Utilização de suporte distribuído na gerência de configuração de *software* análise de um processo evolutivo

André Luís de Andrade Mendes

Janeiro de 2003

Resumo

Na presente dissertação serão discutidos alguns dos problemas de gerência de configuração que ocorrem num ambiente de desenvolvimento de *software* e suas possíveis soluções com a adoção de diversos tipos de tecnologias, tais como *clusters* de computadores e terminais remotos, analisando suas vantagens e desvantagens, ganhos e perdas. Porém não se quer ter, somente, um aumento de desempenho e diminuição de tempos de resposta, mas sim, resolver problemas relacionados à gerência de configuração, como tempo para se obter uma liberação de produto (*release*), tempo necessário para a liberação de um posto de trabalho, ou seja, problemas que estão num nível de abstração acima do tempo de compilação ou velocidade de execução de um produto, onde os *clusters* já demonstraram ser soluções viáveis.

Controle de configuração e métricas são importantes para qualquer tipo de projeto, mesmo que isso seja feito manualmente. Com a automação que certas ferramentas podem trazer, podemos melhorar a produtividade, aumentar a taxa de correção de erros e ao mesmo tempo liberar a equipe de programadores de qualquer trabalho diferente do desenvolvimento das soluções e produtos.

Os procedimentos de gerenciamento de configuração guiam e armazenam certos atributos, dos quais podemos extrair informações como: em qual liberação de produto uma certa linha de código foi escrita, qual foi o desenvolvedor que realizou essa modificação, quando foi modificada e qual característica do produto essa mudança implementa. Com toda esta informação estatística coletada é possível ir além, e através de análises matemáticas, melhorar o processo de desenvolvimento e o próprio processo de coleta e análise.

Neste trabalho, serão apresentadas as tecnologias necessárias para alcançar estes pontos e serão analisadas as diversas propostas que existem para um ambiente de desenvolvimento de *software* para telecomunicações. Ao mesmo tempo, será mostrada uma nova proposta baseada nos resultados encontrados das análises com *clusters*. Estas análises poderão guiar e mostrar que esta tecnologia pode ser usada para maximizar a utilização do ambiente de desenvolvimento, bem como minimizar o tempo gasto com a implantação, manutenção e atualização dos diversos postos de trabalho que um ambiente de desenvolvimento requer.

Abstract

Herein we can see different configuration management problems that are inherent in a software development environment and solutions to solve this complex arrangement, using different kinds of technologies, like a cluster of computers and remote terminals. At the same time, the advantages and disadvantages will be presented to improve all variables and subsets of operations, not only the computing performance, but mainly in the configuration management. So there is the possibility to use this computing power to help the configuration management solves its problems too, not only to improve the compilation or execution time.

Configuration control and measurement are very important for any kind of project and it must be done, even if it is done manually. But with the automation that different tools can bring to this scenario it is possible to improve the error checking, productivity and at the same time freeing staff from performing tasks that are different from developing.

The configuration management procedures track several process-related attributes, so it is possible to determine in which release a line of code was created, who made these modifications, when they were made and what feature they implement. With these in place, it is possible to go even further, starting collect some statistics data, make some analysis on top of them and improve the development process and verification methods that are used.

In this document basic technologies will be presented to achieve these goals and analyze different ideas that were used before in a managed software configuration environment. After that, going one step further, a cluster computer will be used to facilitate the distribution of the development environment and improve the response time of a development team. At the same time, it is possible to reduce the overload on the configuration management administrator and all costs that were involved with this practice.

Em momentos de crise, só a imaginação é mais importante do que o conhecimento.

(Albert Einstein)

Agradecimentos

Gostaria de agradecer a todas as pessoas que permitiram o desenvolvimento deste trabalho; compreendendo os momentos em que não pude dar a devida atenção, que ajudaram a nortear corretamente a evolução do processo e àqueles(as) que souberam manter-se serenos e confiantes em momentos críticos, onde eu achava que não conseguiria chegar ao fim.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A presente dissertação de mestrado apresenta o processo de implantação da gerência de configuração de *software* em um ambiente de desenvolvimento voltado à produção de soluções para a área de telecomunicações, área esta, que possui algumas características específicas, entre elas, a necessidade de seguir diversos padrões estabelecidos pelas entidades internacionais, *softwares* que devem ser mantidos por longos anos e um ambiente de desenvolvimento que deve garantir certos níveis de qualidade identificados previamente. Simultaneamente, são descritas e avaliadas tecnologias de suporte distribuído que possam contribuir para o incremento da produtividade, nos diversos aspectos da produção de *software* nesse ambiente.

Os relatos aqui descritos e analisados mostrarão um processo incremental e evolutivo, partindo de um ambiente completamente desestruturado, com desenvolvedores autônomos, até chegar em um ambiente com gerência de configuração plenamente implantada. Para que a comparação entre as diversas etapas percorridas ao longo deste processo, seja possível, são definidas métricas de produtividade de *software* que permitem comprovar os ganhos trazidos pelas metodologias e tecnologias introduzidas no processo de desenvolvimento.

Uma tecnologia de especial importância nesse contexto é a de *clusters* de computadores, ou seja, computadores associados, através, de uma rede local e *software* básico apropriado para permitir a cooperação entre eles na resolução de algum problema específico. Normalmente, a tecnologia de *clusters* é empregada quando se deseja maior desempenho computacional ou maior disponibilidade de recursos para resolver problemas matemáticos de grandes dimensões, sem ter de lançar mão de super-computadores caros e pouco acessíveis. Muitos *clusters* estão substituindo super-computadores ao redor do mundo, por serem mais baratos e mais flexíveis que os os mesmos e apresentarem desempenho similar.

Entretanto, a busca de maior poder de processamento não é o único fator que motiva o uso de *clusters*. Estes também vêm sendo usados em soluções de alta disponibilidade, empregando tecnologias adequadas para tolerância a falhas e balanceamento de carga entre os nós do *cluster*. Este trabalho analisa o impacto do uso de *clusters* no ambiente de desenvolvimento de *software* descrito, avaliando suas possíveis contribuições para o processo de desenvolvimento de *software* e a gerência de configuração.

Além da tecnologia de *clusters*, serão também analisadas outras tecnologias capazes de trazer benefícios para o processo de desenvolvimento, como as tecnologias de terminais remotos e de replicação de discos. A agregação de todas essas tecnologias visa alcançar os seguintes benefícios:

- Facilidade de integrar novos desenvolvedores ao ambiente de desenvolvimento;
- manutenção de um ambiente de desenvolvimento controlado e coerente, uniforme para todos os desenvolvedores;
- facilidade na geração de versões e *builds*, e também na restituição de versões anteriores e das ferramentas de desenvolvimento a elas associadas;
- melhoria no desempenho dos computadores usados no desenvolvimento;
- rapidez na disponibilização de postos de trabalho;
- aproveitamento racional do conjunto de computadores instalados.

Os itens acima indicados serão abordados ao longo do texto, buscando mostrar que as vantagens apontadas são realmente relevantes em um ambiente de desenvolvimento de *software* no qual atuam simultaneamente, diversas equipes de desenvolvimento.

Nesta dissertação serão apresentadas as principais tecnologias e processos necessários para a implementação e adoção de um ambiente de desenvolvimento de *software* com gerência de configuração e será realizada uma análise do impacto dessa implantação no desenvolvimento de soluções de *software* para a área de telecomunicações. Esta dissertação é dividida em seis capítulos, que são:

- Capítulo 1 - Introdução

  Breve resumo dos assuntos tratados durante toda a evolução do documento;

- Capítulo 2 - O ambiente de desenvolvimento de *software*.

  Identifica os componentes básicos que formam um ambiente de desenvolvimento de *software* e seus relacionamentos, para que o conjunto represente da melhor maneira possível os interesses e necessidades de cada parte;

- Capítulo 3 - Gerência de configuração de *software*

  Neste ponto, será mostrada a necessidade de haver uma gerência de configuração de *software* num ambiente de desenvolvimento de programas, para que assim existam as condições mínimas para a concepção, criação, desenvolvimento e evolução de um produto de *software*, bem como uma descrição do processo de implantação da gerência de configuração, desde os seus conceitos mais básicos até os mais avançados;

- Capítulo 4 - Configuração do ambiente de trabalho

  Estes são os assuntos deste capítulo:

  - Verificação da gerência de configuração para o desenvolvimento de *software*.
  - O processo deverá chegar a um ponto, onde os produtos de trabalho dos desenvolvedores estarão sob controle, chegando ao ponto de gerir a configuração do próprio ambiente de trabalho.

- Capítulo 5 - Resultados obtidos

  Após a implantação de uma determinada idéia, foram coletadas algumas informações as quais serão apresentadas neste capítulo, tentando comparar os diversos resultados obtidos entre as diversas soluções. Vendo ainda que nem todos os problemas são resolvidos com a adoção de uma determinada solução;

- Capítulo 6 - Conclusões e perspectivas

  As conclusões das diversas abordagens utilizadas, seus problemas relativos e possíveis novas propostas estão contemplados no capítulo final.

Capítulo 2

O ambiente de desenvolvimento de *software*

Num ambiente de desenvolvimento de *software*, existem inúmeros componentes que são as peças formadoras desta estrutura. Sem estas peças organizadas, provavelmente, haverá um ambiente de desenvolvimento com grande desperdício de esforços, dinheiro, foco e dinamismo.

Com a adoção de um sistema de gerenciamento de configuração, que permeia todos estes componentes, é formado um ambiente controlado e passível de ser mensurado e aperfeiçoado. Para interagir e compreender melhor cada componente deste sistema, será necessário descrevê-los, procurando identificar os inter-relacionamentos existentes.

2.1 Componentes Básicos

Um ambiente de desenvolvimento de *software*, normalmente, é composto das seguintes partes: computadores, periféricos, programas básicos, bibliotecas, programas auxiliares, ferramentas de teste, códigos-fonte, desenvolvedores, testadores, gerentes e processos.

A seguir, será apresentada uma descrição destes diversos componentes do ambiente de desenvolvimento.

2.1.1 Componentes de *hardware*

Computadores e/ou estações de trabalho

No ambiente de desenvolvimento, o computador é uma peça essencial para que os produtos de trabalho sejam obtidos. Este é necessário para que o desenvolvedor cons-

trua, escreva e compile os seus programas para gerar o produto final. Este componente, deve ser o mais rápido possível, do ponto de vista do desenvolvedor, mas na realidade com uma boa gerência de configuração, uma economia razoável pode ser obtida, colocando as melhores máquinas nos pontos críticos do desenvolvimento e ao mesmo tempo, agradando aos desenvolvedores com relação a desempenho e características, tais como: quantidade de memória, tamanho do monitor e aceleradores gráficos.

Periféricos

Este é um outro item do ambiente de desenvolvimento que pode causar problemas. Normalmente, são produtos que devem ser compartilhados no ambiente de desenvolvimento, pois são caros, tais como emuladores de *hardware*, ou emuladores de placas que estão sendo desenvolvidos, simultaneamente, ao produto de *software*, além dos dispositivos de armazenamento de dados e os de entrada e saída, tais como: *scanners* e impressoras. Estes devem ser mantidos na maior parte do tempo em estado funcional para que não atrapalhe ou atrase o desenvolvimento do projeto como um todo.

Equipamentos de medição ou teste

Outros tipos de aparelhos que podem fazer parte de um ambiente de desenvolvimento, são os equipamentos de testes ou de medição, que ajudam a encontrar erros nos produtos de *firmware* que estão sendo desenvolvidos, ou executam medidas para verificar se o programa de controle de determinada placa está executando o seu trabalho corretamente. Pode-se enquadrar nesta classe de aparelhos, os osciloscópios, analisadores lógicos, analisadores de protocolo, entre outros.

Hardware proprietário

Ainda pode-se ter neste ambiente de desenvolvimento, equipamentos já desenvolvidos, que são necessários para a correta depuração dos novos produtos, ou são necessários para que todo o ambiente de utilização seja reproduzido em laboratório, permitindo assim que se chegue o mais próximo do ambiente de produção, onde o novo produto será inserido.

2.1.2 Componentes de *software*

Programas Básicos

Os programas chamados de básicos num ambiente de desenvolvimento, são: sistemas operacionais, compiladores e afins, como: *link* editores, cross-compiladores, *assemblers*.

Neste item, são encontrados os maiores problemas. Cada equipe de desenvolvimento deve ter uma certa característica de ambiente, por exemplo: num grupo de dez desenvolvedores, dois deles devem instalar um sistema operacional *Windows NT* versão 4 com o compilador Visual C versão 6, o outro grupo de quatro pessoas necessita de um ambiente *Red Hat Linux* versão 7 com o compilador GNU. Os demais precisam de um ambiente proprietário para o desenvolvimento de *firmware* para as placas emuladoras. Tudo isso, deve ser gerenciado e customizado, conforme a necessidade de cada um e ainda mantido, no sentido de atualizações, onde devem ser aplicadas novas bibliotecas e códigos-fonte em comum.

Para ajudar nessa manutenção, surge um outro grupo de programas básicos, os gerenciadores de versão. Esses programas são responsáveis por garantir que cada pessoa ou grupo ganhe a sua versão do produto e dos códigos-fonte, conforme a necessidade de cada fase do projeto e de cada parte do grupo de desenvolvimento.

Bibliotecas

As bibliotecas são grupos de códigos-fonte que foram compilados e agrupados de maneira a formar um pacote de programas que serão usados por outros grupos no decorrer do desenvolvimento. Esses pacotes podem ser gerados pela própria equipe de desenvolvimento ou comprados externamente, de terceiros, os quais neste caso, acabam sendo responsáveis por atualizações.

Normalmente, neste caso, são atualizações em formato binário, obrigando que todo o pacote seja substituído de uma única vez, gerando ainda a necessidade de uma recompilação do código-fonte da aplicação que as utiliza.

Códigos-Fonte

Estes são os produtos desenvolvidos e escritos para resolver o problema que foi proposto no início do projeto pelos desenvolvedores. Estes são a propriedade intelectual que deve ser controlada e armazenada; é o produto de trabalho da equipe que vai gerar o produto a ser vendido, a ser colocado no mercado. Assim, eles devem ser entregues com o máximo de qualidade possível, no prazo e com o custo controlado para garantir o lucro da empresa, permitindo que novos projetos sejam feitos ou que os projetos em andamento sejam atualizados.

Documentos

Os documentos também podem ser considerados produtos de trabalho, pois muitas vezes, estes são os resultados de outras fases que compõem o desenvolvimento completo

de um novo produto. Por causa desta característica, estes também devem ser mantidos sob controle de versão, pois assim pode-se controlar uma determinada característica quando implementada e testada, ou ainda, como esta será testada e seus resultados armazenados para posteriores comprovações e liberações.

Em muitos casos, um documento controlado pode nortear o desenvolvimento a atingir um melhor resultado, concentrando os seus esforços nas soluções dos problemas cruciais para cada liberação existente, cada um a seu tempo, conforme os documentos resultados das fases de análise, ou teste do produto.

Programas Auxiliares

Os programas auxiliares podem ser caracterizados como sendo os programas de apoio para o desenvolvimento, como por exemplo: editores de textos, geradores de gráficos, gerenciadores de apresentação para que se gerem especificações, documentos e relatórios de maneira consistente e padronizada, facilitando o acesso de todos os envolvidos no projeto às informações necessárias para o desenrolar contínuo e constante do projeto. Possibilitando, assim, o acompanhamento por parte dos gerentes, da evolução e necessidades do projeto, sejam elas necessidades de: pessoal, equipamentos ou realocações para tentar fazer cumprir a meta idealizada na fase de planejamento.

Esses programas devem estar disponíveis para toda a equipe de desenvolvimento, para não impedir ou atrasar o produto final ou deixar alguém sem acesso a algum tipo de informação, que seja necessária para a evolução do projeto.

Programas de teste

Os programas de testes são os programas que auxiliam o desenvolvedor ou o testador a replicar testes de módulo, testes de regressão e testes de carga de maneira controlada com a possibilidade de repetição e com relatórios dos testes gerados para uma aprovação do código, do módulo ou do produto, segundo uma qualidade aprovada e acordada anteriormente, ou requisitada na especificação do projeto.

2.1.3 Componentes Humanos

Desenvolvedores

Desenvolvedores são as pessoas envolvidas em todo o trabalho para a implementação da solução do problema que, depois de concluído, será o produto gerado que será entregue ou vendido. Essas pessoas têm algumas características interessantes que merecem ser citadas [GGC, 2001]:

- os programadores/desenvolvedores tendem a subestimar seus projetos em 20% a 30%
- as estimativas de um pequeno projeto costumam ter um erro maior que 100%
- um grande projeto costuma atrasar um ano
- dos projetos maiores que 12.500.000 LOC (*Lines Of Code*) em C, apenas 15% deles são entregues no prazo

Por causa dessas características, um melhor desempenho e a qualidade das equipes de desenvolvimento devem ser procurados, maximizando a utilização dos recursos, sejam eles humanos ou materiais, administrando as divergências para tentar fazer com que estas máximas não aconteçam. É bom lembrar que os desenvolvedores são os responsáveis, em última instância, pela finalização ou não do projeto, e que também são seres humanos com ideais diferentes, perspectivas de vida diversas e capacidades, competências e conhecimentos diferentes e ao mesmo tempo necessários para a equipe como um todo.

Testadores

Os testadores têm as mesmas características dos desenvolvedores, mas são os responsáveis por garantir a funcionalidade de um módulo do sistema, da integração entre estes diversos módulos e por garantir o funcionamento e a qualidade do produto final.

Normalmente, essas pessoas têm necessidades diferentes dos desenvolvedores, pois precisam ver ou ter um ambiente de testes semelhantes ao ambiente para o qual o produto foi desenvolvido, a fim de garantir o mínimo de equipamentos ou de características necessárias para o correto funcionamento do produto.

Gerentes

Os gerentes são os responsáveis pelo bom andamento dos projetos, pela manutenção da equipe, pelas necessidades de pessoal e materiais. Normalmente, são as pessoas que mais influenciam nas decisões, pois sempre querem o máximo de funcionalidades com o mínimo de gastos/custos e se possível sem nenhum investimento. Tudo isso é conflitante, mas a figura do gerente é extremamente importante para que os problemas sejam filtrados, conflitos sejam resolvidos, investimentos sejam feitos e custos sejam controlados, para assim manter a equipe de desenvolvimento focada na solução do problema.

2.1.4 Processos

Nesta dissertação, não será tratado o processo de desenvolvimento de um novo produto como um todo, e sim da base necessária para que se crie um suporte mínimo e coerente para que, numa nova etapa, se agregue novos processos mais sofisticados, tais como: análise e gerenciamento de requisitos, análise e gerenciamento de erros.

Plano de gerência de configuração

Assim, para a gerência de configuração, que será melhor descrita no próximo capítulo, a primeira fase de um projeto começa quando é necessário armazenar produtos de trabalho e disponibilizar postos de trabalho coerentes com o desenvolvimento deste novo produto, gerando assim a necessidade de haver uma análise dos requisitos de entrada deste novo produto para a gerência de configuração, visando: verificar se alguns desses requisitos não são conflitantes, validar o conjunto de requisitos em relação às ferramentas e equipamentos disponíveis, garantir que todas essas ferramentas, *software* e *hardware* estejam disponíveis e configuradas para o início do projeto e alocar o tempo dos desenvolvedores de modo adequado em relação aos postos de trabalho disponibilizados. Isso feito, as condições e pré-requisitos necessários para começar o desenvolvimento do produto estão satisfeitos.

Testes do produto gerado

Existem diversos pontos no desenvolvimento de um produto de *software*, onde se deve fazer testes para garantir alguns pré-requisitos básicos para a continuidade do projeto como um todo. Estes testes partem de um simples teste de módulo ou teste de objeto, onde o próprio desenvolvedor/programador faz o seu teste de compilação, para ver se o arquivo não possui erros graves de sintaxe ou semântica que impeçam a continuidade de evolução do grupo de desenvolvimento; passam por um teste de integração para verificar se todos os objetos ou módulos "conversam" entre si e realizam as funções que foram planejadas para que estes módulos as executassem.

Neste ponto, um índice mínimo de qualidade deve ser colocado para que o produto possa ser testado como produto final, totalmente agregado. Assim, as equipes de testes sistêmicos podem ser alocadas para que os testes finais no produto sejam realizados, muitas vezes, junto com outros produtos ou equipamentos. Testes esses que gerarão relatórios de não conformidade em relação às especificações feitas pelo cliente, ou relatórios de falhas sistêmicas, nos quais aparecem falhas do próprio produto ou na integração com o mundo real, ou até influências que este pode causar nos outros

produtos, que porventura deveriam trabalhar em conjunto.

Validação de entrega do produto

Após a fase de testes, a equipe responsável pelo teste sistêmico e integrado para o qual previamente, deve ser estabelecida uma lista de checagem para verificar se o produto faz aquilo que deveria fazer e foi especificado para fazer, deve gerar um relatório de validação.

Neste relatório de validação, devem constar todos os testes realizados e suas devidas respostas, sejam elas positivas ou negativas. Se estas forem negativas, devem contemplar uma ação necessária para torná-las positivas ou então serem justificadas se serão corrigidas, alteradas ou feitas numa próxima versão do produto.

Essa validação passa ainda por um acordo prévio do nível de qualidade exigido. Jamais um produto irá sair 100% completo ou isento de erros, pois sem esse acordo prévio e assumindo a regra acima, nunca um produto seria finalizado. Um certo nível de imperfeições deve ser assumido para que um produto fechado possa ser entregue, e que permita uma certa evolução através da inclusão de novas características previstas inicialmente, que teriam um custo elevado numa primeira instância, quando este custo pode ser monetário ou número de programadores/dia ou até mesmo prazo de entrega.

2.2 Suporte a diferentes tecnologias

Num ambiente de desenvolvimento real, suportar diferentes tecnologias é essencial para a produção de novos produtos, porque muitas vezes os problemas apresentados no desenvolvimento de um produto não podem ser resolvidos, somente, com uma técnica. A melhor tecnologia disponível naquele momento, deve ser utilizada para solucionar determinado problema. Às vezes uma linguagem de *script* resolve melhor determinadoo problema do que um programa escrito em linguagem C, num certo instante de tempo devido a restrições impostas externamente, tais como: tempo, compatibilidade, foco do problema.

As tecnologias necessárias a todas as etapas do desenvolvimento do projeto devem ter um suporte por parte da equipe de gerência de configuração. Isto faz com que todo o esforço intelectual do desenvolvedor seja direcionado para a solução do problema e não para implementar estruturas já conhecidas ou encontrar soluções não comuns para acessar determinados dados. É de vital importância que haja capacitação para integrar diferentes sistemas operacionais e garantir uma transparência ao acesso das informações e a própria agilidade, neste acesso. Também é de extrema importância

que sejam suportadas diversas linguagens permitindo ao programador usar a melhor linguagem em cada caso, pois as linguagem são escritas e criadas para facilitar a solução de determinados tipos de problemas, implantando conceitos básicos e assim facilitando o encontro da solução ideal por parte do programador. Este conceito pode ser visto quando se está escrevendo programas que têm restrições e exigências de tempo de resposta, onde geralmente, deve-se deixar de lado momentaneamente, uma linguagem de alto nível e escrever as rotinas em linguagem de máquina para poder atender a tempo, todos os serviços requisitados.

Hoje, num ambiente de desenvolvimento, não é raro encontrar sistemas operacionais diferentes, rodando simultaneamente na mesma rede local e muitas vezes até na mesma máquina, tais como Windows e Linux, linguagem de desenvolvimento diferentes, como C++, Java e outras, linguagens de script como Perl, shell script e ainda ASP, PHP, HTML, Javascript, que seriam utilizadas no desenvolvimento para web. As tecnologias que aparecem e desaparecem no dia-a-dia são muitas, por isso o trabalho para manter e atualizar as tecnologias que fazem a diferença no ambiente de desenvolvimento é contínuo onde as tecnologias que num certo momento não são mais necessárias devem ser descartadas ou modernizadas para não desviar a atenção de nenhum dos membros da equipe. Do mesmo modo as novas técnicas que estão surgindo devem ser incorporadas de maneira sutil ao ambiente de desenvolvimento, tentando minimizar o impacto dessas mudanças [PAULK, 1994].

Por sinal, um dos grandes problemas numa equipe de desenvolvimento, muitas vezes, é a falta de foco e direção [YOURDON, 1989], o que faz com que os componentes da equipe se desmotivem e acabem diminuindo a produtividade por motivos alheios ao da solução do problema. Este é um ponto que uma gerência de configuração sólida e bem aplicada pode ajudar a evitar.

