Gerenciamento de Versões de Páginas Web

MARINALVA DIAS SOARES
RENATA PONTIN DE MATTOS FORTES
DILVAN DE ABREU MOREIRA

ICMC- Instituto de Ciências Matemáticas e de Computação
Caixa Postal 668, 13560-970 São Carlos, SP, Brasil
{mdsoares, renata, dilvan}@icmc.sc.usp.br

Abstract. In this paper we discuss some problems with version control, the evolution of information on the Web and the problems that the authors of web pages faced during development. An implementation that supports the versions control of files on the Web by several authors and allows browser users to retrieve old versions of the pages is presented. The implementation uses the tool for versions control files CVS, CGI and Html forms.

Keywords: version control, evolution of information on the Web, collaboration among authors

1 Introdução

Num mundo computacional em constante evolução, a *Web* se apresenta como um ambiente caracterizado por um desenvolvimento acelerado de suas informações. Além das informações na *Web* sofrerem muitas mudanças e com extrema freqüência, os desenvolvedores (ou autores) enfrentam dificuldades nas suas atividades quando envolvem muitas pessoas trabalhando em paralelo no desenvolvimento de uma página ou de um conjunto de páginas. Isso se deve do fato de que os autores trabalham independentemente em suas próprias cópias, tendo como principal problema a integração dessas cópias em um hiperdocumento final (Sommerville et al., 1998). Além disso, em geral, o volume de documentos envolvidos é significativamente grande e foge a um controle simples da evolução de suas cópias, pois pouco ou nenhum gerenciamento de informação é fornecido.

Em adição, os internautas[1] freqüentemente se surpreendem ao visitar uma página e percebem que esta já não possui o mesmo conteúdo ou até mesmo que ela não existe mais; tudo isso é decorrente da rápida e natural evolução das informações na *Web* (Sommerville et al., 1998).

Dessa forma, o objetivo deste trabalho foi desenvolver uma ferramenta para auxiliar no gerenciamento de versões de páginas *Web* por meio da própria *Web*, apoiando os autores ou desenvolvedores de uma página no trabalho colaborativo, sem que haja perda ou sobreposição de informações, além de possibilitar aos internautas visualizarem diferentes versões de uma mesma página e localizarem as diferenças entre elas. A ferramenta desenvolvida foi denominada *VersionWeb.*

Neste artigo são descritos alguns modelos de versão de *software* que descrevem alguns conceitos relacionados ao controle de versão e algumas formas de representação das organizações das diversas versões que são geradas de um arquivo (Seção 2). Na Seção 3 é apresentada uma descrição básica do CVS (*Concurrent Versions System*), uma ferramenta automatizada para controle de versões de arquivos que foi utilizada neste trabalho. Na Seção 4 é apresentada a ferramenta que foi desenvolvida, denominada *VersionWeb*. Finalmente, na Seção 5 são apresentados brevemente alguns trabalhos relacionados encontrados na literatura e na Seção 6 são apresentadas as conclusões e alguns resultados de testes de usabilidade que foram realizados na ferramenta *VersionWeb*.

2 Modelos de Versão de Software para SCM

É muito comum, no decorrer do desenvolvimento de um *software*, efetuarem-se mudanças para melhoria do *software* ou para correção de seus componentes e relacionamentos entre eles. Portanto, diante dessas mudanças, a necessidade de manter uma cópia de tudo que é feito e modificado torna-se importante, pois as alterações descontroladas podem gerar erros e perdas de informações. Para isso, essas cópias devem ser gerenciadas e controladas de forma que seja possível a recuperação das mesmas quando necessário.

O SCM, ou *Software Configuration Management*, é uma atividade abrangente aplicada em todo o processo de engenharia de *software*, e que é responsável por gerenciar a evolução de sistemas de *software* grandes e complexos (Pressman, 1995). O SCM identifica, controla, faz a auditoria e relata as modificações que inevitavelmente ocorrem quando o *software* está sendo

[1] Internauta é o nome pelo qual, comumente, o usuário que navega pela WWW é chamado.

desenvolvido e mesmo depois que ele é distribuído aos clientes.

