aval do DoD, como o consenso entre a indústria e o governo norte-americano sobre gerência de configuração. Define gerência de configuração como: "o processo gerencial para estabelecer e manter consistência na performance de um produto, atributos funcionais e físicos com relação aos requisitos, projetos e informações operacionais ao longo de sua vida".

O IEEE – Institute of Eletrical and Eletronic Engineer, emitiu na mesma época da EIA a versão atualizada de sua norma IEEE 828-1998 "Standard for Software Configuration Management Plans". Esta norma define: Gerência de configuração de software constitui-se em boas práticas de engenharia para todos os projetos de software, para qualquer fase do desenvolvimento, prototipação rápida, ou manutenção em andamento. Ela aumenta a confiabilidade e qualidade do software através de:

- Providenciar estrutura para a identificação e controle da documentação, código, interfaces e bancos de dados para atender a todas as fases do ciclo de vida

- Dar suporte à metodologia de desenvolvimento e manutenção escolhida para atender os requisitos, padrões, políticas, organização e filosofia gerencial

- Produzir informações gerenciais e do produto com relação ao estado das linhas base, controle de mudanças, testes, liberações de software, auditorias, etc.

A norma NBRISO10007 – Gestão de Qualidade – Diretrizes para gestão de Configuração, emitida em novembro de 1996, pela ABNT – Associação Brasileira de Normas Técnicas, é uma tradução da Norma ISO 10007 de 1995. Esta norma trata de gestão de configuração de uma forma geral, vinculada ao CB-25 – Comitê Brasileiro da Qualidade. Define a gerência de configuração como uma disciplina de gestão que é aplicada ao longo do ciclo de vida de um produto, para prover visibilidade e controle de suas características físicas e funcionais. É aplicável ao suporte de empreendimentos[1], desde a concepção de produtos, passando pelo seu projeto, desenvolvimento, aquisição, produção, instalação, operação e manutenção, até a disposição após o uso.

A norma brasileira segue a orientação das normas internacionais e define o processo de gerência de configuração com as mesmas atividades básicas daquelas normas: identificação de configuração; controle de configuração; relato da situação de configuração e auditoria de configuração.

2.8 Gerência de Configuração de Dados e Documentos

Conforme relatado em LYON (2000), CHAFFREY (1998) e em outras referências sobre gerência de configuração, administração de documentos e trabalho em grupo, os conceitos e padrões de gerência de configuração se aplicam aos dados e documentos usados pelas organizações.

[1] Empreendimento foi usado na norma como tradução de "project" e projeto como tradução de "design".

CHAFFREY (1998) relata exemplos de organizações como o banco CERA da Bélgica, que administra manuais com 18.000 páginas contendo informações comerciais, legais e fiscais e com atualização semanal média de 150 páginas. Na Inglaterra, o Abbey National Bank executa cerca de 300 alterações por mês nas suas 2.000.000 de páginas de informações de produtos, procedimentos e manuais.

Todos estes documentos devem ser controlados com relação à correção e vigência, criando mecanismos de controle de versões e de mudanças totalmente similares aos controles de configuração de equipamentos, sistemas e softwares. Surge a necessidade de ferramentas de software especializadas nestes controles, os chamados EDMS – sigla em inglês de Software de Gerência de Documentos Eletrônicos. Estes produtos, além de executarem as funções regulares de um editor de texto, devem permitir a um grupo de pessoas:

- Realizar a autoria cooperativa da documentação;
- Revisar e anotar detalhes de revisão;
- Modificar a documentação;
- Publicar a documentação;
- Distribuir a documentação na organização;
- Modificar e repetir todo o ciclo.

Produtos como o Interleaf (http://www.interleaf.com), Documentum (http://www.documentum.com) e Filenet (http://www.filenet.com) permitem operar com a documentação em grande escala. Padrões são apresentados também nesta área, conforme já foi dito na seção anterior.

2.9 Estrutura de comparação dos ambientes de Gerência de Configuração

Uma estrutura de análise e comparação (*framework*) pode ser construída com o objetivo de explicitar os critérios mais ajustados aos objetivos deste trabalho.

No capítulo 3 esta estrutura será utilizada para comparar os produtos analisados e na conclusão (capítulo 7) servirá para avaliar o quanto atingimos os objetivos propostos.

A nossa proposta é considerar como itens mais relevantes para avaliar e para julgar os ambientes de GCS os seguintes:

- **Cooperação** no processo de GCS
- Percepção (***awareness***) do trabalho em equipe
- Apoio à equipe com **dispersão** geográfica
- Controle de **mudanças**
- **Semântica** enriquecida
- **Amigabilidade** nos controles de projeto

Capítulo 3

Descrição e Análise Comparativa dos Produtos de GCS

A gerência de configuração é praticamente impossível de ser executada sem a contribuição de ferramentas automatizadas (LYON, 2000, WHITE, 2000, PRESSMAN, 2001). O volume de dados necessários para uma boa administração dos itens de configuração exige o auxílio de sistemas especializados nestas funções.

Um conjunto de ferramentas foi examinado para que se pudesse avaliar o estado da arte nesta área e permitisse projetar uma proposta adequada para uma ferramenta nova. As principais características foram examinadas e comparadas para garantir uma boa decisão. Procurou-se mesclar ferramentas com apelo comercial com outras de cunho mais acadêmico, evitando uma influência maior num sentido ou noutro. Não houve a preocupação de se fazer uma pesquisa exaustiva, pegando todos os exemplares de sistemas de gerência de configuração, ou mesmo de todos os tipos existentes. Procurou-se tipo de ferramenta com boa aceitação no mercado, tanto comercial como acadêmico, e que tivessem características típicas de sua área de atuação.

Diversas ferramentas e ambientes estão sendo desenvolvidas em universidades nacionais e centros de pesquisa, em especial, no Programa de Engenharia de Sistemas e Computação da COPPE temos desenvolvido trabalhos nesta área (MURTA et al, 2000, TEIXEIRA et al., 2001).

Na Tabela 3.1 são apresentadas diversas ferramentas pesquisadas, com algumas de suas características. As principais ferramentas desta pesquisa serão detalhadas posteriormente, com a apresentação de seus aspectos mais relevantes para que se pudesse fazer uma comparação entre elas e concluir por suas melhores qualidades a serem incorporadas em uma proposta de solução.

Produto	Apresentação	Características relevantes
AccuRev	AccuRev Inc. – USA Windows, Unix http://www.accurev.com/i_prod.html	Cliente/Servidor Gerencia Processo
Alchemist	Sequel UK Limited – Inglaterra MVS, Unix, OS/2, Windows http://www.sequeluk.com/alchemist	
Aldon/CMS	Aldon Computer Group – USA AS/400 http://www.aldon.com/	
AllChange 2000 SE	Intasoft Ltd – Inglaterra Windows, Unix Gerência de Configuração e Mudança http://www.intasoft.net	LAN, FTP, E-mail Integração com MS-Explorer e Word Monitores de eventos
BitKeeper	BitMover, Inc. – USA Unix, Windows http://www.bitmover.com/bitkeeper	
ChangeMan eChangeMan	Serena Software – USA Mainframe, Unix, Windows http://www.serena.com/	Integração mainframe, desktop e Web Gerência de Mudança
CCC/Harvest	Computer Associates Int., Inc. – USA Servidor: Unix, Windows NT Cliente: Windows, Unix, OS/2 http://www.cai.com/products/ccc_harvest.htm	Integração com *mainframe* através do Endevor® for MVS Suporte a SAG Natural
Clear Case	Rational Corp - USA Unix, Windows http://www.rational.com/clearcase/	Gerência de Release Gerência de Configuração Gerência de Mudanças
CMVision	Expertware – USA Windows, Unix http://www_cmvision.com	Gerência de Release Gerência de Configuração Gerência de Mudanças
Code Co-op	Reliable Software - USA Windows http://www.relisoft.com	Ferramenta distribuida Controle de versões
ContinuusCM	Continuus Soft Corp. – USA Servidor: Unix, WinNT Cliente: Unix, WinNT, Win9x www.continuus.com	Workflow e cooperação Web
CVS	Free Software Foundation Servidor: Unix Clientes: Unix, Windows, Macintosh http://www.cyclic.com	Controle de versões
Enabler	Softlab – Alemanha Mainframe, Unix, Windows http://www.softlab.com	Gerência de Documentos Gerência de Software
Perforce	Perforce Software Todas as plataformas	
Plus-One (+1)	+1 Software Engineering – USA Sun/Solaris http://www.plus-one.com	
PVCS®	Merant – USA Windows, Unix http://www.merant.com/products/pvcs/	
Razor	Visible Systems Corp – USA	

	Windows, Unix http://www.razor.visible.com/	
RCS	Free Software Foundation Linux/Unix http://www.cyclic.com	Base de manipulação de arquivos versionados
SCCS	Free Software Foundation Linux/Unix http://www.cyclic.com	Usa o RCS
Sky	Vertical Sky – USA Windows, Unix http://www.verticalsky.com	Web
StarTeam	StarBase – USA Windows http://www.starbase.com	Cooperação Gerência de Configuração Gerência de Mudanças
TRUEchange	McCabe & Associates – USA Windows, Unix http://www.mccabe.com	
Visual Source Safe	Microsoft Co - USA Windows http://www.microsoft.com/ssafe	Ferramenta embutida nos pacotes de desenvolvimento da Microsoft
XStream	Prosoft Inc. – USA Windows, Unix http://www.prosoftcm.com/	

Tabela 3.1 – Produtos de Gerência de Configuração

3.1 ClearCase

O ClearCase (WHITE, 2000) é um produto desenvolvido por diversos pesquisadores, que após experiências iniciais na área acadêmica, fundaram a Rational Software Corp., empresa de software e consultoria dedicada a software Orientado a Objetos. O produto e o seu uso adequado se subordinam a um modelo gerencial chamado UCM – acrônimo de Gerência Unificada de Mudança (*Unified Change Management*), que define a filosofia de gerência de mudanças no desenvolvimento de sistemas de software, desde a fase de definição dos requisitos até a liberação do produto. O modelo UCM é concretizado através de um processo de gerência e de ferramentas. O ClearCase é a ferramenta usada para gerenciar os artefatos produzidos pelo projeto de software, incluindo os artefatos de sistema e os de gerência de projeto. A ferramenta chamada ClearQuest gerencia as requisições de tarefas de projeto, correção de defeitos e os pedidos de melhoria e provê os gráficos e relatórios necessários para o acompanhamento do progresso do projeto.

O modelo de trabalho de desenvolvimento usando o ClearCase é descrito, de forma sucinta, da seguinte maneira:

1. Os arquivos e diretórios de software são organizados em componentes *versionados* (cada componente pode ter várias versões diferentes)

2. Os gerentes criam projetos e designam equipes de projeto para trabalhar na construção dos componentes destes projetos

3. Os desenvolvedores realizam mudanças nos componentes, arquivos e diretórios baseados em atividades a que foram designados (requisições de tarefas, defeitos e mudanças)

4. Novas versões de arquivos e diretórios são criadas durante o desenvolvimento e associadas a atividades (conjuntos de mudanças)

5. Uma vez completadas, as atividades e os conjuntos de mudanças associados são entregues e integrados em uma área compartilhada de integração

6. Linhas base de novos componentes são criadas, testadas e promovidas

7. Linhas base são montadas em um sistema

8. Sistemas são testados e liberados

O modelo UCM, apresentado de forma gráfica na Figura 3.1, propõe papéis para os membros da equipe de desenvolvimento, que podem ser exercidos por uma ou mais pessoas ou mesmo uma pessoa acumulando mais de um papel, dependendo do porte e complexidade do projeto. Os papéis e suas responsabilidades são os seguintes:

- O **Arquiteto** – é o responsável pela definição do modelo de implementação e deve ser profundo conhecedor de arquitetura de sistemas. Determina como a

arquitetura do sistema será fisicamente realizada, ou seja, como os vários objetos de projeto serão agrupados e mapeados em arquivos e diretórios para implementar este projeto

- O **Gerente de Configuração** – deve ter familiaridade com os processos de gerência de configuração e gerência de mudança da organização e com as ferramentas utilizadas. É o responsável por criar e manter a infraestrutura física necessária para a implementação do projeto. Isto basicamente envolve a criação e manutenção de repositórios e a importação de arquivos e diretórios existentes

- O **Gerente de Projeto** – entende do processo de gerência de mudanças da organização e das políticas de gerência de projeto. Em termos de gerência de configuração, os gerentes de projetos são responsáveis por definir e programar tarefas de trabalho e assinalar componentes de software aos membros das equipes de projetos. Uma vez que os componentes de software estiverem definidos, o gerente de projeto é responsável por criar fisicamente o projeto no ClearCase e definir as políticas pelas quais o projeto será governado. Rotineiramente, fará o monitoramento do estado das tarefas de projeto, realimentando todo o processo

- O **Desenvolvedor** – recebe tarefas assinaladas pelo gerente ou formuladas por si próprio, dependendo do grau de formalidade da organização, para serem executadas, alterando arquivos e diretórios e entregando o resultado ao integrador de sistema. Um desenvolvedor é qualquer um que crie e modifique arquivos e diretórios no repositório de gerência de configuração: o gerente modificando um plano de projeto, o documentador modificando um manual, um testador modificando algum script de teste, o analista alterando um modelo de projeto, ou um autor de WEB modificando uma página HTML

- O **Integrador** – tem familiaridade com a estratégia de montagem e ferramentas de montagem usadas no projeto e com os processos e as

ferramentas de gerência de configuração usadas pela organização. O integrador recebe as atividades dos desenvolvedores, cria novas linhas base de componentes, monta os componentes do sistema, garante que os produtos foram testados e promove a nova linha base quando os testes estiverem completos

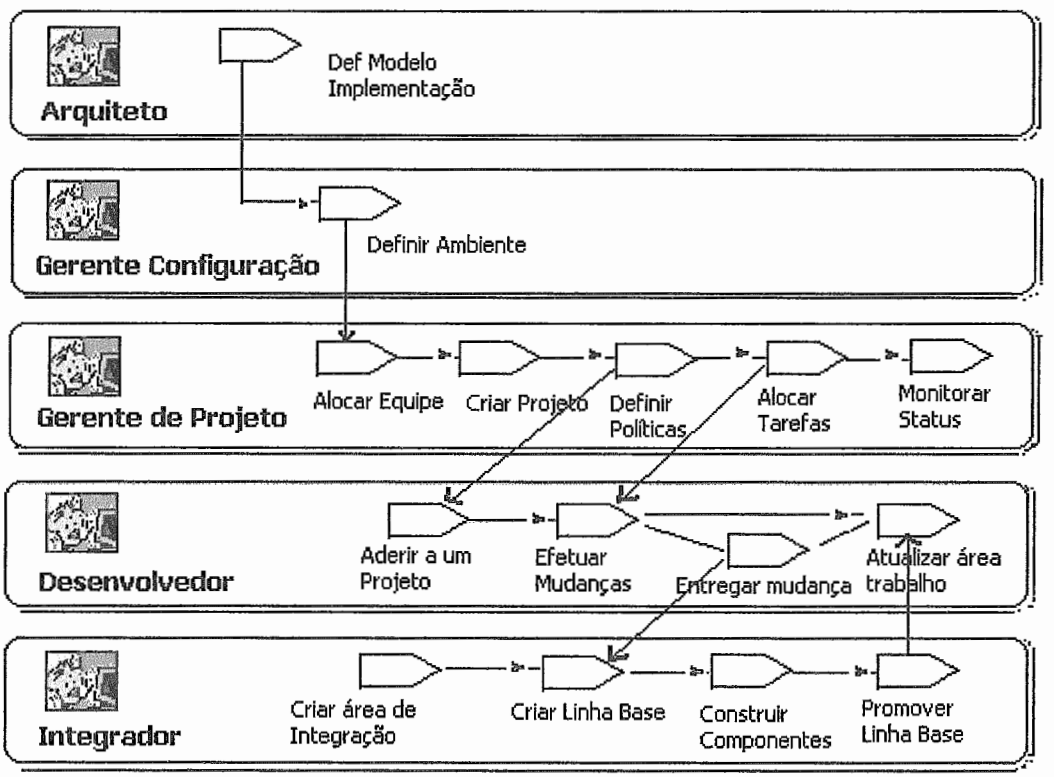

Figura 3.1 Papéis no desenvolvimento (ClearCase)

Conforme podemos observar, a abordagem de gerenciamento, embutida no modelo proposto, é de cunho prescritivo, própria para organizações grandes e hierarquizadas. Talvez para equipes abertas, distribuídas geograficamente e sem vinculação empresarial muito forte entre si, que é um tipo de grupo de desenvolvimento em crescimento, este estilo não seja o mais adequado.

O repositório do ClearCase é denominado VOB – Base de Objetos Versionados (*Versioned Object Base*) e é definido como um repositório permanente de dados no qual podem ser armazenados arquivos, diretórios e metadados. A Rational afirma que o VOB

é um banco de dados escalável, tolerante a falhas, distribuído e replicável. É constituído de dois tipos primários de bases de dados: o próprio VOB, que é compatível com o sistema de arquivos da máquina hospedeira, e, portanto, pode ser visto de forma similar aos demais arquivos e diretórios do sistema operacional; e o PVOB – VOB de Projeto, que contem os objetos associados ao ambiente de projeto como projetos, componentes, atividades e linhas base.

A área de trabalho do desenvolvedor (*Workspace*) pode ser construída de duas formas: tipo instantâneo fotográfico (*snapshot*) ou visão dinâmica. No primeiro caso, será feita uma cópia dos arquivos e diretórios do release de software que o usuário deseja alterar. Apesar de precisar um tempo maior de carga e mais espaço em disco, tem a vantagem de facilitar as tarefas de edição, compilação e montagem. No segundo, ele terá uma visão do próprio repositório. A carga será muito rápida e o espaço de disco ocupado menor.

O objeto atômico do ClearCase é chamado de **Elemento**, que pode ser um arquivo ou diretório. Um **Ramo** (*branch*) é um objeto que especifica uma seqüência de versões diferentes do mesmo elemento ao longo do tempo. Pode ser usado para executar um desenvolvimento em paralelo ou para manter variantes do sistema. A **Versão** é uma instância específica do elemento, criada por um desenvolvedor num certo instante do tempo. Existe ainda a possibilidade de ser criado um **Rótulo**, que associará uma versão a um nome externo de referência. A Figura 3.2 mostra de forma gráfica uma **Árvore de Versão**, contendo todos estes elementos. O ClearCase permite obter este tipo de informação por linha de comando ou por interface gráfica.

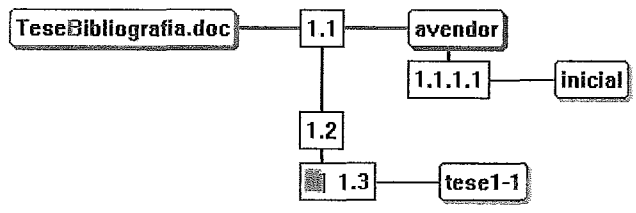

Figura 3.2 – Árvore de versão (Clear Case)

É importante salientar que os diretórios também são mantidos em versões, no ClearCase. Isto permite que se possa recuperar uma estrutura de diretórios e arquivos que já não existem na versão mais atual[1].

Os elementos devem ser categorizados de acordo com os tipos de dados contidos nos arquivos. Com base nos tipos o ClearCase fará a comparação entre versões e o armazenamento no repositório. Os tipos pré-definidos pelo ClearCase são os seguintes:

- **Arquivo** – cada nova versão de um elemento do tipo arquivo será armazenada como uma cópia completa. Nenhuma computação de delta é usada para economizar espaço.
- **Arquivo texto** – é um arquivo texto e usa armazenamento delta em linha[2]. Este é o tipo primário usado para a maioria dos textos.
- **Arquivo texto comprimido** – usa o mesmo mecanismo delta do tipo texto, mas com compressão do arquivo delta, para maior conservação de espaço em disco.
- **Arquivo comprimido** – é do tipo Arquivo, com compressão de cada cópia de arquivo.
- **Arquivo binário delta** – realiza o armazenamento apenas do delta entre arquivos binários. O objetivo é conservar espaço em arquivos binários longos e muito modificados.
- **Diretório** – novas versões de diretórios de arquivos são armazenadas neste tipo, usando um formato interno. Permite administrar versões de diretórios de projeto.

---

[1] O comando **rmname**, do ClearCase, remove um elemento da versão atual do diretório, mas não o remove do repositório. Já o comando **rmelem** remove do repositório, impedindo que possa ser usado posteriormente.

[2] O armazenamento "delta em linha" usado pelo ClearCase é diferente do delta à frente (*forward*) e do delta reverso, normalmente usado em sistemas de GCS.

Os tipos de elementos podem ter supertipos. Por exemplo, pode ser criado um tipo "arquivo pascal" que tenha como supertipo "arquivo texto", herdando suas características básicas de armazenamento.

Existem alguns tipos pré-definidos adicionais construídos sobre os elementos básicos. São os seguintes:

- **Linguagem de marcação de texto** (*html*) – identifica e armazena arquivos formatados em HTML. O supertipo é "arquivo texto", portanto usa o armazenamento delta em linha.

- **Microsoft Word** – é usado para identificar e administrar documentos criados com o Microsoft Word. O gerente de tipos usa as ferramentas de comparação e junção (*merge*) do Microsoft Word. O supertipo é "arquivo", sendo, portanto, armazenada a cópia completa.

- **Modelo ROSE** – identifica e gerencia diagramas do Rational Rose. Usa o integrador de modelos do Rose para fazer comparações e junções. O supertipo é "arquivo texto" e, portanto usa este tipo para armazenar os dados.

- **eXtensible Markup Language** (*xml*) – identifica e gerencia arquivos XML. O gerente de tipos usa as novas ferramentas de comparação e junção do XML para suportar o desenvolvimento paralelo de arquivos XML. O supertipo é "arquivo texto" e usa o armazenamento delta em linha.

O usuário avançado do ClearCase pode definir tipos que utilizem mecanismos de compressão e delta proprietários. O administrador pode também definir como fazer a comparação e a junção de duas ou mais versões, para um tipo de usuário.

O ClearCase é um produto de Gerência de Configuração orientado a processo. Inclui, portanto, mecanismos de definição e administração do processo de software. Os objetos especiais, ou metadados, que permitem executar esta gestão, implementando funcionalidades e associações adicionais, são os seguintes:

- **Rótulos** – são instâncias do *tipo rótulo* que são ligadas a uma versão de um elemento. Permitem identificar um conjunto consistente de versões de elementos. Por exemplo, podemos marcar todas as versões de elementos que formam a "Release 1" de um software, atribuindo o rótulo RELEASE1 às versões dos elementos que compõem este release

- **Atributos** – os atributos são tipos formados por conjuntos nome/valor que podem ser associados com praticamente todos os objetos do ClearCase. Por exemplo, pode-se criar um tipo de atributo chamado status_da_revisão, com os valores possíveis de "passou", "falhou", "pendente". Uma instância deste tipo pode ser ligada a cada versão de um **elemento, indicando o status daquela versão.**

- **Hiper-ligações** (hiperlinks) – definem relacionamentos entre objetos. Por exemplo, pode-se ligar os documentos do projeto detalhado ao código fonte correspondente

- **Gatilhos** (triggers) – são eventos definidos pelo usuário que disparam quando ocorrem operações do ClearCase. Combinados com atributos e hiper-ligações permitem automatizar a criação de relacionamentos interobjetos, dados de ligação e regras de negócio baseadas em eventos. Os gatilhos podem ser de dois tipos: pré-eventos e pós-eventos. Os pré-eventos são disparados antes da ocorrência do evento e podem cancelar a ocorrência do mesmo. Servem para garantir os procedimentos e as políticas definidos para o evento. Os pós-eventos permitem comunicar a quem de direito, por exemplo, que um determinado evento ocorreu

Apesar do ClearCase não ter implementado de forma nativa procedimentos de cooperação entre os membros da equipe de desenvolvimento, a base para esta implementação existe de forma muito boa. Basta que o responsável pela configuração da ferramenta proceda de forma a utilizar os recursos disponíveis.