2.3 Conclusão

Após a apresentação dos componentes básicos para montar um ambiente de desenvolvimento de *software*, suas técnicas e conceitos serão descritos no capítulo seguinte, mais profundamente, os conceitos necessários para a compreensão da gerência de configuração de *software*, sua importância e como alcançá-la, visando, assim, otimizar o ambiente de produção e desenvolvimento de *software* em diversas direções simultaneamente.

Capítulo 3

Gerência de configuração de *software*

Um problema muito importante que deve ser tratado em um desenvolvimento de *software* e na sua manutenção é o controle de configuração e revisão, uma das áreas que a gerência de configuração de *software* trata. A tarefa de manter um sistema de desenvolvimento de *software* consiste, basicamente, em manter diversas revisões e configurações, tantas quantas forem necessárias, muito bem organizadas, para que em qualquer instante seja possível saber como reconstruir qualquer versão do produto, bem como, voltar a qualquer ponto do desenvolvimento de maneira rápida e fácil. Existem muitas ferramentas que são capazes de ajudar neste trabalho, gerenciando diversas revisões de arquivos texto, tais como, código-fonte, documentações e dados de teste, além de automatizar as tarefas de armazenar, recuperar, fazer *logs* e identificar as diversas versões.

3.1 Conceitos básicos

Se uma pessoa nunca usou uma ferramenta de gerência de configuração (*FGC*), ela pode ser "enganada" por algumas suposições. O que normalmente, acontece é que as pessoas imaginam que uma *FGC* é usada para funções distintas e aparentemente sem correlação, que são: a gravação de registros (histórico) e a colaboração (trabalho em paralelo e em grupo).

A gravação de registros torna-se necessária para que um certo desenvolvedor possa, a qualquer instante, comparar o estado de um programa com uma revisão criada em algum momento, no passado, ou ainda controlar a própria evolução do sistema. Para que o produto evolua, em certos momentos, deve tornar-se instável ou incompleto: estes são os momentos em que os desenvolvedores estão implementando as novas características que foram planejadas. Nestes pontos, o programa que estava funcional deixa de

funcionar até que as novas características fiquem prontas e sejam liberadas.

Se nesse ínterim, um problema é detectado na liberação que estava "pronta", este deve ser corrigido o mais breve possível. Neste caso, a gravação de registros (históricos) é essencial para poder retornar ao ponto no qual o programa foi liberado, para que o problema seja corrigido e depois, possa ser re-anexado ao desenvolvimento principal do produto. Sem este tipo de tecnologia, a implantação do controle de configuração torna-se difícil e incompleta e para as *FGC*s este é um serviço básico intrínseco à própria ferramenta.

Por exemplo, através de uma *FGC* é simples atender a solicitações feitas por um grupo de desenvolvedores, de códigos-fonte de três semanas atrás, ou então uma liberação entregue a um cliente para realizar qualquer tipo de correção de problemas. Para que estas solicitações sejam possíveis, as *FGC*s implementam diversos tipos de comandos de seleção, que serão apresentados no decorrer deste capítulo.

Para a parte de colaboração ou desenvolvimento em paralelo, as *FGC*s podem implementar vários mecanismos que permitem a alteração dos arquivos por diversos programadores, sendo os mais comuns: *lock-modify-unlock* e o método *copy-modify-merge*. Cada ferramenta, normalmente, implementa um certo modelo, mas o modelo de *copy-modify-merge* é o que permite que grandes equipes de desenvolvimento, muitas vezes, separadas por grandes distâncias e impossibilitadas de se comunicar, consigam desenvolver o produto de maneira controlada e gerenciada.

3.1.1 Controle de revisões ou versionamento

O controle de revisões de arquivos, mais comumente conhecido como versionamento, serve para gerenciar múltiplas revisões de arquivos, e este deve automatizar o armazenamento, a recuperação, o *log* das operações executadas, a identificação e o agrupamento das revisões [BOLINGER, 1995]. Isto é muito útil para arquivos-texto que são freqüentemente revisados, como: códigos-fonte de programa, documentações, planilhas para geração de gráficos entre outros.

Normalmente, o controle de revisões de arquivos é feito através, de uma árvore de revisões, nas quais são guardadas as diferenças entre a versão atual e a versão anterior, e os diversos meta-dados que são necessários para um ambiente de desenvolvimento, tais como: carimbos de data/hora, o usuário que fez a modificação, comentários, etc. Assim, o sistema é capaz de responder a qualquer instante perguntas do tipo:

- Quem fez determinada modificação ?
- Quando essa mudança foi realizada ?

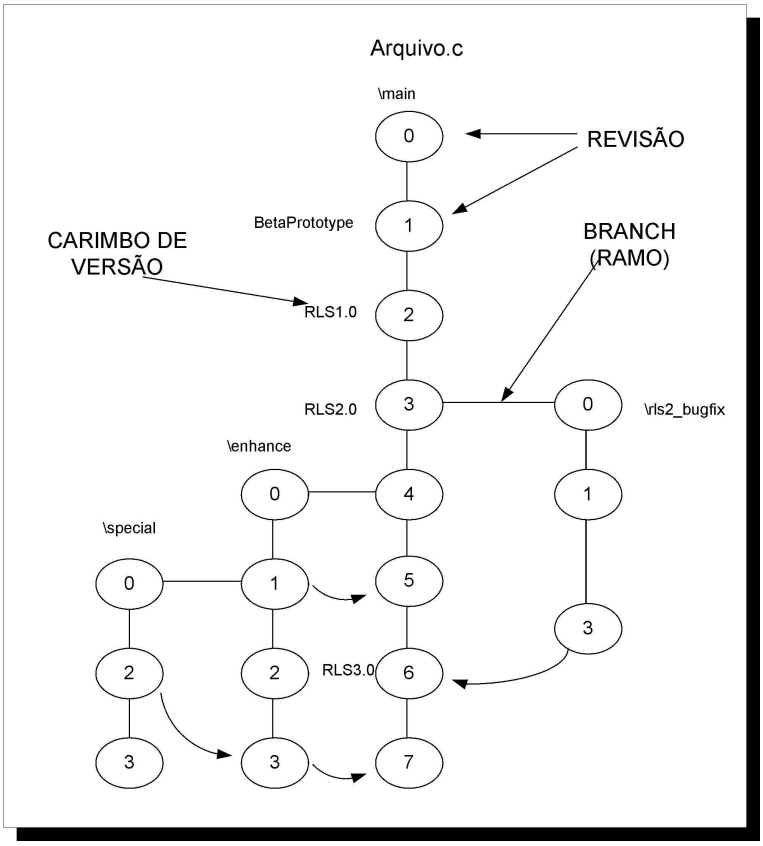

Figura 3.1: Árvore de revisões de um arquivo

- Por que a alteração foi necessária ?
- O que foi modificado ?
- Existem outras modificações que foram realizadas ao mesmo tempo ?

O sistema de controle de arquivos, uma vez implementado, é a base para controlar o ambiente de desenvolvimento de produtos de *software* e é necessário para se implantar um sistema de gerência de configuração adequado.

3.1.2 Representação das revisões através de deltas

A tecnologia por trás do controle de arquivos, que é a representação das revisões através de diferenças (deltas), na prática, foi o que deu o grande impulso para a utilização deste tipo de serviço, conservando espaço em disco. Ao mesmo tempo que não existe a degradação de desempenho por se manipular cada vez mais quantidades de

dados, além é claro, de que a interface de comandos do sistema de controle de versão esconder completamente, esta característica.

Estas diferenças são seqüências de comandos, que são capazes de transformar uma cadeia de caracteres em outra cadeia de caracteres. Estes comandos são baseados em alterações ocorridas em linhas do arquivo, e os comandos existentes para este caso, são a inserção e remoção de linhas. Caso ocorra a mudança de apenas uma letra numa determinada linha, a linha inteira é considerada diferente. Se fosse usado um algoritmo de diferenças de caracteres, o ganho seria muito maior, mas isso não acontece, o que não justifica o aumento de complexidade da solução, e muito menos o aumento de tempo computacional.

Novamente, pensando no compromisso espaço/tempo, a utilização do algoritmo de diferenças reduz a ocupação do espaço em disco, mas aumenta o tempo de acesso ao arquivo. Uma ferramenta desse tipo não deve influenciar ou interferir nas tarefas do dia-a-dia de um desenvolvedor. É muito comum, quando ferramentas de controle de versão introduzem no ambiente de desenvolvimento atrasos sucessivos e excessivos, que estas deixem de ser utilizadas ou que sejam substituídas por procedimentos pouco ortodoxos, comprometendo a integridade do sistema, causando diversas não conformidades ao processo.

Para garantir o mínimo de interferência e ter um ganho na velocidade de acesso ao arquivo, tanto no momento da edição, quanto, no momento de compilação - outro ponto bastante crítico - o *RCS*, ferramenta usada como exemplo, utiliza o seguinte formato e algoritmo: sempre, a revisão mais recente do ramo principal é armazenada de maneira intacta, e todas as demais revisões são armazenadas como deltas reversos, ou, as diferenças descrevem como voltar atrás no histórico do arquivo em questão. Aplicando esses comandos, um desenvolvedor consegue obter a revisão desejada, se utilizar a revisão sucessiva como base. A principal vantagem desta idéia é que é extremamente rápido acessar a última versão (*check-out*) porque é apenas uma operação de cópia; para realizar o *check-in* também é muito rápido, pois o comando deve apenas inserir a nova revisão de maneira completa e trocar a revisão prévia pelo cálculo da diferença entre a última revisão e a sua imediatamente anterior, mantendo os demais deltas intactos.

Para os ramos, é necessário um tratamento especial, pois se fosse armazenado da mesma maneira do ramo principal (tronco), o gasto com o espaço em disco seria muito grande. A solução mais eficiente para esses casos é a utilização de deltas progressivos. Para conseguir a revisão que o programador quer, o sistema extrai a última revisão do tronco, aplica as diferenças reversas até chegar no ponto em que houve a aparição do ramo. A partir daí, a ferramenta aplica os deltas progressivos até alcançar a revisão que o desenvolvedor deseja, que está guardada no ramo lateral. A figura 3.2 ilustra

bem como todas essas operações se sucedem. Desse modo, o sistema entrega ao usuário as informações que este necessita e ao mesmo tempo otimiza a utilização do espaço de armazenamento.

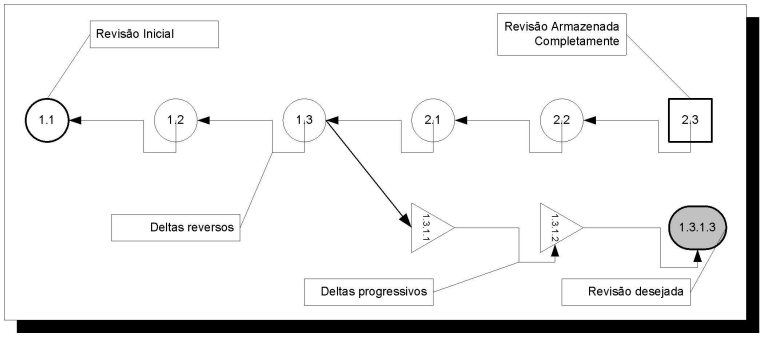

Figura 3.2: Árvore de revisão com deltas reversos e deltas progressivos

Com esta solução de armazenamento, sempre a última revisão de maneira completa, é um ganho muito grande no momento de realizar um *check-out*, mas para conseguir o acesso a qualquer outra revisão, é necessário que o tempo seja proporcional ao número de deltas aplicados ao arquivo. No exemplo da figura 3.2 para regerar a revisão que o usuário deseja, é preciso aplicar 6 deltas e mais o *check-out* da revisão inicial. Como na maioria dos casos, cerca de 95% das vezes, sempre é feito o trabalho com a última versão do arquivo, o ganho que é alcançado em utilizar um algoritmo que privilegia o tempo de *check-out* é muito significante e importante para ser deixado de lado, ou não tentar realizar esta otimização. A necessidade do trabalho em outras revisões é imprescindível, e o algoritmo que aplica as diferenças, sejam elas reversas ou progressivas, deve ser muito bem escrito para minimizar o atraso na aplicação dos deltas.

3.1.3 Limitações da gerência de configuração

As ferramentas de gerência de configuração não são capazes de resolver todos os problemas, assim a lista apresentada mostra os principais pontos, onde pode-se pensar em utilizar uma ferramenta de gerência de configuração, mas o qual ela não é a melhor solução.

1. Ferramenta de gerência de configuração não substitui o sistema de *build*

Uma ferramenta de gerência de configuração deve ser independente do sistema de construção de liberações (*build system* - `MAKE`). Obviamente, toda a estrutura que a

FGC agrega ao sistema, bastante rígida em certos casos, ajuda em muito o sistema de construção.

Mesmo com a *FGC* existe muito trabalho para o sistema de construção. Neste caso, a ferramenta pode ajudar a manter o histórico dos arquivos de configuração da ferramenta de construção [ORAM, 1991]. Podendo ajudar muito e com o auxílio de um programa de geração de interdependência automático, pode-se diminuir o tempo de *build*, realizando somente, as compilações dos módulos que sofreram impacto pelas mudanças realizadas nos códigos-fontes.

2. Ferramenta de gerência de configuração não substitui a comunicação entre os programadores

Uma *FGC* não é capaz de resolver conflitos que existam no mesmo arquivo ou em grupo de arquivos que se interrelacionam. Além do mais, os desenvolvedores sempre tentam resolver os problemas mais simples de maneira direta, enquanto que os conflitos complexos, normalmente, são muito difíceis de ser resolvidos sem uma direta comunicação entre os envolvidos.

Uma *FGC* não é capaz de resolver os problemas lógicos, que podem ocorrer quando existem modificações em um arquivo ou em grupo de arquivos. As ferramentas de `diff` e `merge`, integradas as *FGC*s, são capazes de detectar somente, conflitos textuais. Para exemplificar um conflito simples, mostrado na figura 3.3, que a *FGC* não é capaz de detectar, seria o seguinte: supondo que um programador esteja modificando um arquivo A, que contém uma chamada a uma função `X(a,b)`, e esta modificação está sendo realizada no número de parâmetros da chamada, fazendo com que ela se torne `X(a,b,c)`. Se no mesmo instante, um outro programador mexendo num arquivo B, adiciona novas chamadas a função `X(a,b)`, e, este só conhece a definição antiga, na prática, isto acaba impedindo que o programa continue compilando.

Este é um exemplo clássico e simples de conflitos nos quais uma *FGC* não é capaz de fazer nada, e o diálogo entre as pessoas que compõem a equipe de desenvolvimento que deve tomar conta da solução deste conflito, além é claro do respeito às especificações feitas, anteriormente.

3. Ferramenta de gerência de configuração não substitui o gerenciamento

Os gerentes e líderes de projeto devem manter um diálogo constante com os desenvolvedores para terem um certo conhecimento dos tempos envolvidos, vendo se os cronogramas estão sendo cumpridos, ter conhecimento dos pontos de união (*merge points*) e datas de liberação.

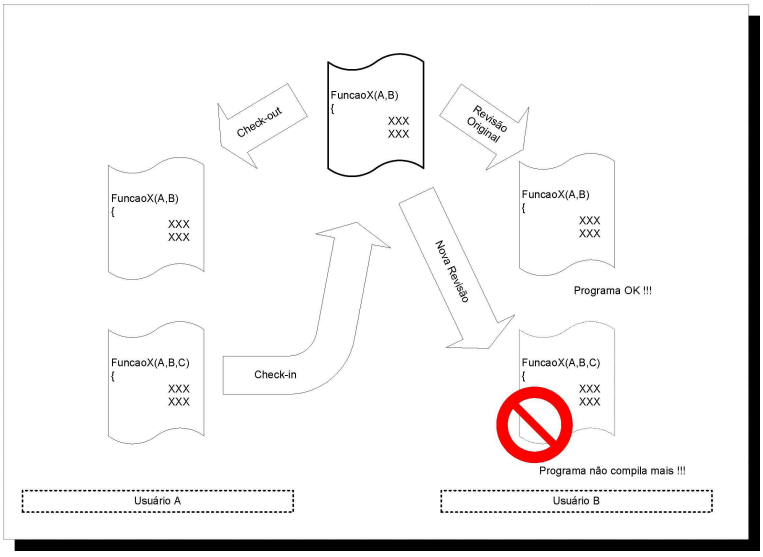

Figura 3.3: Conflitos lógicos

4. Ferramenta de gerência de configuração não tem modelos de processos embutidos

Uma *FGC* normalmente, não tem um sistema que modela processos embutidos no controle de configuração, mas como no caso do sistema de construção, uma *FGC* pode ajudar a garantir alguns passos de um modelo de processos. Através, de algumas soluções simples, como arquivos de controle padronizados ou procedimentos de gatilhos, é garantido que determinados ramos do desenvolvimento sejam usados para a implementação de certas características, ou que as mudanças que ocorrem em diversos pontos sejam avisadas ou aprovadas antes de se realizar um *check-in*. Para alcançar um modelo de processos garantidos é melhor integrar a ferramenta de gerência de configuração a uma ferramenta de gerência de *workflow*. Conseqüentemente, o resultado final terá uma melhor qualidade.

5. Ferramentas de gerência de configuração não automatizam processos de testes

Outro ponto interessante, é a parte de testes, na qual as *FGC*s também não são capazes de ajudar na automatização destes testes. Uma *FGC* pode ajudar a controlar os cadernos de testes ou relatórios gerados, pois estes são documentos, como qualquer outro.

Novamente, o uso de ferramentas de testes, que geram relatórios, podem ou devem

ser integradas a *FGC* e também ao sistema de *workflow*, para garantir a qualidade durante todo o processo.

6. Ferramentas de gerência de configuração não têm controle de modificações e características do produto

Este tipo de serviço deve ser feito por ferramentas apropriadas a esta tarefa, visto que o controle de modificações, normalmente, envolve várias modificações em vários documentos, simultaneamente. Isto fica a cargo das ferramentas de gerência de requisitos, que são ferramentas que utilizam a gerência de configuração como base para executar o seu trabalho.

Entretanto, quando o gerenciamento de mudanças é lembrado, pode ser que seja um procedimento ou tarefa de *bug tracking*, onde deve existir um banco de dados com os problemas encontrados e suas possíveis soluções e após uma decisão, tomada em conjunto entre os analistas, gerentes e programadores, deve permanecer cadastrada, e não o processo de gerência de requisitos ou modificações.

Qual será a solução adotada, quais programadores irão implementá-la, em qual revisão este *bug* deverá ser corrigido, se o cliente que detectou o problema está de acordo com a solução adotada? Estes são os pontos quando uma *FGC* pode ajudar o sistema de *bug tracking* a controlar os arquivos, que serão modificados para satisfazer a solução de determinado problema.

Outro aspecto que pode ser visto no caso de controle de modificações é a questão de verificação do estado de cada característica (acompanhamento) - como está a evolução da implementação de cada parte. Também, neste caso, uma *FGC* normalmente, não consegue fazer o trabalho sozinha.

3.2 Ferramentas para versionamento

Algumas ferramentas que podem ser utilizadas para o controle de versionamento serão exemplificadas a seguir, mostrando as suas características principais. Alguns conceitos serão explicados durante o desenrolar do capítulo.

3.2.1 *RCS*

O *RCS*[1] é uma das ferramentas mais básicas e simples que existem para o controle de revisões de arquivos [TICHY, 1991]. Esta ferramenta permite que sejam gerencia-

[1] *Revision Control System*

das múltiplas revisões de arquivos. Além disso, como não deveria deixar de ser, este automatiza o armazenamento, a recuperação, a gravação de registros de históricos, a identificação das mudanças e a mixagem entre revisões.

Principais funções

- Armazenar e recuperar múltiplas revisões de arquivos textos

  O *RCS* é capaz de armazenar as diferentes revisões de texto de um arquivo em um modo muito eficiente: os arquivos delta, que foram explicados anteriormente. Assim, uma modificação não mais destrói uma revisão original, pois todas as revisões continuam tendo a possibilidade de serem acessadas a qualquer instante.

- Manter um histórico das modificações

  O *RCS* mantém um histórico de todas as modificações que são feitas sobre um arquivo que esteja sob seu controle. Este guarda informações importantes sobre as modificações, tais como: o autor da modificação, a modificação propriamente dita, o carimbo de data/hora na qual esta alteração foi feita, textos descritivos com comentários sobre as alterações, entre outras coisas. Isto ajuda em muito, quando é preciso descobrir o que aconteceu com determinado arquivo ou módulo, sem ser necessário recorrer aos desenvolvedores originais.

- Resolver conflitos no acesso a arquivos

  Quando dois ou mais desenvolvedores precisam modificar a mesma revisão de certo arquivo, o *RCS* é capaz de alertar sobre tal acontecimento e impedir que a modificação de um usuário interfira na alteração do outro usuário.

- Manter uma árvore de revisões

  Através de uma árvore de revisões o *RCS* é capaz de manter linhas de desenvolvimento diferentes para cada arquivo ou módulo, ao mesmo tempo, que guarda os relacionamentos hierárquicos entre estas diversas revisões.

- Mixar revisões e resolver conflitos

  Linhas diferentes dentro do mesmo arquivo podem ser facilmente, integradas no mesmo arquivo pelo próprio *RCS*, enquanto que modificações numa mesma linha são apenas alertadas para que os usuários possam resolver este conflito sobreposto.

- Controle de configuração e liberação

O *RCS* permite que sejam colocados carimbos nas revisões, facilitando assim o controle das liberações, que podem ser marcadas como liberada, estável, testes, experimental, etc.

- Identificação automática

  O *RCS* pode identificar, automaticamente, cada revisão, com um número de revisão, com a data de criação e o nome do autor. Esta identificação pode funcionar como uma etiqueta, ser colocada no código binário final, e assim permitir que sejam identificadas as revisões que compuseram determinado arquivo binário.

Entretanto, o *RCS* tem ainda alguns problemas, e assim foram necessárias mais pesquisas e avanços para que novas ferramentas fossem idealizadas para resolver alguns problemas que o *RCS* ainda possuía.

3.2.2 *CVS*

O *CVS*[2] é outra ferramenta de controle de revisões que permite que sejam mantidas diferentes revisões de um arquivo, normalmente código-fonte, e também manter registros históricos de quem, quando e por que estas modificações aconteceram [CEDERQVIST, 1992,1993]. Até aqui, parece uma ferramenta igual ao *RCS*, entretanto, não é tão simples assim. O *CVS* não trabalha apenas com um arquivo ou dentro de um determinado diretório como o *RCS*, mas é capaz de gerenciar e operar sobre uma coleção hierárquica de diretórios que contenham arquivos versionados, e é isso que diferencia o *CVS* do *RCS*.

Por causa desta característica, o *CVS* ajuda muito no gerenciamento de liberações, no controle de edição simultânea de arquivos entre vários programadores e ainda tem um sistema de gatilhos, que permite controlar as diversas operações da ferramenta *CVS*, permitindo até que esta seja usada em redes WAN.

Outra característica importante, é que o *CVS* mantém apenas uma cópia dos arquivos mestres, que é chamada de "repositório". O repositório central contém todas as informações necessárias para que qualquer programador, em qualquer lugar, possa requisitar e reconstruir uma revisão que tenha sido previamente, armazenada no sistema.

Como o *CVS* é um aperfeiçoamento do *RCS* e utiliza o mesmo método de armazenamento, os arquivos individuais são compatíveis entre os dois sistemas de versionamento,

[2] *Concurrent Versions System*

assim o *CVS* possui as mesmas funções que o *RCS* possui, somente, que estendidas para suportar as árvores de diretório.

Conceito dos *sandboxes*

O *CVS* usa uma maneira diferente dos demais sistemas de versionamento, os desenvolvedores podem sempre trabalhar com os arquivos de forma concorrente. Normalmente, esta é uma operação simples e direta, um certo usuário faz um *check-out* de uma estrutura do repositório, edita e realiza as modificações necessárias nos arquivos de seu interesse, e após executa um *commit*.

Se neste momento, outro usuário realizou alguma modificação nos mesmos arquivos, o *commit* irá falhar e para corrigir o problema deve ser feito um *update* a partir do repositório, que irá mixar todas as modificações que existam no repositório, com as modificações que o próprio usuário fez ao arquivo.