O SCM pode ser visto como uma disciplina de apoio ao gerenciamento e desenvolvimento de *software*. No caso de apoio ao gerenciamento, o SCM gerencia o controle de alterações dos produtos de *software* fazendo a identificação dos componentes do produto e suas versões, o controle de alterações (pelo estabelecimento de procedimentos a serem seguidos quando uma alteração é realizada), a contabilidade de *status* (registrando os *status* dos componentes e pedidos de alteração), a análise e a revisão (garantia de qualidade das funções dos componentes para preservar a consistência do produto) (Kilpi, 1997). Já no caso de apoio ao desenvolvimento, o SCM fornece funções que auxiliam os programadores na realização coordenada de alterações nos produtos de *software*. Além disso, o SCM auxilia os desenvolvedores com a composição dos produtos de *software* a serem controlados registrando, assim, suas revisões (versões subseqüentes, seriais) e variantes (versões paralelas, independnetes), mantendo a consistência entre componentes dependentes, reconstruindo configurações de *software* previamente gravadas, construindo objetos derivados (código compilado e executável) de seus fontes (programa texto) e construindo novas configurações baseadas nas descrições de suas propriedades.

Os objetos (ou componentes) de *software* e seus relacionamentos constituem o espaço do produto, já as suas versões estão organizadas no espaço da versão. Existem vários modelos de versão de um *software*, mas cada um deles é caracterizado pela maneira como o espaço da versão está estruturado, pela decisão de quais objetos serão controlados e pela forma que a reconstrução de velhas e a construção de novas versões serão realizadas (Conradi e Westfechtel, 1998).

Um modelo de versão define os itens a serem controlados, as propriedades comuns compartilhadas por todas as versões de um item, os deltas e a maneira com que os conjuntos de versões estão organizados (introduzindo dimensões de evolução tais como revisões e variantes) (Conradi e Westfechtel, 1998).

Para um item que está sendo controlado, cada versão deve ser unicamente identificada através de um identificador de versão (VID - *version identifier*). Vários sistemas SCM geram automaticamente um número para cada versão e oferecem nomes simbólicos (definidos pelo usuário) que podem servir como chave primária de identificação. Todas as versões de um item compartilham propriedades comuns, onde cada uma dessas propriedades pode ser representada por relacionamentos ou atributos que não são alterados. Entretanto, a decisão de quais propriedades serão compartilhadas pelas versões depende do modelo de versão adotado e também da forma como esse modelo foi adaptado para uma certa aplicação.

O controle de versão é realizado com diferentes propósitos. Uma versão destinada a substituir seu predecessor é chamada revisão (versão serial). Além disso, uma revisão pode ser definida como sendo a forma mais simples de criar novas versões a partir de uma modificação na versão anterior. Dessa forma, as versões formam uma lista encadeada que pode ser referida como uma cadeia de revisões (Munch, 1995). Geralmente, uma nova revisão é gerada quando se torna necessário fazer alguma melhoria na versão anterior, prover algum aumento de funcionalidade, corrigir problemas, adaptar a versão anterior a algum outro ambiente, etc.

Por outro lado, versões que não substituem seu predecessor são chamadas variantes (versões paralelas). Por exemplo, variantes de estruturas de dados podem se diferenciar pelo consumo de armazenamento, eficiência de tempo de execução e operações de acesso. Já no que se refere ao produto de *software*, este pode ter variantes por suportar múltiplos sistemas operacionais e diferentes sistemas de janelas, por exemplo. Um outro exemplo pode ser o seguinte: considere que um simples programa é composto pelos componentes 1, 2, 3, 4 e 5. O componente 4 é usado somente quando o *software* é implementado usando monitores coloridos. O componente 5 é implementado para monitores monocromáticos. Neste caso, duas variantes de versão podem ser definidas: uma contendo os componentes 1, 2, 3 e 4; e outra contendo os componentes 1, 2, 3 e 5 (Pressman, 1997).

Finalmente, as versões podem também ser mantidas para apoiar a cooperação. Neste caso, múltiplos desenvolvedores trabalham em paralelo em diferentes versões e cada desenvolvedor opera em um *workspace* que contém as versões criadas e usadas. Para isso, algumas políticas de cooperação são necessárias para controlar quando as versões devem ser exportadas de um *workspace* ou importadas para dentro de um *workspace* (Conradi e Westfechtel, 1998).