3.2 Code Co-op

A Reliable Software desenvolveu o Code Co-op como o seu principal produto. É uma pequena organização composta por quatro programadores, um deles vivendo em Seattle – EUA e os outros três na Polônia (Gdansk e uma outra pequena cidade polonesa) e auxiliada por mais alguns poucos estudantes de computação, um deles na Universidade de Calgary - Canadá. O Code Co-op se intitula *Sistema de Controle de Versão Sem Servidor* (*Server-less Version Control System*), pois sua principal característica é a de não possuir um servidor para armazenar o repositório de projetos. A Figura 3.3 apresenta a arquitetura do sistema. O repositório de projeto é totalmente replicado nas máquinas denominadas de *Hub* (*Centro!*). Quando alguém modifica algum arquivo e faz o check-in, é gerado um script da diferença do arquivo editado em relação à versão retirada do repositório. Este script é distribuído a todas as máquinas que estão trabalhando no projeto e sincronizado com o respectivo repositório. A distribuição pode ser feita diretamente pela rede local, ou por alguma forma de transferência de arquivos, como disquete ou zip drive, ou por correio eletrônico. Caso exista algum conflito na sincronização ele é resolvido e uma nova versão do arquivo é criada.

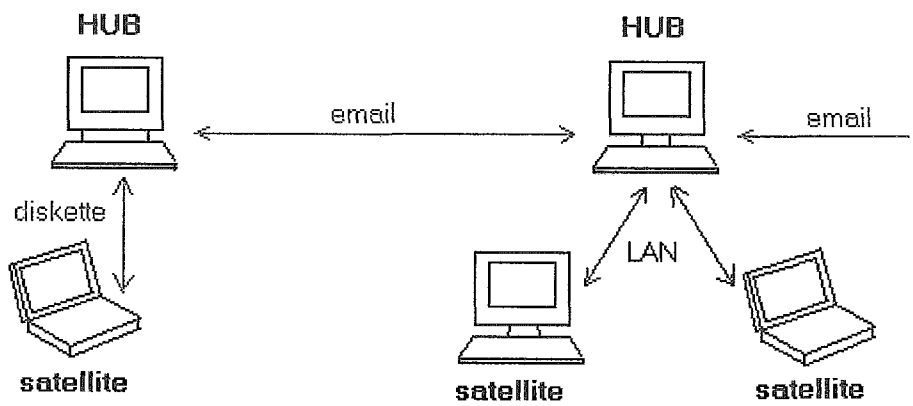

Figura 3.3 Arquitetura do Code Co-op

As ações normais de trabalho com o Code Co-op podem ser descritas da seguinte forma:

1. O usuário cria um projeto e passa a ser seu administrador

2. Inclui (check in) arquivos e diretórios neste projeto

3. Outros desenvolvedores aderem ao projeto e realizam modificações nos arquivos, gerando novas versões, que são replicadas em todas as máquinas que participam do projeto

4. Ao fazer o check in de um módulo o desenvolvedor deve, como boa prática de projeto, primeiro fazer a reconciliação dos scripts recebidos de outras máquinas. No caso de conflitos entre as atualizações propostas o usuário deve resolver estes conflitos, o que pode gerar interações com os demais membros

Existem três papéis de usuários no Code Co-op: o administrador, o membro votante e o observador. O observador tem uma caracterização mais comercial do que técnica. Todos os usuários com licença de uso expirada[3] são transformados em observadores e não podem mais modificar os itens de projeto. O Administrador é o usuário que criou o projeto. Os membros votantes podem juntar-se ao projeto através do envio de um script de adesão dirigido ao administrador, que será respondido, positivamente ou não, formalizando a entrada no projeto e cadastrando-os nos repositórios. Os membros podem solicitar o afastamento de um projeto. Se, eventualmente, o administrador solicitar afastamento, será realizada uma eleição de um novo administrador, dentre os membros votantes do projeto. Um candidato se apresenta e os demais membros aprovam ou não a candidatura.

Os dados do repositório são armazenados em uma estrutura própria do Code Co-op. A versão mais atual dos arquivos é mantida na área de trabalho do usuário, dentro da

[3] O software é distribuído pela Internet com licença de experimentação de 31 dias, findos os quais o usuário deve adquirir uma licença de uso, ou reinstalar o produto, ou permanecer apenas como *observador* nos projetos

estrutura normal do projeto, com o único cuidado de manter os arquivos com o atributo de *somente leitura*. As diferenças são mantidas no repositório do sistema, inclusive para os arquivos binários. Se houver algum problema qualquer de edição fora do controle do Code Co-op, será necessário executar uma função de reparo do repositório. Normalmente a ferramenta tem condições de reparar inconsistências devido à redundância implícita no repositório. Algumas vezes outros membros do projeto podem ser convocados a ajudar no processo de recuperação. Para evitar dificuldades de recuperação de arquivos, é recomendado que qualquer operação seja sempre executada "por dentro" do Code Co-op, principalmente mudanças de nomes e alterações de estruturas de diretórios de projeto.

Sempre que um novo tipo de arquivo for colocado sob controle de versão, o Code Co-op solicitará que o usuário categorize o tipo de arquivo. Existem quatro categorias previstas: cabeçalhos, fontes, textos e binários. Para todos o armazenamento é feito de forma incremental. A diferença entre eles é que os arquivos binários não podem ser fundidos e, se houver conflito entre dois arquivos binários, será feita uma cópia do arquivo conflitante (criando "cópia de ....").

É permitido ao usuário criar um braço de projeto (*branch*) a partir de uma versão original. No entanto, não pode haver junção novamente ao tronco principal de desenvolvimento. Ou seja o novo braço de projeto segue como um projeto independente.

A operação do sistema é feita através do módulo básico Code Co-op e de um agente chamado *Dispatcher*, que permite distribuir os scripts. O módulo Code Co-op, conforme pode ser visto na Figura 3.4, apresenta diversas abas para controle dos *projetos*, *arquivos*, *área de check in*, *caixa de correio*, *área de sincronização de scripts* e *histórico das versões*. A aba de arquivos relaciona os arquivos de um determinado projeto, mostrando o estado do arquivo, que pode ser "in" para arquivos sob controle de versão, "out" para arquivos liberados para edição, ou "--" para arquivos não incluídos no controle do sistema. Também, são apresentados a identificação global e o tipo de arquivo.

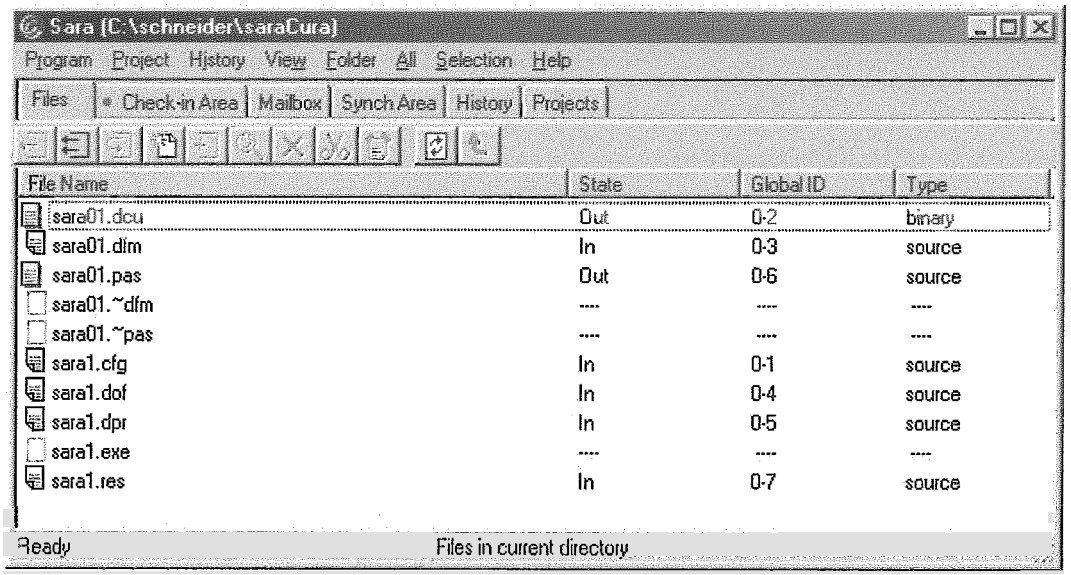

Figura 3.4 Aba de arquivos do Code Co-op

A aba de histórico, Figura 3.5, permite visualizar as diversas versões do projeto. O estado de um item no histórico pode ser pendente, se ainda não foi confirmado por todos os membros do projeto. Status de confirmado significa que todos os membros aprovaram o script através do reconhecimento (*acknowledgement*) do script.

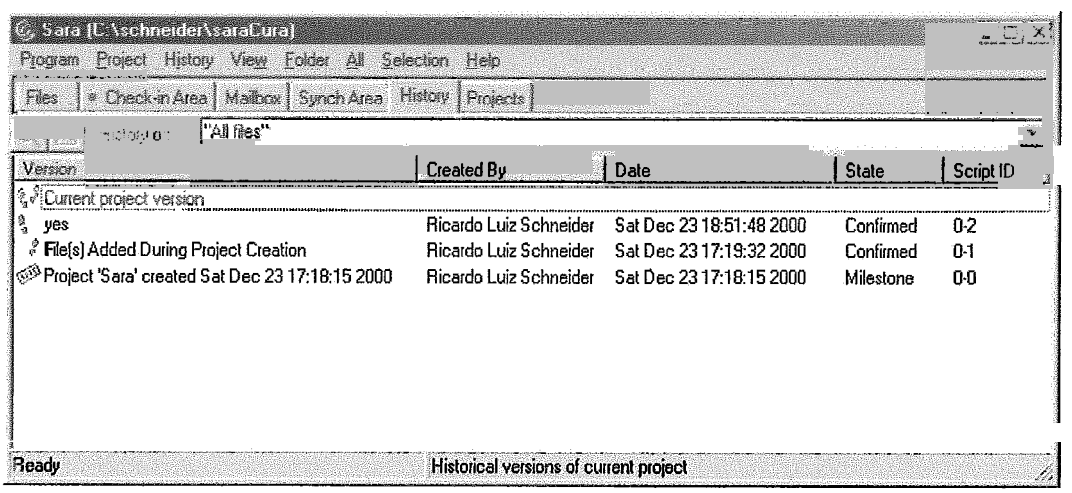

Figura 3.5 - Aba de histórico do Code Co-op

O *Dispatcher* é um agente colocado na barra de tarefas (ver Figura 3.6) do Windows (*System Tray*), podendo ser acionado automaticamente quando algum script for colocado nos diretórios de script do Code Co-op ou de algum dos projetos. O ícone do *Dispatcher* apresenta uma seta vermelha para cima quando existir algum script a ser enviado para um *hub* externo e uma seta vermelha para baixo quando existir um algum script a ser aplicado ao *hub* local. O recebimento e envio de correio pode ser automático, se o software cliente de correio instalado for totalmente compatível com a interface Simple MAPI da Microsoft. O Eudora, Outlook, Outlook Express possuem compatibilidade total. Já o Netscape Messenger precisa de alguma intervenção do operador para funcionar adequadamente.

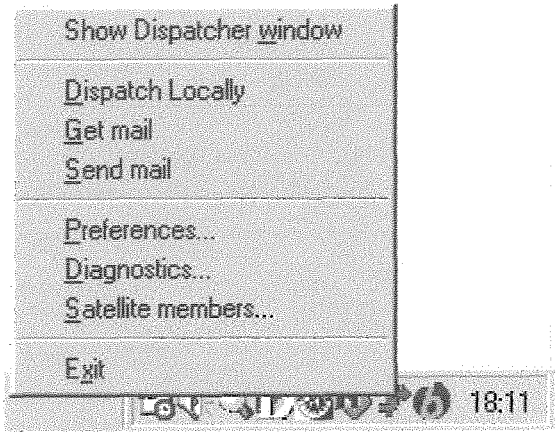

Figura 3.6 *Dispatcher* na barra de tarefas

Na Figura 3.7 é apresentado o *Dispatcher* ativado para realizar o controle dos scripts. O projeto UsuariosNT possui um script de saída e os demais projetos não possuem nenhum script a receber ou enviar, no momento.

O Code Co-op possui interface com a SCC API da Microsoft. Esta interface de controle de código fonte permite operar com check in, check out, diferenças entre módulos e outras operações de gerência de configuração com produtos como o Microsoft Developer Studio (C++, Java, VB), CodeWright, VisualAge, Cold Fusion, HomeSite e o Rational ROSE.

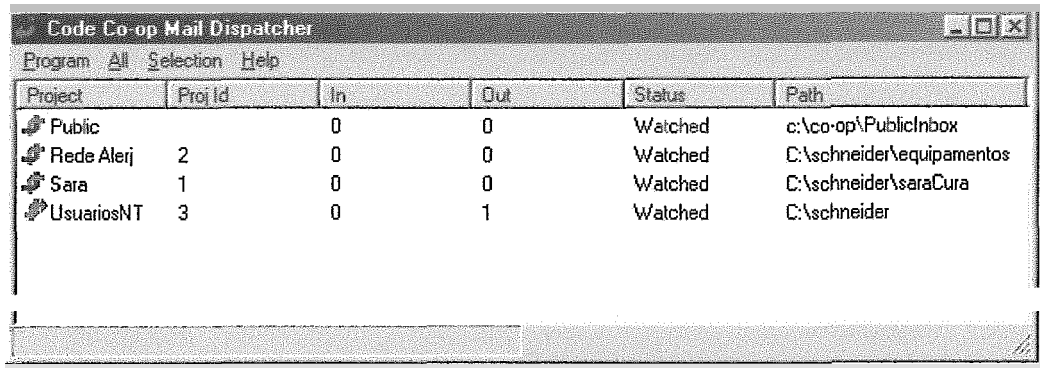

Figura 3.7 Controle de scripts do *Dispatcher*

Outras características interessantes do Code Co-op são a ajuda do sistema, que é toda desenvolvida em *html* e uma interface intercambiável entre usuário principiante e experiente. Esta interface, no modo principiante, conduz o usuário na utilização do sistema, informando os passos em execução e orientando em relação às ações consecutivas.

O Code Co-op é produto simples na sua concepção. Não incorpora mecanismos de administração do processo de desenvolvimento, sendo, portanto, um sistema de controle de versões de código fonte, mas possui características de cooperação entre os desenvolvedores bastante interessantes, devido ao processo de aceitação dos scripts de alterações encaminhados pelos outros membros da equipe. Esta cooperação é assíncrona, mas eficiente porque é obrigatória.

3.3 CVS

O CVS – *Concurrent Versions System*, é uma ferramenta aberta, desenvolvida sob os auspícios da Free Software Foundation (CEDERQVIST, 1993), e distribuída com licença GNU. É tipicamente uma ferramenta cliente/servidor. O servidor CVS é um serviço que executa sob Linux/Unix. O cliente normal é um programa de linha de comando, que executa em lote. Existem, no entanto, diversos clientes em interface gráfica como o WinCVS, que roda sob Windows 9x, NT, 2000, o jCVS, roda nas várias

plataformas que funcionam com Java e tkCVS, desenvolvido em TCL/TK, também executa nas diversas plataformas onde este ambiente opera: Linux/Unix, Windows, Macintosh.

No CVS o usuário desenvolve o sistema em sua área de trabalho, criando a sua estrutura de projeto com diretórios, sub diretórios e arquivos. Quando o trabalho estiver pronto submete ao servidor através de uma conexão TCP/IP. Pode neste momento desfazer toda a estrutura do projeto na área de trabalho, porque, quando precisar, pode transferir toda a estrutura de volta ao seu computador. O armazenamento dos dados no repositório do projeto é feito usando o formato RCS de arquivos. O formato RCS foi desenvolvido também pela Free Software Foundation e é, ele próprio, um sistema de controle de versões. Armazena os dados de forma incremental, colocando somente as diferenças entre uma versão e a seguinte. O CVS incorporou novas funcionalidades sobre a base do RCS.

No WinCVS, cuja tela principal é mostrada na Figura 3.8, é criada uma pequena estrutura de controle, junto ao diretório de trabalho, para facilitar a interação com o servidor CVS. Esta estrutura é constituída de um diretório com arquivos contendo a relação dos módulos da área de trabalho, o diretório raiz do CVS e o caminho do projeto atual no repositório. No painel superior direito, podemos ver a relação dos módulos do projeto, com as informações: objeto é de escrita ou não, ícone descritivo do objeto (vermelho significa que já foi alterado, linhas indicam texto, 01 indica que é arquivo binário, ...), nome, revisão, etc.

Uma característica marcante do WinCVS é que, apesar de ser uma ferramenta com interface gráfica, guarda um comportamento típico de ferramenta conduzida por linha de comando. Podemos identificar este fato pelo painel inferior direito (na Figura 3.8) que apresenta anotações (*log*) dos comandos executados. Outro exemplo é encontrado no menu "CVS Admin", onde encontramos a opção "Command Line..." que permite escrever diretamente uma linha de comando para o servidor CVS. Estas opções poderiam ficar embutidas na interface, isolando o usuário dos detalhes de operação do servidor CVS.

O CVS possui flexibilidade, como o seu próprio nome indica, para acesso concorrente de diversos usuários. Existem três níveis possíveis de acesso aos arquivos do repositório:

- A colocação de cadeado (lock) que inibe o acesso ao arquivo enquanto o usuário, que colocou o cadeado, não o liberar

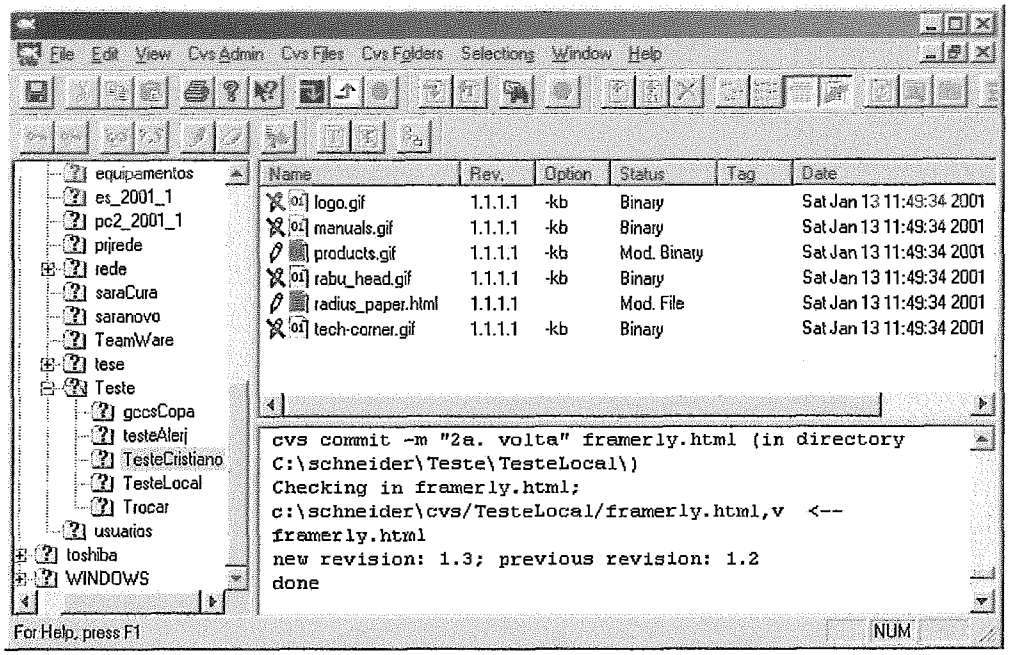

Figura 3.8 - WinCVS

- A colocação de uma **sentinela** (*watch*) que avisa quando algum usuário estiver editando um determinado arquivo. Pode, inclusive, enviar um e-mail sempre que algum outro usuário pegar o arquivo para edição.

- O uso **concorrente** sem nenhuma restrição. Neste caso, cada usuário que realizar o check in de um arquivo gerará uma nova revisão do mesmo. Todas as variantes do arquivo serão guardadas no repositório. Possíveis conflitos de edição deverão ser resolvidos para poder realizar a inclusão do arquivo no repositório

O histórico dos arquivos é mantido com riqueza de detalhes de cada revisão e rótulo incluído, como no exemplo da Figura 3.9. Nela se vê as releases iniciais que o CVS cria automaticamente (*avendor* e *arelease*), a Release-1 incluída pelo usuário e as diversas revisões (1.1, 1.2 e 1.3) criadas para o arquivo "framerl3.gif".

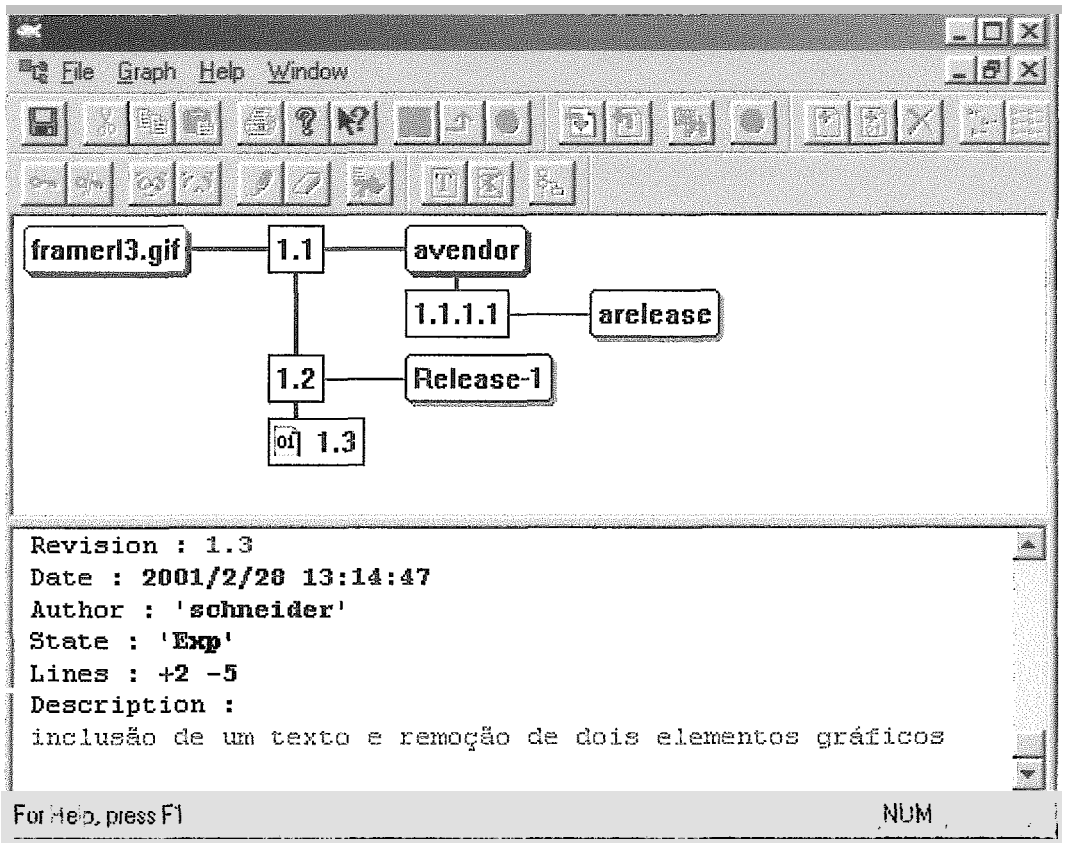

Figura 3.9 - Visão gráfica das revisões do CVS

O CVS distingue apenas dois tipos de arquivos: Textos e Binários. E usa sempre o formato de armazenamento incremental (através do RCS). Trata-se, portanto, de um controlador de versões e não é um sistema de controle de mudança mais completo.