Algumas vezes, esta mixagem não pode ser feita automaticamente, por exemplo: quando os dois usuários realizaram mudanças na mesma linha de código, chamado conflito. Os conflitos ocorrem muito raramente. Quando isso ocorre o *CVS* coloca as duas modificações com separadores entre si, e permite ao usuário que o resolva, editando o arquivo, novamente, para resolver o impasse ocorrido nesta determinada linha [CVS, 2002]. Depois disso é só refazer o *commit* que desta vez irá funcionar.

Este método de trabalho, onde cada usuário reside numa caixa-de-areia (*sandbox*), tem muitas vantagens, tais como: modificações feitas por outros usuários são isoladas até o momento em que um certo usuário quer fazer um *check-in* para enviar as suas modificações para o repositório; evita os bloqueios ou impedimentos que outros sistemas impõem, quando um arquivo está em estado de *check-out*; qualquer desenvolvedor pode trabalhar com os seus arquivos sem ter que estar em contato direto com o servidor de arquivos. O acesso ao servidor de arquivos é necessário apenas, no momento, de um *update* ou um *commit* serem realizados.

Terminologia

O *CVS* usa uma terminologia semelhante aos demais sistemas de versionamento, porém com significados diferentes para o *CVS*. Exemplo: alguns termos listados a seguir e seus significados num ambiente de desenvolvimento que utiliza o *CVS*:

- *Checkout*

  Se refere à primeira cópia dos dados de um módulo inteiro a partir do repositório central;

- *Commit*

  É a ação de enviar ao repositório as modificações realizadas pelo usuário;

- *Export*

  Refere-se a operação de retirar um módulo inteiro do repositório, sem as informações de controle (meta-dados) do *CVS*. Os módulos exportados não estão mais sob controle do sistema de versionamento;

- *Import*

  É a operação de criar um novo módulo no repositório, através, da importação de uma estrutura inteira de diretórios para dentro do repositório;

- *Module*

  É uma estrutura de diretórios que representa um projeto de *software* dentro do repositório central do *CVS*;

- *Release*

  É a versão de um produto como um todo, uma liberação;

- *Revision*

  É a revisão de um determinado e único arquivo num dado momento;

- *Sandbox*

  É o conceito que o *CVS* implementa, onde os usuários estão trabalhando isolados uns dos outros;

- *Tag*

  É um nome simbólico, uma etiqueta, dada a um conjunto de arquivos, num dado instante do desenvolvimento do produto, que representa alguma mudança significativa;

- *Update*

  Fazer uma atualização dos arquivos locais de trabalho a partir das informações que estão no repositório central.

Inconvenientes

Algumas operações não são nativas ao *CVS*, tais como: movimentar arquivos e renomear arquivos [FOGEL, 1999,2000]. Para realizar estes tipos de operações, deve ser feito o uso de algumas soluções de contorno (*workaround*), usando as operações de remover e adicionar arquivos ao sistema de controle do *CVS*.

3.2.3 *ClearCase*

O *ClearCase* é um sistema de gerenciamento de configuração e de controle de versão, que foi idealizado e desenhado para equipes de desenvolvimento que trabalham numa rede local [ABELL, 1998]. Um grande diferencial deste sistema, é: mesmo que este esteja sendo executado em diferentes plataformas de *hardware*, as revisões são completamente, compatíveis.

Este sistema é capaz de manipular diferentes variações de um sistema de *software*, controlar quais revisões compõem um determinado *build*, realizar uma compilação completa ou de parte de um produto, conforme a especificação do usuário, garantindo certas políticas de desenvolvimento específicas de cada grupo ou de um departamento.

A grande diferença entre o *ClearCase* e os demais sistemas mostrados anteriormente, é que: este é baseado num banco de dados que é chamado de repositório de dados seguro ou VOB[3]. Neste local, estão todos os dados que devem ser compartilhados entre os usuários do sistema, como: as revisões, os registros de histórico, meta-dados, etc; este repositório só pode ser manipulado pelos comandos disponibilizados pelo *ClearCase*, evitando assim que estragos acidentais ou até modificações maliciosas sejam realizadas sem um controle ou um histórico de quem/quando/como esta foi feita.

Área virtual para acessar os arquivos - *Views*

Para que os usuários possam visualizar o conteúdo destes repositórios, estes devem possuir uma `view`. Uma `view` é um ambiente isolado e virtual para o desenvolvimento, permitindo o acesso de cada usuário, somente, as informações que lhe são pertinentes e necessárias, ao mesmo tempo que mantém o isolamento entre os demais membros da equipe de desenvolvimento.

Além disso, esta estrutura de `view` permite ao sistema acessar diretamente, o repositório central para os dados que estão em modo *read-only*, e acessar os dados nos banco de dados individuais, para os dados que estão em modo *read-write*. Isto minimiza a ocupação de espaço em disco e a replicação de dados, garantindo que qualquer

[3] *Versioned Object Base*

modificação feita num arquivo seja visível apenas pelo usuário que está realizando tal modificação ou teste [ABELL, 1998].

Quando estas alterações estiverem concluídas ou concretizadas, podem ser enviadas para o repositório central para serem re-distribuídas para os demais integrantes da equipe de desenvolvimento. Neste caso, o trabalho de um desenvolvedor é transparente para os demais membros da equipe até a sua conclusão.

Versionamento estendido

Um dos principais requisitos para uma ferramenta de gerência de configuração é o controle de versionamento. O *ClearCase* implementa este conceito e não é somente para arquivos-texto, como em outros sistemas de versionamento, mas o *ClearCase* consegue manipular, além de arquivos-texto, arquivos-binários e diretórios, que permite ao sistema controlar as modificações ocorridas na organização dos arquivos dentro da árvore de diretório. As modificações que podem ser controladas são: criação, renomeação e exclusão de arquivos [ABELL, 1998]. Estas modificações são tão importantes quanto as modificações ocorridas dentro dos próprios arquivos-fonte.

3.3 Versionamento ou controle de arquivos

Versionamento é a tarefa de manter os sistemas de *software* que são compostos de diversas versões e configurações muito bem organizadas.

Para o versionamento de arquivos, a principal função é gerenciar os grupos de revisão. Um grupo de revisão é uma coleção de documentos de texto chamados de revisão, que evoluem de um para outro. Normalmente, as ferramentas que ajudam a fazer e cumprir essas tarefas, são chamadas de ferramentas de controle de versão ou ferramentas para versionamento.

Uma nova revisão pode ser criada a partir da edição de uma revisão mais antiga ou uma nova revisão pode ser criada a partir da revisão a qual o trabalho estava sendo feito. Assumindo essa estrutura como válida, surge uma estrutura de ancestrais, como uma árvore genealógica e é neste conceito que a maioria das ferramentas existentes no mercado se baseiam.

Outro conceito interessante que deve ser citado e já foi conceituado anteriormente, é o armazenamento dos dados de cada arquivo. Suas revisões são feitas a partir das diferenças existentes entre o arquivo ou revisão atual e seu imediato predecessor. Se essa estrutura for assumida como verdade, a partir de um nó 0 (zero), onde o arquivo é vazio, e o arquivo for evoluindo, através, dos diversos novos nós desta árvore estariam

as novas revisões do arquivo.

3.3.1 Operações básicas

Para o versionamento de arquivos e demais objetos de um sistema, onde se está implantando a gerência de configuração é necessário que a ferramenta utilizada seja capaz de realizar três operações básicas, *check-in*, *check-out* e *stamping*. Estas operações são as responsáveis por manter a estrutura de controle de versão e garantir que as diversas operações não sejam conflitantes e que todas as modificações possam ser controladas. Os exemplos dessas operações, serão feitos, usando uma das ferramentas mais comuns que existem para controle de versão, o *RCS*[4]. Supondo que um determinado arquivo texto, `arq.c` está sob controle do *RCS*, utilizando o comando de *check-in*:

```
ci arq.c
```

será criada uma nova revisão com o conteúdo do arquivo `arq.c`, que será considerada a revisão inicial, a qual será numerada como 1.1 pela ferramenta. Essa versão será armazenada ou guardada no arquivo `arq.c,v`. Normalmente, esse comando requer uma descrição, que ficará guardada junto com o arquivo `arq.c,v`. A partir daí, será requerida uma nova descrição para cada novo comando de *check-in*.

Todos os arquivos que têm a extensão `,v` são chamados de arquivos do *RCS*. Essa extensão `,v` vem da palavra versão e os demais arquivos são chamados de arquivos de trabalho. Para se obter o arquivo de trabalho a partir de um arquivo *RCS*, deve ser executado um comando de *check-out*:

```
co arq.c
```

este comando irá extrair a última revisão do arquivo `arq.c` que está armazenado no grupo de versões (arquivo `arq.c,v`) e irá escrever esta revisão no arquivo `arq.c`. Este poderá ser editado e modificado. Quando estiver pronto, poderá ser feita uma nova operação de *check-in* com o comando:

```
ci arq.c
```

após a finalização deste comando, este colocará como a última revisão do arquivo `arq.c` o valor 1.2, indicando que é a segunda revisão deste arquivo. Normalmente, essa é a seqüência de operação de um desenvolvimento, baseado em controle de versão de

[4] *Revision Control System*

arquivos. A qualquer momento, é possível voltar atrás, com as modificações realizadas, caso estas não sejam eficazes.

3.3.2 Identificação automática

Uma ferramenta de versionamento de arquivos como o *RCS* é capaz de carimbar o código fonte e conseqüentemente, o código objeto com identificadores, que são similares a números de série de outros produtos. Isto é a capacidade de *stamping*, para obter esta identificação, coloque a seguinte cadeia de caracteres:

```
$Id$
```

dentro do arquivo texto que contém a revisão na qual se está trabalhando. Quando a operação de *check-out* é executada, esse identificador é substituído pelo comando `co` pela seguinte cadeia de caracteres:

```
$Id: gerevol.tex,v 1.1 2003/01/08 11:38:23 root Exp root $
```

através desta cadeia de caracteres, sempre é possível saber quais foram os componentes que geraram determinado programa. Essa facilidade é imprescindível para os programadores, quando se necessita fazer uma manutenção.

Para ajudar na documentação do código, existe um outro identificador que acumula as mensagens de *log* de cada operação de *check-in*, o que nos permite colocar esta informação diretamente, dentro do código-fonte, conseguindo um histórico completo das modificações feitas no arquivo.

3.3.3 Árvore de revisões

A grande maioria das ferramentas de controle de versão usa uma tecnologia de árvore de revisões, como visto nos conceitos básicos, para manter e controlar as diversas revisões. No caso do *RCS* que está sendo usado como exemplo, ele arranja as revisões numa árvore de ancestrais. O comando `ci` realiza a inicialização desta árvore e um comando auxiliar chamado de `rcs` manipula esta árvore.

Nesta árvore, a revisão que corresponde ao nó raiz, normalmente, é chamada ou numerada de 1.1 e as sucessivas revisões vão sendo incrementadas a partir desta, do seguinte modo: 1.2, 1.3, 1.4, etc. O primeiro campo, corresponde ao número de liberação e o segundo campo deste número, equivale ao número de revisão. Sem que nada seja

dito ao comando `ci`, este sempre incrementa o número da revisão, a partir, do número de revisão da edição anterior do arquivo.

O número de liberação sempre deve ser incrementado, explicitamente, através do comando `co -r`. Como exemplo o seguinte comando pode ser executado:

`ci -r2.1 arq.c` ou `ci -r2 arq.c`

que irá criar a partir de agora, revisões numeradas a partir do 2.1. Um subseqüente *check-in* irá criar uma revisão 2.2 e o próximo 2.3 e assim por diante. Normalmente, o número de liberação deve ser incrementado, quando da liberação de uma nova versão de *software*, ou quando exista uma modificação muito grande no desenvolvimento do produto.

3.3.4 A importância dos ramos

Uma árvore de revisão muito nova, normalmente, é esguia, ou seja, consiste apenas do ramo principal, o qual é chamado de tronco. Conforme esta árvore vai envelhecendo, os ramos podem começar a aparecer, e esses ramos são necessários em quatro situações distintas, as quais serão detalhadas a seguir:

Consertos temporários

Tomando como exemplo uma árvore de revisão que contenha 5 revisões agrupadas em 2 liberações, como é mostrado na figura 3.4:

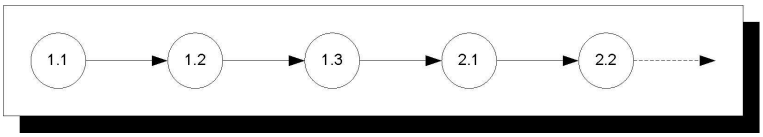

Figura 3.4: Árvore de revisão jovem e sem ramos

Neste caso, a revisão 1.3 é a última revisão da liberação número 1, e está em operação no cliente, enquanto a liberação número 2 é a que está ativamente, em desenvolvimento. Imagine que existe um problema detectado no cliente o qual, este está requisitando que uma solução seja implantada. O *RCS* não permite inserir uma nova revisão no meio da árvore de revisões, pois isto, não representaria a verdadeira história do desenvolvimento do produto, então como seria resolvido o problema?

Este é um dos casos que a ramificação é necessária, assim deve ser criado um ramo na revisão 1.3 e numerá-lo como 1.3.1, assim as revisões que porventura sejam

necessárias neste ramo, seriam numeradas de 1.3.1.1, 1.3.1.2, e assim por diante. Este tipo de numeração é necessária, permitindo que um segundo ramo parta da mesma revisão, como por exemplo: 1.3.2, onde a numeração neste caso seguiria a seguinte lógica: 1.3.2.1, 1.3.2.2, etc.

Os comandos necessários para a realização dessa operação são os seguintes:

`co -r1.3 arq.c` - *check-out* da revisão 1.3
`vi arq.c` - modificações para corrigir o problema
`ci -r1.3.1 arq.c` -*check-in* da revisão no ramo 1.3.1

esta seqüência de comandos transforma a árvore de revisão da figura 3.4 na árvore de revisão representada pela figura 3.5.

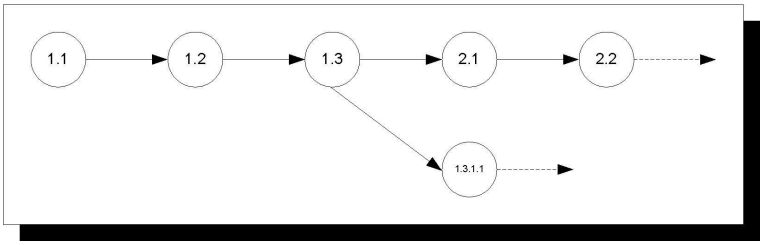

Figura 3.5: Árvore de revisão com um ramo lateral

Distribuição de novas liberações ou correções do cliente

Assumindo que a situação descrita na figura 3.4, onde a revisão 1.3 já está em operação em diversas localidades do cliente, enquanto os desenvolvedores trabalham na liberação número 2, e o nosso cliente fez a requisição para a solução do seu problema específico. Estas modificações não podem ser anexadas ao ramo principal de desenvolvimento, porém devem ser mantidas em um ramo lateral. Quando a nova distribuição (liberação) estiver pronta para ir para o cliente, as modificações que estavam no ramo lateral devem ser agregadas ao desenvolvimento principal e assim gerar-se um novo ramo lateral que contenha as modificações particulares de um cliente com o agregado das novas funcionalidades da nova liberação. Seguindo com o exemplo, aparece um ramo na revisão 1.3, que corrigiu o problema requerido pelo cliente, e o desenvolvimento que prosseguiu até a versão 2.3. Para o cliente específico, deve ser feita uma união do ramo principal com o ramo lateral, gerando a versão 2.3.1.1, como mostra a figura 3.6.

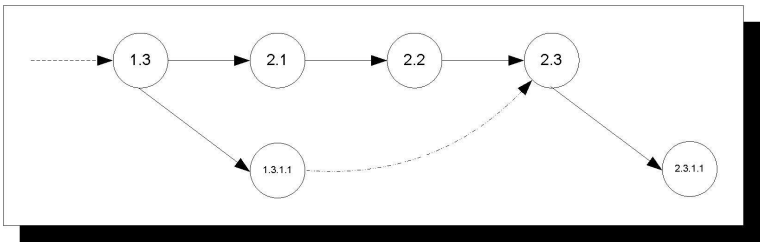

Figura 3.6: Árvore de revisão com modificações requeridas pelo cliente

Desenvolvimento em paralelo

Muitas vezes, é necessário ou desejável um desenvolvimento em paralelo com o a linha ou ramo principal, a fim de testar um projeto ou um algoritmo alternativo ou uma diferente técnica de implementação, sem que isto afete, altere ou impeça o desenvolvimento principal. Para que isto ocorra, deve ser lançada mão de um ramo lateral no desenvolvimento, e quando esta experimentação estiver concluída, e for bem sucedida deve ser re-integrada ao desenvolvimento principal ou então ser descartada.

Atualizações conflitantes

Uma situação bastante comum, no momento de um desenvolvimento em paralelo, ocorre, quando um determinado programador executa um *check-out* numa determinada revisão e deve ficar com esta revisão durante um bom tempo para solucionar o problema. Neste tempo, um outro programador deve executar uma modificação prioritária; o segundo programador poderia passar as modificações para o primeiro agregar ao arquivo, já em estado de *check-out*, mas o problema é que como o primeiro programador estava executando modificações no arquivo, que ainda não foram finalizadas, este não poderá realizar o *check-in* deste arquivo, inviabilizando a solução.

A solução surge através, dos ramos laterais, onde o segundo programador executa o *check-out*, realiza as modificações necessárias no arquivo, faz o *check-in* num ramo lateral que será posteriormente, re-integrado ao desenvolvimento principal, quando for possível.

3.4 Versionamento ou controle de diretórios

Como foi citado anteriormente, o versionamento de diretórios é muito importante para a gerência de configuração. A manutenção correta de um sistema de versionamento garante um produto com melhor qualidade e facilidade de identificar e corrigir os erros,

implementações de novas características são menos traumáticas, citando apenas alguns exemplos.

No caso de gerenciamento de diretórios, as operações que manipulam o diretório devem ser controladas, e não as operações de escrita/leitura que ocorram num arquivo, para serem realizadas modificações nos arquivos, é indispensável fazer um *check-out* e depois um *check-in* do mesmo, sem que seja preciso fazer um *check-out* do diretório, onde se encontra este arquivo.

3.4.1 Operações básicas

Devido ao fato de se tratar o diretório como um elemento versionado, todas as operações que modifiquem a sua estrutura devem ser controladas, porque a qualquer instante, pode haver a necessidade de retorno desta estrutura de diretório, completamente reconstruída.

As operações que devem ser mantidas sob controle, são as seguintes:

- Adicionar novos arquivos
- Renomear arquivos existentes
- Excluir arquivos
- Adicionar novos sub-diretórios
- Mover arquivos ou diretórios
- Adicionar atalhos
- Remover atalhos

Para qualquer destas operações descritas, um *check-out* e um *check-in* do diretório que será manipulado deve acontecer, assim o controle das mudanças ocorridas neste momento, na árvore de diretórios, fica garantida e obtém-se a capacidade, a qualquer instante, de reverter modificações ocorridas.

Nem todas as ferramentas implementam este tipo de controle, no escopo desta dissertação, a *FGC* que tem estas características constituídas é somente o *ClearCase*.

3.4.2 Seleção de revisões

Um dos grandes problemas no gerenciamento de diretórios, ocorre no momento da seleção de revisões. Como podem haver alterações nos arquivos sem que haja uma

mudança na versão do diretório? Isto é possível, porque a alteração de um arquivo é um problema para o gerenciamento de arquivos e não para o gerenciamento de diretórios.

Quando se tenta retirar uma revisão de diretório no qual o arquivo não existia, o usuário não verá o respectivo arquivo, pois o arquivo "sumiu" da estrutura do sistema de arquivos sob controle da ferramenta de gerência de configuração. Entretanto, quando este retirar uma revisão posterior à criação do referido arquivo, este estará presente e controlado.

Na realidade, o usuário não perdeu nada, apenas o arquivo dele foi escondido pela ferramenta, pois na revisão do diretório que este pediu, o arquivo ainda não existia ou tinha outro nome.

Neste caso, a seleção de revisões a partir de nomes simbólicos ou etiquetas, resolve o problema de seleção por revisões diferentes (será visto no item **Controle de configuração**), pois se for aplicado o mesmo carimbo, tanto no arquivo quanto no diretório a serem vistos, sempre que for selecionada aquela revisão, tudo o que é necessário será visto, sem nenhum problema.

3.5 Versionamento ou controle de produtos

O versionamento de produtos é responsável por controlar aquilo que foi entregue ao cliente, quando esta determinada versão foi liberada e para qual cliente o produto foi disponibilizado. Tendo este sistema implantado, é extremamente simples implementar modificações específicas para cada cliente, sendo isso uma requisição ou não. Realizar manutenções preventivas ou correções, também, é mais simples, pois sempre é alcançado o controle total do produto entregue ao cliente. Além disso, todas as modificações realizadas para um cliente, caso seja pertinente e necessário, podem ser incorporadas para as versões distribuídas para os outros usuários com extrema facilidade, devido à utilização da *FGC*.

Dependendo do tipo de ferramenta de gerência de configuração que é utilizada, é até possível incorporar ao arquivo binário, entregue ao cliente, informações da gerência de configuração, as quais permitirão reconstruir toda a árvore de revisões, sem que o ambiente de desenvolvimento seja completamente montado. É claro que esse recurso deve ser pensado como última alternativa, pois o trabalho necessário para esta tarefa é grande, dependendo do tamanho do produto e da quantidade de arquivos-fonte necessários para construí-lo, mas é factível.

3.6 Controle de ferramentas

O controle de ferramentas também é um ponto crucial para a gerência de configuração, pois muitas vezes é preciso ter sob controle os programas auxiliares, tais como compiladores, bibliotecas, produtos comprados de terceiros, entre outros.

Tudo é necessário para num momento futuro, ser reconstruído todo o ambiente de desenvolvimento que existia no instante o qual o produto foi compilado ou estabilizado. Nem sempre, uma versão posterior de um determinado programa ou biblioteca é completamente compatível com uma versão mais antiga, o que pode injetar erros no programa que não existiam no momento em que foi compilado, ou que não fazem parte da requisição de entrada para as modificações para as quais se está remontando todo o ambiente.

Num ambiente sem *FGC* freqüentemente é visto, os desenvolvedores/programadores enfrentando problemas com as ferramentas e a compatibilidade entre elas, e não resolvendo os problemas contidos no desenvolvimento do produto que foi planejado. Para que isso não ocorra, é útil pensar uma *FGC* como uma ferramenta de apoio e ajuda para o desenvolvedor, e não como mais um processo que deve ser seguido dentro da empresa.

Para estes casos, a *FGC* é capaz de organizar o problema, porque podem ser guardadas junto ao projeto, as versões dos produtos adquiridos de terceiros para realizar uma posterior instalação ou para uma remontagem do ambiente de desenvolvimento. Através da *FGC* pode ser feita a seleção dos componentes que faziam parte de uma determinada liberação, e tais como os arquivos-fonte, as ferramentas que apareceriam após a seleção feita, seriam as versões necessárias para aquela revisão que está sendo solicitada.

Outro conceito que pode ser usado é o da criação de um documento com as explicações necessárias para a instalação de um novo posto de trabalho, e esse documento ir evoluindo junto com o projeto em si. Assim, a cada nova liberação, esse documento deve ser editado ou não, para refletir as necessidades da determinada liberação. Após uma escolha realizada, aparece o documento válido para aquele conjunto de código.

São dois modos diferentes de solucionar o mesmo problema, e devem ser avaliados, conforme a necessidade e possibilidade de cada grupo de trabalho, visando sempre garantir a reprodutibilidade do ambiente de desenvolvimento.

3.7 Controle de configuração

O controle de configuração é necessário para que, em qualquer momento, se saiba quais são as revisões individuais de cada componente de um produto, que tenha sido entregue, para poder refazer a liberação.

Uma configuração nada mais é do que um conjunto de revisões de arquivos que formam uma liberação, e estas são selecionadas conforme um critério qualquer que determina os inter-relacionamentos entre os arquivos. Para exemplificar o conceito e simplificar o entendimento, é possível citar a construção de um compilador. Para obter um compilador que funcione, completa e corretamente, e possa ser vendido é preciso ter as corretas revisões dos arquivos que formam o analisador léxico, o analisador sintático e o analisador semântico e o gerador de código, pois todos estes agrupados é que formarão o compilador.

3.7.1 Maneiras de selecionar uma determinada revisão

Seguindo com o exemplo, um pouco mais das idéias de gerência de configuração são exibidas usando a ferramenta *RCS*, nos seguintes parágrafos.