Com base nos conceitos sobre modelos de versão de *software* descritos até aqui, serão ilustradas aqui algumas formas de representação de organização de versões. Existem várias formas de representar o espaço da versão, mas a maioria dos sistemas SCM faz essa representação através de grafos. Um grafo de versão consiste de nós e arestas que correspondem às versões e seus relacionamentos, respectivamente.

No caso mais simples (organização unidimensional), um grafo de versão consiste de um conjunto de versões conectadas por relacionamentos de um único tipo chamado **sucessores**. Um grafo de versão deste tipo representa a evolução histórica de um item,

onde "v_2 é sucessor de v_1" significa que "v_2 derivou de v_1" através de alguma alteração aplicada em v_1. Os grafos de versão podem ser representados de diferentes formas, como mostra a **Figura 2.1** . Na maioria dos casos, as versões podem estar organizadas em uma seqüência de revisões como está ilustrado na **Figura 2.1 (A)**. Na representação através de árvores, **Figura 2.1 (B)**, os sucessores de versões que não são folhas podem ser criados, por exemplo, de forma a manter as versões anteriores já distribuídas. A **Figura 2.1 (C)** representa um grafo acíclico, onde uma versão pode ter múltiplos predecessores (isso ocorre quando é realizada uma operação *merging*).

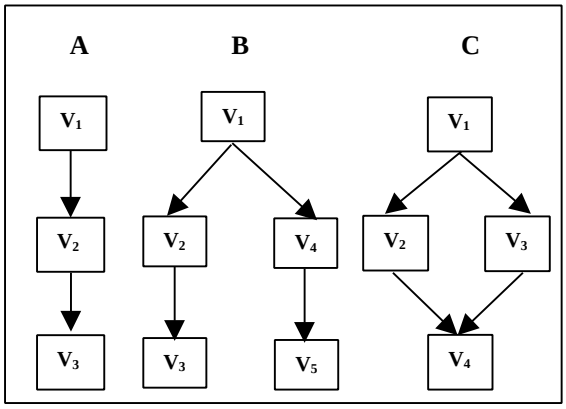

Figura 2.1 - organização unidimensional

Um grafo de versão em organização bidimensional é composto por *branches* (ramificações laterais), sendo que cada *branch* consiste de uma seqüência de revisões. Neste caso, no mínimo dois relacionamentos são necessários: **sucessores** (dentro de uma *branch*) e **descendentes** (entre as *branches*), como ilustra a **Figura 2.2**.

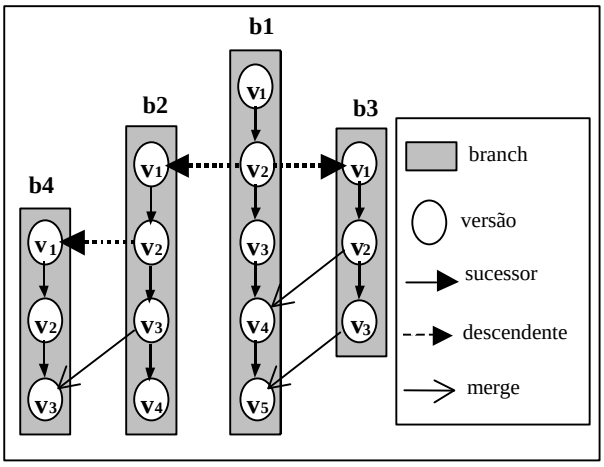

Figura 2.2 - organização bidimensional

Para economizar espaço nos dispositivos de armazenamento secundários (como disco, por exemplo) a maioria das ferramentas de controle de versão armazena versões na forma de deltas. Delta é o conjunto de diferenças entre duas versões subseqüentes de um mesmo arquivo (Conradi e Westfechtel, 1998). Em outras palavras, delta pode ser definido como uma seqüência de comandos de edição que transformam uma *string* em outra (Tichy, 1985).