3.4 Perforce

O Perforce é um produto oferecido pela Perforce Software, empresa inglesa especializada em sistemas de gerência de configuração. Sua principal propaganda é em cima de sua velocidade. Auto intitula-se como "*O mais rápido software de gerência de configuração*".

Funciona em mais de trinta plataformas diferentes, como Windows, Linux, Macintosh, etc. Permite utilização pela Internet (Web), Lan e Wan.

A tela principal da ferramenta é apresentada na Figura 3.10

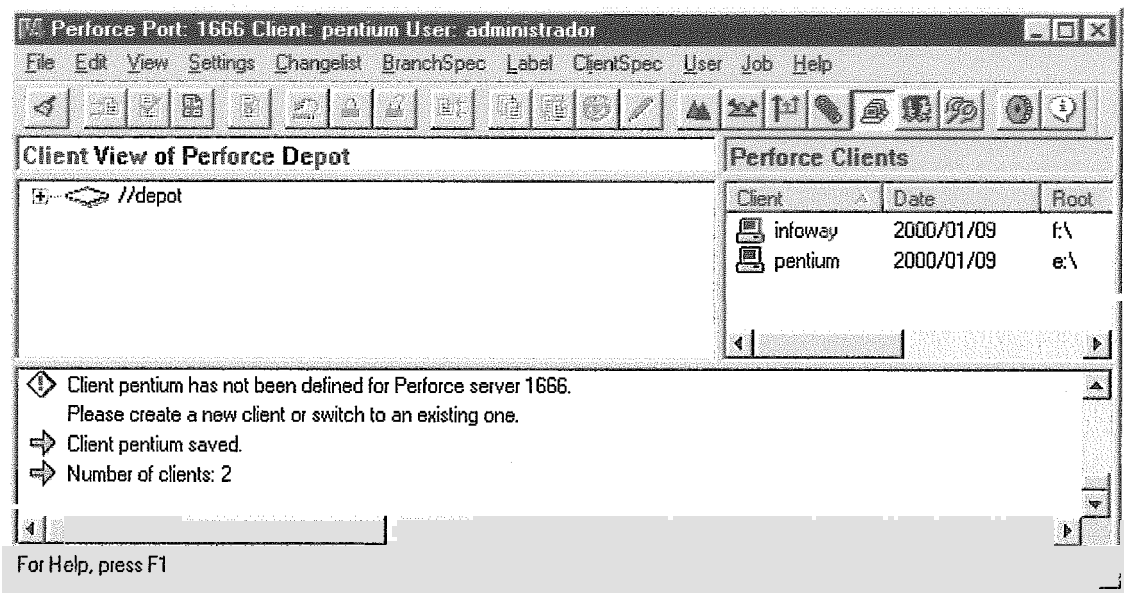

Figura 3.10 – Tela principal do Perforce

3.5 PVCS

O PVCS® é uma família de produtos da MERANT™, empresa da Califórnia, EUA, que "*automatiza a qualidade de software*" provendo ferramentas para: 1) Gerência de Configuração de Software; 2) Gerência de Lançamentos (*issue*) de Software e 3) Automação do Processo de Software. As ferramentas que permitem esta administração são as seguintes:

- **PVCS Version Manager**: Habilita equipes de qualquer tamanho a coordenar desenvolvimento concorrente, enquanto controla mudanças em múltiplas revisões
- **PVCS VM Server**: Estende o poder do *Version Manager* para a Internet e Intranet usando um cliente seguro de WEB
- **PVCS Configuration Builder**: Automatiza e padroniza o processo de montagem (*build*) de programas
- **PVCS Tracker**: Captura, gerencia e comunica solicitações de melhorias e defeitos de software
- **PVCS Professional**: Forma o coração do desenvolvimento efetivo em equipe pela combinação do *Version Manager*, *Tracker* e *Configuration Builder* em um único pacote
- **PVCS Developer's Toolkit**: Providencia uma interface de programação de aplicação (API) para as funções do PVCS
- **PVCS Dimensions**: Define o ciclo de vida e gerencia o processo de desenvolvimento para acelerar a entrega de software de alta qualidade

A proposta do PVCS é auxiliar numa gerência de configuração que organize e gerencie os esforços das equipes de desenvolvimento na criação dos componentes de software desde os primórdios até à montagem final das liberações de produtos, de forma

consistente e reproduzível. As solicitações de melhorias e os relatórios de defeitos são rastreados e não se perdem nos meandros da burocracia. Tudo isto é proposto que seja executado através de ferramentas e procedimentos que identifiquem, gerenciem e controlem os componentes de software à medida que sejam desenvolvidos e modificados. Propõe-se, portanto, a ser um conjunto de ferramentas com forte aderência ao processo de desenvolvimento de software.

A ferramenta central na sistemática de gerência de configuração é o VM – Version Manager™. Os conceitos utilizados pela ferramenta que possuem características diferenciadas em relação a outros produtos de gerência de configuração e que, portanto, permite visualizar a sua forma de funcionamento, são os seguintes:

- **Banco de Dados de Projeto**: É uma coleção hierárquica de projetos, subprojetos, e versões de arquivos. Um banco de dados de projeto define uma configuração comum para todos os projetos e subprojetos contidos nele. Os usuários são definidos para cada banco de dados

- **Projetos e subprojetos**: Projetos são agrupamentos lógicos de subprojetos e versões de arquivos. Subprojetos são projetos contidos em um projeto. Apesar do banco de dados de projeto definir valores de configuração para todos os projetos e subprojetos contidos nele, é possível sobrepor estas opções a níveis de projeto e subprojeto

- **Arquivo de trabalho**: é qualquer arquivo que tenha sido colocado na área de trabalho do usuário para criar uma nova revisão

- **Arquivo de Versão**: é um arquivo com as revisões associadas. Mantêm informações sobre o nome e localização do arquivo físico que contêm as revisões

- **Arquivo físico (*Archive*)**: é o arquivo que armazena a história evolucionária de um arquivo de versões. Inclui descrição das mudanças, quem fez e quando foram feitas as mudanças. Existe um arquivo físico para cada rótulo de

versão e grupo de promoção

- **Revisão**: é uma instância de um arquivo de versão que pode ser recriada. Quando é feito um check in de um arquivo de trabalho, as mudanças feitas são armazenadas como uma revisão. Quando um ckeck out de uma revisão é realizada é criado um arquivo de trabalho na localização especificada

- **Área de trabalho**: é uma coleção de definições de ambiente para um projeto. Inclui a localização dos arquivos de trabalho, a versão padrão, a versão base e o ramo de versão em efeito para o usuário

- **Usuário**: é uma pessoa que executa tarefas básicas e avançadas de gerência de configuração. Dentre outras tarefas, cria projetos, adiciona e faz check in de arquivos de trabalho, check out de revisões, adiciona rótulos de versão, e promove revisões

- **Administrador**: é a pessoa que realiza tarefas administrativas, tais como criar e manter bancos de dados de projetos, banco de dados de controle de acesso, modelos de promoção, prepara gatilhos de eventos e gera relatórios

- **Grupo de Promoção**: representa um marco no ciclo de desenvolvimento de uma aplicação. Alguns exemplos incluem o Desenvolvimento, Revisão de Qualidade e Produção

- **Modelo de Promoção**: é uma hierarquia de grupos de promoção representando marcos no processo de desenvolvimento de uma aplicação. Quando é criado um modelo de promoção, o Version Manager associa revisões com os grupos de promoção do modelo

- **Linha Base:** é um *instantâneo* de um projeto existente em um ponto específico do desenvolvimento. Para criar uma linha base é feita uma cópia do projeto inteiro em um novo projeto com base num rótulo de versão ou num grupo de promoção. Normalmente, são colocados cadeados nos arquivos colocados numa linha base de projeto, para que sirvam como fontes

de referência para o desenvolvimento seguinte

- **Gatilhos de eventos:** são mecanismos que causam uma ação em resposta a um evento da ferramenta. Gatilhos de eventos são definidos nos arquivos de configuração

- **Banco de Dados de Controle de Acesso:** define os usuários que estão autorizados a executar ações nos projetos. Os bancos de dados de controle de acesso são associados com bancos de dados de projeto

- **Listas de acesso:** definem os grupos de usuários e os usuários individuais que são autorizados a executar ações em arquivos físicos. Cada arquivo físico possui uma lista de acesso associada e uma lista de acesso é um subconjunto dos usuários definidos no banco de dados de controle de acesso

- **Opções de Configuração e Arquivos de Configuração:** Opções de configuração e arquivos de configuração permitem controlar como o Version Manager opera. Existem dois tipos de arquivos de configuração: o arquivo mestre de configuração, que contem opções para o banco de dados de projeto e todos os projetos, e o arquivo de configuração de projeto, que tem efeito sobre um projeto específico

Um fluxo de trabalho básico previsto no PVCS é apresentado na Figura 3.11, onde podem ser destacados os papéis previstos na ferramenta: Administrador, Líder de Projeto e Membro da Equipe de Projeto. Existem tarefas próprias de cada papel e tarefas compartilhadas por mais de um papel, como, por exemplo, criar arquivos de trabalho, que é um tipo de atividade exercida pelo líder ou por algum membro da Equipe de projeto.

Pelas atividades previstas neste roteiro geral, em especial as quatro últimas tarefas, podemos verificar o destaque que o PVCS dá à função de montagem do código objeto. A maioria das outras ferramentas de GCS não inclui a geração e controle do código executável, focalizando sua atenção exclusivamente no código fonte.

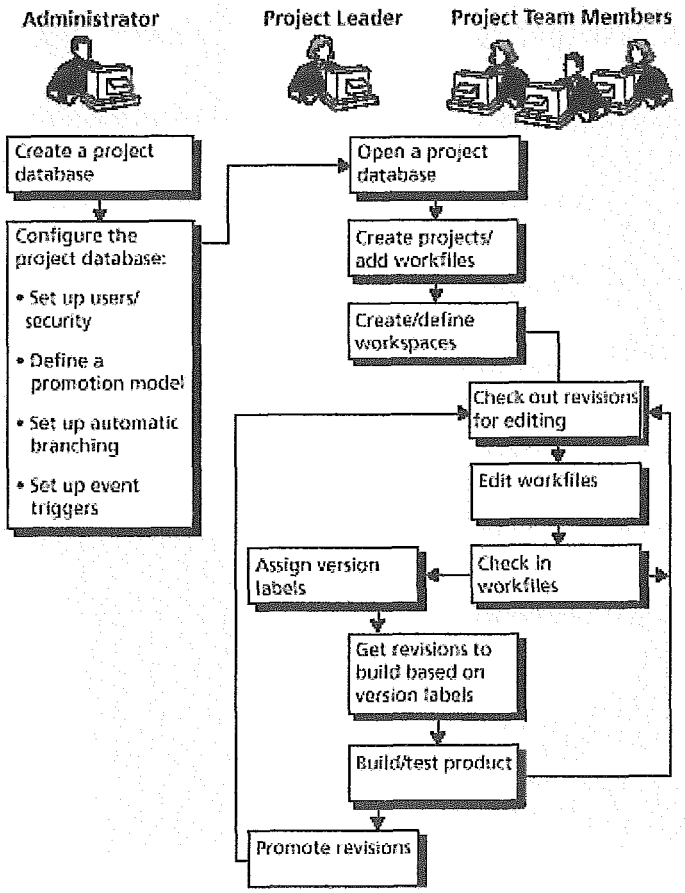

Figura 3.11 - Fluxo de trabalho do PVCS

3.6 RCS

O RCS (VASUDEVAN, 2001) é também um software desenvolvido de forma aberta pela Free Software Foundation. O RCS é o método básico de armazenar versões de arquivos, usado pelo CVS. Como não suporta concorrência de acesso não é mais utilizado de forma corrente, apenas embutido no CVS ou em repositórios arcaicos ainda não substituídos por métodos mais atuais.

O SCCS – Source Code Control System, é um sistema de controle de versões mais antigo ainda da Free Software Foundation e também raramente utilizado. Sua importância reside no fato de ser utilizado por ferramentas mais modernas, como o próprio CVS, como modo básico de administrar arquivos diferenciais.

3.7 Star Team

StarTeam é uma ferramenta desenvolvida pela StarBase, empresa da Califórnia, USA. É o principal produto desta organização dedicada a softwares colaborativos. Possui uma abrangência bastante completa: armazena e controla os arquivos e diretórios de um projeto, controla as solicitações de mudanças, tarefas em execução, comunicação entre os membros das equipes e outros mecanismos importantes para um processo de desenvolvimento colaborativo de software.

Na Figura 3.12 é apresentado um aspecto da tela principal do StarTeam para a visão geral dos arquivos de um projeto, onde se pode observar três painéis: O painel esquerdo com a hierarquia geral do projeto; o painel superior direito que apresenta o primeiro nível de detalhamento do diretório de projeto selecionado no painel esquerdo; e, finalmente, no painel inferior esquerdo os detalhes do item selecionado no painel superior direito. Estes últimos detalhes estão subordinados à aba de histórico, mostrando este tipo de detalhe. Poderia ser selecionada uma outra aba, apresentando outro detalhamento.

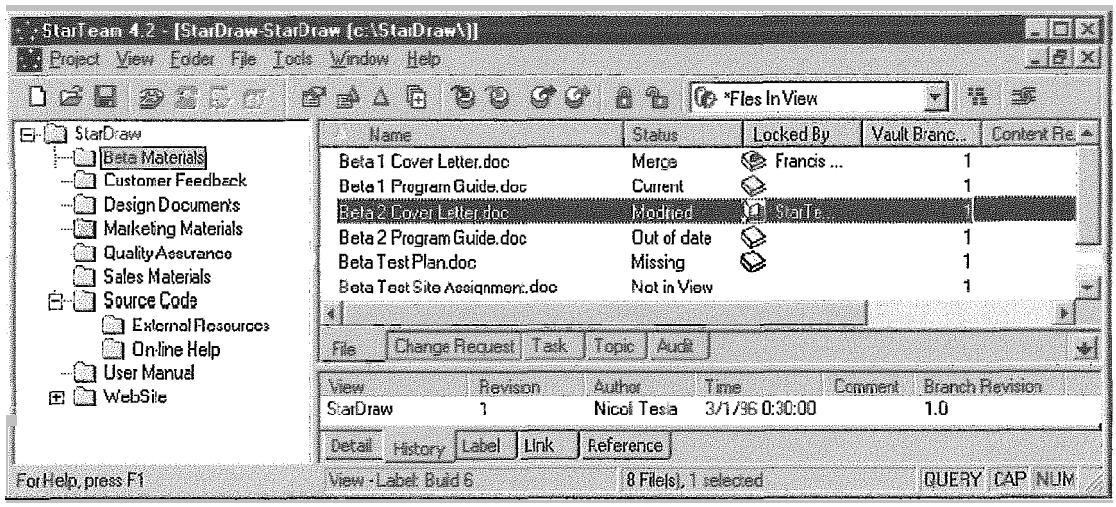

Figura 3.12 – Tela principal do StarTeam

Na Figura 3.13 podemos avaliar a proposta de fluxo de trabalho no uso da ferramenta. De forma especial, podemos destacar o papel do testador (*tester*) com

bastante relevância no ciclo de trabalho – aparece com mais atividades do que o próprio desenvolvedor.

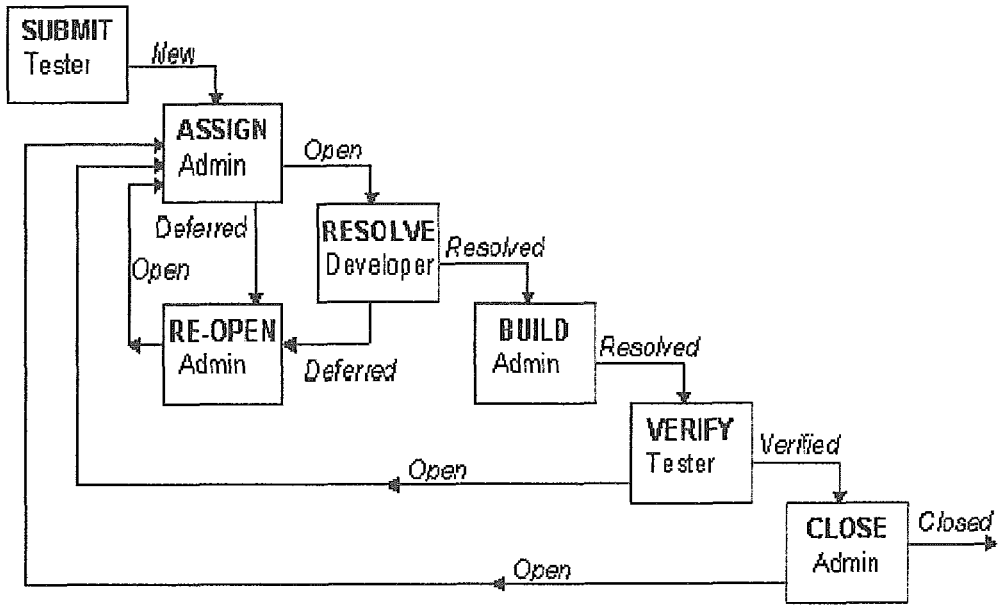

Figura 3.13 – Fluxo de trabalho para Solicitação de Mudança

O Controle de Requisições de Mudanças, apresentado na Figura 3.14, do StarTeam é bastante completo. Características, como a identificação do pedido, uma sinopse, o tipo de pedido (se defeito, melhoria, etc), o status, a severidade, a responsabilidade, são apresentadas de uma forma global e detalhes complementares, como a revisão envolvida, autor de cada revisão, etc, estão também disponíveis.

Poucos produtos de GCS possuem características de tratamento dos Pedidos de Mudanças quanto o StarTeam.

Outro aspecto bem cuidado da ferramenta é o tratamento oferecido às tarefas a serem executadas. Na Figura 3.15 são apresentados os detalhes das tarefas, tais como: identificação, nome, tipo, responsável, datas previstas de início e término, datas efetivas de inicio e término, etc.

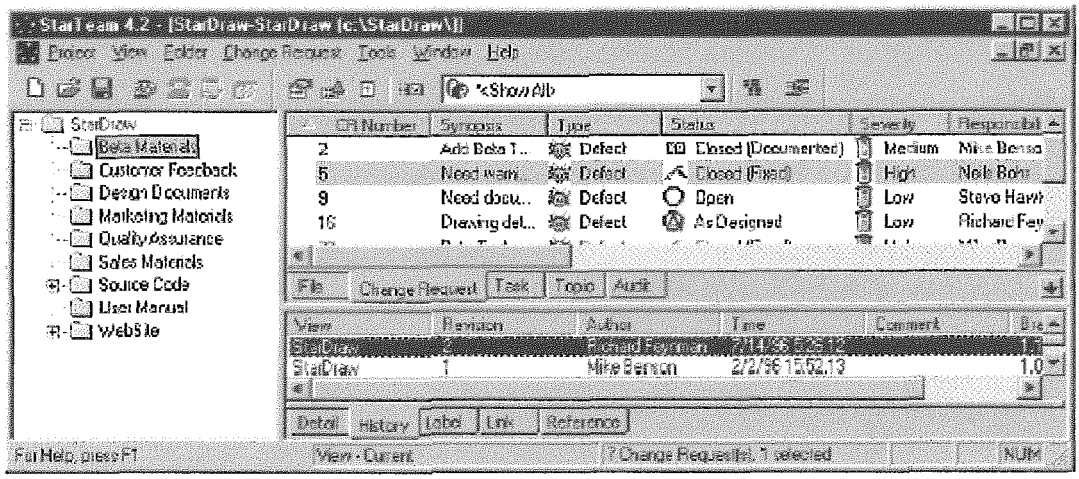

Figura 3.14 – Controle de Requisições de Mudanças

Em suma, o StarTeam é uma das ferramentas mais completas encontradas na pesquisa realizada.

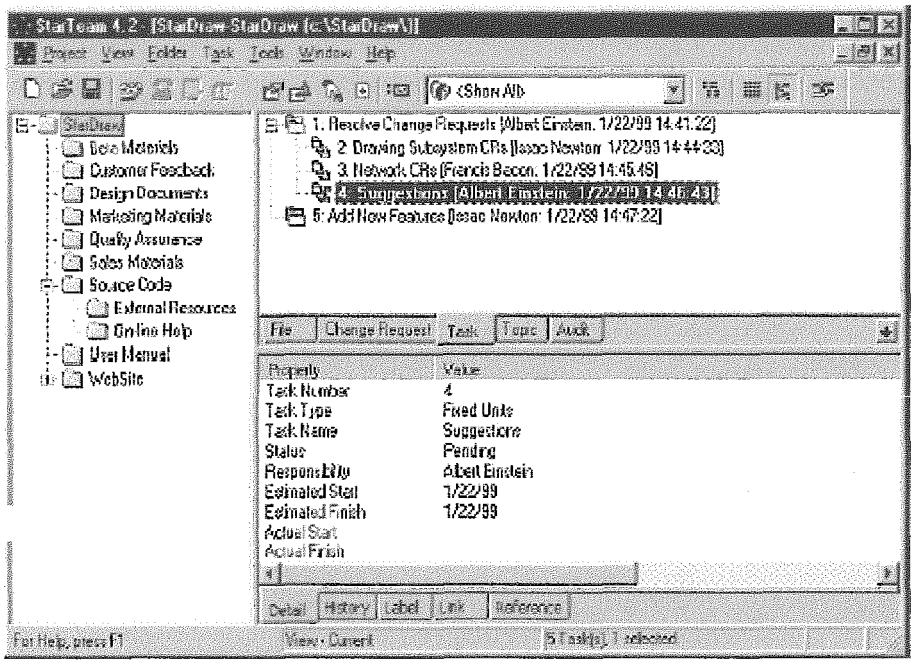

Figura 3.15 – Controle de Tarefas

3.8 Visual Source Safe

O Microsoft Visual SourceSafe™ (MICROSOFT, 1996) é o produto atual de gerência de versões e configuração da Microsoft Corporation. Possui grande penetração comercial em função de estar embutido nos produtos de desenvolvimento de software da Microsoft, tais como o Visual Studio, Visual Basic, Visual C++, Visual FoxPro, etc. Opera basicamente nos sistemas operacionais da Microsoft: Windows NT, Windows 9x, Windows Me, Windows 2000, porém possui versões para Macintosh e Unix, fornecidas por terceiros, MetroWerks e MainSoft, respectivamente.

O Visual SourceSafe armazena os arquivos e diretórios de projetos em repositório próprio sob formato de arquivos incrementais. Tanto para arquivos texto, quanto para arquivos binários[4], o método usado é o *armazenamento delta reverso*, no qual são armazenadas uma cópia completa da revisão mais atual do arquivo e as diferenças com as revisões anteriores. A função de mostrar diferenças entre revisões de arquivos é que é diferente nos dois casos: não é possível mostrar diferenças para arquivos binários.