Seleção *default*

Durante o desenvolvimento, a situação mais comum de ocorrer é a necessidade de ser realizado um *check-out* da última revisão de um arquivo ou de um grupo de arquivos e do ramo principal. Assim, o comando `co` do *RCS* tem como operação padrão essa situação.

Quando se executa o comando:

```
co arq.c
```

o *RCS* é capaz de reconstruir a última revisão do ramo principal.

Seleção baseada nos números de revisão

Especificando um número de liberação ou um número de revisão, a revisão especificada pode ser selecionada. Este tipo de seleção é muito útil quando uma liberação foi feita e a equipe de desenvolvimento caminhou para a próxima etapa de desenvolvimento e de uma nova liberação. Como exemplo, é colocado um comando que extrai as últimas revisões dos arquivos da liberação número 3:

```
co -r3 *,v
```

Seleção baseado no *status*

Quando a última versão de determinada liberação não é a revisão desejada, outras opções mais sofisticadas são lançadas, como as baseadas no estado (*status*) de determinada liberação, ou ainda no autor da mudança realizada. Segue um exemplo de comando:

```
co -r2 -sReleased *,v
```

Seleção baseada na data

Outra possibilidade para que revisões sejam selecionadas, é a seleção baseada na data. Supondo que uma determinada liberação de um certo sistema, tenha sido feita no dia '22 de setembro às 06:37 horário local'. O seguinte comando nos faria o *check-out* de todos os componentes do sistema, independente do número de versão:

```
co -d'September 22, 6:37am LT' *,v
```

Um ponto interessante para esse tipo de seleção é que ele não é fixo, ou seja é um ponto de corte na linha de tempo de evolução dos arquivos dentro de um projeto, assim, ele sempre realizará o *check-out* dos arquivos os quais o *check-in* estiver mais próximo dessa data, mas não a ultrapasse.

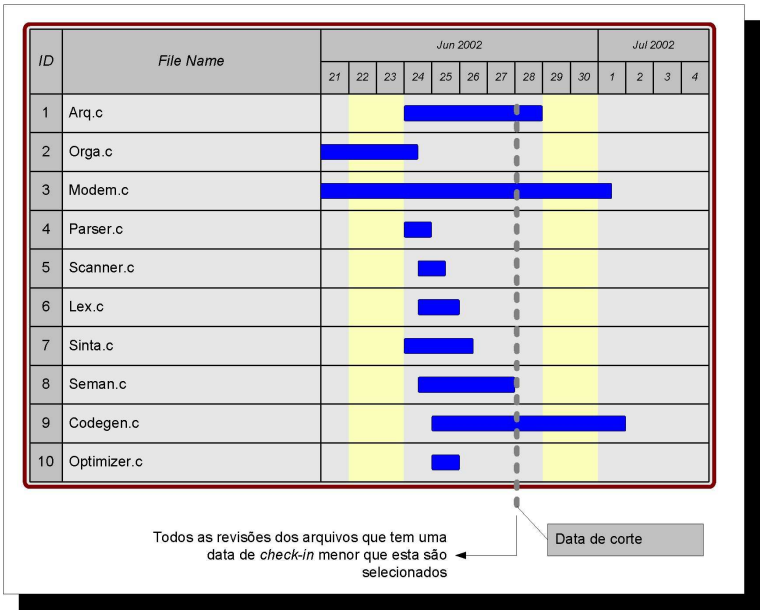

Figura 3.7: Seleção de revisões baseada em datas

Seleção baseada em nomes simbólicos (*tags* ou *labels*)

Talvez, este seja o modo de seleção mais poderoso disponível, o modo de seleção baseado em nomes simbólicos. Este tipo de seleção é que permite que a seleção de revisões seja feita para grandes grupos de desenvolvedores, onde a seleção por data ou a seleção por número de revisão não são apropriados ou suficientes para atender as necessidades.

Exemplificando a utilidade deste tipo de processo de seleção: supondo que dois grupos de desenvolvimento estejam desenvolvendo módulos e que uma terceira pessoa vai agregá-los. Neste caso, uma simples solução de datas pode não ser suficiente, porque os grupos podem ter realizado os *check-in* em momentos diferentes. Pensando em fazer a seleção por número de revisão, também pode não satisfazer ou solucionar o problema, pois pode ter um grupo na liberação número 3 e outro na liberação número 15. É neste contexto, que pode ser usada a seleção por nomes simbólicos, onde pode ser associada a uma liberação número 3 de determinado grupo um "carimbo/etiqueta" (nome simbólico) e para a liberação número 15 do outro grupo é associada a mesma etiqueta (V4 no nosso exemplo). Portanto, quando se executa o comando:

```
co -rV4 arqgrupo1.c,v arqgrupo2.c,v
```

os arquivos corretos de cada grupo são adquiridos para serem manipulados. Este tipo de seleção simplifica muito a gerência de configuração do sistema como um todo, pois é usado, somente, um comando para manipular vários sub-sistemas simultaneamente. Este tipo de seleção é tão importante, que o *RCS* criou um comando exclusivo para realizar um tipo de associação, chamado de `rcsfreeze`.

3.8 Gerenciamento de novas características

Muitos dos erros ou requisitos não atendidos, aproximadamente 20%, são causados por equívocos de interpretação dos requisitos, fazendo com que os analistas comecem a fazer reuniões entre os diversos desenvolvedores, ou entre os desenvolvedores e seus clientes para tentar minimizar as diferenças de compreensão, que existiam e tentar fazer com que o sistema não apresente ambigüidades [JALOTE, 2000].

Depois disso, as simulações baseadas em cenários começaram a ser utilizadas para realizar testes controlados, antes de ser entregue qualquer resultado de trabalho para o cliente. Em mais de 75% dos casos, os clientes sempre esperam mais do que realmente foi feito, porque num momento inicial, não souberam expressar com total clareza ou completude aquilo que desejavam.

Para minimizar os problemas ocorridos nesta fase do projeto, é adotado um processo como o mostrado na figura 3.8. Para evitar problemas que porventura venham a acontecer, por uma análise mal-feita de um novo requisito, acabe alterando outra funcionalidade já implementada.

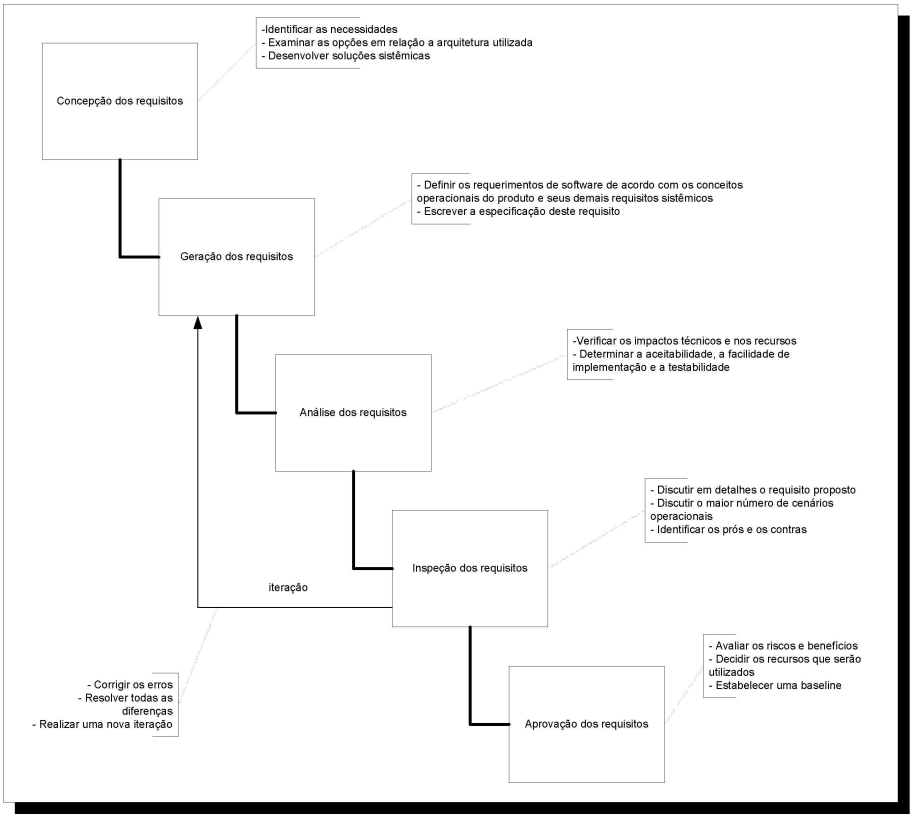

Figura 3.8: O processo da gerência de requisitos num projeto [PAULK, 1994]

Atualmente, um dos grandes obstáculos que se têm no gerenciamento de novas características é a diferença de estilos de documentação, que cada analista ou desenvolvedor utiliza. É necessário encontrar uma ferramenta que auxilie neste processo. Entretanto, este não é o escopo desta dissertação e pode vir a ser um trabalho futuro: encontrar um meio de reduzir essas diferenças de estilo, para que haja uma metodologia mais consistente na gerência de novos requisitos, agregada e incorporada ao gerenciamento de configuração.

3.9 Gerenciamento de erros

O gerenciamento de erros também é um item que deve ser visto com muito cuidado, durante o processo de desenvolvimento. Normalmente, os erros vêm de dois pontos

diferentes, o primeiro, quando é encontrado pelo teste sistêmico do produto, e o outro que vem do cliente, que são detectados na instalação do produto no ambiente do cliente. Estes últimos são os piores, por serem acontecimentos que ocorrem na frente do cliente e podem afetar a confiança do produto [PAULK, 1994].

O gerenciamento de erros deve prover maneiras pelas quais, se evitar ao máximo que um produto saia do desenvolvimento e entre em produção com algum erro que afete o seu funcionamento. Também deve prever as condições, que determinado projeto pode ou não, liberar uma versão para ser entregue ao cliente, verificando as condições de contorno que possam ser aplicadas para diminuir as desconfianças que surjam por parte do cliente final.

3.10 Avaliação do processo - Métricas

Medições são sempre necessárias para avaliar-se qualquer tipo de processo implantado, ou usado num ambiente de desenvolvimento. É quando é julgado válido ou não para um possível reaproveitamento do método ou para uma modificação sobre o mesmo [KAN, 1995]. Ou seja, num caso extremo, o seu abandono e troca por uma outra metodologia de trabalho.

Para se gerenciar um ambiente de forma transparente para todos os envolvidos, deve-se prover um sistema de medição a fim de identificar os possíveis ganhos ou as perdas ocorridas com a adoção de uma certa técnica. E para saber como estas medidas serão interpretadas, devem ser definidos valores ou tabelas comparativas para verificar se os valores alcançados são coerentes ao processo estabelecido.

Existem diversas métricas que devem ser definidas e coletadas para o processo ser avaliado de diversos modos e ângulos diferentes. Este processo de definição destas métricas específicas requer um certo trabalho, se não for feito de maneira coerente pode tornar-se inútil e desgastante.

Como exemplo de medidas bem definidas, são citados os relatórios financeiros de uma empresa, onde os dados coletados podem ser interpretados de formas diferentes, comparados entre si e em diversos períodos diferentes, ou ainda entre empresas diferentes por serem bem definidos. Estes relatórios contém métricas completamente, diferentes e que se inter-relacionam, tais como:

- Valores corporativos
  - Capitalização no mercado
  - Relação entre os preços e os ganhos

  - Ativos fixos

- Gerenciamento do processo
  - Fluxo de caixa
  - Substituição do inventário
  - Lucros líquidos
  - Substituição de clientes

- Expectativas financeiras
  - Divisão do mercado
  - Relação das contas
  - Recuperação por cliente
  - Recuperação por empregado
  - Crescimento industrial

Como pode ser visto pelo exemplo na área financeira e transposto para a área de informática, diversos pontos são refletidos, mesmo que num primeiro momento, não se pareçam co-relacionados [JALOTE, 2000]. Alguns são citados: o tempo necessário para executar *check-in* e *check-out*, que pode influenciar muito no prazo final do trabalho; pensar no espaço ocupado pelos arquivos, o espaço em disco necessário pode ser planejado, entre outros que serão vistos a seguir. Estas métricas devem avaliar todos os aspectos do processo, não só a ferramenta, os processos, os desenvolvedores, mas tudo junto e se possível o inter-relacionamento que podem ter.

Outro ponto importante citado com relação às medidas, é que o processo deve ser automático, tanto para a coleta, quanto para o armazenamento da informação e também, deve ser periódico. Caso contrário pode ser inviabilizado completamente, o uso de métricas no sistema ou no ambiente de desenvolvimento, pois pode tornar-se num processo burocrático e não um processo de auto-avaliação.

Se este processo não for automatizado, além dos usuários terem mais esta tarefa, os dados colhidos podem ser contaminados, cada um pode acabar adotando um certo método de medida ou formato de armazenamento, fazendo com que os dados sejam inválidos, incompatíveis ou inúteis para o sistema de medição.

As métricas se bem feitas, coletadas de forma automática, periódica e de modo dinâmico, serão com toda a certeza um dos melhores resultados de trabalho que a implantação de uma *FGC* pode trazer para o ambiente de desenvolvimento.

3.10.1 Tempo de execução dos comandos: *check-in* e *check-out*

Essa métrica deve ser levada em consideração, podendo influenciar nos prazos acordados para a entrega das liberações, conforme o produto vai crescendo. O número de revisões e de ramos ou a quantidade de arquivos farão com que algo que levava segundos para ser realizado, passe a levar minutos e assim, esse tempo deve ser levado em consideração no momento da realização do cronograma do produto.

3.10.2 Sobrecarga de armazenamento

Uma segunda métrica interessante que pode ser analisada é a sobrecarga que uma *FGC* pode causar no ambiente de desenvolvimento investigado, em relação ao espaço em disco ocupado. Observando um ambiente de desenvolvimento onde está implementada a gerência de configuração pode-se ver que não existe uma sobrecarga no armazenamento de informações com a adoção de uma ferramenta de gerência de configuração, como muito usuários poderiam pensar, uma vez que sempre estaremos salvando diversas revisões do mesmo arquivo.

Entretanto, através, da tecnologia de deltas que já foi introduzida nesta dissertação, a ocupação de disco é minimizada, e muitas vezes chega a ser menor do que fazer uma cópia de uma estrutura de diretórios completa para guardar uma versão de um certo módulo ou mesmo produto inteiro.

Ao assumir que as revisoes são mais ou menos do mesmo tamanho, ou seja, tem um tamanho médio aproximado, podemos ter boas estimativas do espaço em disco necessário para as revisões extras que estão sendo criadas para a finalização do produto, e assim garantir que se terá meios de armazenamento até a conclusão do projeto, assim, esta métrica pode ser útil para se estimar a necessidade de novas compras de unidades de armazenamento.

3.10.3 Tamanho médio dos arquivos

Uma métrica deste tipo pode ser muito útil para o administrador do sistema para planejamento de aquisição de novos equipamentos para armazenamento, ou para o instante em que um novo projeto tem início, é necessário fazer algum tipo de estimativa de espaço que este vai precisar. Essa métrica é muito importante para que o administrador da ferramenta de controle de configuração possa se planejar, vendo onde e qual máquina, pode alocar para o determinado projeto. Prevendo eventuais necessidades, bem como recursos para cópias de segurança.

3.10.4 Número de revisões dos arquivos

Esta pode ser uma métrica interessante para ser colhida dentro de um ambiente de desenvolvimento com gerência de configuração aplicada, esta pode informar se os usuários estão usando corretamente, ou não, a ferramenta de gerência de configuração e ao mesmo tempo, ainda indicar com está o andamento do projeto.

3.10.5 Número de linhas nos arquivos

Uma métrica interessante para ser coletada, se for possível, não somente identificar, a quantidade total de linhas, mais identificar e diferenciar as linhas de código, linhas de comentários e linhas em branco, dentro do arquivo. Através de uma análise mais detalhada do relacionamento entre todos estes aspectos é verificada a produtividade de um determinado desenvolvedor, ou a velocidade com que esta determinada pessoa é capaz de escrever código em uma determinada base de tempo.

Novamente, através de análises estatísticas sobre estes dados coletados de forma periódica e automática, também é investigado em que parte do projeto se está, se no início, onde existe sempre uma grande variação na quantidade de linhas de código, enquanto que numa fase mais adiantada esse número de linhas tende a crescer menos ou até mesmo diminuir, podendo mostrar que é uma fase de manutenção do projeto. Esse tipo de informação extraída do ambiente de desenvolvimento é de extrema utilidade para os planejadores e pessoas que fazem o acompanhamento do projeto como um todo, ajudando os gerentes a tomar decisões críticas, tais como alocar mais recursos para o projeto ou retirá-los e transferi-los para outras atividades, quando estes já não são mais necessários.

3.10.6 Número de classes, métodos e atributos

Seguindo essa linha, é avaliado, no caso de programas escritos em linguagens orientadas a objetos, algumas medidas como: a quantidade de classes, e de métodos ou ainda a relação entre o número de métodos privados e públicos, ou a relação dos atributos [JACOBSON, 1992]. Estas medidas são, neste tipo de programação, melhores do que o número de linhas de código. A característica de herança, muitas vezes, pode mascarar resultados ou mesmo não mostrar nenhum valor neste tipo de programação.

3.10.7 Número de erros detectados

É uma métrica muito interessante para a avaliação da qualidade do produto final. Através dela, é observado: se o produto foi melhorando ou não, conforme as liberações são entregues, permitindo que os gerentes e responsáveis possam tomar as ações necessárias, evitando entregar produtos com baixa qualidade, ou verificar como a equipe de um certo projeto está progredindo e evoluindo com o seu conhecimento, tanto do produto, quanto pela tecnologia utilizada na construção do mesmo.

Esta métrica também pode servir de *milestone* para a liberação de uma certa versão do produto, e definindo nas especificações do projeto que este só poderá ser liberado se não tiver nenhum erro-crítico e no máximo 5 (cinco) erros-menores, somente, como um exemplo indicativo.

Chegando a definição do que é um erro-crítico, um erro-maior e um erro-menor, o que também e feito no início do projeto. Em linhas gerais, um erro-crítico é aquele que impede o funcionamento do produto, um erro-maior é aquele que limita alguma funcionalidade e tem algum tipo de *workaround*. Já um erro-menor é aquele que não atrapalha em nada a utilização do sistema e não afeta o resultado final do processamento. Normalmente, são erros que acontecem na interface gráfica com o usuário, posicionamento de janelas ou objetos gráficos.

3.10.8 Considerações gerais

As métricas citadas anteriormente são números indicativos para ajudar as pessoas responsáveis por tomadas de decisão, a terem através, de dados estatísticos um número de acertos maiores nas decisões tomadas, ajudando não só em um projeto mas tentando ver os relacionamentos que existem nos diversos projetos que, normalmente, se desenvolvem em paralelo. Desta forma pode ser aumentada a ocupação dos recursos, melhorando a distribuição de pessoal e diminuindo os custos gerais, pontos cruciais que os gerentes sempre estão perseguindo.

Métricas são as ferramentas para realizar este tipo de trabalho. Com a adoção de uma *FGC*, a implementação desta, fica mais facilitada. Muitos dos serviços que o usuário deveria realizar passam a ser feitos de forma automática, periódica e consistente pelo próprio ambiente de gerência de configuração. Desta forma, tirando uma pesada carga dos ombros dos programadores, que muito freqüentemente não são adeptos a realizar tarefas diferentes à programação.

3.11 Aplicação dos conceitos

Nesta seção será abordado um processo evolutivo de como aplicar a gerência de configuração, desde os conceitos mais simples até os conceitos avançados de controle de configuração, visando assim, tentar chegar ao máximo de controle possível sobre o processo de desenvolvimento de produtos. Desta forma, busca-se deixar sempre que possível o desenvolvedor livre dos problemas de gerência de configuração e livre para pensar na solução do seu problema de desenvolvimento.

3.11.1 1º Caso - Projeto sem controle

Um projeto sem controle, o que acontecia, normalmente, em empresas pequenas e com poucos programadores nos anos 70 e meados dos anos 80, sempre causaram grande *stress* às equipes de gerenciamento e às equipes de desenvolvedores.

Um projeto que tinha como plano inicial gastar R$200.000,00 e levar seis meses para ser concluído e sem controle algum, acabava durando um ano ou dois e tinha um gasto bem superior ao previsto, acarretando uma série de problemas, devido à dificuldade em avaliar o custo do desenvolvimento e portanto o custo do produto final. Era muito difícil avaliar corretamente, em quanto tempo, o investimento no desenvolvimento seria depreciado totalmente e quando começariam a aparecer os lucros para a empresa e seus acionistas, bem como novas fontes de investimentos para novos produtos.

Não havendo nenhum controle, os arquivos do projeto eram guardados em simples diretórios, o que causava sérios problemas, tais como:

- dificuldades no desenvolvimento em paralelo ;
- replicação de informações, quando se tinha que desenvolver uma nova versão do produto;
- custo excessivo de armazenamento;
- qualidade do produto final;
- etc.

Falta de controle

A falta de controle no processo de desenvolvimento era responsável muitas vezes, pela interrupção ou pela finalização prematura de um projeto ou produto, pois o custo já tinha ultrapassado um valor pelo qual era possível conseguir-se um certo retorno,

quando o produto fosse vendido. Se fosse vendido, pois muitas vezes o *time to market* é perdido.

Outro grande problema que surge, quando existe um projeto deste, sem o controle adequado, é que os gerentes, diretores ou investidores têm um certo temor em investir mais e mais, pois não se consegue planejar de quanto será o retorno e quando este virá, entre outras coisas, fazendo com que as pessoas fiquem retraídas e não confiantes no resultado. Essa falta de confiança acaba chegando até aos desenvolvedores, que imaginam que o projeto no qual estão trabalhando não tem importância, diminuindo assim a auto-confiança e a auto-estima, afetando a produtividade final dos indivíduos da equipe e por conseguinte da equipe como um todo.

Para equipes de desenvolvimento que contenham muitas pessoas, é muito difícil planejar novos projetos, planejar a mudança das pessoas de um projeto para o outro ou de uma tarefa para outra, ou planejar a ocupação ou realocação dos equipamentos. Freqüentemente se ouve a frase "sempre se está apagando um incêndio" o que, na realidade, é uma frase que resume muito bem o que acontece num projeto tratado dessa maneira.

Desenvolvimento em paralelo

O desenvolvimento deste tipo de projeto em paralelo fica bastante prejudicado, porque na prática, passa a ser uma tarefa difícil de ser gerenciada. Este desenvolvimento, sem uma ferramenta de controle, insere uma grande quantidade de erros no produto final, inerentes ao processo que se usa neste tipo de compilação, que sempre é um processo manual de "mixagem" dos arquivos *merge*.

Isso sem contar na quantidade de dados duplicados que devem ser disponibilizados para as diversas equipes de desenvolvimento ou para os usuários que irão utilizar os diversos módulos, uma vez que os próprios módulos estão sob desenvolvimento. Assim, o problema de retorno dos erros encontrados na utilização destes módulos, insere no desenvolvimento mais uma variável, que é o problema de se gerenciar os arquivos em uso, para verificar se as mudanças estão sendo agregadas de forma correta e não afetam outros grupos de desenvolvedores.

Esta é uma difícil tarefa para aplicar, sem a ajuda de uma ferramenta de gerenciamento de arquivos, pois se gasta um tempo enorme e mesmo assim a qualidade do produto final pode ficar comprometida.

Otimização de recursos

Outro grande problema dessa falta de controle, é que a utilização dos recursos fica bastante prejudicada, normalmente, existe uma grande duplicidade de informações espalhadas por vários lugares na rede, ou ainda o que é pior, nas máquinas dos desenvolvedores. No momento, de uma liberação ser gerada, não é conhecida a versão válida para cada módulo que compõe o produto final, sendo necessária a presença de cada desenvolvedor para gerar essa liberação.

Quando é recebida uma notificação de falha em campo, é muito difícil controlar, onde essa correção deve ser feita, porque o desenvolvimento continuou após a entrega do produto, no momento do retorno de alguma falha, os códigos já não são os mesmos que foram usados para gerar o produto que foi entregue. Caso haja retorno, é necessário realizar mais testes para garantir que o produto esteja corretamente integrado.