Os sistemas de *software* consistem de muitos componentes e, durante o desenvolvimento e manutenção de um sistema, os componentes sofrem mudanças. Os componentes resultantes de transformações são chamados de versões, e essas podem ser distinguidas em dois tipos: variantes (versões paralelas) e revisões (versões seriais) (Bieliková, 1999). As versões de um componente de *software* são muitas vezes organizadas em uma árvore histórica (ancestral), a qual tem uma versão principal (*root*) que representa a primeira versão de um componente de *software*. A árvore histórica inicial é simples, consistindo somente de uma *branch*, chamada tronco, mas com o prosseguimento do desenvolvimento, *branches* laterais podem surgir. As *branches* podem surgir, na maioria das vezes, nas seguintes situações:

- no desenvolvimento simultâneo entre múltiplos usuários;
- no desenvolvimento distribuído em vários *sites*;
- quando versões anteriores ainda precisam ser melhoradas (alteradas);
- quando há a necessidade de criar versões com propósitos alternativos.

Uma tarefa essencial do SCM é armazenar a árvore histórica de versões eficientemente e, para isso, várias técnicas têm sido propostas (Bieliková, 1999). A idéia principal para economizar espaço de armazenamento é a seguinte: se uma versão for derivada de uma outra versão (ou seja, um sucessor na árvore histórica) então as duas versões provavelmente têm uma grande parte em comum e um pequeno número de diferenças. Portanto, para economizar espaço, uma versão é armazenada completa e a outra em forma de delta, ou seja, armazena-se somente as diferenças de conteúdo. Durante a análise de armazenamento dos deltas, dois aspectos são importantes: **a)** como gerar um delta entre dois arquivos **b)** como aplicar o delta na árvore histórica de versão, ou seja, qual versão armazenar de forma completa e em qual versão aplicar o delta.

Atualmente, as técnicas de armazenamento de deltas mais conhecidas são o delta *reverse* e o delta *forward*. Para gerar um delta entre dois arquivos, um algoritmo para isolação da seqüência comum entre os dois é usado (Tichy, 1985).

A técnica mais simples de delta é o delta *forward*, ou seja, as diferenças entre dois arquivos são criadas de forma que o delta seja aplicado para transformar a versão mais velha em uma mais nova. Os deltas *forward* são calculados entre a primeira e a segunda versão, a segunda e a terceira, e assim por diante, pois somente a

primeira versão é armazenada completa, e as posteriores são armazenadas como deltas *forward* (como acontece, por exemplo, no SCCS - *Source Code Control System*) (Rochkind, 1975). Já os deltas *reverse* são calculados a partir da versão mais recente até a versão requerida na árvore histórica de versões, pois, neste caso, a última versão é armazenada completa e as anteriores são armazenadas como deltas *reverse* (essa técnica é utilizada, por exemplo, no RCS - *Revision Control System*) (Tichy, 1985).

No caso de somente versões mais recentes serem requeridas com mais freqüência, a técnica *forward* apresenta desvantagens em relação à técnica *reverse*, pois o tempo de reconstrução de uma versão mais recente é maior através do delta *forward* (Bieliková, 1999).

3 CVS (*Concurrent Versions System*)

O CVS foi originalmente desenvolvido por Dick Grune em 1986, e consistia de um conjunto de scripts shell do UNIX. Em 1989, ele foi projetado e codificado na linguagem de programação C por Brian Berlinger (Cederqvist, 1993) e, mais tarde, Jeft Polk acrescentou o projeto de módulos e suporte à geração de *branches* no CVS.

Em grandes projetos de desenvolvimento de *software*, é comum que duas ou mais pessoas tenham a necessidade de modificar o mesmo arquivo ao mesmo tempo (Hung e Kunz, 1992; Berliner, 1990). Com o RCS ou SCCS isso é impossível, pois os arquivos são bloqueados (*locked*) ao fazer o *checkout* para um diretório de trabalho, fazendo com que somente uma pessoa possa obter uma cópia do arquivo para escrita por vez.

Embora o bloqueio de arquivos seja desejável na teoria, isso traz conseqüências indesejáveis ao grupo de desenvolvedores, pois esse bloqueio causa a serialização do processo de desenvolvimento fazendo com que o restante do grupo fique esperando a pessoa que está com o arquivo fazer as alterações e efetuar o *commit*. Feito isso, o arquivo é liberado e, então, um outro desenvolvedor pode obter uma cópia do arquivo para escrita.