Existe um tratamento especial para arquivos HTML. Para estes arquivos é possível identificar a integridade das ligações de Hiper-texto. Quando a ligação for para arquivos locais é constatada a existência do destino. Quando a ligação for para servidores externos é reportado um alerta para verificação manual posterior.

Uma implementação interessante do Visual SourceSafe é a opção de ser gerado um arquivo de auditoria (*journal*) que permite ao administrador ou um líder de projeto acompanhar as atividades ocorridas em um projeto por um período de tempo.

É possível realizar operações especiais com arquivos no SourceSafe, tais como: compartilhamento (*sharing*) de arquivos, pregar (*pinning*) uma versão de arquivo, criar ramificações (*branch*) e juntar (*merge*) versões de arquivos que foram ramificados. A

operação de compartilhamento implica no uso em projetos diferentes do mesmo arquivo básico. Caso o arquivo seja modificado, a nova versão será automaticamente visível, nos demais projetos. A operação de *pregar* uma versão de arquivo implica em congelar uma versão mais estável, por exemplo, para uso em outro projeto. A ramificação e a junção usam os conceitos convencionais deste tipo de operação.

Outro recurso do SourceSafe é o uso do conceito de Pasta Sombra (*Shadow Folder*). Este recurso permite gerar uma pasta com as versões mais recentes do projeto. Sempre que algum arquivo do projeto for modificado automaticamente a pasta sombra é também alterada. Este recurso permite que pessoas que não têm acesso normal ao SourceSafe possam examinar, ou mesmo compilar, arquivos do projeto sem ter acesso ao projeto em si.

3.9 Quadro comparativo das ferramentas de GCS

Uma comparação entre as ferramentas examinadas é apresentada na Tabela 3.2. As ferramentas foram avaliadas com graus entre 1 até 5, dentro dos critérios selecionados para a comparação (itens do *framework* de comparação apresentados na seção 2.9). O grau 5 corresponde a uma avaliação ótima, 4 a muito boa, 3 a boa, 2 a regular e 1 a ruim.

Considerando que a influência dos diversos critérios utilizados é aproximadamente equivalente um com os outros, podemos utilizar a média das avaliações dos diversos itens como uma base para uma avaliação sintética das ferramentas. Colocamos numa última coluna da tabela a ordem de qualidade das ferramentas, de acordo com estes critérios.

---

[4] A identificação do arquivo binário é feita pela constatação da existência do caractere h'00'. A identificação também pode ser decidida pelo usuário, explicitamente.

Ferramenta avaliada	Itens avaliados (*)						Ordem
	-1-	-2-	-3-	-4-	-5-	-6-	
Clear Case	4	3	4	5	5	4	2
Code Co-op	4	5	5	2	3	3	3
CVS	3	4	5	2	3	3	4
PERFORCE	3	3	4	3	3	3	6
PVCS	3	3	4	3	4	3	4
RCS	2	2	3	1	2	2	8
Star Team	4	4	4	5	5	4	1
Visual Source Safe	3	3	4	3	3	3	6

(*) Itens avaliados:

-1- Cooperação embutida na ferramenta

-2- *Awareness* do trabalho em equipe

-3- Apoio à dispersão geográfica da equipe

-4- Controle de mudanças

-5- Semântica dos modelos controlados

-6- Amigabilidade nos controles de projeto

Tabela 3.2 – Comparação dos produtos de GCS

Da avaliação apresentada concluímos que as melhores ferramentas, dentre as avaliadas, são o Star Team, o Clear Case e o Code Co-op. Num segundo nível estão o CVS e o PVCS. Ferramentas menos completas são o PERFORCE e o Visual Source Safe. O RCS é uma ferramenta bastante incipiente para os requisitos propostos, o que é natural tendo em vista o propósito com que foi criada e seu tempo de vida.

Capítulo 4

A importância da Colaboração no Desenvolvimento de Aplicações

O processo de desenvolvimento de software é cooperativo por sua própria natureza. Visto como um processo de aprendizado (ARMOUR, 2000), no qual os técnicos de sistemas e os futuros usuários do sistema em desenvolvimento devem entender a dinâmica do sistema, para modelar os seus objetos e características, para só então, construir a nova "encarnação" do mesmo, este processo deve obrigatoriamente contar com a operação conjunta, ou cooperação, de toda a equipe envolvida.

Neste capítulo faremos uma revisão dos sistemas de automação do fluxo de trabalho (*workflow*) e *groupware* sob o enfoque de utilização no apoio ao processo de desenvolvimento de sistemas, e mais especificamente, visando a gerência de configuração.

Os conceitos colocados neste capítulo foram obtidos, principalmente, em CHAFFREY (1998) e KHOSHAFIAN & BUCKIEWICZ (1995).

4.1 Introdução aos Sistemas Colaborativos

A forma de operar com sistemas está passando, hoje em dia, por uma transformação significativa. No modelo antigo, o individuo interagia com o computador para uma grande quantidade de tarefas apoiadas por sistemas computacionais e, quando precisava interagir com outras pessoas, usava recursos convencionais, como comunicação pessoal, reuniões, telefone etc. Na forma atual, tudo pode ser feito via computador. As pessoas interagem com os sistemas e outras pessoas usando os recursos computacionais de forma completa: solução das necessidades de automação, comunicação pessoal e interação com o grupo. Este novo paradigma, o dos sistemas

cooperativos, permite que os membros de uma equipe compartilhem idéias, informações, realizando suas tarefas de uma maneira muito mais eficiente. Evidentemente, a necessidade social de interação pessoal continua sendo satisfeita, e até potencializada, com os recursos deste novo paradigma. Isto porque os limites do afastamento no tempo e espaço são eliminados, ou pelo menos diminuídos, nesta forma de atuação.

No mundo dos negócios, este novo enfoque tem ocasionado verdadeiras revoluções, como o BPR – Reengenharia dos Processos de Negócios (*business process reengineering*), proposta por HAMMER (1993) ou programas de gerenciamento de qualidade total ou de gerência de processos.

Esta revolução está chegando também ao desenvolvimento de sistemas. As equipes de projetos de sistemas têm utilizado, em escala crescente, ferramentas e sistemas com recursos de cooperação.

4.2 Funções e Aplicações de Grupo

Sistemas de trabalho em grupo (*groupware*) auxiliam as pessoas a trabalharem em conjunto através de três funções básicas:

- **Comunicação** – é a característica central dos sistemas de trabalho em grupo, que permite o compartilhamento e o envio de informação a outras pessoas.

- **Colaboração** – é o ato de operar em conjunto na solução de um problema de negócio ou na execução de uma tarefa. Sistemas de trabalho em grupo podem reduzir os problemas das reuniões convencionais, tais como: encontrar um local e um horário apropriado, a falta de informação disponível ou, mesmo, o predomínio de um indivíduo sobre os demais, em uma reunião.

- **Coordenação** – é o ato de garantir que uma equipe está trabalhando efetivamente e atingindo os objetivos propostos. Isto inclui distribuir tarefas aos membros da equipe, rever o desempenho e outras tarefas similares.

Um resumo das aplicações de grupo é apresentado na Tabela 4.1. São apresentadas funções básicas de trabalho em grupo, as aplicações típicas e a utilização em Engenharia de Software.

Funções Básicas de *Groupware*	**Aplicações**	**Utilização em Engenharia de Software**
1. E-mail e envio de mensagens	Correio eletrônico, processamento de formulários	Comunicação assíncrona da equipe, Revisões assíncronas
2. Gerência de documentos e compartilhamento de informações	Disseminação de informação	Controle da documentação, Controle de versões
3. Co-autoria	Desenvolvimento de documentos em equipe	Diagramas e documentos em co-autoria
4. Conferência	Conferência em texto, Vídeo-conferência, Quadro branco	Revisões síncronas
5. Gerência do tempo	Agenda e calendário de grupo	Gerência de mudanças Controle de tarefas
6. Gerência de *groupware* e suporte à decisão	Uso remoto e distribuído de recursos	Gerência de configuração
7. *Workflow* ad hoc	Colaboração com acoplamento fraco	Controle de tarefas
8. *Workflow* estruturado	Gerência estruturada de tarefas	Gerência cooperativa de Configuração

Tabela 4.1 – Funções Básicas de *Groupware*

Em relação ao tempo e espaço, em que ocorrem os eventos, os sistemas de trabalho em grupo são classificados em quatro grandes grupos:

- **Tempo diferente e mesmo local:** Uso típico de e-mail e gerência de fluxo de trabalho de uma equipe atuando próxima.

- **Tempo diferente e local diferente:** E-mail, conferência de texto e gerência

de fluxo de trabalho para uma equipe dispersa.

- **Mesmo tempo e mesmo local:** Reuniões presenciais ou trabalho em equipe apoiada por "quadro branco".
- **Mesmo tempo e local diferente:** Reuniões síncronas e videoconferências

4.3 Sistemas Baseados no Fluxo de Trabalho

Sistemas ou ferramentas baseadas em fluxo de trabalho, ou *workflow* em inglês, são sistemas especializados na assistência ao trabalho cooperativo na consecução de processos de negócios. Um Sistema de Gerência de Fluxo de Trabalho é, segundo a WfMC – Workflow Management Coalition, "um sistema que define, cria e gerencia a execução do fluxo de trabalho através do uso de software, operando uma ou mais máquinas de fluxo de trabalho, as quais são capazes de interpretar a definição do processo, interagir com os participantes do fluxo de trabalho e, quando necessário, acionar ferramentas e aplicações de tecnologia da informação".

Exemplos clássicos de aplicações de fluxo de trabalho são os sistemas de processamento de tramitação de atendimentos, como apólices de seguro, itens de materiais, cadastramento de objetos em banco de dados, etc.

Sistemas de fluxo de trabalho fazem grande diferença para a eficiência operacional dos processos. As razões básicas são devidas à assistência aos gerentes na coordenação das tarefas executadas pelos membros da equipe e o conjunto de informações disponíveis para ajudar na execução das tarefas. O ganho se dá em tempo e custos.

Os benefícios principais de um sistema de fluxo de trabalho são:

a. **Maior eficiência do processo.** O processo se torna mais eficiente, porque um tempo menor é necessário para executar as tarefas e o custo é reduzido, com o aumento do volume de trabalho ou a redução da equipe.

b. **Obtenção de padronização do processo.** A automação do processo aumenta sua qualidade de serviço pela colocação de modelo de padronização das tarefas sendo executadas.

c. **Aumento da disponibilidade das informações.** A entrega direta, por meio eletrônico, das informações na estação de trabalho do usuário aumenta significativamente a disponibilidade das informações.

d. **Alocação automática de tarefas à equipe.** As tarefas são automaticamente alocadas a partir de uma fila de fluxo de trabalho, ao invés de uma alocação manual.

e. **Monitoramento automático do processo.** Isto é possível graças a ferramentas, tabelas e mapas de métricas mostrando a performance da equipe e dos indivíduos.

4.4 Análise e Modelagem de Processo

Ao desenvolver a análise e modelagem de sistemas baseados em processo, aliás como acontece para qualquer tipo de sistemas, devemos responder à pergunta do *quê* o sistema deve fazer? Na análise de sistemas de fluxo de trabalho precisamos especificar a forma como os processos de negócios devem funcionar. Isto inclui a especificação das peças de trabalho que a equipe deve completar e a ordem em que as mesmas precisam ocorrer.

Os principais métodos de definição de processo são: baseados em atividades; baseados em comunicação e orientados a objetos.

As informações necessárias para caracterizar um fluxo de trabalho são as seguintes e desenvolver um modelo adequado são:

- **Tarefas:** processos, atividades ou funções e detalhes das sub tarefas e o tempo para completar cada uma;
- **Entregas:** produto ou serviço entregue pelo processo;
- **Dependências:** entre as tarefas ou a seqüência das tarefas;
- **Recursos:** que irão executar as tarefas e os papéis ou responsabilidades dos mesmos

4.5 Projeto de Aplicações Colaborativas

A passagem da análise para a implementação exige um esforço de projeto, onde diversas etapas devem ser cumpridas. A Figura 4.1 apresenta uma visão geral da ligação entre a análise, projeto e implementação de sistemas de fluxo de trabalho.

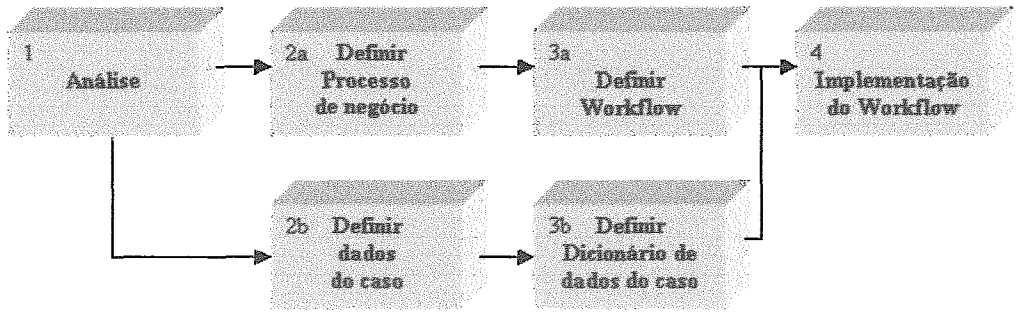

Figura 4.1 Modelo de projeto de sistemas de *workflow*

4.6 Implementação de Software Colaborativo

A implementação, o desenvolvimento final e a entrega do produto de software colaborativo se baseiam na execução adequada de tarefas técnicas e gerenciais. Fatores não técnicos extremamente importantes são a aceitação do sistema por parte dos futuros usuários e das mudanças que o mesmo ocasionará no ambiente de trabalho.

Uma das formas recomendadas de diminuir os problemas de aceitação, e também o conflito dos usuários com o novo sistema, é a implantação sob a forma de protótipo. A importância desta forma de abordar o problema se apresenta pelas seguintes preocupações:

- Testar que as definições do processo e as regras de negócios foram corretamente capturadas na análise e convertidas em processo executáveis na máquina de fluxo de trabalho;

- Testar se o volume do fluxo de trabalho é gerenciável pelos recursos humanos existentes;

- Verificar o desempenho no processamento das transações e no atendimento das pesquisas que geram as listas de trabalho. Pode o fluxo de trabalho ser processado pelo hardware e software disponíveis?

- Testar a integração dos diferentes aspectos dos dados. É fácil passar do fluxo de trabalho para o caso de dados?

- Permitir que os usuários sejam iniciados no novo sistema de uma forma gradual, tendo em vista que é muito freqüente que os mesmos não estejam acostumados à nova forma de trabalhar.

Sistemas e ferramentas de fluxo de trabalho, aplicadas aos próprios técnicos de desenvolvimento podem apresentar problemas adicionais. É muito comum haver uma resistência bastante grande a controles por parte dos analistas e programadores. Eles fazem a automação dos seus usuários, mas, muitas vezes, não gostam de melhorar os seus próprios passos. Um cuidado especial deve ser, portanto, dedicado ao conforto e facilidade do uso do sistema.

Capítulo 5

Uma proposta de solução para os problemas da GCS

Neste capitulo são apresentados conceitos gerais que permitam consubstanciar uma proposta adequada para um método de Gerência Cooperativa de Configuração de Software – a GCCS. Os conceitos básicos, já apresentados nos capítulos anteriores de forma embutida nas ferramentas pesquisadas ou colocados de forma didática, serão repassados neste capítulo com o intuito de formalizar a solução proposta.

Inicialmente, será apresentada uma visão global do método. Posteriormente serão detalhados os conceitos de gerência de configuração e software colaborativo. Por fim, será apresentada uma caracterização de uma ferramenta para apoiar o método.

5.1 Apresentação Geral do Método GCCS

A proposta básica do método GCCS – Gerência Cooperativa de Configuração de Software, é o de apoio ao desenvolvimento de sistemas de software que facilite, ou até obrigue (*enforce*), o trabalho em equipe no processo de desenvolvimento, garantindo um nível de qualidade elevada. Os pontos chaves da proposta são:

- Controle das versões dos objetos manipulados no desenvolvimento
- Gerência das mudanças ocorridas no desenvolvimento
- Controle sobre o processo de desenvolvimento
- Parametrização do nível de colaboração entre os membros da equipe

O Controle de versões é a peça fundamental do método. Desde as definições preliminares de um projeto de sistemas até sua implementação final são manipulados e gerados os mais diversos objetos, componentes do sistema alvo e do processo de

desenvolvimento deste sistema. O processo de desenvolvimento, conforme já ressaltado anteriormente, pode ser concebido como um aprendizado no qual examinamos uma parte do mundo real que está nos interessando particularmente – o **Sistema**. Analisamos e modelamos este sistema com a ajuda de ferramentas adequadas – construindo um **Modelo** do novo sistema. Finalmente, concebemos e projetamos uma nova encarnação para ele, construímos os elementos necessários para dar vida ao **Novo Sistema** e, finalmente, fazemos a avaliação e o teste da obra, para garantir que ela atingiu os objetivos propostos inicialmente.

O conjunto dos objetos que manipulamos no processo de desenvolvimento pode ser categorizado sob dois eixos distintos: a constituição física e a permanência após o processo de desenvolvimento. Segundo a constituição física podemos identificar dois tipos bem característicos de objetos. Aqueles que podem ser guardados em repositório de dados de uma ferramenta automatizada por computador e os que não permitem este armazenamento. No primeiro caso encontramos documentos redigidos e gravados em formato eletrônico, arquivos fonte e objeto de programas de computador, modelos de análise e projeto gerados em ferramentas de desenvolvimento, etc.. Todos eles conceituados como *arquivos* de computador. No segundo grupo encontramos documentos em papel, equipamentos, sistemas existentes e as ferramentas de compilação, edição, montagem, etc., e que não podem, ou não interessam, serem armazenados no repositório do projeto. Ambos os tipos devem ser guardados na ferramenta de controle de versões: os que possuem representação na forma de arquivos de computador, como versões de arquivos, e, os que não possuem esta representação, na forma de uma referência que permita encontrá-los no lugar correto e quando necessário.

De acordo com o outro eixo de referência, tanto os objetos que comporão o sistema alvo quanto aqueles que servem apenas como componentes do processo de desenvolvimento, deverão ser controlados pelo método. Numa analogia com a construção de uma obra de engenharia civil, pretendemos que, tanto o edifício em construção, quanto as fôrmas de concreto que sustentam as vigas e lajes durante a construção, sejam gerenciadas pelo sistema de controle de configuração.

O gerenciamento das mudanças que ocorrem durante o processo de desenvolvimento de sistemas é o segundo ponto chave do método proposto. Em alguns dos produtos avaliados encontramos este tipo de gerenciamento integrado com a gerência de configuração, mas, na maioria, não existe integração ou nem existe ferramenta para tal fim. Entendemos que a desvinculação das duas tarefas faz com que o controle de versões acabe caindo em desuso. Portanto, um ponto chave é a vinculação direta do controle de mudanças com a gerência de configuração. Esta forma de organizar o método obriga à vinculação das tarefas de desenvolvimento com o fluxo de trabalho, forçando a um nível de colaboração maior entre os membros da equipe de desenvolvimento.

Um modelo adequado para gerência de mudanças parece ser o proposto pela ferramenta ClearQuest da Rational. A proposta se baseia no processamento de pedidos de alteração de software, através de um fluxo de trabalho que irá gerar tarefas, hierarquizadas adequadamente até criar passos de trabalho, que possam ser assinalados a cada membro da equipe. Os estágios que devem ser acompanhados para cada solicitação de tarefa a ser executada são:

- Submissão do pedido de mudança
- Avaliação da mudança
- Decisão de implementação ou não da mudança
- Implementação das mudanças aprovadas, distribuindo as tarefas para a equipe
- Verificação da completeza do pedido de mudança
- Fechamento da solicitação de mudança

O controle de processo do desenvolvimento será implementado através das solicitações de mudanças que irão gerar tarefas no nível de granularidade compatível com o porte do projeto e o tamanho da equipe. A partir da definição das tarefas serão gerados eventos, que indicam a ocorrência do final de uma tarefa e, em geral, são pontos de convergência das atividades de mais de um participante da equipe de desenvolvimento.

O método proposto deverá permitir que a colaboração dos membros da equipe seja parametrizada de forma a garantir, em grau maior ou menor, a obrigatoriedade de participação das pessoas nas decisões e no trabalho de desenvolvimento. Estes parâmetros serão fixados pelo administrador de gerência de configuração.

5.2 Conceitos de GCS re-visitados

Apresentamos a seguir uma definição específica dos conceitos de gerência de configuração com o objetivo de registrar como estes conceitos serão usados e implementados na nossa proposta. Já que, na literatura e nos métodos e ferramentas analisadas, existem muitas diferenças em relação a estes conceitos.

Repositório – é um conjunto de diretórios e arquivos usados para guardar os dados de um ou mais projetos de desenvolvimento. São armazenados e disponíveis para manipulação em uma máquina servidora, que pode, eventualmente, ser a própria estação de trabalho do usuário, mas possui isolamento em relação à área de trabalho do mesmo. O GCCS permitirá que diversos repositórios sejam administrados, mas apenas um estará em uso em cada momento.

Área de trabalho – é o conjunto de diretórios e arquivos que o usuário utiliza na sua máquina para realizar as tarefas de criação, edição, compilação, montagem de arquivos com o objetivo de desenvolver o software. Constitui uma réplica do repositório, no ambiente computacional do usuário, dos objetos necessários para que o mesmo execute as suas tarefas dentro do projeto. Quando o usuário tiver realizado modificações em arquivos, normalmente ao final de uma jornada de trabalho, fará a sincronização da sua área de trabalho com os repositórios, enviando as modificações que fez e recebendo o que outros membros da equipe realizaram. A área de trabalho poderá ser removida após a sincronização com os repositórios, porque tudo que estava na mesma estará salvo nos repositórios. A área de trabalho pode ser pública, quando os

arquivos estão visíveis para outros colegas de desenvolvimento, ou privada, quando só a pessoa que está operando com os dados tem visibilidade dos mesmos. Na implementação inicial do método, a área de trabalho será considerada como pública com o objetivo de aumentar o grau de cooperação entre os membros da equipe. Um mapeamento completo entre o que foi retirado dos repositórios e a área de trabalho, permitirá que a ferramenta conduza o usuário na sincronização de sua área de trabalho com os repositórios.

Projeto – é a unidade lógica de desenvolvimento de software. Congrega todos os diretórios, arquivos e outros objetos de um processo de desenvolvimento num único repositório. Se mais de um projeto compartilhar arquivos, como é cada vez mais incentivado pela filosofia de reutilização de software, eles deverão ser duplicados em cada projeto para que se garanta a unicidade e integridade do mesmo. No futuro, pode-se tentar evitar esta duplicidade de objetos em diversos projetos.

Objeto – o *objeto de sistema* é todo componente de interesse para efeito de modelagem e desenvolvimento do sistema, definido dentro de um projeto. Inclui: diretórios; arquivos dos mais variados tipos, como planos, especificações, fontes, diagramas, textos, manuais, etc.; manuais e documentos em papel; equipamentos físicos; etc. Será feita uma distinção, visual e nas propriedades, daqueles objetos que estão armazenados no repositório do projeto em relação aos que estão fora do controle do repositório. Um objeto pode ser **original** ou **derivado**. Um objeto original constitui o objeto de definição ou especificação, numa cadeia de objetos vinculados por uma relação de especificação e construção. Um objeto derivado é aquele construído em função das definições contidas nos objetos originais a que está ligado[1].