O custo dos equipamentos também é maior, porque todas as máquinas devem ter discos de maior capacidade a fim de guardar as cópias necessárias, para que cada desenvolvedor consiga realizar a compilação do produto. Quando uma liberação é gerada, para resolução do problema anterior, guardando uma cópia das informações, existe a replicação de todos os dados, para que seja obtida a capacidade de reproduzir uma certa versão. Ao mesmo tempo, se vier em mente que o problema foi resolvido, outro pode ser criado: o transporte de uma correção qualquer ocorrida numa versão anterior para a posterior, ou para a versão que os programadores estão escrevendo. É extremamente complicado lembrar de todas as modificações executadas nos diversos arquivos, que se somam para montar um determinado módulo. Com muito trabalho o contorno de todos esses problemas é alcançado. Mas muito tempo é gasto e existe um grande desgaste na equipe de desenvolvimento, e entre a "empresa" e o seu cliente.

O controle por parte dos gerentes, para a alocação de determinado funcionário para uma nova tarefa, também é bastante prejudicado, pois é difícil ter a certeza de que um projeto vai acabar, ou que um certo equipamento vai ficar disponível para ser re-utilizado em outro projeto.

3.11.2 2º Caso - Acrescentando o controle de arquivos

Um projeto, onde é acrescentada a gerência de arquivos ou controle de arquivos, começa a ficar mais maleável, é quando se começa a ter um pequeno aumento no controle das informações. Isto faz com que uma aproximação melhor das projeções seja obtida, que são feitas em termos de recursos, gastos e equipamentos necessários para o completo desenvolvimento da solução proposta.

A gerência de arquivos não resolve todos os problemas, mas já existe um grande

incremento na precisão das informações geradas, bem como possibilita começar um desenvolvimento em paralelo de maneira controlada, onde existem ferramentas que gerenciam e controlam a aquisição dos arquivos antes de dar início a um trabalho de desenvolvimento de uma determinada solução ou algoritmo. Este processo de aquisição dos direitos de manipular os arquivos (*check-out*), garante que uma modificação nunca será perdida no processo de desenvolvimento, a fim de que este arquivo faça parte do repositório do produto final. Os diversos desenvolvedores que "pegaram" o arquivo para si, devem necessariamente, realizar uma operação de armazenamento (*check-in*). Como a ferramenta faz este controle, todas as modificações serão mixadas, antes que o último programador consiga realizar o seu armazenamento, tornando todas as suas modificações disponíveis para todos os grupos, dentro do ambiente de desenvolvimento e corretamente integradas.

Desenvolvimento em paralelo

Com uma ferramenta de gerenciamento de arquivos, ou mais comumente, chamada de ferramenta de gerenciamento de configuração (*FGC*), o desenvolvimento em paralelo torna-se possível de ser implementado de forma eficiente, pois as alterações nos arquivos serão controladas e elencadas. Desta maneira, sempre é possível verificar quais foram as modificações introduzidas, quem foi o responsável por essas alterações e quando foram feitas. Com todas estas informações "em mãos" a ferramenta é capaz de liberar ou não o acesso a um determinado arquivo ou forçar o usuário a executar um *merge* das informações contidas nas diversas versões do arquivo em questão ou mesmo retornar ao ponto de partida, caso a implementação não solucione o problema.

Com este controle implementado, existe a possibilidade de diversas equipes de desenvolvimento, trabalharem em paralelo, e todos os produtos de trabalho por elas geradas podem ser usados sem perder o controle ou sem que exista perda de informação, ou acontecer o retorno de algum erro já corrigido anteriormente.

Otimização dos recursos

Outro ponto que tem um ganho considerável, quando passa a se utilizar uma *FGC* é na questão de armazenamento das informações. Para mostrar este conceito, eis os exemplos:

Num ambiente de desenvolvimento sem uma *FGC* a organização de informações em forma de diretórios tem que ser feitas, onde se acham as versões que estão "congeladas" ou "estabilizadas" para a utilização de possíveis outros grupos. Enquanto os responsáveis por este ou aquele módulo estão trabalhando num outro diretório, para

dar continuidade à implementação e à solução dos problemas por eles enfrentados.

E isso só tende a se agravar, conforme o projeto vai evoluindo e novos módulos vão sendo necessários. Ao mesmo tempo, as liberações que tenham sido feitas começam a necessitar de correções de defeitos, ou modificações requisitadas pelos clientes.

Com a colocação da *FGC* este problema pode ser minimizado, pois individualmente, os arquivos já são armazenados na forma de deltas, onde sempre são guardadas, somente, as diferenças entre uma revisão e outra. Através, da utilização de *check-out's* baseados em etiquetas, o trabalho pode ser feito num mesmo diretório, evitando a duplicidade dos arquivos para os diversos grupos de trabalho. O grupo que precisa do módulo, somente, para compilar o resto do produto, pode retirar através de uma operação de *check-out* uma revisão intermediária da árvore de revisões, enquanto os programadores do módulo em questão trabalham em uma revisão posterior sem que haja conflito algum.

Para o caso de liberações já realizadas dos produtos, a *FGC* tem o recurso de ramos (*branch*), onde também se tem a ajuda da ferramenta para minimizar a ocupação do disco e também, para permitir a modificação em paralelo dos arquivos em questão.

3.11.3 3º Caso - Acrescentando o controle de produtos

Com a colocação da gerência ou controle de produtos, logo após a adoção da gerência de arquivos, aumenta mais o controle sobre os produtos de trabalho gerados pela equipe de desenvolvimento, permitindo mais trabalho em paralelo, aumentando, simultaneamente, a satisfação do cliente e a ocupação dos desenvolvedores.

Através da gerência de produtos, os problemas que foram detectados em um determinado cliente podem ser catalogados e analisados para serem resolvidos, implementados, ou agendado para uma nova liberação do produto. Ao mesmo tempo, a equipe de desenvolvimento continua os trabalhos de maneira transparente para todos os níveis, sejam eles operacionais ou gerenciais.

Suporte tecnológico

O suporte tecnológico necessário para colocar este tipo de ação em prática, é basicamente, o suporte que uma *FGC* pode dar através dos ramos ou *branch*. É através desta tecnologia, que é alcançada a possibilidade de deixar alguns programadores trabalhando numa versão anterior, fazendo as modificações necessárias para corrigir os problemas detectados, sem interferir no trabalho dos demais programadores que foram adiante com o desenvolvimento do projeto.

Através da adoção desta característica, o ramo principal continua a sua evolução dentro do projeto, implementando os requisitos planejados no início do projeto, enquanto as equipes responsáveis em atender os diversos clientes abrem um ramo lateral e fazem os seus respectivos trabalhos em paralelo ao desenvolvimento principal, usando todo o código comum para os dois casos, e é claro com o suporte da ferramenta. As modificações realizadas paralelamente, caso também sejam necessárias no ramo principal, (por exemplo, na correção de um erro que se apresentaria em todas as liberações) podem ser facilmente incorporadas ao resto do desenvolvimento em qualquer momento futuro, através de uma operação de *merge*.

Problemas residuais

Através da adoção de uma ferramenta de gerência de configuração com todo o seu suporte tecnológico para o controle das revisões de arquivos, com a possibilidade de haver trabalho com ramos (*branch*) para solucionar o problema de várias equipes de desenvolvimento, trabalhando em paralelo e em diversas liberações do produto. Sendo ele para clientes diferentes ou não, ainda existe um problema que tem que ser atacado, o do controle das ferramentas usadas no dia-a-dia para realizar o projeto, tais como compiladores, bibliotecas, editores, etc, etc. Este problema será visto no próximo caso.

3.11.4 4º Caso - Projeto com controle de ferramentas

Sem um controle das ferramentas utilizadas, ainda existem problemas. Todos os sistemas envolvidos no desenvolvimento de um programa também estão em constante evolução para acompanhar a evolução tecnológica, seja ela de *hardware* ou de *software*. É sabido que sempre que o *hardware* torna-se mais poderoso, os desenvolvedores de *software* resolvem colocar mais uma ou outra característica no seu produto, fazendo com que este volte a rodar na mesma velocidade antes da evolução do *hardware*.

Isto garante uma evolução contínua dos equipamentos e sempre traz novas facilidades para o usuário, que muitas vezes não consegue mais viver sem ela. Normalmente, estas facilidades trazem consigo um aumento da produtividade.

É neste ponto que muitos administradores de sistemas cometem um grande erro. Para o usuário comum, é claro que certas evoluções de *software* trazem um grande benefício, mas para os desenvolvedores, isso é uma grande "dor de cabeça", muitas vezes essas características de *software*, mudam o ambiente, onde o desenvolvedor está trabalhando, mudando uma configuração ou uma biblioteca. Acrescentando ou retirando uma destas bibliotecas, um grande atraso pode ser motivado na solução de um problema, porque algo que o programador estaria pensando ser causado pelo seu pro-

grama, na realidade é originado pela troca da determinada biblioteca, gerando uma incompatibilidade momentânea, mas que desvia a atenção do programador que estava empenhado em solucionar o seu problema.

É obvio que este novo problema deve ser atacado, mas no escopo correto. Quando se acha que o programa desenvolvido está apto a interagir com os demais programas do sistema operacional em questão, este deve passar por um bateria de testes de compatibilidade. Neste momento, os desenvolvedores estarão preocupados com a compatibilidade do produto, e não em solucionar o problema da sua aplicação. Assim, a necessidade de um controle nas ferramentas usadas no desenvolvimento de uma aplicação é essencial e necessária para a obtenção de uma boa qualidade no produto que está sendo gerado.

Reprodução posterior do ambiente

Outra necessidade de um ambiente de desenvolvimento é a reprodutibilidade, ou seja, em qualquer momento futuro, de ter a possibilidade de montar o ambiente de desenvolvimento completo.

Através da utilização diária de uma *FGC*, incluindo a gerência de ferramentas, é possível que isso ocorra. Caso um projeto seja retomado, por algum motivo, depois que este tenha sido finalizado, ou ainda, uma versão mais velha que determinado cliente importante possua e onde não seja possível fazer uma atualização para uma liberação mais nova, existe a total probabilidade de "voltar" no tempo e reproduzir o ambiente, completamente, e executar as modificações e correções necessárias ao programa.

3.11.5 5º Caso - Projeto com controle de configuração

Com todas estas características citadas anteriormente, postas em prática de maneira simultânea, é possível se ter um ambiente com total controle da configuração dos projetos, onde podem ser destacados os seguintes pontos:

- possibilidade dos ambientes a serem reproduzidos;

  Com essa característica, o ambiente pode ser reproduzido, posteriormente, se for necessário retomar o projeto, como também remontar o ambiente em outro lugar, se houver um aumento na equipe de desenvolvimento e esta não esteja no mesmo lugar da equipe inicial. Isto pode ser uma necessidade do projeto, caso este seja um projeto de desenvolvimento colaborativo, ou onde tenha que ser entregue mais rapidamente. Então, podem ser alocado mais recursos para o mesmo.

- ferramentas e programas auxiliares sob controle;

Isto é de extrema valia, pois em muitos casos o programador já está extremamente envolvido com a solução dos seus problemas, que a inserção de condições externas não controladas podem gerar um atraso enorme no desenvolvimento, o que normalmente não está contemplado no cronograma inicial do projeto. Neste caso, pode ser garantido que os programadores estejam resolvendo os problemas relativos ao desenvolvimento do produto, deixando os problemas de compatibilidade para um teste posterior, muitas vezes até mesmo realizado por uma equipe que faz parte do teste sistêmico do produto.

- diferentes liberações simultâneas;

  Este item nos traz várias vantagens, porque pode atender diversos clientes simultaneamente, colocar o produto sob teste o mais rápido possível e, ainda em diferentes ambientes que seriam muito difíceis de serem reproduzidos, permitindo que um alto grau de compatibilidade seja atingida e que seja possível realizar o desenvolvimento de um determinado produto de forma escalonada, onde é liberado um produto inicial com as características básicas e que é acrescentado as demais funcionalidades, assim que o projeto vai ganhando corpo e estabilidade.

- possibilidade de equipes trabalhando em paralelo;

  A possibilidade das equipes trabalhando em paralelo garante aos gerentes que a ocupação dos diversos recursos, sejam materiais ou humanos, e que seja a melhor possível, diminuindo assim os custos e gerando um reaproveitamento de materiais, de código e de conhecimento mais otimizado. Isto também, nos permite que a alocação dos recursos e seus respectivos planejamentos sejam melhores e mais próximos da realidade.

- controle das modificações realizadas nos arquivos;

  Este é um dos principais pontos a serem levados em consideração num desenvolvimento. Com esse controle é possível, saber quando, o quê e quem realizou determinada modificação, além de permitir que sejam feitos experimentos, sem que se atrapalhe os demais membros da equipe de desenvolvimento. Se existe a possibilidade de fazer experimentos, também é possível voltar atrás, a qualquer instante. Caso não se chegue ao resultado desejado, descartando todas as modificações realizadas de maneira muito rápida e com garantia que tudo voltou ao ponto inicial.

- minimização da ocupação de espaço no local de armazenamento;

Com a adoção de uma *FGC* acontece a otimização do espaço de armazenamento, uma vez que não será mais necessário realizar cópias de estrutura de diretórios inteiras para as diversas equipes envolvidas no desenvolvimento, e nem será mais necessário esta mesma cópia para as diversas liberações do produto. Além, de evitar a duplicidade de informação que pode gerar inconsistências no produto final, a ferramenta usa a tecnologia de armazenamento em deltas, o que garante uma ótima ocupação de espaço em disco, quando comparada com qualquer outro método.

- possibilidade de coleta de métricas;

  Após a adoção de todos esses procedimentos, se cria um ambiente homogêneo, com facilidade para implantar métodos automáticos de verificação da adoção dos procedimentos, e automatização da coleta de informações para gerar métricas para os mais diversos pontos de observação que serão necessários, durante o desenrolar do projeto. Tudo fica muito facilitado, manejável, coerente e consistente.

3.12 Conclusão

Após a adoção de uma ferramenta de gerência de configuração em todos os projetos, não se pode parar de procurar novas soluções, pois quando o trabalho é feito no caos total e a ordem vai sendo trazida, certos problemas foram desaparecendo, e outros que estavam obscurecidos, começaram a vir à tona.

É nesse ponto que muito ainda, deve ser feito para melhorar mais e mais um ambiente de desenvolvimento com gerência de configuração, num ambiente fácil de se dar manutenção (satisfazendo os administradores de rede, os gerentes de configuração, os administradores de banco de dados), altamente configurável (satisfazendo os desenvolvedores, programadores e usuários em geral), facilmente reprodutível e confiável (agradando os líderes de projetos e supervisores) e muito mais maleável e previsível (agradando os gerentes e diretores, que têm uma melhor percepção do que cada um está fazendo e quanto está custando).

Após a descrição da gerência de configuração, das técnicas e ferramentas necessárias para entender e utilizar um ambiente assim constituído será visto a partir de agora, como implantar um ambiente de desenvolvimento com a gerência de configuração aplicada.

Capítulo 4

Configuração do ambiente de trabalho

Mesmo com o conceito de gerência de configuração definido e conhecido, ainda há problemas para resolver num ambiente de desenvolvimento, mesmo após a adoção de uma *FGC*.

Já foram solucionados vários problemas, que existiam quando não se era usada uma ferramenta de controle. Porém, estes controles foram colocados e os problemas sendo resolvidos. Então, os tipos de problemas que estavam eclipsados por dificuldades maiores começam a ser vistos e a partir de agora, um dos principais problemas que estava obscurecido, começa a ser atacado: o da instalação e replicação dos postos de trabalho.

Estes postos de trabalho, normalmente, são muito dependentes das características dos projetos, mas num mesmo projeto, freqüentemente os postos de trabalho são parecidos, quando não são iguais. Quando existe uma diferença, esta está nas configurações de um usuário e não na configuração da máquina.

Diversas soluções para o problema devem existir, e é isto que será discutido e mostrado a partir deste momento.

4.1 Instalação controlada

A instalação controlada vem como primeira opção para solucionar o problema da replicação de postos de trabalho. Ela pode ser vista como uma solução muito boa, e realizada de maneira muito simples, às vezes até com a introdução de uma espécie de *check list*, onde estão descritos todos os programas que devem estar instalados para que um desenvolvedor tenha o seu ambiente completamente configurado e pronto para começar a trabalhar.

Esta solução é bastante boa, para pequenos grupos com poucas configurações para

serem mantidas, e através de um documento de gerência de configuração bem escrito, uma pessoa pode manter as máquinas dos programadores corretamente, configuradas e ser capaz de reproduzir este ambiente de desenvolvimento em qualquer lugar a qualquer instante, desde que este tenha disponível todas as ferramentas necessárias para realizar a instalação. Entretanto, existem alguns problemas, os quais serão descritos a seguir:

- Documento de gerência de configuração não atualizado ou mal escrito;
- Tempo para a disponibilização do posto de trabalho;
- Dificuldades de se realizar atualizações;
- Número de pessoas envolvidas no processo de desenvolvimento.

4.1.1 Documento de gerência de configuração não atualizado

Este tipo de controle de instalação deve ter um documento de gerência de configuração muito bem escrito e controlado constantemente, para não aparecer nenhuma surpresa. Muitas vezes, numa instalação controlada, onde estão envolvidas muitas ferramentas que devem ser instaladas e configuradas, pode acontecer incompatibilidades entre elas, se forem instaladas em uma determinada ordem, enquanto que em ordem inversa, não surgiriam problemas.

Deve ser levado em consideração que este documento pode descrever além da ordem de instalação, as versões correspondentes dos programas para serem instalados. Para haver uma correta gerência de configuração, estes programas e suas respectivas versões devem ser controlados.

4.1.2 Tempo para a disponibilização do posto de trabalho

Uma das grandes desvantagens deste método é o tempo necessário para que um posto de trabalho fique pronto. Muitas vezes, é necessário instalar o sistema operacional na máquina deste o início, para obtenção de um total controle sobre a instalação e aplicar os diversos corretivos (*patch*) sobre o sistema antes de começar a realização das instalações dos aplicativos necessários para o desenvolvimento, teste ou suporte. Neste caso, devem ser verificadas as versões das ferramentas ou bibliotecas e seus respectivos corretivos.

Quase sempre todo este trabalho para uma instalação padrão de um posto de trabalho pode levar de dois a três dias, o que para uma equipe pequena de quatro a dez pessoas é ainda viável. Acima disso, pode começar a se tornar um problema para a equipe responsável por gerência de configuração.

4.1.3 Número de desenvolvedores

O número de postos de trabalho necessários e indispensáveis para a realização do projeto é um ponto bastante crítico neste tipo de solução. Conforme há necessidade de novos postos, o tempo mínimo indispensável para liberar o posto de trabalho para o programador cresce. Por conseqüência, o tempo que a equipe de gerência de configuração tem para manter o ambiente diminui, consideravelmente.

Isto pode acabar gerando um gargalo e uma sobrecarga de trabalho para os responsáveis por esta parte, dentro de um projeto, o que pode inviabilizar datas importantes dentro do cronograma de trabalho de uma equipe ou ainda gerar grande insatisfação dentro do grupo, por ficar sempre à espera do equipamento a ser usado.

4.1.4 Dificuldades de se realizar atualizações

Outro ponto que deve ser levado em consideração é que um projeto, normalmente, não leva um ou dois dias, leva meses, às vezes, não muito raramente, até anos. Isto significa que se deve manter os equipamentos dos programadores por um bom período de tempo, nos levando a um problema muito sério para a gerência de configuração: as atualizações de *software*.

O mundo dos programas de computador está sempre em constante evolução, por motivos diversos, tais como:

- novas necessidades;
- constantes evoluções dos equipamentos;
- correções de falhas e erros;
- compatibilidade entre os diversos aplicativos e equipamentos;
- integração entre sistemas;
- novas facilidades para os usuários.

por todos estes motivos, a necessidade de serem realizadas constantes atualizações nas instalações dos postos de trabalho é grande, e assim :

- como garantir que todos estão usando as versões corretas das ferramentas e bibliotecas?
- como distribuir estas informações de forma rápida e coerente?

- como replicar novas instalações com estas correções já aplicadas?

Ainda, são muitos os problemas a serem resolvidos para que a solução através de uma instalação padrão seja definitiva. A busca de uma solução mais ampla tem que ser contínua.

4.2 Clone de máquinas

Dando continuidade à procura por uma melhor solução para todos os problemas que uma equipe de gerência de configuração enfrenta, durante a evolução da implementação de uma gerência cada vez mais eficiente, é encontrada a possibilidade de realizar *clone de máquinas*, utilizando-se da tecnologia de imagem de discos, que será apresentada posteriormente.

O termo clonagem na área de informática, na realidade é apenas uma maneira de se designar o processo de realizar uma cópia idêntica do seu disco rígido em um determinado instante. Este pode ser usado tanto para se replicar uma máquina ou mesmo para fazer uma cópia *backup* dos seus dados, necessária quando há perda de acesso às informações armazenadas no disco rígido.

4.2.1 Imagem de discos

A tecnologia de clonagem baseia-se na criação de uma cópia do seu disco rígido, que é armazenada em um arquivo chamado de arquivo-imagem, ou mais comumente, de imagem de disco, que pode ser guardada em um meio de armazenamento removível, ou em outro disco, ou até em uma outra partição, dependendo da utilização que será feita deste arquivo-imagem.

De posse deste arquivo-imagem, pode ser restaurado com muita facilidade um sistema danificado ou passar uma cópia do sistema, atualmente instalado numa máquina para outra, economizando um bom tempo nas instalações padronizadas, que devem ser reproduzidas em diversos computadores.

Muitas vezes, alguém pode pensar que o *software* responsável por fazer a clonagem de disco pode ser comparado a um sistema ou programa de *backup*, mas existe uma enorme diferença entre estes dois tipos de programa. O programa de clonagem opera na camada de disco ou de uma partição, enquanto que o programa de *backup* opera no nível de arquivos. O programa de clonagem está projetado para a criação de imagens estáticas dos discos rígidos, o que permite aos gerentes de configuração reproduzirem máquinas idênticas em pouco tempo, ou restaurar o estado de determinada máquina em um ponto conhecido e confiável.

O arquivo-imagem gerado pelo programa de clonagem captura e armazena todos os detalhes do disco rígido original ou da partição que se está clonando, incluindo todos os arquivos que estiverem no disco naquele instante, sem se importar com os atributos destes arquivos, tais como *read-only* ou *hidden*. Só para exemplificar, vamos ver os arquivos `IO.SYS` e `MSDOS.SYS` que são arquivos ocultos, mas são essenciais para a partida de qualquer sistema *Windows 9X*: estes arquivos serão copiados para o arquivo-imagem sem nenhum problema.

Como a um nível mais baixo tudo pode ser tratado como *bytes*, tudo que é relacionado com os dados de partições, com o sistema operacional, aplicações instaladas, utilitários, valores armazenados e arquivos de configuração são copiados para o arquivo-imagem, pois o programa de clonagem realiza uma cópia *byte* a *byte*, assim dentro deste arquivo-imagem, é encontrado tudo para reconstruir o sistema seja onde for.

Com esta tecnologia de cópia *byte* a *byte* existem um duplicação de dados. Os problemas, sempre existem, mas para este caso há uma boa solução, que é a compressão dos dados. Através da compressão pode-se reduzir o tamanho do arquivo em até 50% e caso seja necessário aumentar a velocidade da geração do arquivo-imagem, a estrutura de diretórios e arquivos é passível de ser interpretada e a cópia é realizada, dos setores que efetivamente, contenham dados e assim o arquivo-imagem pode tornar-se ainda menor.

Apesar de ser muito útil, se for usada a técnica de armazenar somente os setores que contenham efetivamente dados, tanto na questão velocidade, quanto na questão espaço de armazenamento, existe uma grande desvantagem. Os programas que forem realizar esta cópia deverão ser escritos para entenderem a estrutura do sistema de arquivos, o que torna o programa pouco genérico e dependente das versões de sistemas operacionais que se está rodando.

Retorno das informações armazenadas

Até agora, foi discutido e apresentado como é feito para gerar um arquivo com as informações indispensáveis, num formato conhecido, de fácil manuseio, com o menor tamanho possível, o que resulta no menor tempo necessário para ser obtido o arquivo imagem. Desde o início, foi focada a solução do problema na variável que estava tentando minimizar, ou seja, o tempo que é preciso para se apresentarem novos postos de trabalho, prontos para que os desenvolvedores possam realizar a sua tarefa.

De posse destas informações, o que pode ser feito? Basicamente, há quatro possibilidades:

- Restauração do disco rígido em caso de falha;

Normalmente, o programa de clonagem ou criação de imagem de disco é um dos meios mais úteis para ser recuperado uma falha do disco rígido. Na realidade, o processo é o mesmo que é feito para replicar a máquina, somente que o disco será usado no computador origem.