Com o sistema CVS, cada desenvolvedor pode obter uma cópia de uma versão do arquivo para escrita de todo o código fonte de um projeto para dentro do seu diretório de trabalho sempre que quiser. O CVS é um sistema de controle de versões que permite gravar o histórico de arquivos fonte (Cederqvist, 1993; CVS, 1999) e, como qualquer outra ferramenta de gerenciamento de configuração de *software* ou de controle de versões, ele possui um repositório central que armazena as cópias mestres de todos os arquivos que estão sob o gerenciamento de versões.

O CVS foi projetado para manter a localização de alterações feitas nos arquivos por grupos de desenvolvedores e é baseado no RCS, porém o CVS fornece algumas características adicionais como: permitir o trabalho paralelo entre desenvolvedores através do acesso concorrente aos arquivos, dar suporte à geração de *releases* e evitar sobreposição ou perda de informações quando duas ou mais pessoas estão trabalhando no mesmo arquivo simultaneamente (Hung e Kunz, 1992).

Dentre as funcionalidades básicas do CVS, podemos citar:

- mantém um histórico de todas as alterações feitas em cada árvore de diretório que ele gerencia; usando esse histórico, o CVS pode recriar estados anteriores da árvore, ou mostrar a um desenvolvedor quando, porque e por quem uma alteração foi feita;
- armazena e recupera versões anteriores de arquivos eficientemente;
- fornece controle de arquivos através da rede de forma transparente para grupos de desenvolvedores;
- suporta desenvolvimento paralelo, permitindo que mais de uma pessoa trabalhe em um mesmo arquivo ao mesmo tempo;
- fornece acesso seguro às árvores de diretórios de *hosts* remotos usando protocolos Internet.
- permite adicionar, remover e alterar arquivos e diretórios do repositório (repositório é a hierarquia de diretórios onde residem os arquivos que estão sendo gerenciados);
- permite agrupar uma coleção de arquivos relacionados em módulos e, então, o módulo passa a ser gerenciado, em vez dos arquivos separadamente;
- tags simbólicas podem ser associadas a um conjunto específico de revisões;
- executa em várias plataformas como Unix, Windows 95, Windows NT, Macintosh e VMS (CVS, 1999).

O CVS salva todas as informações de controle de versão em arquivos RCS armazenados no repositório, sendo que este é separado do diretório de trabalho do usuário.

Os arquivos históricos (nome do arquivo fonte acrescido de ",v" no final) do CVS contêm informações suficientes para recriar qualquer revisão do arquivo, localizar o autor, data e hora da revisão, além dos comentários da alteração realizada para melhor identificar a razão da geração daquela revisão. Todos os arquivos históricos são criados apenas para leitura e essa permissão não deve ser alterada devido às informações neles contidas para a recuperação de suas revisões. Os diretórios dentro do repositório podem ser habilitados

para escrita por pessoas que tenham permissão para modificar os arquivos dos diretórios.

O CVS mantém as permissões de arquivo para novos diretórios que são adicionados dentro da árvore de diretórios, mas as permissões devem ser dadas manualmente quando um novo diretório tiver permissões diferentes do seu diretório pai.

4 A ferramenta *VersionWeb*

A ferramenta *VersionWeb* foi desenvolvida com o objetivo de auxiliar os autores no desenvolvimento de páginas, especificamente para dar um suporte à evolução das mesmas, de forma coordenada e controlada evitando, assim, perdas ou sobreposições de informações. Outro objetivo da *VersionWeb* foi o de permitir que os internautas, durante a navegação, tivessem acesso às informações que alguma vez estiveram disponíveis, mas que, geralmente, como conseqüência das constantes atualizações das páginas não estão mais, naquele momento, sendo apresentadas. Por meio da recuperação de versões anteriores dessas páginas e localização de alterações, o usuário pode rever informações disponibilizadas em algum momento anterior.

Para o desenvolvimento dessa ferramenta foi utilizado o CVS (descrito na Seção anterior), *software* para controle de versões que viabilizou o desenvolvimento do suporte à evolução das páginas *Web*. A comunicação entre a *VersionWeb* e o CVS foi efetuada através de CGIs (*Common Gateway Interface*) que ficam residentes na máquina servidora juntamente com o CVS. Os programas CGIs foram implementados especialmente para viabilizarem essa comunicação.

A *VersionWeb* proporciona uma interface amigável baseada em formulários HTML que oferece as operações básicas de controle de versão para executar os comandos do CVS, uma vez que este é totalmente orientado a linha de comando. Para incluir os recursos do CVS em uma aplicação baseada na *Web*, uma abordagem arquitetural foi adotada e está ilustrada na **Figura 4.1**.