Revisão – *revisão* de um objeto é uma instância de um determinado objeto que possui as mesmas características funcionais e de identificação, mas apresenta alguma alteração que a distingue das demais. É objetivo do método garantir que todas as revisões significativas sejam armazenadas em repositório, para que eventuais falhas no

[1] O modelo de ligações destes tipos de objetos será apresentado oportunamente no capítulo 6

processo de construção possam ser facilmente sanadas, e permitir que outros membros da equipe acompanhem as modificações sendo realizadas. São consideradas significativas as revisões concluídas pelo usuário, seja pela conclusão efetiva da tarefa, seja pela conclusão de uma jornada de trabalho. A identificação e o controle das revisões é realizado internamente pela ferramenta, porque o objetivo é meramente permitir a recuperação sistemática dos objetos colocados em controle de versão.

Ramo de desenvolvimento – é a possibilidade de construir revisões de objetos alternativos. Os objetivos da criação destas revisões podem ser um desenvolvimento alternativo, como, por exemplo, para línguas diferentes. Ou, pode-se construir ramos para corrigir algum erro de um módulo, sem interromper o fluxo regular de desenvolvimento. No segundo caso, principalmente, é bem provável que seja necessário juntar novamente os ramos para continuar com uma única linha de desenvolvimento.

Release – é uma liberação de um produto operacional. Todas as revisões mais atuais dos objetos usados no release, serão marcadas como vinculadas à mesma, permitindo que, a qualquer momento, se reconstrua o estado do projeto nesta posição. Um release constitui, portanto, uma versão liberada do software desenvolvido. A identificação do release será feita pelo administrador de configuração baseada em regras externas de denominação. Por exemplo, a release 5.7.2 atualiza o produto, cujo release anterior era 5.7.1, através de pequenas melhorias e acertos de problemas. Ou, o release pode ser identificado como 5.8.0, se a alteração tiver sido de porte maior.

Linha base – define um *release* de uma fase intermediária da construção do produto de software. Com o objetivo de garantir uma estabilidade mínima no processo de desenvolvimento, são criados marcos de projeto, onde os objetos que formam um conjunto coerente e íntegro de uma fase do projeto são *congelados* para permitir a execução consistente da fase posterior. A definição e identificação das linhas base estão subordinadas ao processo de desenvolvimento utilizado. Por exemplo, se o desenvolvimento for conduzido por protótipos sucessivos, cada linha base pode ser a liberação de um novo protótipo. Se for utilizado um ciclo de desenvolvimento linear, as fases de definição, especificação, projeto, codificação, testes e avaliação podem

constituir linhas base do desenvolvimento. Novos *releases* de uma linha base podem ser criados, mas a aprovação disto deve envolver níveis mais elevados da organização do projeto. Seja pela escalada a níveis superiores, no caso de uma organização hierarquizada, seja pelo aumento do número de pessoas a autorizar, no caso de uma equipe democrática.

Usuário da ferramenta – é uma pessoa física, identificada de forma única, e que constrói o software através da interação com o método e os repositórios, e todos os objetos contidos nos mesmos. O usuário pode ter diversos codinomes para acessos a serviços, senhas, endereços de correio eletrônico, estações de trabalho utilizadas, projetos alocados, papéis desempenhados, e outras características pessoais e funcionais. No entanto, deve sempre ser possível identificá-lo no seu relacionamento com os objetos de trabalho.

Papel – é o tipo de responsabilidade do usuário para cada objeto de um projeto. O método terá uma proposta básica de papéis a serem assinalados, mas o administrador poderá adaptar ao perfil específico de cada projeto. O conceito de papel utilizado não tem ligação com o controle de acesso dos usuários aos repositórios. O controle de acesso será definido, exclusivamente, em função do perfil do nome de usuário utilizado, quando for feito o acesso a algum repositório ou banco de dados. Os papéis básicos definidos inicialmente são:

- **Administrador do GCCS** – possui a atribuição de definir os parâmetros gerais de trabalho, atendendo às políticas e práticas aplicadas ao projeto

- **Líder de Projeto** – tem como função tomar decisões específicas dentro do seu projeto. Normalmente, este papel terá um representante dos usuários e um da equipe técnica de desenvolvimento

- **Usuário do Sistema** – tem a responsabilidade de conceber e aprovar os objetos do sistema, nos diversos estágios do desenvolvimento, em especial

na especificação e na implementação

- **Analista** – é o responsável pela concepção e especificação dos objetos do sistema, a partir da interação com o usuário do sistema

- **Projetista** – tem a responsabilidade pelo projeto (*design*) e detalhamento dos objetos originais do projeto

- **Construtor** – responsável pela construção ou implementação do objeto derivado de um objeto original de definição ou especificação.

- **Revisor** – é o responsável por fazer a revisão e/ou teste de um objeto, que tanto pode ser original ou derivado.

Um usuário pode desempenhar diversos papéis dentro do projeto, porém serão colocadas restrições para papéis desempenhados em relação a um objeto. Por exemplo, o construtor não pode desempenhar o papel de revisor dos seus próprios objetos.

5.3 A Colaboração no Processo de Gerência de Configuração

No método GCCS são usados diversos mecanismos de garantia da colaboração ou cooperação entre os membros das equipes de projeto. A sistemática de trabalho deve permitir a parametrização dos níveis de envolvimento ou participação das pessoas. As principais situações de envolvimento e cooperação são as seguintes:

- Qualquer manipulação de objeto, simultaneamente com outro usuário, deve gerar uma notificação síncrona, se os usuários estiverem com *visibilidade* entre si, através dos recursos de rede;

- Quando houver o conhecimento da manipulação de um objeto, mas o acesso direto não estiver disponível, será enviada uma notificação, através de uma mensagem de correio, de uma mensagem síncrona ou outro meio disponível;

- Advertência (*awareness*), através de uma comunicação síncrona ou o envio de uma mensagem de correio eletrônico, toda vez que um usuário da mesma equipe estiver trabalhando ativamente no projeto. Permite o estabelecimento de uma comunicação síncrona entre os usuários, se algum deles assim o desejar;

- Sempre que for criada uma revisão de algum objeto do repositório, será iniciada uma sessão assíncrona de avaliação do módulo. São revisores naturais do módulo: o usuário do sistema, o analista, o projetista e pelo menos um revisor independente. Outros construtores que já tenham feito alterações no módulo também são candidatos naturais a fazerem revisão. Deverá ser configurado o número ou percentual de usuários para cada módulo e o administrador deverá fixar este parâmetro para cada projeto. O coordenador do projeto específico será chamado a configurar adequadamente as pessoas específicas que farão a revisão, dependendo da ocupação atual de cada uma;

- Quando for iniciado um processo para criação de um novo release do sistema, todas as revisões, dos módulos vinculados a este release, devem estar aprovadas, como condição preliminar. Será, então, executada uma sessão síncrona para aprovação final do release, na qual todos os envolvidos podem participar e o administrador fixará os parâmetros mínimos para aprovação;

- A alocação de tarefas para solução de pedidos de mudanças será efetuada pelo coordenador do projeto, com a assistência direta de um processo automático, que irá informar a disponibilidade, a carga de trabalho e os papéis desempenhados pelas pessoas envolvidas. A criação de um fluxo de trabalho será a base para a solução dos problemas de controle do processo de desenvolvimento;

Uma das formas de acesso previstas no CVS prevê uma situação de acesso concorrente a um objeto sem nenhuma restrição ou aviso. Esta situação não é adequada,

do ponto de vista da cooperação entre os usuários. Portanto, sugerimos que este tipo de facilidade do CVS não seja utilizado. Sempre será realizada uma notificação, quando mais de um usuário estiver manipulando o mesmo objeto.

5.4 O Software Servidor de Gerência de Configuração

A definição adequada de um software servidor para a gerência de configuração é um dos componentes importantes do método GCCS. Inicialmente, a proposta era construir um servidor próprio. No entanto, algumas ponderações importantes nos levaram a selecionar um servidor existente ao invés de investir na definição e desenvolvimento de um outro servidor.

O ponto básico, é que este trabalho pretende aplicar cooperação ao processo de gerência de configuração, mostrando a importância desta abordagem. E não, simplesmente, construir mais um sistema completo de gerência de configuração. O desenvolvimento de uma solução completa demandaria um esforço bem acima das possibilidades deste trabalho. Inclusive, diversos pesquisadores, a quem a proposta deste trabalho foi apresentada, sugeriram exatamente que fosse concentrado esforço nos aspectos de cooperação, que estão bem mais próximos do método de trabalho do que na definição de software.

A sugestão é utilizar como componente servidor o CVS (CEDERQVIST, 1993). O CVS é um servidor de gerência de configuração bastante robusto, utilizado por uma comunidade bastante considerável e, o fator que mais pesou na decisão de usá-lo, é um produto *aberto* (*free*) de software – subordinado à licença GNU.

O objetivo do servidor de gerência de configuração é o de armazenar os objetos de desenvolvimento e suas diversas revisões, permitindo gerenciar as revisões de objetos de desenvolvimento e dos releases de sistemas.

5.5 A Necessidade de um Servidor Adicional de Banco de Dados

Uma série de objetos, do processo de desenvolvimento, está diretamente ligada ao ambiente de trabalho de cada usuário. Outros objetos estão vinculados ao relacionamento entre os usuários e não se referem diretamente aos objetos armazenados no servidor de gerência de configuração. Outros ainda, como a relação dos próprios servidores de gerência de configuração, não ficam armazenados nos mesmos. Estes tipos de dados devem ser armazenados em algum local com segurança e confiabilidade.

Muitas ferramentas utilizam arquivos de parâmetros para realizar a persistência destes dados. Como, por exemplo, o próprio cliente CVS armazena dados relevantes em um diretório chamado CVS, subordinado ao diretório principal da área de trabalho do projeto em edição.

Esta arquitetura é bastante frágil. Resolvemos, para melhorar este aspecto da questão, recomendar que o método lance mão de um sistema de gerenciamento de banco de dados adicional para realizar este armazenamento.

Um sistema gerenciador de banco de dados como o Interbase da Borland, foi utilizado nas experimentações realizadas, para suportar a persistência destes dados.meramente por uma questão de comodidade. O método e as ferramentas de apoio deverão permitir o uso de diversos SGBDs, como Oracle, MS-SQLServer, ACCESS, ou o próprio Interbase.

5.6 A Interação do Usuário com os Objetos de Desenvolvimento

O usuário da Gerência Cooperativa de Configuração de Software deve ter o apoio de métodos e ferramentas para interagir com o seu ambiente de trabalho, suas ferramentas de edição de diagramas e programas, os servidores de gerência de

configuração, o processo de desenvolvimento e de gerência de mudanças, e, principalmente, com o restante da equipe de desenvolvimento.

A proposta é que o usuário possa interagir com as ferramentas de gerência de configuração e de mudanças automaticamente através de um conector (*plug-in*) colocado nas ferramentas de edição usualmente utilizadas, como o editor do Rational ROSE, editor de linguagens como C++, Visual Basic, Delphi, etc.

Apesar de existir um apelo forte no sentido de realizar interações deste tipo usando recursos de acesso de navegador de Internet, como Internet Explorer ou Netscape Navigator, decidimos recomendar a implementação usando uma arquitetura cliente servidor convencional, através do ambiente Delphi 5 da Borland. A justificativa básica para esta opção é a necessidade de interação com muitos componentes simultaneamente: banco de dados relacional, servidor CVS, servidor SMTP, conexão TCP/IP direta, outros produtos de software de terceiros, como editores de diagramas e linguagens de programação, etc.

5.7 A Necessidade de um Agente Automático

Um componente adicional no ambiente de trabalho proposto para dar apoio ao método de GCCS é um agente automático que permaneça sempre ativo para executar tarefas rotineiras de coordenação e comunicação entre os diversos membros da equipe durante todos os trabalhos de desenvolvimento de projetos e os diversos componentes de dados. Será carregado automaticamente no início da sessão do usuário (a menos que o mesmo explicitamente iniba esta opção) e ficará residente na barra de ferramentas do sistema (*system tray*).

Os agentes inteligentes de software, de acordo com CASE (2001), devem apresentar algumas ou todas dentre as seguintes capacidades: cooperação, *proatividade* e adaptabilidade. Cooperação implica em comunicação com outros agentes para realizar

tarefas em conjunto. Proatividade (termo aportuguesado de *proactivity*) significa a capacidade de iniciar uma atividade sem o estímulo explícito do usuário e adaptabilidade permite que o agente aprenda com as experiências passadas e altere o seu comportamento nas situações encontradas.

O agente automático proposto terá capacidades bastante avançadas de cooperação e proatividade e algum grau de adaptabilidade nas funções de garantir a comunicação entre os diversos usuários e o usuário local com seus compromissos e tarefas. Todos os eventos marcantes do processo de desenvolvimento e de gerência de mudanças serão registrados e acompanhados pela agente, que fará as conexões e emitirá os avisos necessários.

As principais tarefas do agente de comunicação serão as seguintes:

- Sincronizar os bancos de dados dos diversos usuários e/ou máquinas ativas na rede;
- Registrar a localização atual de rede (nome da máquina ou endereço de IP) do usuário da máquina em todos os bancos de dados que estiverem disponíveis.
- Procurar algum usuário na rede de dados, sempre que seja necessário um contato com ele. Se o usuário procurado estiver ativo no momento, basta procurar no banco de dados de registro do mesmo. Caso contrário pode ser necessário realizar a pesquisa em diversas máquinas ou mesmo uma procura via correio eletrônico.
- Estabelecer e coordenar a comunicação síncrona entre os usuários.
- Enviar mensagens de correio entre os usuários e/ou máquinas para estabelecer contatos;
- Coordenar o registro das sessões de trabalho, síncronas ou assíncronas entre os usuários de uma equipe;

- Fazer a convocação dos usuários, acionada por um gatilho do sistema, para o estabelecimento de sessões de revisão síncrona ou assíncrona;

- Disparar avisos sonoros e/ou visuais nos eventos agendados de reuniões, início previsto de tarefas ou marcação de algum compromisso atribuído pelo sistema ou líder de projeto ou outro usuário habilitado;

Capítulo 6

Especificação detalhada da solução de GCCS

Será apresentado, neste capítulo, o detalhamento dos pontos relevantes de especificação do método de GCCS – Gerência Cooperativa de Configuração de Software. Sugestões de implementação e apresentação de algumas decisões de projeto para uma proposta de ferramenta de apoio ao método também serão examinadas. Um protótipo de uma implementação estará disponível como um complemento deste trabalho de tese.

A forma de apresentação da especificação detalhada será através de modelos parciais dos grupos de classes de objetos e menus e telas, que permitam apresentar a interação dos usuários com o método.

Na Figura 6.1 é apresentada a arquitetura geral do ambiente de gerência cooperativa de configuração de software. As diversas estações de trabalho servem de base para os usuários realizarem suas tarefas de desenvolvimento de software, com o uso do método de GCCS e apoio da ferramenta. O método GCCS é utilizado, em cada estação, sempre que o usuário realize interações com os repositórios de gerência de configuração (Servidor CVS). Os servidores de gerência de configuração serão servidores CVS, conforme explicado na seção 5.4 – O Software Servidor de Gerência de Configuração. Estas interações estão indicadas por linhas contínuas na figura. Os usuários precisam armazenar dados de trabalho, tais como projetos em desenvolvimento, repositórios onde estão localizados os projetos, atributos dos usuários envolvidos nos projetos, e outras informações que não fazem parte do escopo dos objetos administrados pelos repositórios. Tais informações serão armazenadas em bancos de dados de uso geral, que podem residir na própria estação do usuário ou em servidores próprios (BD Adicional). Um servidor de banco de dados pode atender a um ou mais usuários. Linhas pontilhadas na figura indicam este tipo de interação.

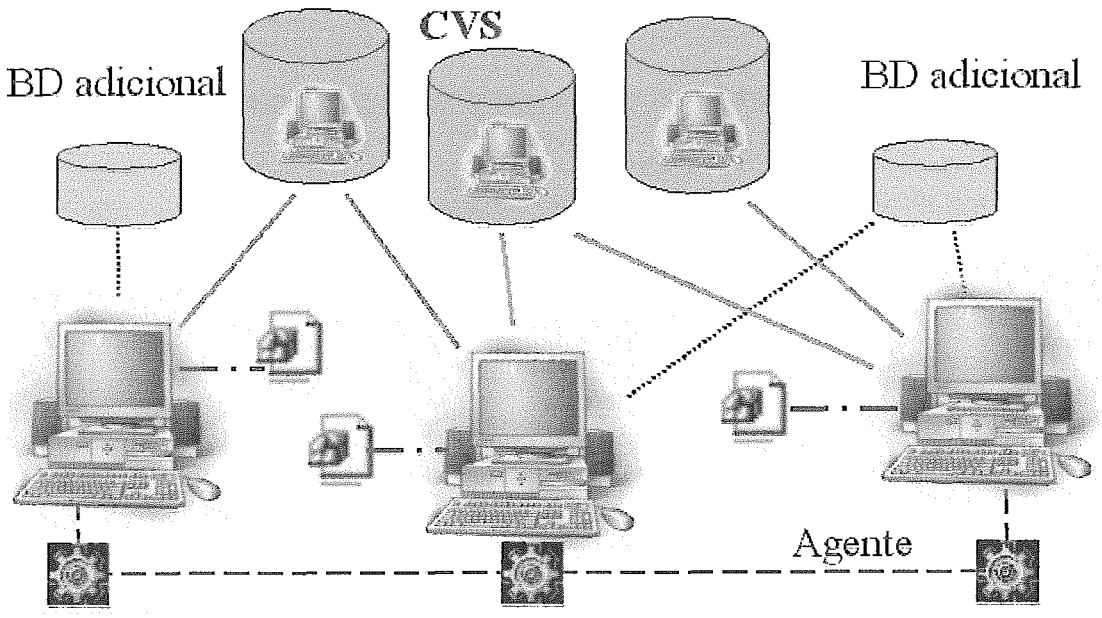

Figura 6.1 - **Arquitetura da ferramenta de GCCS**

Outro processo estará executando, em cada estação de trabalho, na forma de um *agente* automático, chamado AC – Agente de Cooperação, que garantirá a comunicação entre as máquinas e executará procedimentos automáticos de controle. Os agentes estarão em comunicação entre si através de um protocolo *Peer-to-peer* (estação a estação!) sem intervenção direta de servidor. O estabelecimento deste protocolo pode ser problemático quando alguma das estações de trabalho estiver colocada em uma rede privativa. Soluções para este tipo de problema estão sendo propostas em projetos como o Gnutella Next Generation (CLARK, 2001) que está desenvolvendo o GPulp (protocolo geral de localização de URL). A linha tracejada da figura indica a interação deste protocolo que, nas situações mais críticas, será executado via correio eletrônico.

Finalmente, determinados dados de relacionamento dos usuários e da ferramenta com a máquina específica que o usuário está usando num determinado momento, serão armazenados no sistema de registro (*registry*) do Windows, ou outro mecanismo local de persistência no caso de outro sistema operacional. Estas interações estão indicadas por linhas com traços e pontos na figura.

6.1 Os objetos do repositório de módulos

Os objetos de gerência de configuração serão armazenados no servidor CVS. O modelo de classes deve incorporar, no que for possível, a concepção implícita no repositório CVS, evitando a necessidade de persistência adicional no banco de dados local. A Figura 6.2 apresenta o modelo básico do repositório com os objetos que o constitui.

Um repositório pode incluir diversos projetos, mas cada projeto deve ser armazenado em um único repositório. Esta é uma das limitações do CVS que poderia ser contornada, mas implicaria em controles adicionais. Numa implementação futura poderia ser considerada uma melhoria neste aspecto.

As classes: *projeto*, *diretório*, *módulo* e *revisão* formam uma hierarquia típica da situação de agregação de classes. Repositório é uma agregação de projetos. Projeto é uma agregação de diretórios. Diretório pode agregar outros diretórios, de forma recursiva, e/ou módulos. Um módulo pode conter uma ou mais revisões.

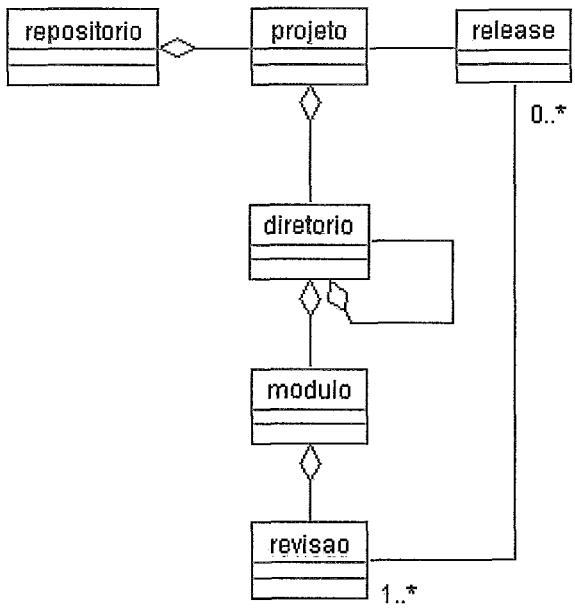

Figura 6.2 – Modelo do repositório

Um projeto pode estar associado a diversos *releases*. Um *release* está associado a diversos módulos, através de uma única revisão de cada um deles. Uma revisão de um módulo pode estar associada a nenhum ou a diversos *releases*.

Um release pode estar num estado de *programado* ou *efetivado*. O estado inicial é de programado e passará a efetivado quando as revisões de módulos que implementam o release forem associadas com o mesmo. Inicialmente, é obrigatória a previsão de, pelo menos, um release, no cadastramento de um novo projeto. Na seção 6.2, referente ao processo de desenvolvimento, será feita a vinculação do release com as tarefas e eventos do processo.

Os diversos repositórios devem ser cadastrados, para tornar transparente o acesso por parte dos usuários, em um painel semelhante ao apresentado na Figura 6.3.

As propriedades cadastradas do repositório são: a identificação, o caminho de rede para encontrar o repositório, o diretório raiz do CVS, a porta de acesso, se existir. O acesso remoto ao servidor CVS pode ser feito de várias formas, usando o serviço de segurança SSH, arquivo *rhosts*, arquivo local ou arquivo de *password*. Nesta primeira implementação será permitido o acesso através de arquivo de *password* ou arquivo local apenas.

Os projetos devem ser cadastrados no banco de dados local e, evidentemente, também estarão incluídos em algum repositório CVS.

Esta redundância se faz necessária para garantir o conhecimento prévio dos projetos controlados. Existe uma limitação no CVS, que exige o nome do projeto (diretório principal no CVS) para dar acesso aos objetos de um projeto. Um release de projeto também estará replicado no CVS e no banco local, pelo mesmo motivo.