É muito útil ser mantida uma cópia do disco de inicialização do sistema com os aplicativos necessários, mais recente e atualizada possível para que possa ser restaurado em um novo dispositivo caso um desastre maior venha a ocorrer.

- Regressão do sistema a um ponto anterior conhecido;

  Novamente, é vista a aplicação e evolução das tecnologias e técnicas conhecidas, que geram efeitos colaterais, acabam gerando novas necessidades e possibilidades de uso, como é o caso da regressão.

  Ao criar e guardar os arquivos-imagem em momentos específicos, surge a possibilidade de regressar o sistema nos pontos específicos da criação das imagens. Isto é muito útil nos momentos em que são feitas instalações de novos programas ou novos equipamentos, resultando por ter algum tipo de incompatibilidade ou redução drástica de desempenho. Através do arquivo-imagem, há um retorno rápido ao ponto em que o sistema estava funcionando.

  Deve ser lembrado que como o sistema de geração de imagens não é um sistema aditivo, ou seja, sempre é feita uma cópia completa e exata do sistema. Qualquer outro arquivo ou modificação que tenham sido agregados ao sub-sistema de discos é perdido, no momento da regressão. Porém este não é um sistema de *backup* incremental, e caso exista algum arquivo que deva ser mantido no sistema após a regressão, este deve ser copiado e re-introduzido, manualmente, a partir de outra mídia, se ainda for possível o acesso ao sistema computacional.

- Atualização do sub-sistema de discos;

  Nos últimos anos, cada vez mais, o custo por megabyte de armazenamento em disco rígido vem diminuindo e os sistemas operacionais e aplicações agregadas a eles vem crescendo, surgindo uma nova necessidade para os usuários de computadores, a troca da unidade de disco rígido. Nesta troca pode-se pensar em instalar todo o sistema operacional novamente. Sim é uma solução, mas é a mais demorada e a que demanda o maior esforço. Não é, somente, a questão de instalar tudo novamente, e sim, a questão de reconfigurar todos os programas e dispositivos que são anexados ao computador.

Pode ser usado um sistema de *backup*, porém neste caso, seria necessário reinstalar no mínimo o sistema operacional, para ter acesso à ferramenta de *backup*. O problema é que este processo pode trazer transtornos, como aplicativos que não funcionam por falta de uma ou outra parte, ou uma ou outra configuração. Através do processo de criação de arquivos-imagem, este processo é muito mais simples, e o resultado é muito melhor e mais rápido. Para realizar esta atualização, basta criar a imagem do disco de inicialização em um meio removível ou em outro disco. Um novo disco é instalado e a imagem é trazida para este novo disco. No momento em que o processo finalizar, se dá inicio ao novo disco, o "velho sistema" completo estará de volta e com as mesmas características, exceto pela capacidade livre extra para novas necessidades.

- Replicação de máquinas através da clonagem.

  A clonagem, seria a utilização mais nobre para o processo de arquivos-imagem, e na prática, resume todos os casos anteriores. O processo de gerar o arquivo-imagem e baixá-lo em um momento futuro seja na máquina original, com o disco original, ou numa outra máquina ou em um outro disco, as informações estarão sendo replicadas de qualquer um dos modos.

  Para os gerentes de configuração ou para os gerentes de TI, o processo de replicação garante que todos os indivíduos receberam computadores padronizados, com instalações controladas e da maneira mais veloz.

4.2.2 Problemas residuais

Novamente, esta solução traz certas vantagens, agilizando o processo de disponibilização de novas máquinas para os diversos grupos, mas existem o problema das atualizações posteriores, instalações de novas ferramentas, sub-utilização dos equipamentos, entre outros. Isto significa que ainda há possibilidade de identificar pontos de melhorias e buscar outras soluções mais vantajosas.

4.3 Terminais

Existe uma tecnologia que já foi muito usada, a tecnologia de terminais. Ela vai ser aplicada num conceito prático voltado para a gerência de configuração, no quesito replicação de postos de trabalho. A técnica de terminais é quando há um computador central onde todo o processamento é efetuado, e todas as operações de interface com o usuário são executadas em terminais remotos. Assim o usuário é capaz de realizar a

entrada de dados e após o processamento, feito totalmente, num processador central, as respostas são enviadas de volta para o terminal do usuário.

Este conceito se tornou anacrônico com o advento do *downsizing* e com a utilização de ambientes gráficos (GUI), que exigiam uma banda passante enorme para o tráfego de informações. Isso trouxe vários problemas para a administração do ambiente, bem como para a gerência de configuração.

Com o desenvolvimento da tecnologia de redes, que trouxe mais e mais velocidade na transmissão das informações entre os diversos equipamentos, que podem compor uma rede, o conceito de terminais, agora gráficos, voltou a ser uma solução atrativa para o desenvolvimento e conseqüentemente, para a gerência de configuração, onde há a solução para muitos problemas que se encontram e que ainda existem, são eles:

- **Instalações padronizadas**

  Como toda a instalação do sistema está centralizada numa única máquina, todos os usuários desta máquina terão sempre o mesmo tipo de instalação, com as mesmas características necessárias para o desenvolvimento do produto ou projeto, enquanto as suas configurações pessoais podem ser preservadas, como será mostrado a seguir.

- **Velocidade nas instalações novas**

  Novamente, quando é finalizada a instalação do servidor central, todos os usuários já têm o ambiente disponível para começar o trabalho, sendo desnecessária a espera para se realizar replicação ou clonagem de máquinas.

- **Atualizações**

  Este é um dos itens que mais têm ganhos, a partir de um ponto do desenvolvimento, onde precisa ser realizado um tipo de atualização em algum produto que compõe o ambiente de desenvolvimento. Basta realizá-lo uma única vez e todos os programadores receberão a atualização, e ao mesmo tempo, a garantia de que não haverá no mesmo ambiente de desenvolvimento, dados de configuração ou ferramentas em diferentes versões.

- **Individualidade dos usuários**

  Apesar de haver uma instalação centralizada e padronizada, onde todos os programadores recebem um ambiente de trabalho idêntico, cada um, sempre, tem certos hábitos, que devem ser preservados, o quê influencia muito na produtividade de cada um. Este tipo de configuração permite uma solução, pois todas as particularidades de um usuário são guardadas nos seus diretórios individuais,

chamado de diretório */home*, enquanto que as demais configurações, pertinentes a um projeto ou ferramenta, ficam centralizadas no ambiente controlado pela gerência de configuração.

Usando este sistema de terminais, existe uma área para otimizar todo o processo, a utilização dos terminais dos usuários de forma dinâmica. Uma vez que a maioria dos equipamentos terminais, quando estão trabalhando na forma de processamento centralizado, ficam ociosos por um bom tempo. Essa idéia será melhor trabalhada mais adiante neste trabalho.

4.3.1 Terminais magros e consoles remotos

Este tipo de tecnologia permite aos usuários que utilizem computadores de pequena capacidade para as atuais aplicações, uma vez que a cada ano, sempre, é necessário mais e mais poder de processamento. Através da utilização de operações remotas que são executadas num servidor e a saída e entrada de dados feitas em outro computador, o terminal magro, o usuário sempre tem disponível o poder computacional do servidor, facilitando assim, a administração do sistema, bem como as atualizações do ambiente computacional como um todo.

Na parte de *software*, este permite que a manutenção dos aplicativos seja feita apenas em uma máquina: a servidora, enquanto os demais, as clientes, recebam automaticamente essas atualizações nos próximos *reboots* ou *log-ins*. Na parte de *hardware*, acontece a mesma coisa, a preocupação maior deve ser sempre com a máquina servidora, em fazer as atualizações pertinentes para um maior poder de processamento, aumento de memória ou novos periféricos. Nesta máquina servidora, os clientes automaticamente, terão estes serviços disponíveis. No caso dos clientes, caso aconteça algo com o "seu" terminal, basta dar outro terminal para o usuário, que este terá de volta o seu ambiente de trabalho.

Outra grande vantagem deste tipo de tecnologia, é que o usuário (cliente) não precisa se preocupar em levar o "seu" computador para qualquer lugar que ele precise ir, pois basta que tenha acesso a um terminal e pronto, ele terá o seu ambiente, completamente, configurado com todos os seus programas disponíveis num tempo muito curto. Além disso, dependendo do tipo de tecnologia utilizada, um usuário pode manter a sua sessão, com todos os aplicativos abertos e nas posições desejadas.

4.3.2 X Terminais

O sistema gráfico dos ambientes Unix, mas conhecidos como X11 ou *X Window System* [XFREE86, 2002], foi concebido e criado desde o início para ser "transparente" para a rede local, permitindo que ele seja portável para diferentes plataformas e que este seja executado em diferentes arquiteturas.

O *X Window System* é baseado numa estrutura cliente/servidor, onde o programa servidor executa na máquina que tem o `display` gráfico, recebendo as entradas do usuário, processando e enviando as respostas para os clientes. Na situação mais comum, o servidor e o cliente executam na mesma máquina, mas pela estrutura idealizada. Estes mesmos clientes podem executar em outras máquinas de modo transparente.

Esta característica faz com que um sistema X11 possa executar, remotamente, e exportar o `display` para um console gráfico remoto, permitindo assim que os X Terminais sejam utilizados como consoles remotos, disponibilizando uma área de trabalho para um outro computador. Entretanto esta configuração tem um problema: como usa-se a estrutura cliente/servidor, o acoplamento entre os dois sub-sistemas é muito grande, e se o servidor perde contato com o cliente, todos os aplicativos que estavam rodando neste cliente morrem, e o usuário deve re-inicializar o ambiente gráfico, perdendo as suas informações.

Algumas vantagens tidas com a adoção e utilização deste tipo de solução:

- Custo

  Pode-se reutilizar PCs antigos para montar os X terminais. Conforme a tecnologia vai evoluindo, novas aplicações requerem mais e mais recursos, entretanto o *hardware* do `display` gráfico não evolui tão rapidamente. É provável que várias atualizações no servidor serão necessárias, antes de ser necessário a atualização dos X terminais.

- Manutenção

  Administrar um PC, uma estação de trabalho ou servidor, é muito mais simples do que administrar vários PCs. A necessidade de configuração de um X terminal é apenas do dispositivo gráfico, do teclado e do mouse, o que também é mais simples.

- Desempenho

  X terminais normalmente, aparecem com uma solução muito boa em termo de custo, a economia aparece, mas no final há apenas um computador para dividir

entre diversos usuários, mesmo que haja um servidor de aplicação multiprocessado, com um desempenho excepcional. Neste caso, é apenas um computador. A solução com terminais é uma questão de compromisso, pois há vantagens e desvantagens em relação a usar um computador em cada mesa.

Mas por que isso acontece? Há várias respostas para isso, tanto no âmbito econômico quanto no âmbito técnico, como será mostrado a seguir:

- *Caching* no servidor

  Se uma aplicação já está sendo usada por outro usuário no momento em que vai ser utilizada, esta já está carregada na memória. Como todos os usuários fazem um compartilhamento das instâncias do programa, o *startup* de uma aplicação é mais veloz, e a solução com X terminais acaba se beneficiando do uso anterior de um dado.

- Não existe gargalo na rede

  Se alguém compartilha algum tipo de recurso no servidor de arquivos, o servidor pode fazer um cache local das informações, porém se vários clientes requisitam o mesmo recurso através da rede, cada um deve transferir este dado e ter uma cópia local dessa informação, causando um enorme tráfego na rede para computadores diferentes, mas com o mesmo conteúdo.

  Entretanto, quando se usa um X terminal, todos os usuários estão acessando o recurso, localmente, na memória do servidor, onde a velocidade de acesso a informação é maior do que em qualquer tipo de rede que possa aparecer.

- Otimização

  Para evitar o congestionamento na rede pode-se duplicar os arquivos em cada posto de trabalho, porém, além deste esforço resultar num trabalho extra, outro problema pode acabar ocorrendo, o problema da acuidade, que é como evitar cópias desatualizadas das informações. Assim, o uso dos X terminais ajuda na otimização dos recursos.

- Postos de trabalho inter-cambiáveis

  Como cada usuário tem o seu próprio diretório particular no servidor, onde ficam as configurações e dados pessoais, tais como:

  - Ícones da área de trabalho;
  - favoritos de um navegador internet;
  - pastas de e-mail.

Isto permite a um usuário, que este tenha mobilidade de usar qualquer X terminal que esteja disponível, sem perda de informação ou produtividade o que é muito importante, como no caso de uma unidade X terminal qualquer que falhe, pode ser re-alocado este usuário para outra unidade, rapidamente.

- Backups

  Como tudo está armazenado no servidor, a estratégia de backup é bastante simplificada, basta salvaguardar os dados do servidor e todos os dados estarão sob proteção.

- Backup de energia através de baterias (UPS)

  Como a única máquina importante é o servidor de aplicação e onde todos os dados estão armazenados e as aplicações estão sendo executadas, colocar um UPS ou um No-break nesta máquina, protege o trabalho de todos os usuários. Se fosse adotada numa solução convencional com um computador por desenvolvedor, deveria existir um UPS por máquina, aumentando assim o custo do ambiente de desenvolvimento em relação ao custo do projeto.

- Flexibilidade

  O conceito chave por trás do uso de X terminais é a flexibilidade no seu uso. Por exemplo, citando algumas características importantes que podem ser tidas, quando se é feito uso de X terminais:

  - Possibilidade de usar os mesmos programas em qualquer lugar
  - Possibilidade de sessões remotas aninhadas

  As soluções com X terminais são muito boas, porém as conclusões não devem ser tiradas precipitadamente, porque esta solução não resolverá todos os problemas. É aconselhado fazer alguns experimentos no ambiente de desenvolvimento em questão, e tirar as conclusões necessárias para verificar quem pode ou quem não usar este tipo de tecnologia. Entretanto, uma avaliação é necessária com relação a gerência de configuração, para ver se não existem ganhos maiores em detrimento de uma pequena perda de desempenho para um ou outro usuário.

4.3.3 *VNC* (*Virtual Network Computing*)

VNC significa *Virtual Network Computing* como foi citado anteriormente. Esta tecnologia permite, através de um protocolo de apresentação remoto, que se tenha

acesso a uma área de trabalho de um computador a partir de qualquer outra máquina que esteja na rede de dados, usando qualquer tipo de plataforma ou arquitetura que possua o cliente necessário para interpretar este protocolo de apresentação remota [VNC, 2002].

O *VNC* difere dos demais métodos de console remoto que existem, como por exemplo o uso de X-Terminais que já foi comentado anteriormente. Os principais pontos que podem ser ressaltados nesta tecnologia, são:

- Não existe estado armazenado no visualizador

  Isto permite que haja uma troca de um computador para outro sem perder os trabalhos que estavam sendo feitos.

- O visualizador é pequeno e simples

  O programa de visualização é pequeno e não exige instalação, podendo, inclusive, executar a partir de um disquete.

- É independente de plataforma

  O cliente pode rodar em diferentes arquiteturas, inclusive, através de um navegador internet com uma máquina virtual Java.

- Sessões compartilhadas

  As sessões criadas pelo *VNC* podem ser compartilhadas entre diversos clientes. Assim, um ambiente de trabalho pode ser visualizado a partir de vários computadores, permitindo o uso da tecnologia CSCW[1].

4.3.4 Metaframe e RDesktop

Existe outra tecnologia muito usada, que é a utilização de terminais no ambiente *Microsoft Windows*, muitas vezes utilizada para disponibilizar serviços de *Office* para um ambiente de desenvolvimento.

Este serviço está baseado na utilização de um servidor de aplicação e no protocolo ICA[2], que é uma marca registrada da *Citrix* [CITRIX, 2002]. Este é um protocolo de apresentação remota que fornece a devida base para transformar qualquer tipo de dispositivo, num cliente-magro (*thin client*). Esta tecnologia é composta de três partes, um componente de *software* no servidor, o protocolo de rede e um programa cliente.

[1] *Computer Supported Cooperative Work*
[2] *Independent Computing Architecture*

- Componente de *software* no servidor

  A arquitetura ICA, através de um componente de *software* instalado no servidor, permite separar toda a lógica da aplicação da sua interface com o usuário. Todo o processamento é feito no servidor e apenas a interface com o usuário é transmitida pela rede.

- Protocolo de rede

  Nesta arquitetura, um dos grandes responsáveis pelo seu sucesso, é o protocolo de transporte de rede, que transmite apenas a interface com o usuário do servidor para o cliente e as teclas digitadas e os movimentos do *mouse* do cliente para o servidor. Isto garante que apenas uma pequena quantidade de dados sejam movidos através da rede, não ocasionando nenhum tipo de congestionamento na rede.

- Programa cliente

  O programa cliente é um pequeno *software* capaz de interpretar o protocolo ICA, e mostrar a interface do programa que está sendo executado no servidor, na tela do dispositivo cliente. Ao mesmo tempo, este também é capaz de interpretar as teclas digitadas e os movimentos do *mouse*, empacotar estes dados e enviar ao servidor.

Este protocolo foi comprado e licenciado pela Microsoft, que implementou pequenas modificações e colocou estes recursos nos seus sistemas operacionais mais novos, como o Windows 2000 e o Windows XP. Esta nova versão do protocolo ficou conhecida com RDP[3], e para acessar os servidores que utilizam este tipo de protocolo é utilizado um cliente **RDesktop** [RDESKTOP, 2001].

Terminais ICA em UNIX

É sabido que o protocolo gráfico padrão do mundo UNIX é excelente, como foi mostrado anteriormente, mas este possui dois grandes problemas, que, muitas vezes torna a sua utilização inviável. O problema da necessidade de haver uma banda passante bastante larga para suportar o enorme tráfego de dados que é gerado por este protocolo, e a manutenção de estado no programa cliente, o que em alguns momentos não é aceitável ou não permite certos tipos de aplicações.

Assim, foi implementado, com o uso do protocolo ICA [CITRIX, 2002], um servidor e um cliente que rodam em ambiente UNIX e fazem o uso deste tipo de tecnologia para

[3] *Remote Desktop Protocol*

suplantar essas limitações que o protocolo X11 possui, melhorando ainda mais as características dos ambientes gráficos para UNIX, além de disponibilizar esta importante tecnologia para os grupos que utilizam este tipo de plataforma.

4.4 *Clusters*

Um outro conceito citado anteriormente, e o que agora é aplicado na prática, tentando melhorar ainda mais o ambiente de desenvolvimento de *software*, é a técnica de *clustering*. Para que haja um melhor compreensão de todos os conceitos apresentados, certas técnicas devem ser descritas.

4.4.1 *Clusters* do tipo Beowulf

O que é um sistema Beowulf? Este termo Beowulf está sendo aplicado, atualmente, para mostrar uma nova estratégia em computação de alto desempenho, que explora as tecnologias de mercado e não as tecnologias proprietárias de supercomputadores, que têm seus preços determinados pelos seus fabricantes. Esta tecnologia permite aos cientistas, engenheiros e outros mais, que façam a utilização de sistemas de alto desempenho para resolver os seus problemas, abrindo assim novos caminhos para o desenvolvimento da tecnologia e ciência.

A motivação para a utilização de *clusters* do tipo Beowulf, que será a configuração adotada nesta solução, em conjunto com o balanceador dinâmico implantado pelo Mosix, é a facilidade da utilização de processadores que estão no mercado de massa, ou seja, já se tornaram *commodities*. Os equipamentos necessários para a interconexão dos equipamentos em alta velocidade (*FastEthernet, Gigabit,* e outros), também se tornaram *commodities* e aliados a sistemas operacionais abertos confiáveis, capazes de serem configurados para operarem com as configurações de computação paralela. Eles trazem a vantagem de oferecer o suporte à computação paralela ou distribuída por frações do custo das soluções comerciais.

Uma grande vantagem de tudo isso é que todos esses ganhos vão sendo agregados conforme o sistema vai crescendo, ou seja, a escalabilidade do sistema sempre existe, e não existe um grande gasto no momento de se começar a montar o ambiente, uma vez que pode-se iniciar o *cluster* com apenas duas máquinas.

CoW (*Clusters of Workstations*)

Um outro modelo que poderia ser usado seria o CoW, porém um problema sério que existe quando é montado um *cluster* deste tipo é que geralmente, as máquinas

(*workstations*) são de propriedade de alguém, por exemplo: uma pessoa, um grupo, um departamento ou até uma organização. Normalmente, estas são dedicadas exclusivamente, aos seus "donos".

Esse "certificado" de posse muitas vezes traz grandes problemas, quando é criado um CoW, pois podemos identificar três tipos de "donos":

- Aqueles que usam os seus computadores, na maior parte do tempo, para acessar o seu e-mail ou preparar alguns documentos, por exemplo quadro administrativo, secretárias, etc;
- Aqueles que usam os seus computadores para o desenvolvimento de produtos de *software*. Neste caso, o trabalho envolve a edição, compilação, *debugging* e ciclos de teste;
- Aquelas máquinas que estão envolvidas em simulações, que freqüentemente, requerem um enorme poder computacional;

sempre lembrando que sempre se está tratando de postos de trabalho, e não de servidores. O tempo ocioso de CPU de uma máquina poderia ser utilizado por outra pessoa de maneira informal, porém, isto requer negociações delicadas entre pessoas e não traz o benefício desejado, então se deve voltar à idéia de formar o *cluster*, disponibilizando todo esse poder de processamento para as diversas equipes igualmente, sem prejudicar ou favorecer um certo grupo.

Configuração de um *cluster*

Talvez a idéia da utilização de *clusters* num ambiente de desenvolvimento, não seja a melhor que possa existir, mas deve ser analisada para que os prós e contras deste conceito venham a tona. Com a adoção deste tipo de sistema, há a centralização da administração de todo o ambiente de desenvolvimento, a utilização de tempo de CPU ocioso das máquinas clientes, ganhando em muito, poder de processamento; que são sempre variáveis que são visadas para que haja uma maximização. A seguir serão listados alguns problemas que podem aparecer na disponibilização de um *cluster*, podendo ou não atrapalhar na implantação do sistema:

- Dificuldades de instalação

  Cluster é um sistema mais complexo que necessita de muito mais trabalho para ser colocado em funcionamento, principalmente se o administrador resolver não utilizar o método de instalação manual dos clientes, e sim utilizar instalações através da rede.

- Dificuldades de manutenção

  Um *cluster* de médio a grande porte pode gerar um enorme trabalho para o gerente de configuração, quando chega um momento qualquer em que é necessário realizar algum tipo de atualização. Normalmente isto ocorre ao efetuar um trabalho repetitivo em vários nós do sistema, que é um processo que não está livre de erros.

- Dificuldades de adaptação

  Adaptações que sejam necessárias no *cluster*, novamente, podem gerar uma enorme quantidade de trabalho, pois sempre é necessário realizar as alterações em uma grande quantidade de nós do sistema, e este sempre está susceptível a erros.

Muitas vezes, aparecem problemas que este tipo de solução não é capaz de resolver, se adotada de modo individual, ou seja, o problema de manter o ambiente atualizado e conciso com as necessidades da equipe de desenvolvimento e de alocar novos postos de trabalho de maneira rápida e eficiente, proporcionando a todos os colaboradores um ambiente necessário para o desenvolvimento do seu trabalho. As suas individualidades devem ser mantidas, e ao mesmo tempo deve-se fazer com que os usuários não se preocupem onde encontrar certa ferramenta, biblioteca ou determinado arquivo.

A noção de *cluster* por si só, poderia ser considerada um retrocesso, após a apresentação de todos os problemas e possíveis soluções num ambiente de desenvolvimento com gerência de configuração implantada. Este só traria um ganho em termos de desempenho para todos os envolvidos no processo construtivo de um novo produto. Porém, com o pensamento no futuro e apresentando uma solução, onde a união de diversas tecnologias vêm ao encontro dos problemas apresentados até agora e a chegada a uma solução adequada aos problemas atuais com a tecnologia disponível para solucioná-los, conforme será visto a seguir na evolução da nossa proposta.

4.4.2 Mosix

Inicialmente esta extensão ao *kernel* foi desenvolvida para computadores PDP-11 montados em um *cluster* e rodando uma versão de UNIX version 7. Posteriormente, novas versões foram sendo desenvolvidas, sempre baseadas nas últimas tecnologias existentes, até a versão que será usada neste trabalho de dissertação, para analisar a utilização deste tipo de suporte distribuído em ambiente de desenvolvimento.

O Mosix é um sistema operacional distribuído [BARAK, 1993], e é uma sigla que tem como significado *Multicomputer Operating System for UNIX*. Ele é uma extensão

ao *kernel* dos sistemas operacionais baseados em Unix, tal qual o Linux. Usando esta característica, se pode formar um *cluster* de computadores interconectados com baixo acoplamento, numa máquina virtual UNIX única.