Para que o CVS faça o controle dos arquivos, estes devem estar armazenados em um repositório CVS. E, para tornar esse repositório acessível através da *Web*, decidimos colocá-lo na mesma máquina em que estão localizados o servidor de *Web* (servidor HTTP) e os programas CGIs por razões de segurança e de desempenho. Certamente, o repositório CVS poderia estar em uma outra máquina diferente daquela onde reside os CGIs e o servidor de *Web*. Porém, seria mais uma conexão a ser estabelecida para a execução dos pedidos do usuário (que normalmente são comandos CVS), pois o CGI (programa que faz a chamada do CVS) teria que localizá-lo nessa outra máquina, enviar os pedidos do usuário e o CVS, depois de executá-los, teria que devolver o resultado para o CGI que o chamou na primeira máquina para que o resultado fosse enviado ao usuário.

De acordo com a abordagem arquitetural da *VersionWeb* esquematizada na **Figura 4.1**, um cenário de uso geral da ferramenta foi desenvolvido.

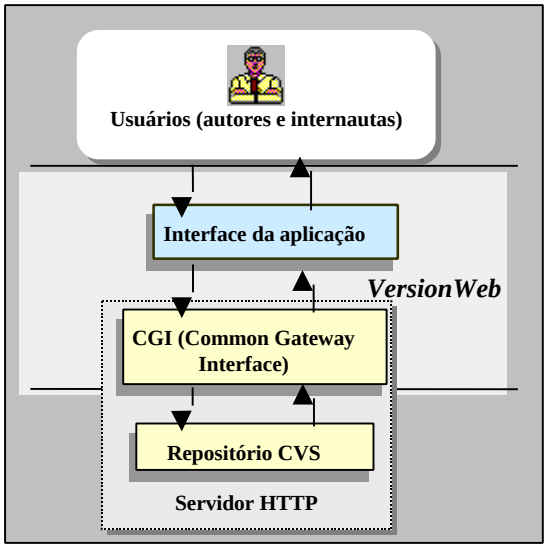

Figura 4.1 - Arquitetura básica da *VersionWeb*

No cenário de interação do usuário com a ferramenta (representado na figura 4.1), juntamente com os seus dois componentes funcionais principais (interface da aplicação e os programas CGIs), é possível o seguinte procedimento para a realização das tarefas de controle de versão através da *Web*:

- os usuários, que estão em suas próprias máquinas (locais ou clientes) em qualquer lugar do mundo, interagem com o CVS através da interface de aplicação da *VersionWeb* (baseada em formulários HTML) e submetem pedidos a um programa CGI que reside fisicamente na mesma máquina que o servidor de *Web*; esses pedidos, geralmente, são comandos CVS a serem executados como *checkout*, *commit*, etc.;
- o servidor de *Web* recebe os pedidos do usuário e os passa ao CGI;
- o CGI recebe e decodifica os pedidos do usuário;
- o CGI que recebeu o pedido faz uma chamada ao CVS (também fisicamente localizado na mesma máquina) com os comandos a serem executados;
- o CVS executa os comandos e retorna o resultado ao CGI que o chamou;
- o CGI envia os resultados ao servidor de *Web*;

- servidor de *Web* devolve os resultados aos clientes, ou seja, os usuários que iniciaram a interação com a ferramenta.

A partir dessa arquitetura, o usuário precisa apenas de um *browser Web* em sua máquina local para utilizar a *VersionWeb*, o que favorece ainda mais a sua mobilidade em relação ao acesso à ferramenta. Os arquivos ficam em um repositório central (repositório do CVS) no servidor, mas as modificações nos arquivos podem ser remotas (no próprio servidor) ou locais (na máquina do cliente).

Para que pudesse ter os recursos do CVS disponibilizados, via *Web*, a ferramenta foi construída com base em três funcionalidades principais, para as quais foram desenvolvidas interfaces básicas: **a)** gerenciamento dos usuários **b)** gerenciamento dos arquivos que estão sob o controle de versão e **c)** navegação para acesso às versões anteriores da página que estiver sendo visitada.