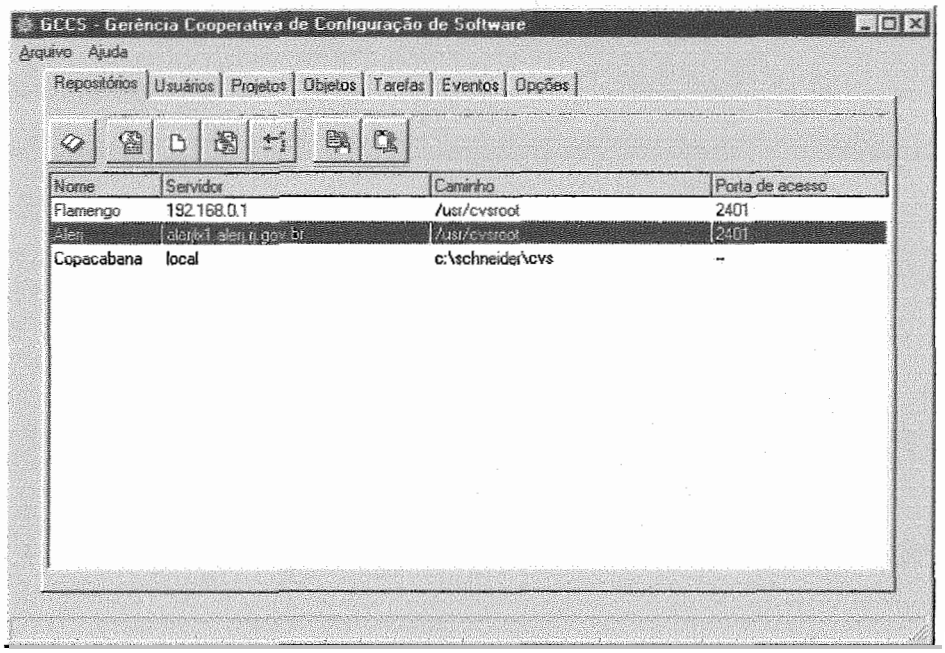

Figura 6.3 – Lista dos repositórios

A lista dos projetos, ver Figura 6.4, inclui dados como: a identificação, o nome, o repositório onde está armazenado e o usuário coordenador do projeto. Diversos parâmetros de configuração associados aos projetos, estarão vinculados ao mesmo.

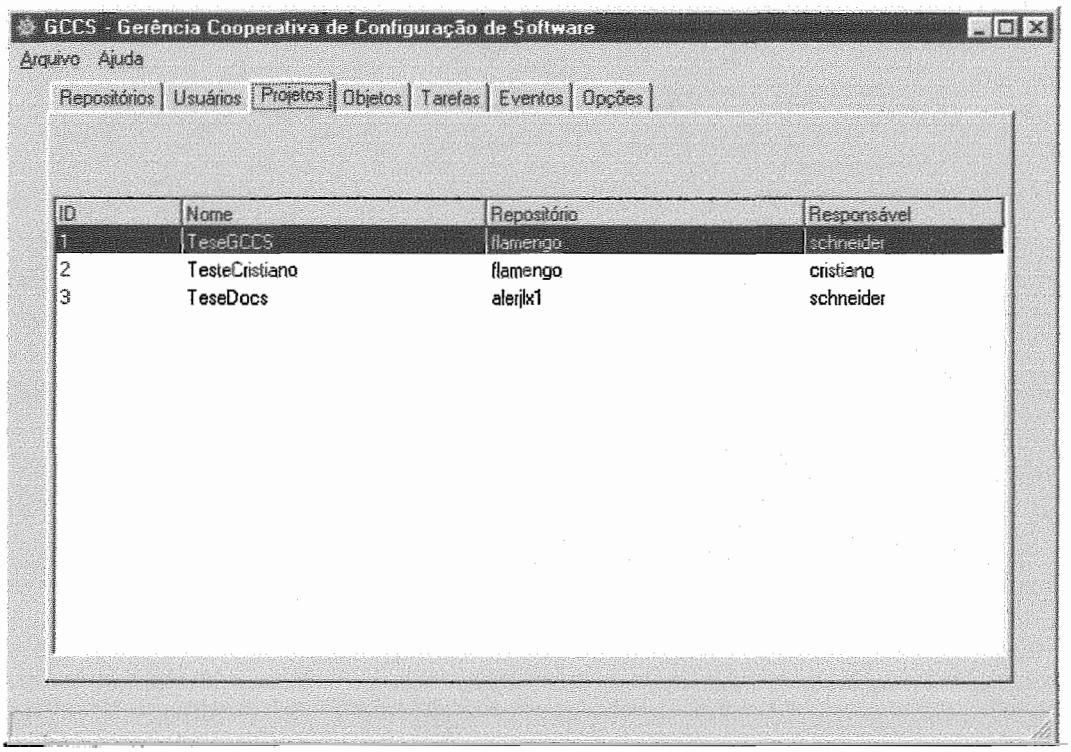

Figura 6.4 – Lista de projetos

Um release de projeto de projeto, conforme será detalhado mais à frente, estará associado a uma fase do processo de desenvolvimento ou a uma versão do produto em desenvolvimento.

6.2 O controle do processo de desenvolvimento

As tarefas executadas pelos usuários da ferramenta são os elementos chave de vinculação entre a gerência de configuração e o processo de desenvolvimento. Uma hierarquia de tarefas consubstancia o processo de desenvolvimento de um projeto, conforme apresentado na Figura 6.5. Cada *tarefa* pode ser desmembrada em outras tarefas até as tarefas elementares que possuem propriedades particulares.

Um projeto novo deve conter no mínimo duas tarefas, ao ser criado. Uma tarefa de desenvolvimento e uma tarefa de administração. À medida que módulos forem sendo criados novas tarefas surgem naturalmente.

A uma instância de *tarefa* podem ser associadas duas ou mais instâncias de *evento*, no mínimo o evento início e o evento fim da tarefa. Outros eventos, como a revisão da tarefa ou dos produtos gerados e/ou modificados pela mesma, podem ser definidos e acompanhados.

Uma tarefa pode estar associada a zero, um ou mais módulos. Esta ligação indica módulos que são *trabalhados* na tarefa. Podem existir tarefas, no entanto, que não geram ou modificam nenhum módulo. Um módulo deve sempre ter, pelo menos, uma tarefa que o implemente.

Ao alterar um módulo podemos desencadear alterações em outros módulos. Os quais, por sua vez, também deverão ter associadas outras tarefas para implementa-los. Este processo cíclico vai resultar numa rede de módulos e tarefas para trabalhá-los. Em especial, será importante acompanhar a cadeia de tarefas que implementam um ciclo típico de engenharia:

- Módulos que especificam características do sistema;

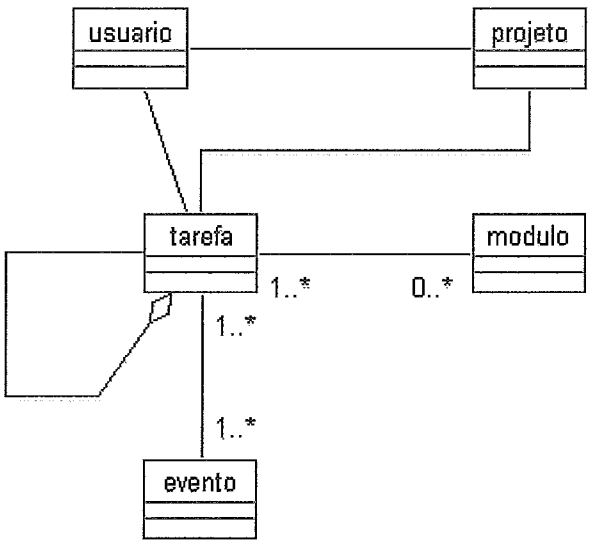

Figura 6.5 – Modelo de processo

- Módulos que detalham o projeto (*design*) de outros módulos;
- Módulos que implementam outros módulos;
- Módulos que testam a validade e veracidade de outros módulos.

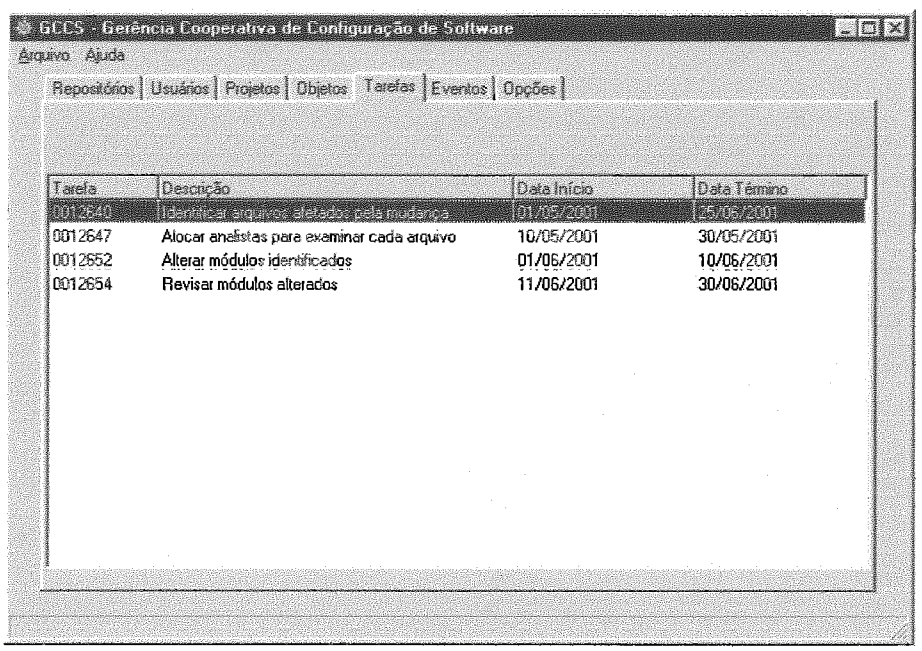

Figura 6.6 – Lista de Tarefas

Um modelo de lista das tarefas, relativas a um projeto, terá a aparência da Figura 6.6.

6.3 Os usuários do processo de desenvolvimento

Na Figura 6.7 apresentamos a modelagem dos usuários. Os *usuários* estão organizados em *equipes* naturais, definidas pelo envolvimento de trabalho com um projeto. Os usuários possuem papéis definidos dentro do projeto como um todo e em cada módulo ou mesmo revisão de módulo. Um usuário especial é o que inclui o projeto no repositório. Este usuário terá o papel de coordenador do projeto e executará uma série de tarefas privativas. Se houver necessidade, o coordenador poderá nomear um substituto para exercer este papel. Mas, somente o coordenador em exercício poderá modificar o status de algum usuário para o de coordenador.

Cada *revisão* de um *módulo* possui apenas um usuário como autor. É aquele que inclui a revisão no repositório. O módulo, então, como abstração relativa a várias revisões, poderá ter vários usuários responsáveis.

Os usuários da equipe de um projeto sempre podem ser alocados pelo coordenador do projeto, e isto será natural em uma estrutura hierarquizada de participação. Em projetos abertos, no entanto, os usuários podem candidatar-se a trabalhar nos mesmos, e serem aceitos ou não, com base em critérios especificados como parâmetro do projeto. Por exemplo, um usuário indicado por alguma pessoa precisaria da aprovação de pelo menos mais um outro usuário, para ser aceito. Já um candidato sem indicação alguma poderia precisar de 3 aprovações e nenhuma indicação contrária.

Uma lista dos usuários de uma determinada equipe de projeto pode ser vista na Figura 6.8.

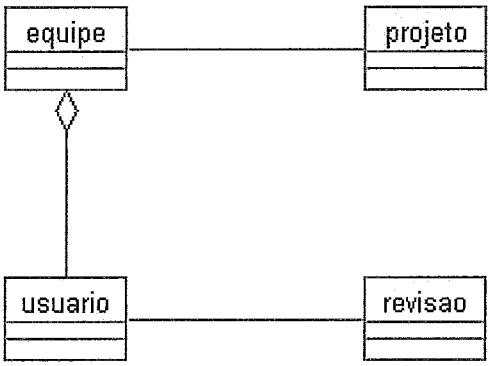

Figura 6.7 – Modelagem dos usuários

Os papéis previstos, por padrão, são os seguintes:

- Coordenador – é o *proprietário* do projeto. Possui atribuições únicas e é um papel privativo do nível de projeto, não existirá no nível de módulos;
- Analista – é o responsável pela tarefa de especificação de módulos;
- Usuário – responde por tarefas de especificação, testes e avaliações do projeto e de módulos junto ao analista;
- Projetista – se encarrega de tarefas de *design* do projeto e de módulos;
- Construtor – constrói efetivamente o código e outros registros de criação do software;
- Testador – encarregado por testar os documentos e módulos de um projeto e dos módulos do mesmo;
- Avaliador – realiza a avaliação de módulos, releases ou tarefas do projeto.

Outros papéis podem ser incorporados na parametrização especial de cada projeto.

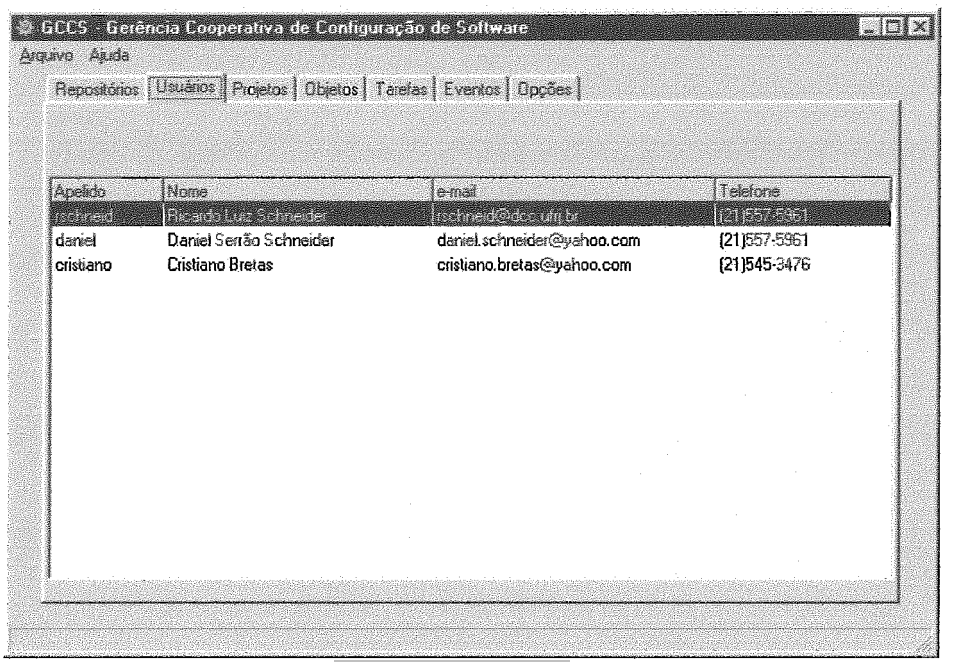

Figura 6.8 = Lista de usuários de uma equipe

6.4 A área de trabalho do usuário

A área de trabalho do usuário será montada, pela ferramenta, de forma similar à estrutura dos projetos no repositório. Cada instância de pasta, arquivo e revisão de arquivo poderá ter uma instância na máquina de cada usuário da equipe. Na Figura 6.9 é apresentado o modelo do repositório, re-visitado para realçar a posição dos objetos que fazem parte da área de trabalho. Estas instâncias são "clones" dos objetos originais do repositório. E, à medida que se diferenciem dos objetos originais, passarão a ser objeto do processo de sincronização com o repositório.

É importante salientar que os módulos podem estar planejados e ainda não existirem fisicamente na área de trabalho do usuário. Só se materializarão por ocasião da execução de sua tarefa criadora.

Os módulos podem estar em diversos estados diferentes, dependendo de sua situação em relação ao armazenamento no CVS, ao registro no banco de dados local ou em relação à área de trabalho local.

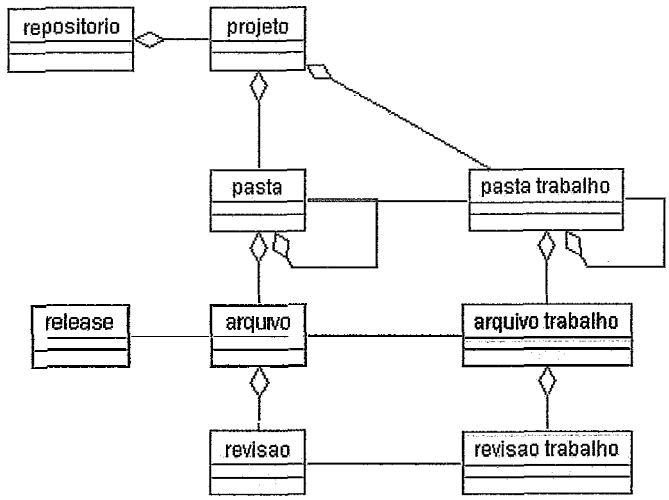

Figura 6.9 – Modelo da área de trabalho

A apresentação dos objetos na área de trabalho poderá ser visualizada pelas ferramentas normais de exploração e edição utilizadas pelo usuário.

As informações sobre os objetos da área de trabalho do usuário são armazenadas no banco de dados relacional. Quando necessárias para a interação com o repositório CVS, são também colocadas em disco, na área de trabalho normal do usuário para edição. Mas, neste último caso, de forma escondida do próprio usuário, por questão de segurança para evitar que o usuário possa sobrepor ou remover informações críticas.

A tabela 6.1 apresenta os estados dos módulos, correlacionando com as ações possíveis de serem executadas em cada estado.

Estado do Módulo	Caracterização do estado	Ações possíveis
Atualizado	Módulo do repositório e da área de trabalho são idênticos	Editar Diferença (com outra revisão) Adicionar a um *release* Remover do repositório Remover do local
Modificado no local	Módulo local é mais recente do que o módulo no repositório.	Editar Diferença (com o repositório) Diferença (com outra revisão) Desfazer edição Remover do repositório Remover do local
Removido do local	Removido do local sem	Editar (trazendo para o local)

	remover do repositório	Remover do repositório
Removido do repositório - existe no local - não existe	Módulo foi removido do repositório. Pode, ou não, estar disponível no local	Remover do local (se existir) Desfazer Remoção
Modificado no repositório - necessita *merge* - resolver conflito	A cópia do repositório é mais atual do que a local. Pode haver necessidade de *merge* ou resolução de conflitos	Atualizar local (fazendo *merge* ou resolvendo conflito)
Local	É um módulo local válido, mas ainda não foi incluído no repositório	Check in (adiciona e comete ao repositório)
Não controlado	É um módulo local cujo perfil (tipo de módulo) não interessa controlar	Check in
Planejado	É um módulo com previsão de criação futura. Não existe no repositório nem no local (somente no banco de dados)	Criar/Editar Remover do banco de dados

Tabela 6.1 – Estados dos módulos

6.5 As sessões de avaliação de tarefas e módulos

Cada tarefa do processo de desenvolvimento terá um evento de avaliação. Se a tarefa estiver associada a um módulo, haverá uma avaliação também deste módulo. Estas avaliações serão automaticamente iniciadas quando o usuário tentar concluir uma tarefa folha na árvore de tarefas ou um módulo. A primeira situação acontecerá quando o responsável por uma tarefa tentar digitar a sua data de término. E a segunda situação ocorrerá quando o usuário tentar colocar o módulo de volta no repositório (*commit*).

As sessões de avaliação e teste dos objetos de desenvolvimento serão registradas automaticamente pela ferramenta. Quando a revisão for de uma tarefa ou módulo será iniciada uma sessão assíncrona de validação do objeto alterado. Ou seja, os usuários selecionados para participar poderão enviar suas mensagens relativas à sessão de avaliação independentemente dos outros participantes estarem conectados ao sistema ou não. Na Figura 6.10 é apresentado o sumário de uma sessão deste tipo.

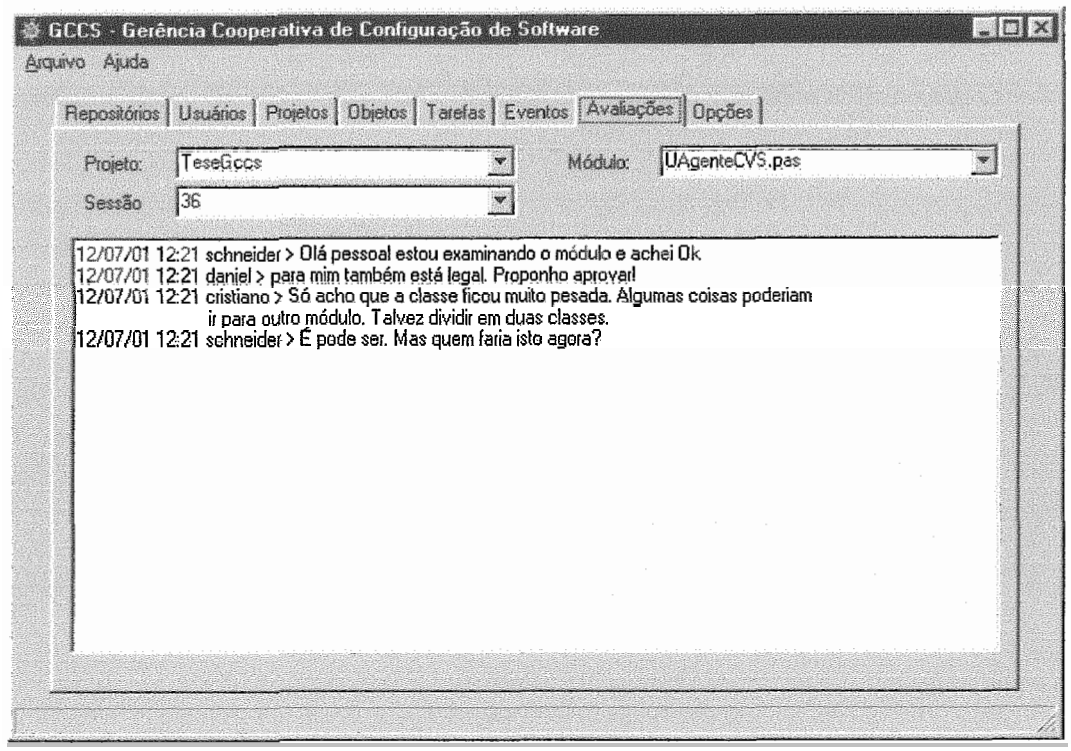

Figura 6.10 – Sessão assíncrona de avaliação de um módulo

Caso o objeto em alteração seja uma especificação ou projeto de algum módulo será obrigatória a execução de uma revisão formal do mesmo. Caso o módulo seja de implementação, além da revisão formal será exigido um teste individual e de integração.

Na Figura 6.11 são apresentadas as classes relativas às avaliações síncrona e assíncrona, relacionadas com as classes do repositório. Uma avaliação assíncrona será executada em relação a cada revisão inserida no repositório. Uma avaliação síncrona será realizada para cada release criado. Evidentemente a avaliação síncrona estará automaticamente vinculada às diversas avaliações assíncronas dos objetos que fazem parte do release criado.

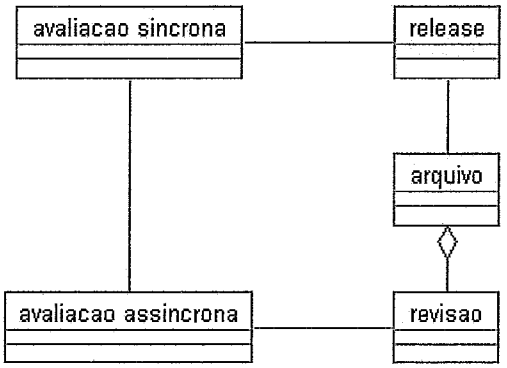

Figura 6.11 – Modelo das avaliações

Não será permitido criar um novo release do projeto, se existir algum objeto em modificação. Ou seja, todos os módulos, e também as tarefas vinculadas a estes, deverão estar *estáveis* quando for gerado um novo release. Este é um mecanismo importante de garantia (*enforcement*) da qualidade do desenvolvimento do software.

6.6 As alternativas de parametrização do método GCCS

Diversos parâmetros da ferramenta podem ser alterados pelo administrador ou pelo coordenador de projeto, quando afetarem exclusivamente um único projeto.