Essa integração de diversos tipos de computadores inclui a capacidade de ser independente e transparente à rede. Assim, essa cooperação entre os diversos computadores formadores desta máquina-virtual provê serviços que ultrapassam as fronteiras de uma máquina, suportando configurações dinâmicas e um processo de migração automática que provê o sistema de *load balance*.

Como o Mosix implementa este esquema de migração automática, a máquina virtual (*cluster*) tem a capacidade de ser altamente escalonável e maleável, onde se pode agregar ou retirar máquinas-componentes sempre que for necessário menos ou mais poder de processamento.

As principais características do Mosix são:

- Transparência à rede

  A infraestrutura de rede é completamente transparente para o usuário do *cluster*, o usuário final, que vê somente uma única máquina-virtual.

- Autonomia

  Cada nó que compõe o *cluster* é completamente independente, e pode rodar o seu sistema operacional nativo sem qualquer tipo de interconexão. Assim um nó da rede, que está sempre conectado, pode ou não fazer parte do *cluster* ou ainda atuar sem qualquer tipo de conexão, mantendo suas características originais.

- Cooperação

  Os diversos nós podem trabalhar em conjunto para prover serviços através da rede.

- Controle descentralizado

  Todos os nós do sistema são capazes de tomar as suas próprias decisões, independente de qualquer outro nó.

- Migração dinâmica de processos

  Os processos podem ser transferidos (migrados) entre os diversos nós do *cluster* sem qualquer restrição.

- Balanceamento de carga

  O processo de migração permite que se aloque os processos aos nós processadores de maneira quase-ótima, observando as seguintes regras:

  - Nunca direcionar um usuário para um nó não funcional;
  - Direcionar os novos usuários para os nós que estão menos carregados e são capazes de dar o melhor desempenho;
  - Suportar de forma transparente e pró-ativamente a migração dos processos em momentos críticos, tais como excesso de carga ou *shutdowns*;
  - Balanceamento automático das cargas, conforme a capacidade de cada nó e conforme a carga atual de cada um, no momento inicial da carga de um programa;
  - Rebalanceamento das cargas nos nós, conforme, o andamento do processo;
  - Rebalanceamento equilibrado entre os nós restantes no momento de alguma falha;
  - Reutilização automática do nó que foi reparado, após um defeito ou utilização automática de um novo nó que for inserido no *cluster*;

- Configuração dinâmica

  Os nós podem ser agregados ou retirados do sistema de maneira quase transparente, sem quase nenhum efeito colateral.

- Aumento da disponibilidade

  Como os processos podem ser distribuídos em diferentes nós, temos o aumento de disponibilidade automaticamente, implantado pelo sistema.

- Desempenho

  Operações remotas são quase tão eficientes quanto as operações locais, excluindo as operações de I/O.

- Confiabilidade

  Um grande grau de confiança é apresentado pelo sistema, uma vez que podemos isolar as diversas falhas que possam ocorrer nos nós.

- Núcleo e Recursos replicados

  O mesmo sistema é replicado nos diversos nós de maneira transparente, o que facilita muito a instalação e montagem do *cluster*.

- Escalabilidade

  O sistema pode ser montado inicialmente com poucos nós, dois no mínimo, e ir crescendo, conforme a necessidade de processamento exija.

- Compatibilidade

  Como o sistema Mosix é uma extensão ao *kernel* do Linux, este pode rodar em diferentes tipos de máquinas, onde o sistema Linux pode rodar, mantendo a compatibilidade binária.

Entretanto uma das principais vantagens de se usar o sistema Mosix é que as aplicações que rodam sobre o sistema, onde foi aplicado a extensão não precisam ser modificadas para ganhar ou receber as vantagens que o sistema de *cluster* é capaz de prover.

4.5 *Cluster* com utilização de terminais

A idéia da utilização de *clusters* para um ambiente de desenvolvimento com gerência de configuração facilita muito a implantação do sistema, permitindo que os programadores/desenvolvedores comecem os seus trabalhos o mais cedo possível dentro do cronograma estipulado pela equipe de gerência de projetos, bem como ajuda em muito, à vida do administrador/gerente de configuração. A maioria dos problemas que foram descritos até agora são facilmente, resolvidos com a adoção do sistema de *clusters*.

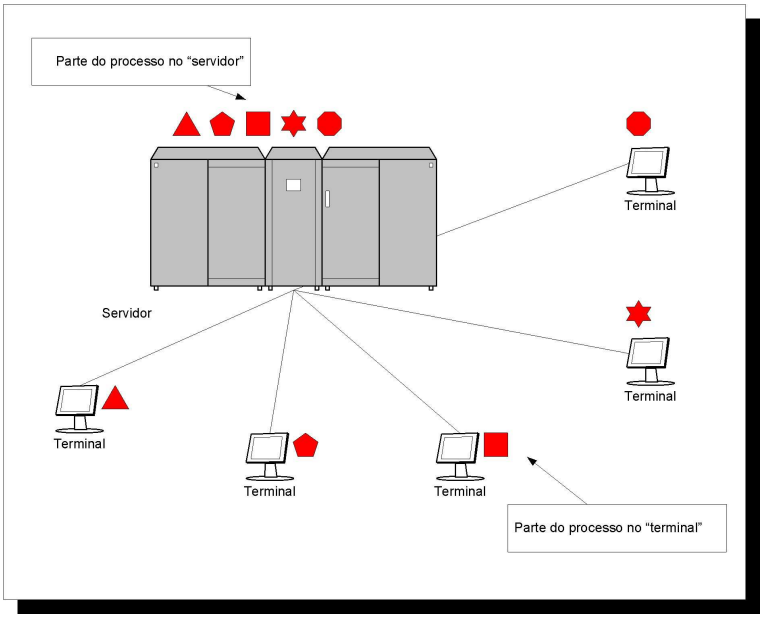

Figura 4.1: *Cluster* com terminais. Duas tecnologias simultaneamente aplicadas

Porém, o problema de manter o ambiente de trabalho sempre atualizado e conciso,

só será conseguido com a adoção de um sistema de *cluster*, com a utilização de terminais, ou seja a união destas duas tecnologias. Conforme foi mostrado na figura 4.1, o *cluster* permite o máximo de ganho em desempenho, porque será usado todo o tempo de CPU disponível para a realização de algum tipo de trabalho.

O trabalho realizado pelo *cluster* será de grande valia para o conjunto dos processos realizados durante o desenvolvimento de um produto. Também será possível fornecer novos postos de trabalho, de maneira quase que instantânea para novos colegas que se agreguem à equipe de trabalho, ou ainda disponibilizar as últimas atualizações para todos os usuários, de modo quase que imediato. Uma vez, atualizada a máquina central do *cluster* todos os envolvidos estarão com as novidades em mãos.

Com este sistema implantado, quando um novo programador for introduzido no ambiente de desenvolvimento, um novo terminal também será introduzido neste ambiente, para que este novo desenvolvedor possa realizar o seu trabalho, e por conseqüência, mais poder de processamento será agregado ao grupo de máquinas. Isto fará com que mesmo com a entrada de um novo membro na equipe, o desempenho de todo o ambiente não degrade de maneira significativa.

Com as máquinas disponíveis, atualmente, a probabilidade de se ganhar em poder de processamento será maior do que a idéia que as pessoas têm, que é mais uma pessoa sobrecarregando o servidor de aplicação. Hoje, mesmo com as máquinas de menor porte, conhecidas no jargão da informática como máquinas *entry-level*, são máquinas poderosas para o tipo de utilização que um ambiente de desenvolvimento requer.

4.5.1 Como implementar este conceito

Acabou de ser verificado que uma das melhores soluções para os problemas da gerência de configuração aplicada, é a adoção de um sistema de *cluster* com terminais. Para colocar esta idéia em prática, são adotadas algumas ferramentas conhecidas no mercado e citadas anteriormente, nas tecnologias conhecidas, como X terminais e um sistema Mosix. Integrando estas duas tecnologias será possível, disponibilizar um ambiente de *cluster*, implantado pelo Mosix e os terminais para acesso dos desenvolvedores, implantado pelo conceito de X Terminais do Unix.

Uma das grandes vantagens dessa idéia é a utilização da técnica de terminais ou servidor de terminais, o que facilita a criação e manutenção do *cluster*. Se fosse feita a montagem de um *cluster* de modo padrão, este poderia tornar-se caro demais e acabar inviabilizando a utilização do mesmo, se junto com a equipe de programadores tivesse um especialista ou uma equipe de administradores de sistema para manter o ambiente de desenvolvimento e não para ajudar a desenvolver o produto.

4.5.2 Considerações na implantação

O maior problema neste tipo de solução, é possivelmente o maior gargalo nesta implementação: é a arquitetura de rede. Para o sistema funcionar de maneira adequada, deve ser muito bem planejado e pensado na arquitetura de rede mais veloz que estiver disponível no instante de instalação do *cluster* para ser usado em um ambiente de desenvolvimento com gerência de configuração.

Como foi citado anteriormente, quanto mais rápido e com menor latência for a infra-estrutura de rede, melhor será o desempenho final de todo o conjunto. Entretanto não é só isso que se deve levar em consideração, pois os ganhos gerais que se têm, principalmente, na condição de manutenção e atualização do ambiente de desenvolvimento, com a utilização desta estrutura de *cluster* com terminais compensam quaisquer pormenores que venham a ser detectados na infra-estrutura de rede.

Porém, este é um ponto a ser observado, sempre que houver reclamações de desempenho, este é o sub-sistema mais exigido de todo o ambiente, e é facilmente sobrecarregado. Quando um novo usuário é agregado ao grupo que está usando este tipo de solução, o gerente de configuração deve ficar atento, pois pode aparecer a necessidade de uma re-adequação da infra-estrutura de rede para manter o desempenho a níveis aceitáveis.

Outro ponto que deve ser levado em consideração, é se houver a necessidade de se usar diferentes sistemas operacionais ou diversas configurações do mesmo sistema operacional, o que pode gerar a necessidade de se montar diversos *clusters* simultaneamente, solução esta que pode satisfazer esta nova necessidade.

4.6 Conclusões

Este capítulo apresentou como deve ser feita a configuração de um ambiente de trabalho para desenvolvimento de *software*, as tecnologias envolvidas neste processo, suas vantagens e os ganhos que se pode alcançar quando se faz uso destas técnicas.

No próximo capítulo, serão abordados e analisados os diversos tipos de projetos de *software*, num processo evolutivo, mostrando os resultados obtidos com a adoção e aperfeiçoamento de um ambiente de desenvolvimento de *software* com gerência de configuração integrados.

Capítulo 5

Resultados obtidos

Neste capítulo, serão apresentados e analisados dados coletados em diversos projetos de *software* que ocorreram em aproximadamente 10 anos de desenvolvimento em um ambiente corporativo. Esses dados são de projetos onde, podem ser identificadas diferentes abordagens para o desenvolvimento de *software*, podendo assim, reconhecer as melhorias alcançadas durante a evolução do processo de adoção, aplicação e aperfeiçoamento da gerência de configuração.

Para evitar problemas e manter a imparcialidade, os projetos serão citados de forma anônima, onde os dados materiais e métricas relativas ao ambiente de trabalho serão analisados, evitando a exposição de qualquer pessoa ou produto utilizado nesta análise.

Todos os dados foram coletados em projetos realizados pelas equipes de desenvolvimento de *software* para equipamentos de telecomunicações, onde normalmente, há alguns pré-requisitos diferentes de quando se desenvolvem programas comerciais. Muitas vezes, num ambiente como este, não existe total liberdade para escolher um certo tipo de tecnologia ou outro, mas sim, padrões especificados pela indústria devem ser seguidos, ou pelos órgãos competentes, tais como o ITU-T[1] ou ETSI[2], para que os programas gerados possam se intercomunicar e, também, poder inspecionar, monitorar ou gerenciar equipamentos de diversos fabricantes, simultaneamente, numa rede de telecomunicações heterogênea.

Também, serão descritos os diversos modos e modelos utilizados nos últimos anos para a resolução dos problemas inerentes ao desenvolvimento de produtos de *software* para esse segmento da indústria, mas isso não quer dizer que as soluções aqui apresentadas não sejam úteis em outras áreas de conhecimento.

[1]International Telecommunication Union - Telecommunication Standardization Sector
[2]European Telecommunications Standards Institute

5.1 Comparação entre as diversas equipes

A seguir serão listados os projetos, verificados os possíveis pontos que podem ser comparados e mostrada a evolução que ocorre quando são aplicadas e evoluídas, essas boas práticas de programação, suporte e acompanhamento de projetos.

5.1.1 (P1) - Projeto sem gerência de configuração

O primeiro projeto que será apresentado, foi um projeto composto por 15 integrantes, desenvolvido em aproximadamente 9 meses, onde não foi utilizado qualquer tipo de gerência de configuração e as sub-equipes trocavam entre si os arquivos necessários para a compilação total do produto.

Dificuldades encontradas

Neste projeto, houve várias dificuldades, apresentadas no dia-a-dia do trabalho das equipes de desenvolvimento:

- Falta de controle nos arquivos

  Neste projeto, uma das grandes dificuldades que foi detectada, exatamente pela falta de uma gerência de configuração, foi saber quais arquivos cada sub-equipe estava usando, normalmente. Para que um produto seja gerado, é necessária a presença de todos os envolvidos, para saber quais arquivos deveriam ser usados a fim de gerar o produto final.

- Falta de controle de erros e requisitos

  Outro ponto de difícil controle foi a gerência das características do produto, pois sempre que os novos requisitos eram implementados, todas as versões dali para a frente tinham essa característica incorporada, fosse ou não requisito de um determinado cliente.

- Falta de gerenciamento de versões

  As correções de erros encontrados no cliente ou no teste sistêmico, eram corrigidos nas versões as quais os desenvolvedores estavam escrevendo, o que tornava impossível que versões anteriores do produto fossem reproduzidas.

- Dificuldades na reprodução do ambiente

  Por causa da falta de gerência de configuração era também uma dificuldade muito grande a reprodução do ambiente em qualquer outro lugar que não fosse

o ambiente original. Não havia um controle das ferramentas necessárias para a criação de um posto de trabalho.

- Alocação de pessoal

  Por falta de controle, freqüentemente, era necessária a presença de quase todos os desenvolvedores, ou no mínimo um de cada sub-equipe para ser realizada a compilação do produto. Deste modo a alocação e re-alocação dos componentes da equipe eram muito difíceis, por parte dos gerentes ou supervisores, o que muitas vezes poderia se tornar frustrante para as pessoas envolvidas.

Informações coletadas

Alguns dados coletados neste projeto, para que seja alcançada uma idéia do tamanho do produto que foi gerado.

Arquivos	Quantidade	Total em bytes
.cpp	146	2.797.356
.hpp	171	652.084
total código	317	3.449.440
documentação	65	2.676.224

Tabela 5.1: Dados colhidos no projeto

LOC[3]	LOC/arq	Pessoas	Produção por pessoa (LOC/p)	Tempo (mês)	Produção por mês (LOC/mês)	Produtividade (LOC/p/mês)
72.542	228	15	4.836	9	8.060	537

Tabela 5.2: Métricas calculadas

Considerações

Este não foi um projeto muito grande, conforme é demonstrado pela quantidade de arquivos que eram necessários para gerar o produto final. Porém, as dificuldades encontradas mostravam que algum tipo de gerência de configuração deveria ser envolvida nos novos projetos.

5.1.2 (P2) - Projeto com gerência de arquivos

O próximo projeto o qual se fará uma análise será: um projeto que foi composto por 10 integrantes, teve a duração de 26 meses e passou a ser utilizado o controle de revisões de arquivos, inicialmente o *SCCS*[4] e depois o *RCS*, de maneira centralizada, sendo assim se apresenta como um protótipo para o início de novos projetos, onde a agregação de gerência de configuração é necessária.

Dificuldades encontradas

As dificuldades que foram encontradas neste projeto, mostram que conforme o "caos" vai se disseminando, onde não existia nenhum tipo de controle, e este controle é colocado sobre o desenvolvimento do produto, outros problemas vão aparecendo.

- Dificuldades no gerenciamento de versões

  A ferramenta adotada para o gerenciamento de revisões ajudava muito no controle de arquivos, mas no momento de "congelamento" de uma versão do produto, não havia suporte da ferramenta para este tipo de operação.

- Tempo de instalação/configuração de um posto de trabalho

  A instalação de um posto de trabalho era um processo demorado, levando até dois dias para preparar uma nova máquina para o trabalho. Além de que a configuração de cada equipamento seguia uma *checklist* para garantir que todas as máquinas estivessem com a mesma configuração no final do processo.

- Controle de erros

  Os erros eram corrigidos através, da utilização de ferramenta de controle de arquivos, porém, como o "congelamento" de versões eram feitos por *scripts* havia um trabalho manual a ser feito para corrigir esses *scripts* a fim de poder gerar corretamente a versão do produto. Como era um processo manual, havia uma grande possibilidade de inserção de falhas.

- Gerenciamento de requisitos

  Não houve um gerenciamento de requisitos, conseqüentemente houve re-trabalho em várias áreas do programa, inclusive pela questão de terem sido propostas novas características. O cliente pode requisitar uma mudança e depois voltar para a idéia original, ocasionando um grande desperdício de tempo, o que muitas vezes acaba gerando atraso no cronograma.

[4] *Source Code Control System*

Informações coletadas

Dados foram coletados para mostrar a quantidade de informação manipulada e ter uma idéia do tamanho do projeto.

Módulos	Linhas de código	Quantidade	Total em bytes
aplicacao	110.413	549	3.221.708
administrador	7.674	42	232.388
arquitetura	20.041	124	604211
gerenciador	4.739	13	124844
hci	76.249	227	2.360.174
iscm	4.166	23	127017
tools	15.237	86	377.063
L2	4.143	27	108.078
liboms	36.416	169	1.392.835
libhci	75.014	384	2.888.605
total	354.092	1.644	11.436.923
documentação		73	5.761.024

Tabela 5.3: Dados colhidos no projeto

LOC	LOC/arq	Pessoas	Produção por pessoa (LOC/p)	Tempo (mês)	Produção por mês (LOC/mês)	Produtividade (LOC/p/mês)
354.092	215	10	35.409	26	13.618	1.362

Tabela 5.4: Métricas calculadas

Considerações

Um projeto, onde se começa a agregar algum tipo de controle, no nosso caso, gerência de arquivos, fica evidente que há ganhos significativos. Como este projeto já envolvia um número maior de desenvolvedores, uma quantidade de arquivos maior, a aplicação inicial de uma gerência de configuração, mesmo que sendo somente a gerência de arquivos, já se fez extremamente útil. Com a adoção desta tecnologia a produtividade dos desenvolvedores foi intensificada, como mostra a tabela 5.1.2.

5.1.3 (P3) - Projeto com gerência de versões

O projeto que será analisado nesta seção foi um dos primeiros projetos, onde foi completamente, adotada a a gerência de versões, tendo como base a utilização do

gerenciamento de arquivos ou versionamento. Assim, existe o necessário controle dos produtos de trabalho, para saber o que compunha cada versão gerada e entregue; seja para o teste sistêmico, ou para o cliente final. O projeto foi desenvolvido por 5 pessoas durante aproximadamente 8 meses e será listado o que foi encontrado de pontos positivos e negativos, na adoção desta metodologia.

Pontos positivos

Os pontos positivos que foram encontrados durante o desenvolvimento deste produto, vêm do controle adicional adotado para o gerenciamento das versões e como poderia ser reproduzido o ambiente de desenvolvimento em qualquer instante que fosse necessário.

- Gerenciamento de arquivos

  O controle trazido pelo gerenciamento de arquivo, permite ao gerente de configuração adotar novos métodos e procedimentos para controlar as liberações intermediárias, necessárias ao teste sistêmico, e liberações finais, que são entregues ao cliente.

- Reprodução do ambiente de trabalho

  A reprodução do ambiente de trabalho, seja ela, para um novo desenvolvedor ou após algum tempo para manutenção, foi extremamente facilitada, por esta adoção de métodos de gerenciamento de versões, basta verificar a versão necessária que se quer reproduzir, e retirar as revisões dos arquivos que formam a versão requisitada.

- Otimização do processo

  A melhoria alcançada durante o próprio desenvolvimento do projeto foi significativa, e vem basicamente, por causa dos desenvolvedores aprenderem melhor a utilizar as ferramentas adotadas para a solução do problema.

Pontos negativos

Entretanto, como citado anteriormente, os problemas ainda existem, e foi possível levantá-los, conforme será visto a seguir:

- Reprodução dos postos de trabalho

  A re-criação dos postos de trabalho, é um processo que deve ser feito de maneira manual e controlada, levando ainda, muito tempo para ser realizado. Como é

um processo manual, pode incorrer em erros na instalação final, inviabilizando a utilização do posto de trabalho, atrasando o processo.

- Alocação de pessoal

  A elaboração do cronograma de trabalho, ainda não tem nenhum respaldo técnico medido e identificado, ainda ocorre o problema do mal dimensionamento do tempo necessário para realizar a implementação das diversas características.

- Otimização do processo

  A otimização do processo pode ser visto tanto como ponto positivo, quanto como ponto negativo. A alteração de um processo já adotado durante o desenrolar de um projeto, pode gerar dificuldade de interpretação por parte das diversas pessoas envolvidas, bem como conflitos não previstos.

Informações coletadas

Alguns dados coletados no ambiente de desenvolvimento deste projeto.

Arquivos	Quantidade	Total em bytes
.cpp	120	1.954.816
.hpp	29	127.522
Arquivos de bibliotecas		
.cpp e .c	264	7.576.346
.hpp e .h	409	2.701.316
total código	822	12.360.000
total documentação	36	11.255.808

Tabela 5.5: Dados colhidos no projeto

LOC	LOC/arq	Pessoas	Produção por pessoa (LOC/p)	Tempo (mês)	Produção por mês (LOC/mês)	Produtividade (LOC/p/mês)
71.388	479	5	14.277	8	8.924	1.785

Tabela 5.6: Métricas calculadas

Considerações

Num projeto como este, onde foi adotado um processo mais controlado, tem-se um ganho significativo, pois foi simples executar os *builds*, verificar quais arquivos faziam

parte de uma determinada liberação e re-montar o ambiente após um certo tempo. Entretanto, para este último ponto o processo ainda era falho e lento, o que mostra em qual direção a equipe de gerenciamento de configuração deveria focar os esforços.

5.1.4 (P4) - Projeto com aplicação de terminais

O projeto listado a seguir, foi o projeto que englobou todas as tecnologias idealizadas pelo grupo de desenvolvimento e de gerência de configuração, trazendo um dos melhores ambientes de desenvolvimento alcançados durante este tempo. O projeto teve uma duração de aproximadamente 30 meses, e puderam ser observados os ganhos obtidos com a adoção completa de um ambiente de desenvolvimento com gerência de configuração adotada e aplicada.

Os 17 desenvolvedores que fizeram parte deste projeto tinham uma flexibilidade de poder trabalhar em qualquer máquina que estivesse disponível. Sempre que um novo pacote de informações estivesse compilado, este pacote estava disponível para os demais integrantes da equipe, *builds* eram realizados de maneira centralizada e otimizada no servidor, etc.

Pontos positivos

Neste projeto, os pontos positivos foram bastante relevantes para que alguns conceitos fossem verificados, onde ainda havia dúvidas e que seriam interessantes para um ambiente de desenvolvimento.

- Gerenciamento de arquivos

  O gerenciamento de arquivos executado neste projeto foi feito através, da ferramenta *ClearCase*, que mantém as revisões dos diversos arquivos que compuseram o produto final.

- Gerenciamento de ferramentas

  Neste projeto, também foi feita a gerência das ferramentas que deveriam ser instaladas. Essa gerência foi feita, sendo consideradas várias variáveis que poderiam influenciar no resultado final, tais como: ordem que deveria ser executada a instalação, quais versões dos aplicativos eram necessárias para determinada liberação do produto, entre outros.

- Utilização dos terminais

  A utilização da emulação de terminais também, foi um ponto crucial neste projeto, pois uma vez que as configurações fossem feitas para uma certa liberação do

produto, todos os desenvolvedores, já recebiam um ambiente completamente, configurado e pronto para começar os trabalhos, agilizando assim, os trabalhos e diminuindo o tempo de *startup*.