O acesso a estes parâmetros estará disponível numa aba específica sempre que o usuário conectado à ferramenta tiver autorização para tal.

Na Figura 6.12 é apresentado o menu de parâmetros disponíveis.

Cada projeto terá os seguintes parâmetros configuráveis:

- Periodicidade de controle – período de tempo em que a ferramenta solicitará que cada usuário relate o andamento dos módulos sob seu controle e tarefas em execução;

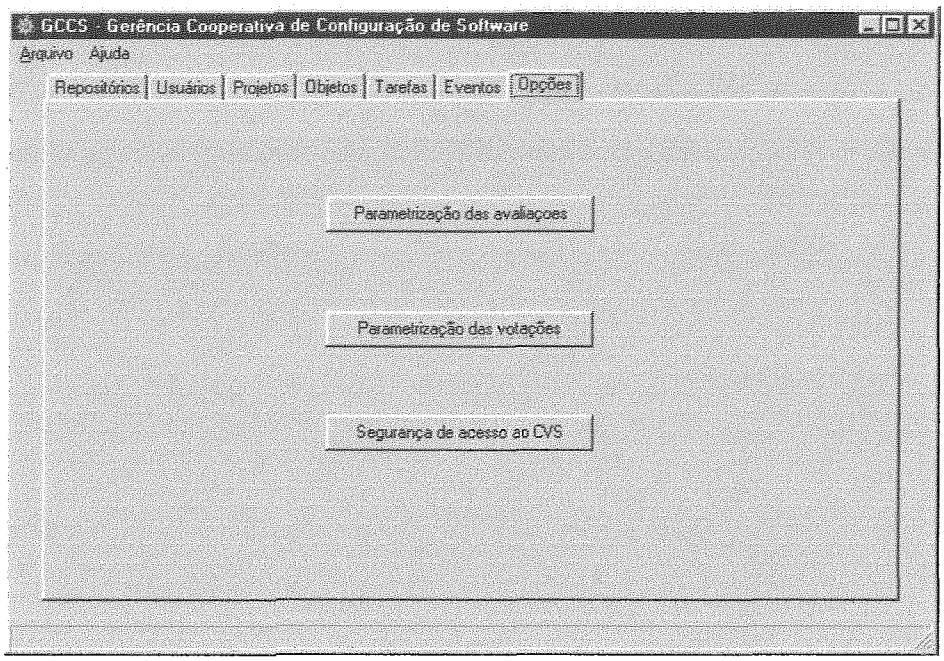

Figura 6.12 – Menu de parâmetros

- Data da execução do último controle – permite a execução periódica do controle do projeto;

- Tipo de projeto – caracterização do tipo de ambiente de desenvolvimento do projeto. Os tipos de arquivos controlados pela ferramenta serão associados a este tipo, e, principalmente, os tipos de arquivos não guardados no repositório (por padrão) serão identificados. Um exemplo de Tipo de Projeto pode ser o DELPHI - incluindo no repositório os arquivos terminados em .pas, .prj, etc. e não incluindo arquivos com sufixos: .~pas, .dcu, .exe, por serem arquivos de backup ou intermediários, só necessários para compilação, montagem ou execução.

Poderão ser fixadas regras com relação aos participantes das avaliações. Por exemplo, os responsáveis pelo desenvolvimento de revisões anteriores de um módulo serão os candidatos naturais para participar da avaliação da revisão seguinte. A escolha efetiva dos avaliadores será definida por candidatura e eventual seleção do coordenador do projeto, em situações especiais de falta ou de excesso de candidatos.

As votações, implícitas nas avaliações assíncronas ou síncronas, poderão ser parametrizadas em cada projeto. O critério, como regra geral, será exigir um rigor maior nas avaliações síncronas, por serem abrangentes e relativas à liberação de um release completo do produto de software, e menos rigorosas nas avaliações assíncronas.

Os parâmetros de votação poderão ser definidos como percentuais do número de participantes da equipe do projeto ou em números absolutos.

Capítulo 7
Conclusão

Neste capítulo apresentaremos as principais conclusões sobre o trabalho desenvolvido. Primeiramente, faremos uma revisão das propostas iniciais, comparando com o efetivamente realizado.

A seguir será feita uma apresentação das lacunas deixadas por este trabalho e a apresentação de justificativas desta situação, para que possam ser evitadas por outros pesquisadores que venham a realizar trabalho semelhante.

Finalmente, serão apresentadas algumas alternativas e sugestões para trabalhos futuros complementares ou semelhantes a este.

7.1 Os objetivos atingidos

Pretendíamos, no início deste trabalho, pesquisar a área de gerência de configuração, como componente importante da engenharia de software. A cooperação entre os membros das equipes de desenvolvimento deveria ser o mote principal da revisão dos conceitos de gerência de configuração. Acredito que este objetivo tenha sido atingido.

Uma avaliação das ferramentas existentes no mercado, tanto acadêmico quanto comercial, foi realizada com sucesso. Descobriu-se que existem muitas idéias e implementações interessantes, orientadas no sentido de facilitar o trabalho de manter as versões dos produtos de software sob controle, ligando cada vez mais estes esforços com um controle efetivo do processo de desenvolvimento, em particular pelo controle de mudanças. As sugestões implícitas nas ferramentas examinadas, como a hierarquização das tarefas do Clear Case, a obrigatoriedade de revisão dos módulos de

colegas de equipe do Code Co-op, foram incorporadas ao presente trabalho, sempre que possível e consistente com a linha mestra de projeto.

Co-relacionando, de forma mais detalhada, o trabalho realizado com os objetivos propostos inicialmente, podemos apresentar as seguintes constatações:

- Cooperação no processo de GCS – O método proposto e o protótipo de ferramenta implementado permitem tratar a cooperação entre os participantes das equipes de projeto de sistemas de uma forma similar às melhores ferramentas examinadas, tais como Star Team e Clear Case. Numa avaliação similar à que foi feita para as ferramentas analisadas no Capítulo 3, podemos avaliar o ambiente proposto com grau 4, na escala de 1 a 5, utilizada na Tabela 3.2;

- Consciência (*awareness*) do trabalho dos colegas – A solução apresentada neste trabalho permite que os usuários envolvidos em um projeto de desenvolvimento recebam informações sobre o andamento dos trabalhos, próprios e de seus colegas, garantindo um efeito moral na equipe, positivo para a condução eficiente dos projetos. Grau 5 na avaliação;

- Apoio à equipe dispersa geograficamente – Outro ponto relevante dos objetivos deste trabalho é o de facilitar que membros dispersos de uma equipe de desenvolvimento possam trabalhar de forma remota o mais facilmente possível. Este aspecto foi resolvido pela utilização de mecanismos de interação síncrona e assíncrona entre os membros da equipe e monitoramento da atividade dos usuários no projeto e outras características da proposta. Grau 5 na avaliação;

- Controle de Mudança – A ligação entre GCS e a administração do processo de mudanças é uma exigência natural devido à ligação forte entre estas atividades. Diversas ferramentas muito conceituadas de GCS não apresentam solução para esta situação, tal como o CVS por exemplo. A

solução deste trabalho inclui estes mecanismos de uma forma bastante inovadora, na medida em que procura ligar a gerência de mudanças com a gerência de projetos de uma forma mais ampla. Grau 4 na avaliação;

- Semântica dos modelos de desenvolvimento utilizados – Foi introduzido o conceito de Tipo de Projeto, que permite definir arquivos a serem administrados e arquivos a serem desprezados, o que permite uma seletividade maior no esforço de controle. Grau 3 na avaliação;

- Amigabilidade no Controle de Projetos – A sistemática de "criar" novas tarefas, à medida em forem sendo introduzidas mudanças no software em desenvolvimento, produz um efeito positivo e amigável no gerenciamento de projetos. Grau 4 na avaliação

A avaliação final da solução proposta neste trabalho (uma auto-avaliação evidentemente!) seria a equivalente à das melhores ferramentas examinadas. Similar ao Star Team ou Clear Case.

Portanto, foi atingido o objetivo importante de construir uma proposta de solução que pudesse aproveitar o máximo dos conceitos e experiências das ferramentas pesquisadas e que tivesse algumas implementações inovadoras, principalmente no que se refere aos mecanismos de obrigatoriedade de colaboração entre os membros da equipe de desenvolvimento de sistemas. Não o foi em maior escala devido a diversos fatores, onde se destacam a complexidade e dinamismo do assunto.

O auxilio de grupos de alunos foi importante para garantir uma implementação adequada. Em particular, o trabalho de final de curso de SCHNEIDER & BRÊTAS (2001) que ajudou a implementar uma parte da proposta aqui apresentada.

7.2 Limitações no trabalho atual

Existem oportunidades de melhorias na solução proposta e em especial na implementação realizada. Por motivos pragmáticos, deve ser reconhecido que muitas pretensões iniciais não foram alcançadas. Reduzimos significativamente o escopo do projeto ao decidir utilizar um servidor de gerência de configuração (o CVS) como o servidor da ferramenta. Boas oportunidades se perderam com esta simplificação e, além disto, trabalhos adicionais tiveram que ser executados para adaptar a ferramenta desenvolvida às idiossincrasias do servidor escolhido.

A implementação em diversos gerenciadores de bancos de dados diferentes não foi possível realizar. Basicamente, o problema do tempo e esforço necessários para estas extensões impediu que isto pudesse ser realizado no escopo desta tese. Apesar de muitas ferramentas propalarem a adesão a padrões ANSI de bancos de dados, a realidade não é bem assim. Todo o processo de geração, principalmente dos metamodelos, é bem diferente nos diversos sistemas gerenciadores de bancos de dados e torna-se bastante trabalhosa a adaptação das aplicações para funcionarem em diversos ambientes.

A implementação da proposta ainda está numa fase de protótipo, pelo menos para diversas funcionalidades, e haveria necessidade de um esforço de alguns meses para depurar e melhorar o desempenho do ambiente.

A ferramenta deveria ser portável para outros sistemas operacionais, como o Linux, Unix ou Macintoch. Estas implementações ainda não foram realizadas.

7.3 Trabalhos futuros

Completar as lacunas relatadas na seção anterior, evidentemente, configura-se como a parte mais óbvia de trabalhos futuros complementares a este. Trata-se de tarefa complementar a ser encetada no âmbito dos trabalhos de pesquisa e

desenvolvimento conduzidos pelo autor deste trabalho. É objetivo realizar estas implementações no âmbito das disciplinas de graduação, ministradas pelo autor.

Outros trabalhos complementares podem, no entanto, ser conduzidos por outros pesquisadores, tanto em nível de projetos de final de curso de graduação, quanto de teses de mestrado.

Uma pesquisa de campo sobre a utilização de técnicas e ferramentas de gerência de configuração no mercado brasileiro, é um ótimo exemplo de extensão possível ao presente trabalho.

Dentro desta linha ainda, um estudo sobre os tipos de ferramentas usadas e a eficiência das mesmas na solução dos problemas que potencialmente a gerência de configuração pretende resolver, seria mais uma sugestão de trabalho a ser desenvolvido.

Um aprofundamento sobre a ligação da gerência de configuração com a gerência de mudanças também é outro assunto potencialmente de bastante interesse.

Capítulo 8
Bibliografia

ARMOUR, P., 2000, "The Five Orders of Ignorance". **Communications of the ACM.** v. 43, n. 10 (Oct). pp. 17-20.

BABICH, W., 1986, **Software Configuration Management**. New York, Addison Wesley.

BUCKLE, K. et al., 1982, **Software Configuration Management**. London, MacMillan Education Ltd.

CASE, S. et al., 2001, "Enhancing E-Communities with Agent-Based Systems". **Computer**, v 34, n. 10 (Jul). pp. 64-69.

CEDERQVIST, P. et al., 1993, **CVS – Concurrent Versions System – Version Management with CVS for CVS 1.10**. New York, The Free Software Foundation. Disponível em: <http://www.cyclic.com>. Acesso em: 12 fev. 2001.

CHAFFREY, D., 1998, **Groupware, Workflow and Intranets – Reenginerring the Enterprise with Collaborative Software**. Butterworth-Heinemann. Woburn, MA.

CLARK, D., 2001, "Face-to-Face with Peer-to-Peer Networking". **Computer**, v 34, n. 1 (Jan). pp. 18-21.

FIELDING, R., 1999, "Shared Leadership in the Apache Project". **Communications of ACM**, v 42, n. 4 (Apr). pp. 42-43.

GABRIEL, C. & SCHNEIDER, D., 2001, **Uma proposta de Ferramenta de Gerência Cooperativa de Configuração de Software.** Dissertação de Bacharelado, UFRJ / IM / DCC, Rio de Janeiro, RJ, Brasil.

GINTELL, J., 1995, "Requirements for a CSCW System for Software Development Organizations", **In: ECSCW 95 Workshop**, pp. 4. Stockholm, Sweden, Aug.

HAAKE, A., & HAAKE, J., 1993, "Take CoVer: Exploiting Version Support in Cooperative Systems", **In: Proceedings of the INTERCHI'93**, pp. 406-413. Amsterdam, The Netherlands, Apr.

HAMMER, M. & CHAMPY, J., 1993, **Reengineering the Corporation: A Manifesto for Business Revolution**. New York. Harper Collins.

HERBERT, J., 1999, **Teste Cooperativo de Software**. Tese de D.Sc., Instituto de Informática/UFRS, Porto Alegre, RS, Brasil.

KHOSHAFIAN, S. & BUCKIEWICZ, M., 1995, **Introduction to Goupware, Workflow, and Workgroup Computing**. Boston. John Wiley & Sons, Inc.

LYON, D., 2000, **Practical CM – Best Configuration Management Practices**. 2 ed. Oxford, UK. Butterworth Heinemann.

MAIDANTCHICK, C., 1999, **Gerência de Processos de Software para Equipes Geograficamente Dispersas.** Tese D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

MERANT., 2000, **PVCS© Version Manager – User's Guide**. Mountain View, California, Merant, Inc. Disponível em:< http://www.merant.com/pvcs>. Acesso em: 10 fev. 2001.

MEYER, B., 1997, **Object-Oriented Software Construction.** 2 ed. Santa Barbara, California, USA, Prentice Hall.

MICROSOFT Co., 1996, **Visual SourceSafe - Project-Oriented Version Control User's Guide.** USA. Microsoft Corporation.

MURTA, L., BARROS, M. & WERNER, C., 2000, "Token: Uma Ferramenta para o Controle de Alterações em Projetos de Software em Desenvolvimento". **In: XIV SBES – Simpósio Brasileiro de Engenharia de Software,** pp. 383-386, João Pessoa, Out.

NOMURA, T., et al., 1998, "Interlocus: Workspace Configuration Mechanism for Activity Awareness". **In: Proceedings of CSCW 98,** pp. 19-28, Seattle, Washington, Nov.

NUMAMAKER, J., 1999, "Collaborative Computing: The Next Millenium". **Computer,** v. 32, n. 9 (Sep), pp. 66-71.

PRESSMAN, R., 2001, **Software Engireering - A practitioner's Approch.** 5 ed. Boston, The McGraw-Hill Companies, Inc.

RAM, S., & RAMESH, V., 1998, "Collaborative Conceptual Schema Design: A Process Model and Prototype System". **ACM Transactions on Information Systems,** v. 16, n. 4 (Oct), pp 347-371.

RELIABLE Software, 2001, **Code Co-op. The Server-Less Version Control System for Collaborative Development.** Relible Software. Disponível em: <http://www.relisoft.com/co_op>. Acesso em: 10 fev. 2001.

SCHNEIDER, R. & BORGES, M., 2000, "Gerência Cooperativa de Configuração de Software". **In: V WTES – Workshop de Teses em Engenharia de Software**, pp. 417-420, João Pessoa, Paraíba, Out.

TEIXEIRA, H., MURTA, L. & WERNER, C., 2001, "LockED: Uma Ferramenta para o Controle de Alterações no Desenvolvimento de Distribuido de Artefatos de Software". **In: XV SBES – Simpósio Brasileiro de Engenharia de Software**, pp. 380-385, Rio de Janeiro, RJ, Out.

VASUDEVAN, A., 2001, **CVS-RCS How-to document for Linux. - Source Code Control System**. The Free Software Foundation. Disponível em: <http://www.linuxdoc.org/HOWTO/CVS-RCS-HOWTO.html>. Acesso em: 10 fev. 2001.

WHITE, B., 2000, **Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction**. Boston, Addison-Wesley.

WU, M & LIN, Y., 2001, "Open Source Software Development: An Overview". **Computer**, v. 34, n. 6 (Jun), pp. 33-38.

Apêndice A

Exemplo de uso da ferramenta

Como apêndice a este trabalho de tese, será apresentado um exemplo de uso do protótipo da ferramenta de Gerência Cooperativa de Configuração de software.

Estão apresentadas as situações mais típicas de utilização enfrentadas pelos membros de uma equipe de desenvolvimento de sistemas. Desde a definição de um novo projeto, até a conclusão da avaliação e entrega de um módulo pronto para o repositório.

A.1 Iniciando o uso da ferramenta

Estando a ferramenta previamente configurada em uma estação de trabalho e nos servidores CVS e de banco de dados, devemos iniciar a operação comandando a execução do programa GCCS.EXE. A Figura A.1 apresenta a tela de boas vindas do sistema e solicita a confirmação ou entrada dos dados sobre identificação do usuário e sua senha no sistema.

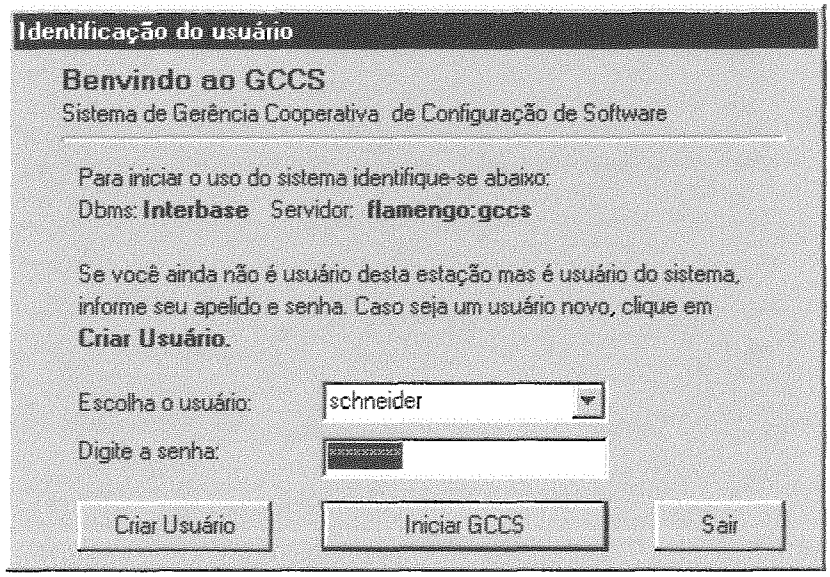

Figura A.1 – Tela de boas vindas

O usuário pode alterar ou confirmar a sua identificação na ferramenta, fornecer a senha adequada e clicar em "Iniciar GCCS" para ter acesso ao sistema, conforme detalhado na Figura A.2 – Detalhes do preenchimento da tela inicial. A lista apresentada na tela é a lista dos usuários que já usaram a estação corrente. O nome selecionado é o do usuário que utilizou o sistema na sessão anterior.

Se o usuário estiver habilitado no banco de dados, com a senha fornecida, passará para o uso normal da ferramenta.

Na primeira vez que o usuário for utilizar o sistema, deverá usar o botão "Criar Usuário", para iniciar o sistema criando o seu perfil de acesso.

O botão "Sair" permite, evidentemente, que o operador abandone o uso da ferramenta.

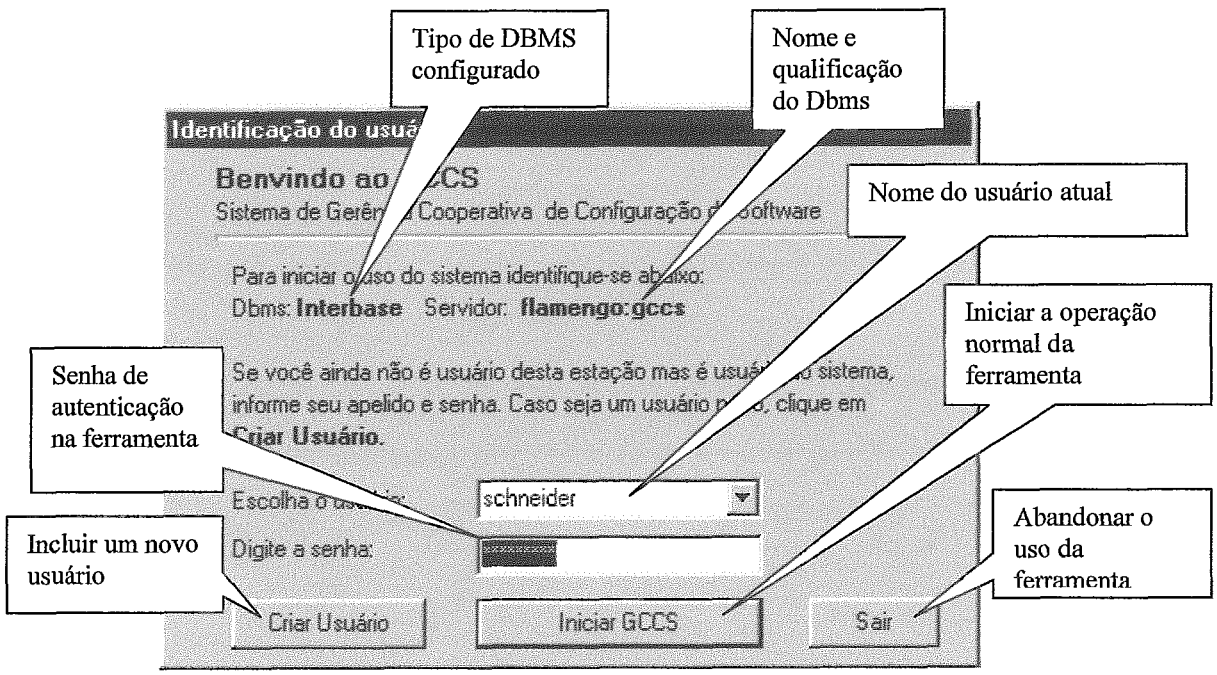

Figura A.2 – Detalhes de preenchimento da tela inicial

Ao entrar no sistema, o usuário receberá a tela mostrada na Figura A.3 – Tela inicial da ferramenta. Esta tela é a primeira de uma série de telas similares que apresentam a interface do operador com a ferramenta. É constituída de quatro partes

básicas, apresentadas em detalhes na Figura A.3: Área de Menus; Informações gerais de contexto, como nome do projeto em administração, diretório local de armazenamento e status da comunicação com o servidor CVS; Área de painéis de controle da ferramenta e Área de Status da ferramenta em geral.

A tela inicial apresenta na área de painéis dados globais do projeto em operação. Inicialmente o projeto em operação é o último projeto trabalhado na sessão anterior.

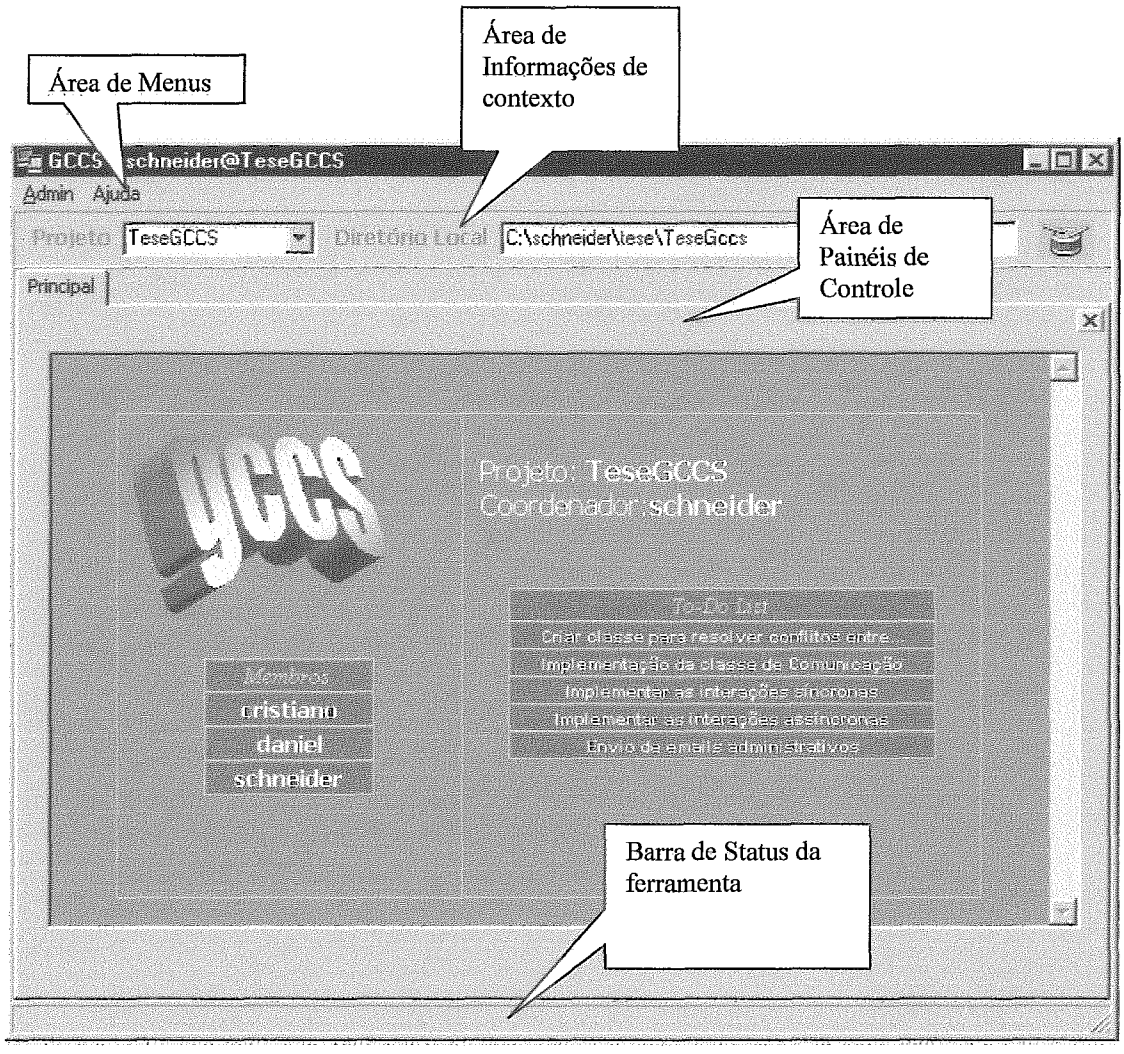

Figura A.3 – Tela inicial da ferramenta

A.2 Criando um novo projeto de desenvolvimento

Para criar um novo projeto, selecionar o Menu Admin / Gerenciar / Projetos, conforme mostrado na Figura A.4 – Menu do painel de projetos.

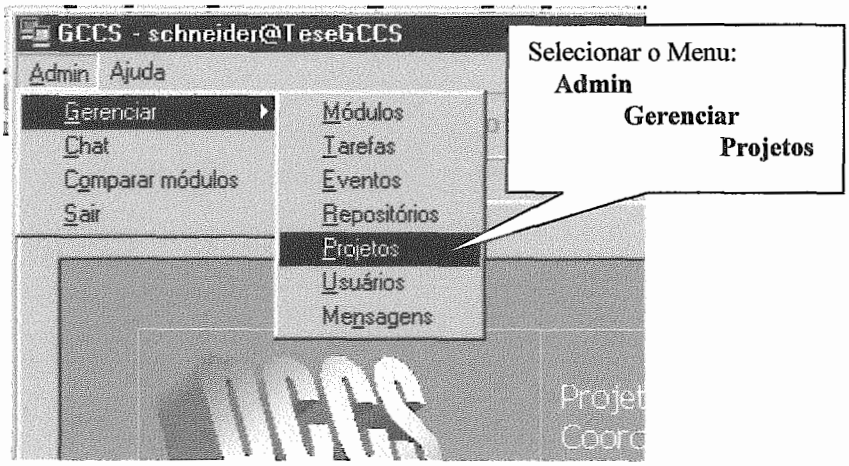

Figura A.4 – Menu do painel de projetos

No painel de projetos, selecionar o botão "Incluir" novo projeto e preencher os dados, conforme mostrado na Figura A.5 – Incluir novo projeto. Os dados básicos do novo projeto são:

A) Dado cadastral básico: **Nome** do projeto;

B) Localização do projeto na estação de trabalho: o **Diretório Local** onde os arquivos estarão armazenados;

C) Repositório CVS de colocação do projeto: **Repositório**;

D) Equipe do projeto: **Coordenador**; **nome** e **papel** dos demais participantes;

E) Tipo do projeto: **nome**; **módulos normais**, que obrigatoriamente serão controlados; **módulos a excluir**, que serão excluídos do controle por serem redundantes;

F) Informação sobre o controle do projeto: **Data último controle**. Permite definir a próxima data de avaliação, dentro da hipótese atual de periodicidade semanal;

G) Parâmetros de Aprovação de cada Módulo nas validações/verificações: **% revisores / total de usuários; % aprovações / número de revisores;**

H) Parâmetros de Aprovação de um novo RELEASE: **% revisores / total usuários; % aprovações/ número revisores; % de módulos revistos** do total de módulos do release;

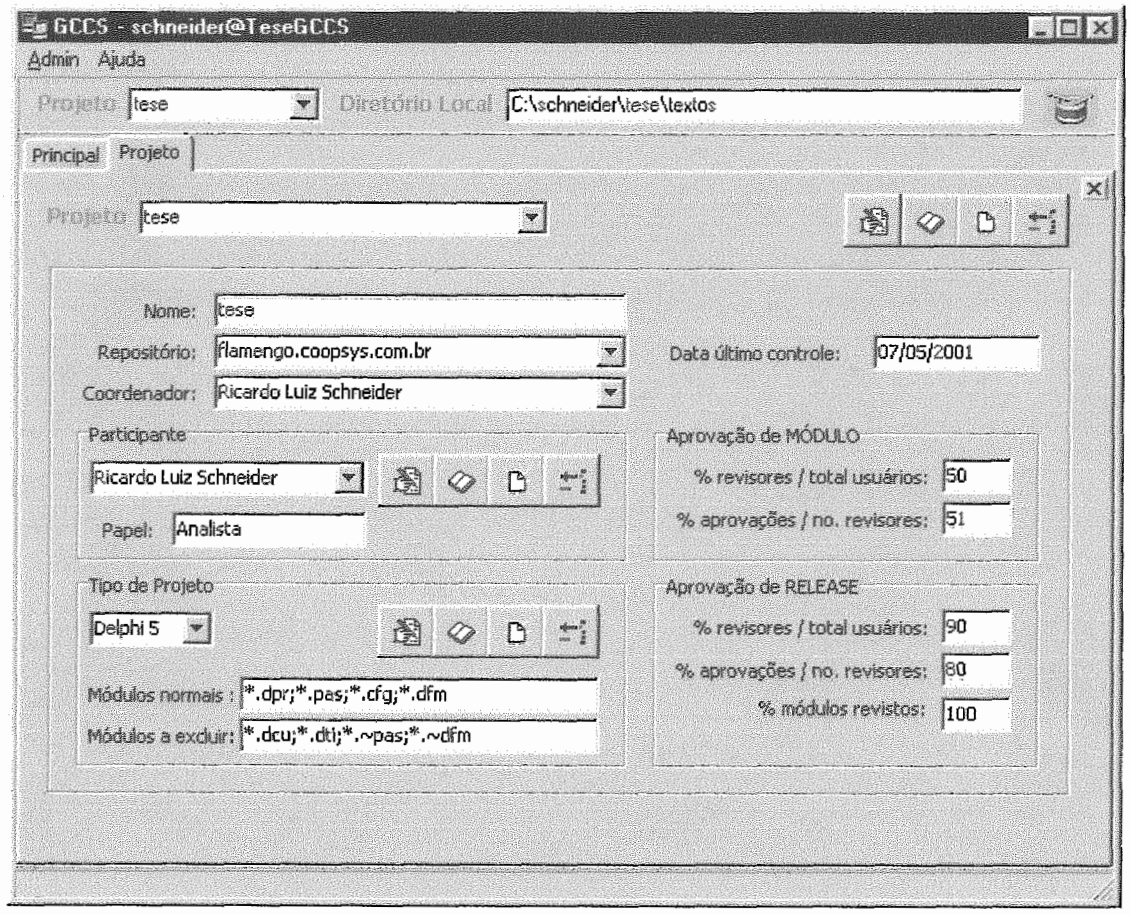

Figura A.5 – Inclusão de um novo projeto

A.3 Incluindo módulos

O próximo passo dentro de um projeto novo é criar arquivos com dados relevantes para os objetivos do mesmo.

A criação dos arquivos em si será realizada com o apoio de algum editor especializado no tipo de documento gerado. Editor da ferramenta de programação, se

o arquivo for um programa fonte, ou um editor de diagramas ou editor de textos, etc.

Uma vez criado o arquivo, devemos colocá-lo sob a gerência de configuração. Isto será feito usando a ferramenta. Selecionar o Menu **Admin / Gerenciar / Módulos** e aparecerá a tela com o painel de controle de módulos posicionado no diretório local indicado no cadastramento (C:\schneider\tese\textos). Ver Figura A.6 – Cadastrar módulos.

No exemplo, os arquivos **Tese.ppt** e **Tese00.doc** até **Tese07.doc** já estão atualizados no repositório (Status **Up-to-date**) e o arquivo recém criado **TeseApendiceA.doc** está com status desconhecido (*Unknown*). Para incluir este arquivo no repositório devemos clicar com o botão direito do mouse sobre o item e selecionar o Menu **Adicionar (Add)** e **Binário**, por se tratar de arquivo binário (MS Word!)

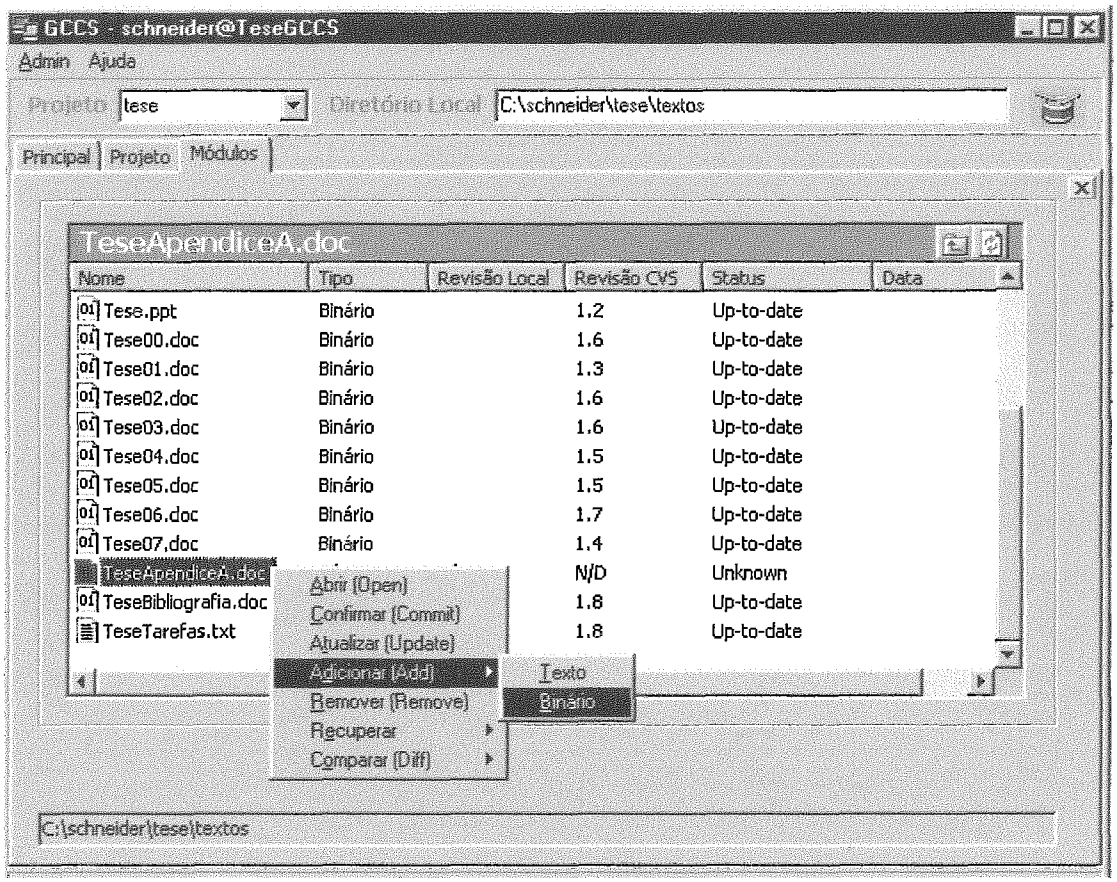

Figura A.6 – Cadastrar módulo

Após esta operação, o arquivo terá sido incluído no repositório CVS, com o status de **Adicionado localmente** (*Locally Added*).

Neste ponto a inclusão do módulo ficará suspensa até ser realizada a inclusão de uma tarefa para desenvolver o módulo recém criado. No item A.4 a seguir será visto como cadastrar uma tarefa associada a um módulo.

Depois de cadastrada pelo menos uma tarefa associada ao módulo em criação/alteração, poderemos iniciar o processo de avaliação do módulo para finalmente poder realizar a confirmação do módulo no repositório, através da operação de **Confirmar**, conforme pode ser visto na Figura A.7 – Confirmar inclusão de módulo. No item A.5 será visto como concluir a entrega de um módulo.

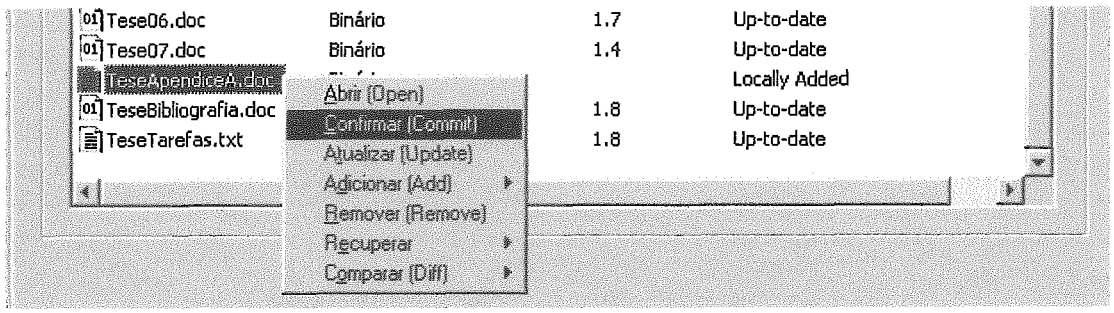

Figura A.7 – Confirmar inclusão de módulo

A.4 Definindo tarefas

Automaticamente, quando for criado um novo módulo, ou quando chamarmos o Menu: Admin / Gerenciar / Tarefas será acionado o painel de cadastramento de uma nova tarefa, conforme mostrado na Figura A.8 – Criar Tarefa.

Além dos dados cadastrais próprios de tarefas, como nome, tarefa superior, tarefas inferiores, data de início, etc., devemos incluir dados sobre os usuários que irão executar a tarefa e dados sobre os módulos a serem alterados pela mesma.

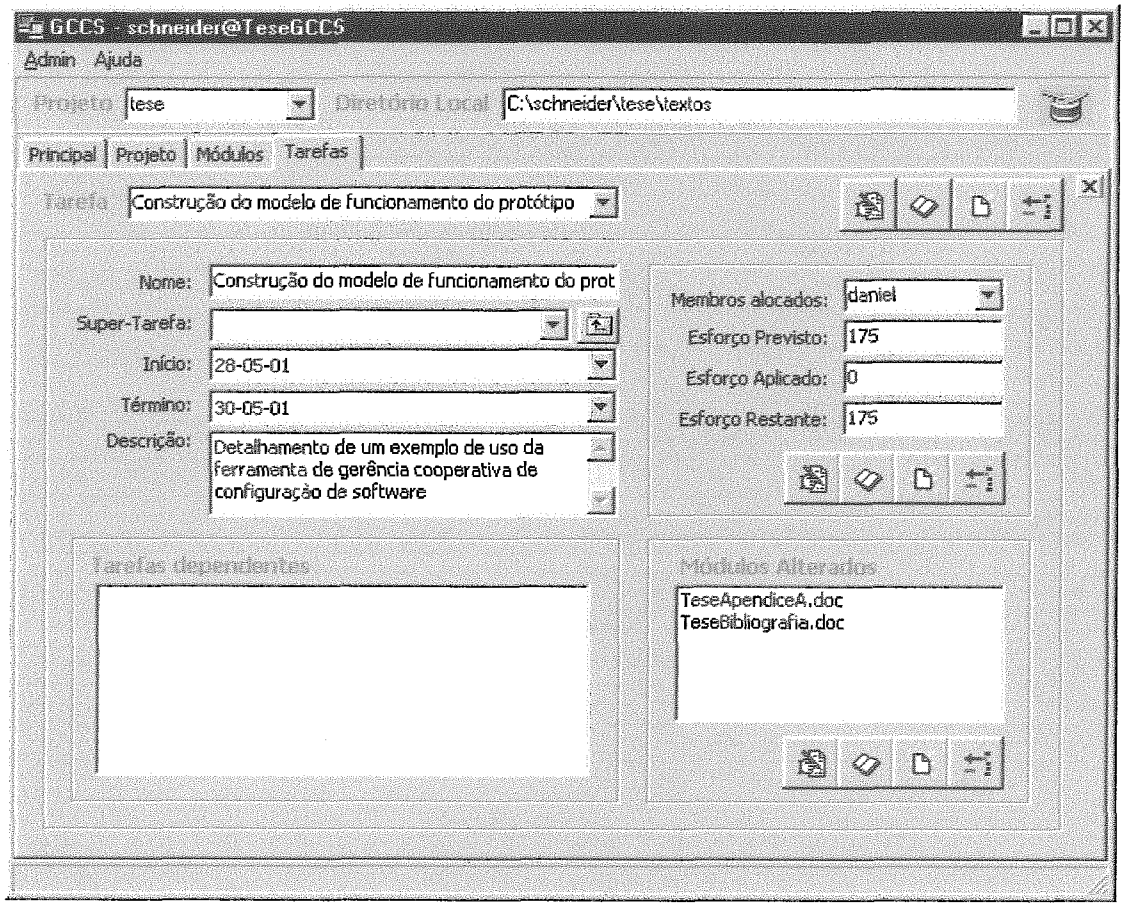

Figura A.8 – Criar Tarefa

A.5 Concluindo a entrega de módulos

Ao tentar fazer a confirmação da entrega de um módulo ao repositório, automaticamente é feita a revisão da avaliação do módulo. Se as avaliações atingirem os parâmetros definidos para o projeto (ver cadastramento de projeto no item A.2) o módulo é liberado para Confirmação de inclusão/substituição no repositório. Caso contrário, será reprovado e deverá ser modificado segundo as sugestões apresentadas pelos avaliadores.

Por exemplo, no projeto apresentado como modelo na Figura A.5, será necessário que mais de 50% dos usuários revejam cada módulo e mais 51% dos avaliadores aprovem o módulo.

Na Figura A.9 vemos um modelo de avaliação do módulo 12 – TeseApendiceA.doc do projeto tese sendo aprovado por um dos participantes do Chat de avaliação do módulo.

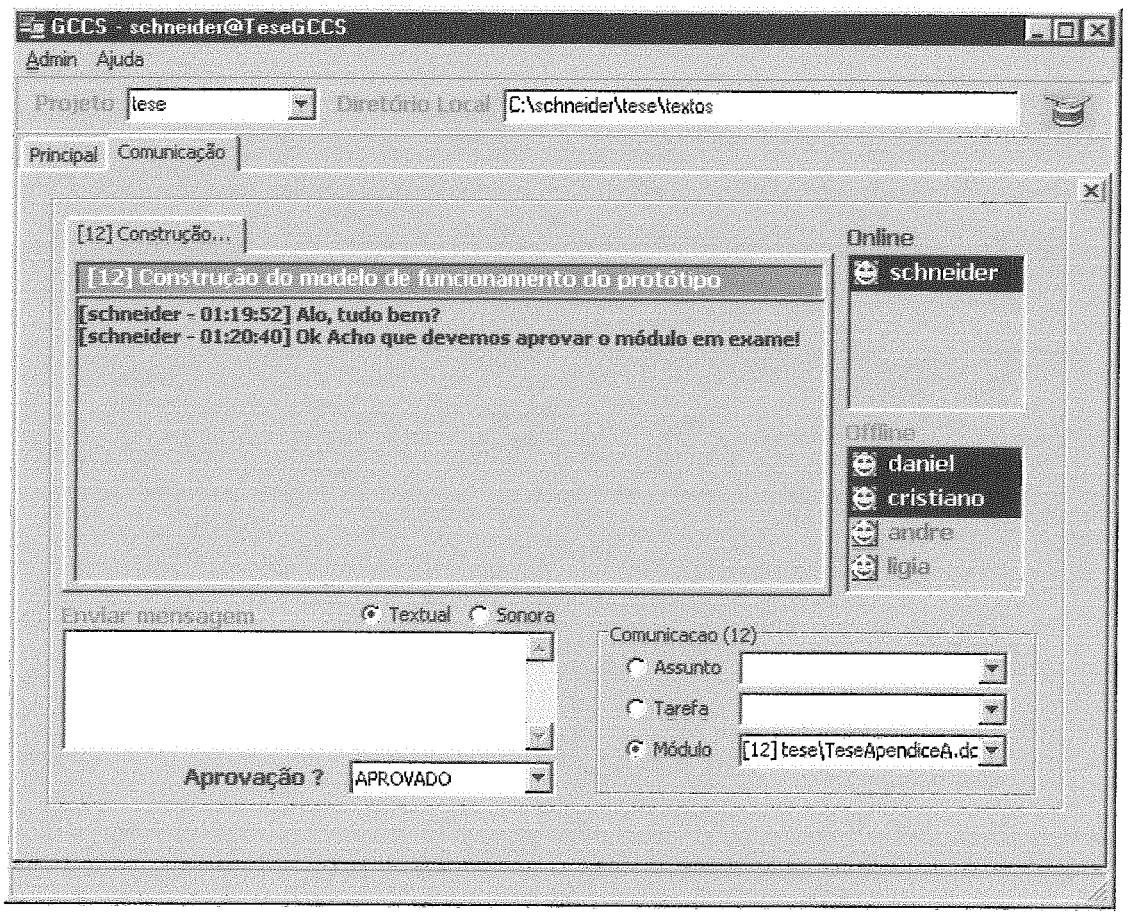

Figura A.9 – Avaliação de um módulo