- Tempo de compilação

  Outro ponto interessante, "efeito colateral" da utilização de terminais, é que a compilação do produto era mais veloz, por estar tudo centralizado em apenas uma máquina, e o reaproveitamento dos arquivos já compilados entre os diversos usuários era mais intensa e produtiva. É claro que em certos momentos, até os arquivos que estavam em cache de disco, eram re-utilizados pelos compiladores, pois já haviam sido acessados pela compilação de um outro programador.

Pontos negativos

Alguns pontos negativos que ainda foram encontrados, após a utilização de uma ambiente como este, nos fez procurar outras soluções.

- Reprodução dos postos de trabalho

  Mesmo com a utilização de terminais, ainda havia máquinas que deveriam ser instaladas localmente, principalmente para a área de testes, e havendo retrocesso ao tempo de horas para disponibilizar uma máquina pronta para ser utilizada.

- Sub-utilização das máquinas terminais

  Como existiam máquinas muito rápidas nos clientes, quando se estava usando a emulação de terminal, as máquinas-clientes ficavam ociosas, sub-utilizadas, trouxe a idéia de continuar a pesquisa por soluções mais eficientes em termos de custo, desempenho e utilização dos recursos.

Informações coletadas

Abaixo serão apresentados os dados coletados do ambiente de desenvolvimento[5]:

Considerações

Este foi um projeto bastante grande, onde havia servidores centralizados de grande porte, mostrando que era uma solução viável e que facilitava muito a gerência do ambiente de desenvolvimento, que normalmente, não é tão estático quanto se pode

[5]os dados de saída foram coletados num dado instante da compilação, pois é um diretório dinâmico

Quantidade	Tipo de dados
11	bibliotecas básicas
2	ferramentas para instalação
1	dados de saída
4	documentação
16	código fonte dos diversos módulos
34	total

Tabela 5.7: Número de VOBs no projeto

Tipo da VOB	Quantidade de arquivos	Ocupação em kbytes
bibliotecas básicas	15170	1.187.011
ferramentas para instalação	22066	1.128.379
dados de saída	2094	708.261
documentação	11446	456.784
código fonte dos diversos módulos	28636	1.732.587
total	79412	5.213.022

Tabela 5.8: Números gerais encontrados neste projeto

LOC	LOC/arq	Pessoas	Produção por pessoa (LOC/p)	Tempo (mês)	Produção por mês (LOC/mês)	Produtividade (LOC/p/mês)
955.777	33	17	56.222	30	31.859	1.874

Tabela 5.9: Métricas calculadas

pensar. A solução de terminais é muito eficiente, uma vez que fazem apenas uma vez a configuração ou atualização, e todos os programadores estão aptos a continuar o seu desenvolvimento. Além disso, com essa técnica foi muito facilitada a reprodutibilidade do ambiente, e por conseguinte, o desenvolvimento em locais e sedes diferentes.

5.1.5 (P5) - Projeto com a aplicação de *cluster* e terminais

O projeto com a aplicação de *cluster* e terminais, simultaneamente, utiliza como base um sistema Linux rodando a extensão de *kernel* Mosix, para chegar a distribuição de carga entre os computadores que compõem o sistema, e a implementação LTSP (*Linux Terminal Server Project*) para implementar a distribuição de X terminais de modo automático através da utilização de *boot* remoto, aumentando mais a facilidade de implementação e flexibilidade do ambiente de programação.

Pontos positivos

Com a implementação de uma solução deste tipo, grandes resultados são alcançados, e é necessário verificar se todos estes ganhos são realmente bons para uma equipe de desenvolvimento e ver se é possível melhorar ainda mais.

- Posto de trabalho

  Com este tipo de solução adotada, há um posto de trabalho completamente utilizável em minutos, com todas as ferramentas instaladas e configuradas para o ambiente de desenvolvimento.

- Utilização das máquinas

  Neste caso, também a melhor utilização das máquinas-clientes foi conseguida, pois pode ser distribuída a carga de processamento, entre as diversas CPUs que estavam à disposição, aumentando o desempenho e ao mesmo tempo diminuído os custos, porque estavam sendo utilizadas máquinas que já se achavam em disponibilidade para realizar a montagem do *cluster*. Mas, o grande trunfo é mesmo o tempo necessário para que um novo integrante consiga começar a produzir, pois um novo posto de trabalho fica pronto em questão de minutos.

Pontos negativos

Ainda não foi possível verificar todos os pontos, pois a equipe envolvida neste projeto é pequena, mas existem alguns tipos de problemas, o que mostra que há espaço para a procura de soluções mais avançadas.

- Infraestrutura de rede

  A infraestrutura de rede neste tipo de solução é muito exigida e deve ser bem projetada, implantada e configurada, para não deixar o sistema inusável. Este é por enquanto o calcanhar de Aquiles deste tipo de abordagem. Qualquer tipo de má configuração, sobrecarga momentânea deixa o sistema incrivelmente, lento.

- Utilização das máquinas-clientes

  Aqui existe um problema sério, utilizar diretamente, as máquinas-clientes, traz alguns empecilhos; muitas vezes as máquinas são desligadas sem um aviso prévio, podendo retardar ou inutilizar algum processo.

Informações coletadas

Alguns dados coletados no ambiente de desenvolvimento deste projeto.

Arquivos	Quantidade	Total em bytes
código-fonte	364	3.780.608
recursos	1.121	21.336.064
total código	1.485	25.116.672
total documentação	42	23.711.744

Tabela 5.10: Dados colhidos no projeto

LOC	LOC/arq	Pessoas	Produção por pessoa (LOC/p)	Tempo (mês)	Produção por mês (LOC/mês)	Produtividade (LOC/p/mês)
47.554	32	4	11.889	6	7.926	1.981

Tabela 5.11: Métricas calculadas

Considerações

Ainda há espaço para tentar melhorar mais a maneira pela qual uma equipe de desenvolvimento pode adotar um processo de gerência de configuração completa. É este espaço que deve ser procurado daqui para frente e não deve ser ignorado nenhum tipo de proposta, porque pode ser observada que a solução dos problemas de gerência de configuração, através, da utilização de *cluster* é extremamente interessante, e deve ser sempre considerada.

Como foi dito durante o desenrolar desta dissertação, os problemas estão dispostos em várias camadas, e conforme vão sendo solucionados, os mais exteriores, outros problemas vão surgindo, muitos deles decorrentes da própria adoção da nova técnica de trabalho. Isso nos mostra que sempre existe espaço para a procura de novas soluções para problemas já conhecidos, o que permite que a evolução sempre continue e que modificações em processos já sedimentados possam ser inseridas, lapidando e moldando o processo como um todo.

5.1.6 Comparação entre o *ClearCase* e o *CVS*

Como, nesta dissertação foram utilizados dois tipos de tecnologias diferentes para a ferramenta de gerência de configuração, é interessante mostrarmos alguns tempos envolvidos, quando são usadas estas ferramentas num ambiente de desenvolvimento de *software*.

Esta comparação foi feita usando-se 12580 arquivos, divididos em 676 diretórios, o que totaliza 440Mbytes de informação. Foi utilizada essa massa de dados para evitar alterações nas medições, que poderia ser causada por qualquer tipo de otimização que o sistema operacional viesse a fazer. As medições realizadas foram feitas sobre os comando `ci` e `co`, comandos esses, os mais utilizados no dia-à-dia de um ambiente de desenvolvimento.

Ferramenta	CI (min)	CO (min)
ClearCase(*dynamic view*)	853	122
ClearCase(*snapshot view*)	267	39
CVS	38	23

Tabela 5.12: Tempos observados em ferramentas diferentes

5.2 Conclusões

Verificando os diversos projetos apresentados, pode-se observar que conforme o aumento da tecnologia empregada na gerência de configuração de um projeto, a produtividade dos programadores aumenta gradativamente, pois os respectivos desenvolvedores ficam mais tempo resolvendo os problemas propostos pelo projeto, as máquinas ficam mais estáveis e a configuração de todo o ambiente é mais controlada. Esta evolução está representada no gráfico 5.1.

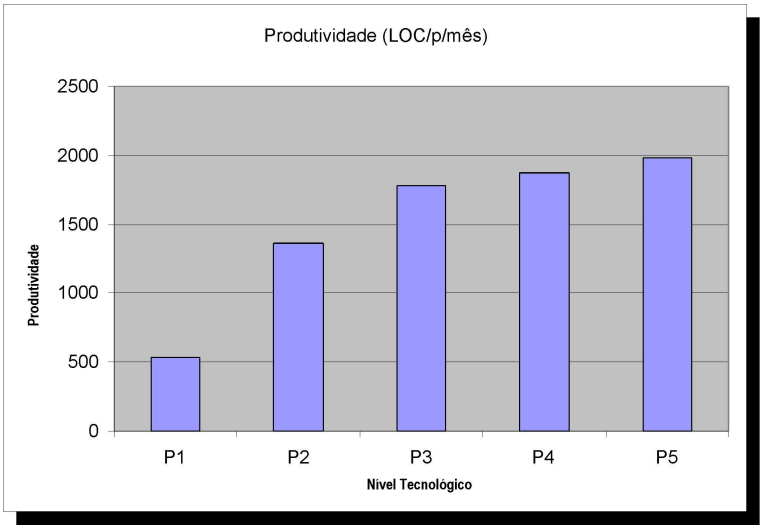

Figura 5.1: Produtividade num ambiente de desenvolvimento conforme a evolução das técnicas usadas na gerência de configuração

Um ponto interessante a ressaltar, é que a partir do momento que se aplica a gerência de configuração, o ganho que se tem é notável, conforme mostra a transição do projeto P1 para o projeto P2, mostrando que esta é uma técnica que ajuda em muito o processo de desenvolvimento.

Um outro ponto que podemos ressaltar é o tipo de ferramenta utilizada no processo de gerência de configuração. Como podemos observar a ferramenta adotada pode ajudar a melhorar ainda mais os tempos de resposta de uma equipe de desenvolvimento, uma vez que está acaba sendo usada com muita freqüência no dia-à-dia dos programadores, e quando mais eficiente for esta ferramenta, melhor e mais rápida será a adoção do sistema de gerência de configuração.

Capítulo 6

Conclusões e perspectivas

Foi apresentado nesta dissertação, um processo evolutivo de desenvolvimento de *software*, onde podem ser reconhecidos os avanços alcançados durante o processo de adoção de um ambiente de desenvolvimento com a gerência de configuração integrada. A principal contribuição de todo este trabalho de dissertação, está no novo modo de se pensar e usar um sistema de *cluster* para ambientes de desenvolvimento. Normalmente, usam-se sistemas distribuídos para aumento de desempenho, porém para o caso aqui analisado, o aumento de desempenho é tratado apenas como mais um item, mostrando assim que podemos enxergar um *cluster* de outro modo e usar as suas características em prol de soluções mais abrangentes e com um grau de abstração mais elevado.

A incorporação de novas tecnologias deve sempre trazer melhorias, mas para que isso aconteça, muitas vezes alguma coisa é perdida ou deve ser modificada. Muitas vezes pode existir uma redução da flexibilização, algumas vezes aumento no tempo de resposta das máquinas, etc, porém estes são os ganhos e perdas existentes em qualquer momento de inovação. Entretanto, essa seria uma visão por parte de um programador, que quando está inserido numa equipe de trabalho, deverá seguir algumas regras para que o trabalho em equipe possa ser realizado e concluído da melhor forma possível, e não pensar somente em seu trabalho e satisfação. Quando se está inserido em um grupo, a solução e satisfação coletiva é a mais importante. Essas regras, não só são necessárias para que haja o trabalho em equipe, mas sim para que o projeto possa ser monitorado, controlado e reproduzido, caso exista uma necessidade posterior.

Como item mais importante, existe a satisfação do cliente que comprou o produto, onde as suas necessidades possam ser atendidas da melhor forma e no menor tempo possível; os problemas que porventura venham a ser detectados nas instalações do cliente, onde todas as variáveis possíveis estarão presentes, onde todas as combinações imagináveis para o ambiente de produção serão contempladas, possam ser monitoradas

e corrigidas, entregando assim o produto com "a cara" que o cliente gostaria que ele tivesse. Normalmente, a interpretação das informações passadas pelas diversas equipes, distorcem os verdadeiros desejos do cliente, ou mesmo as alterações que podem aparecer nas interpretações que acontecem dentro da própria cadeia produtiva de uma empresa.

O processo de gerência de configuração é, por definição, evolutivo e deve ser passível de análises críticas para verificar se as direções tomadas são as melhores, ou ainda se são boas para determinado tipo de projeto. Através dessas análises, baseadas em métricas colhidas durante a evolução do projeto, pode ser verificado se determinada técnica foi ou está sendo adequada à solução do problema, e ainda verificar se esta solução é particular para certo projeto ou pode ser adotada de forma genérica.

Nos últimos anos, foram propostas soluções que fizeram com que o ambiente de desenvolvimento fosse extremamente maleável e gerenciável, e como última, porém, não definitiva, colocada a possibilidade de se utilizar a tecnologia de *clusters* para melhorar ainda mais todo o processo de gerência de configuração, desde o controle puro, até o aumento de desempenho e conseqüente diminuição do tempo de resposta para os desenvolvedores. Ao mesmo tempo, há uma redução do custo total dos recursos computacionais.

Entretanto, conforme foi observado a evolução do processo sempre acaba trazendo novos problemas, que antes não eram vistos ou que aparecem pela adoção desta nova estratégia, ou são problemas inerentes à própria solução. Assim, a solução de *cluster* para uma gerência de configuração avançada que é muito pertinente. Porém, deve ser pesquisada uma melhor solução a fim de realizar e disponibilizar esta solução, pois a utilização direta dos computadores-clientes, como a utilizada, pode causar problemas muito grandes para a equipe como um todo, quando qualquer uma das máquinas-clientes, componentes desta equipe é desligada de modo abrupto.

A aplicação de uma gerência de configuração, cada vez mais otimizada, permite à equipe de desenvolvimento ter um desempenho melhor, em termos de ocupação do pessoal, utilização de equipamentos e também nos prazos de entrega.

Uma das grandes vantagens da utilização de um processo definido e controlado, é que este traz um grande diferencial na qualidade do produto gerado, fazendo com que produtos mais sofisticados possam ser implementados e trazidos a público com maior rapidez, através das propostas colocadas durante todo este projeto, e principalmente, com a utilização de *clusters* de computadores, junto com a tecnologia de terminais, possa ser reduzido o custo de instalação, manutenção, gerência e planejamento, aos custos mínimos, causando o menor impacto no custo final total do produto. Ao mesmo tempo, mantendo o espírito da equipe coeso e criativo para a solução dos projetos propostos.

Após ter controlado e otimizado o processo de gerenciamento de configuração, utilizando-se de diversas tecnologias simultaneamente, abre-se a possibilidade de continuar a pesquisa, começando a envolver um nível a mais de abstração, iniciando, talvez, com o gerenciamento de requisitos, de maneira superposta a toda essa infraestrutura que já está criada e implantada.

Um outro ponto de pesquisa que pode ser atacado, é a utilização desta mesma solução em outros tipos de ambientes de programação, que não sejam ambientes de produção de *software* de telecomunicações, verificando assim se esta é uma solução abrangente ou específica para determinados tipos de ambientes de desenvolvimento.

Existe também, a possibilidade de se montar o mesmo ambiente com máquinas servidoras isoladas das máquinas-clientes, que estão na mesa de cada programador, entretando pode haver, um aumento do custo da implementação da solução, mas pode ser viável, se o ambiente de desenvolvimento for muito dinâmico e com usuários pouco comportados.

Referências Bibliográficas

[RDESKTOP, 2001] RDESKTOP - Remote Desktop. Internet Link: [s.n.], 2001. Disponível em: <www.rdesktop.org>. Acesso em: 27/08/2001.

[CITRIX, 2002] CITRIX - ICA Protocol. Internet Link: [s.n.], 2002. Disponível em: <www.citrix.com>. Acesso em: 27/10/2002.

[CVS, 2002] CVS - Concurrent Versions System. Internet Link: [s.n.], 2002. Disponível em: <www.cvshome.org>. Acesso em: 13/07/2002.

[RAID, 2002] RAID - Redundant Array of Inexpensive Disk. Internet Link: [s.n.], 2002. Disponível em: <www.adaptec.com>. Acesso em: 06/06/2002.

[VNC, 2002] VNC - Virtual Network Computing. Internet Link: [s.n.], 2002. Disponível em: <www.uk.research.att.com/vnc>. Acesso em: 30/10/2002.

[XFREE86, 2002] XFREE86 Project. Internet Link: [s.n.], 2002. Disponível em: <www.xfree86.org/support.html>. Acesso em: 04/11/2002.

[ABELL, 1998] ABELL, V. A.; TEAM, R. D. **ClearCase Administrator's Manual**. Maguire Road Lexington, Massachusetts: Rational Software Corporation, 1998. Document Number 800-010047-000.

[ABELL, 1998] ABELL, V. A.; TEAM, R. D. **ClearCase Reference Manual**. Maguire Road Lexington, Massachusetts: Rational Software Corporation, 1998.

[BARAK, 1993] BARAK, A.; GUDAY, S.; WHEELER, R. G. **The MOSIX Distributed Operating System - Load Balancing for UNIX**. Institute of Computer Science, The Hebrew University of Jerusalem, 91904 Jerusalem, Israel: Springer-Verlag Berlin Heidelberg, 1993.

[BOLINGER, 1995] BOLINGER, D.; BRONSON, T. **Applying RCS and SCCS - from Source Control to Project Control**. USA: O'Reilly, 1995.

[CEDERQVIST, 1992,1993] CEDERQVIST et al. **Version Management with CVS**. Box 204 S-508 02 Linkoping Sweden: Signum Support AB, 1992,1993.

[FOGEL, 1999,2000] FOGEL, K. **Open Source Development with CVS**. [S.l.]: GNU Licensing by Free Software Foundation, 1999,2000.

[GGC, 2001] GGC; GQ. **Apresentação Quality REFRESH ICN 2001**. Curitiba Paraná: Siemens LTDA - ICN ON D, 2001.

[JACOBSON, 1992] JACOBSON, I. **Object-Oriented Software Engineering - A Use-Case Driven Approach**. One Jacob Way, Reading, Massachusetts: Addison Wesley Longman, Inc., 1992.

[JALOTE, 2000] JALOTE, P. **CMM in Practice**. One Jacob Way, Reading, Massachusetts 01867: Addison Wesley Longman, Inc., 2000. SEI series in software engineering.

[KAN, 1995] KAN, S. **Metrics and Models in Software Quality Engineering**. One Jacob Way, Reading, Massachusetts 01867: Addison Wesley Longman, Inc., 1995.

[ORAM, 1991] ORAM, A.; TALBOTT, S. **Managing Projects with MAKE**. USA: O'Reilly, 1991.

[PAULK, 1994] PAULK, M. C. et al. **The Capability Maturity Model: Guidelines for Improving the Software Process**. Carnegie Mellon University: Addison Wesley Longman, Inc., 1994.

[TICHY, 1991] TICHY, W. F. **RCS - A System for Version Control**. West Lafayette, Indiana 47907: Departament of Computer Sciences - Purdue University, 1991.

[YOURDON, 1989] YOURDON, E. **Modern Structured Analysis**. USA: Yourdon Press, 1989.

Apêndice A

Glossário

Application Programming Interface (API)

É um conjunto de definições das bibliotecas de rotinas que permite a desenvolvedores externos ou terceiros a escreverem programas que são portáveis. Pode-se citar alguns exemplos muito conhecidos pela indústria, como: o *Berkeley Sockets* para aplicações que transferem dados pela rede, a GDI do Microsoft Windows, ou ainda a biblioteca Open/GL da Silicon Graphics para a descrição de objetos gráficos tridimensionais.

Largura de banda - *Bandwidth*

É a capacidade de comunicação de uma linha de transmissão, normalmente, medida em bits por segundo, por exemplo: uma rede de 100Mbit/s.

Sistemas *Batching*

Um sistema *batching* é um programa que controla o acesso aos recursos de computação. Normalmente, um usuário submete o seu trabalho (*job*) ao gerente do sistema *batching*, este agente enfileira todos os trabalhos requisitados e os vai disparando um-a-um.

Computação em *cluster*

Um sistema em *cluster* normalmente, é composto por várias estações de trabalho ou computadores pessoais interconectados por uma rede local, que representam ou são vistos como apenas um recurso computacional de grande capacidade. Este arranjo é normalmente, conhecido como CoW (*cluster of workstations*).

flops

Operações de ponto-flutuante por segundo (*Floating point operations per second*); é uma medida de quantas operações simples de ponto-flutuante uma máquina pode realizar, é utilizada para comparar desempenho entre diferentes máquinas.

Heterogêneo

Diz-se que um sistema é heterogêneo, quando este é composto por diferentes tipos de componentes.

Homogêneo

Diz-se que um sistema é homogêneo quando este é composto de componentes idênticos ou do mesmo tipo.

I/O (*Input/Output*) - E/S (Entrada/Saída)

Refere-se aos mecanismos de *hardware* e/ou *software* que conectam um computador ao seus periféricos. Pode-se incluir nestas conexões: a conexão do computador com o sub-sistema de discos, a conexão com terminais ou sistemas gráficos ou, ainda, um sub-sistema de rede.

Internet Protocol (IP)

É o protocolo da camada de rede usado na Internet. O IP é responsável pelo endereçamento e roteamento ponto-a-ponto, pelo encaminhamento de pacotes (*packet forwarding*, pela fragmentação de pacotes e sua re-montagem.

Job

Este é o termo dado às aplicações que são enviadas a um sistema *batching*. Um *job* acaba, quando a aplicação completa a sua execução.

Latência

É o tempo necessário para que um serviço requisite ou entregue uma mensagem independente da natureza da operação ou do seu tamanho.

Load balance

É a maneira de como a carga de processamento pode ser distribuída de maneira equilibrada entre todos os processadores disponíveis num certo ambiente. Um programa executa mais rápido, quando este está perfeitamente balanceado entre todos os processadores, pois assim todos estão compartilhando as suas capacidades de processamento e irão finalizar o seus trabalhos no mesmo instante.

MIPS

Milhões de instruções por segundo (*One Million Instructions Per Second*). Este é um índice de desempenho, normalmente se refere as operações em números inteiros ou a operações de não ponto-flutuante.

Message Passing Interface (MPI)

Um padrão feito pela comunidade de programação paralela, que disponibiliza um biblioteca de subrotinas necessárias para se escrever programas para computadores paralelos ou uma rede de estações de trabalho (NoW) visando a portabilidade dos programas entre as diversas arquiteturas.

Multitasking

Processo de se executar vários programas num único processador. Este conceito normalmente é implementado fazendo-se uma divisão de tempo do processador entre os diversos processos que estão prontos para serem executados, e executando uma troca de contexto entre eles.

Mosix

Extensão do *kernel* para sistemas operacionais UNIX, onde se implementa o um processo de balanceamento de carga através da migração dinâmica de processos.

Rede

É o meio de comunicação físico, constituída de um ou mais barramentos ou segmentos interconectados por equipamentos específicos, como: hub ou *switchs*.

Processo

É a unidade fundamental de uma implementação de *software* num sistema computacional. Um processo é uma parte de código que executa seqüencialmente num processador do sistema.

Enfileiramento - *Queuing*

O enfileiramento é um método pelo qual os diferentes *jobs* são ordenados e enfileirados para acessar um determinado recurso computacional.

Remote Procedural Call (RPC)

Um mecanismo que permite a execução de rotinas individuais em computadores remotos através de uma rede.

Ponto singular de falha - *Single Point of Failure* (SPF)

É a parte do sistema, onde, caso aconteça uma falha, todo o sistema falhará.

Dynamic view

Modo de operação do *ClearCase* onde todas as informações são virtuais e ficam centralizadas num servidor.

Snapshot view

Modo de operação do *ClearCase* onde é feita uma cópia das informações que estão no servidor, para uma área local do usuário.

RAID

É o termo que significa *Redundant Array of Inexpensive (or Independent) Disks.* Um RAID é uma coleção de dispositivos (*drives*) os quais de maneira coletiva atuam como um sistema de armazenamento único, o qual ainda pode tolerar a falha de qualquer dos dispositivos sem que ocorra a perda das informações neles contidas [RAID, 2002].

Supercomputador

É um termo, dependente do momento, que se refere a classe de computadores mais potentes do mercado mundial, num dado momento de referência.

Transmission Control Protocol (TCP)

O TCP é o protocolo de transporte orientado à conexão usado na internet. O TCP provê um método seguro e confiável de transferência de dados entre computadores.