Todas essas funcionalidades podem ser obtidas mediante verificação de que o usuário pertença a um dos grupos autorizados, ou seja, inicialmente o usuário é submetido a um procedimento de autenticação.

A **Figura 4.2** ilustra a estrutura modular da *VersionWeb*, sendo que os nós folhas representam seus três principais módulos, os quais possuem funções específicas.

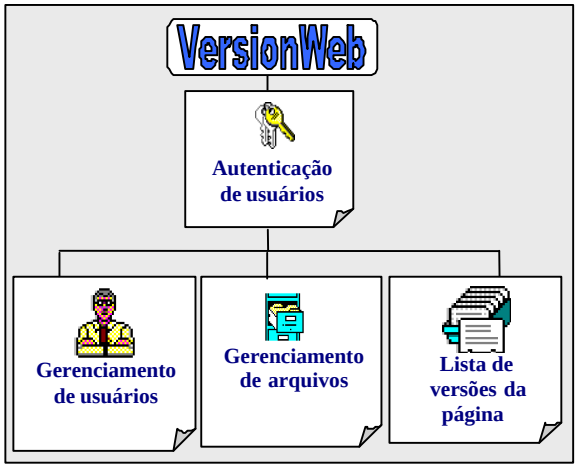

Figura 4.2 - Estrutura modular da *VersionWeb*

Por questões de projeto e de segurança, foi desenvolvido o módulo de autenticação de usuários na *VersionWeb*. Depois de autenticado, o usuário possui as permissões de acesso e, portanto, poderá ser levado, de acordo com a categoria de usuários a que ele pertence, a uma das três interfaces básicas da ferramenta: de gerenciamento de usuários, de gerenciamento de arquivos ou de acesso às diversas versões da página visitada (esta última é no caso dos internautas).

Do ponto de vista de quais os usuários se beneficiariam dos serviços da *VersionWeb*, foi possível identificar inicialmente dois tipos de usuários: os autores das páginas e os internautas e/ou grupos específicos de internautas que navegam pelas páginas através de *browsers*. No entanto, para efeito de se gerenciar esses tipos de usuários, foi definido também um terceiro tipo de usuário, denominado administrador. Dessa forma, a ferramenta *VersionWeb* foi desenvolvida de maneira a atender esses três tipos básicos de usuários:

- **administradores:** pessoas responsáveis por gerenciar todos os usuários da ferramenta;
- **autores:** pessoas autorizadas a manipularem os arquivos que estão sob o controle de versão e que residem no repositório CVS no servidor;
- **internautas e/ou grupos específicos de internautas:** pessoas às quais é permitido apenas visualizar as versões de uma página, as diferenças entre elas, receber notificação de novas versões da página disponíveis e enviar comentários.

O primeiro passo para o uso da ferramenta *VersionWeb* é a autenticação do usuário (obrigatório para autores e administradores), o que dá acesso às funcionalidades principais da ferramenta. Assim, a *VersionWeb* inicia a identificação do usuário através da interface de autenticação ilustrada na **Figura 4.3**, para que a seguir, a interface relacionada com a devida permissão do usuário seja então disponibilizada.

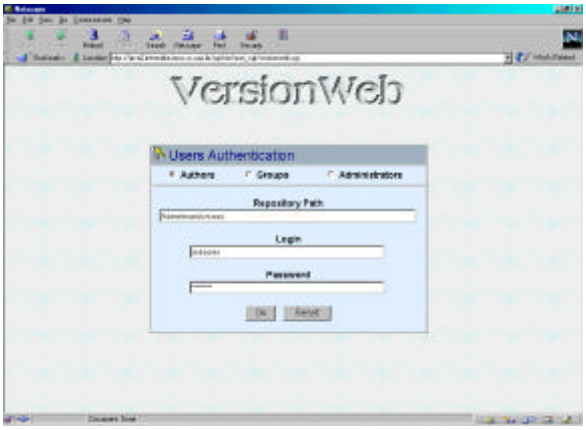

Figura 4.3 - Interface de autenticação de usuários

Após o processo de autenticação, cada tipo de usuário tem acesso a um determinado conjunto permitido de serviços. Se o usuário se autenticou como um administrador, ele é responsável por gerenciar todos os usuários da ferramenta e terá acesso à interface ilustrada na **Figura 4.4**.

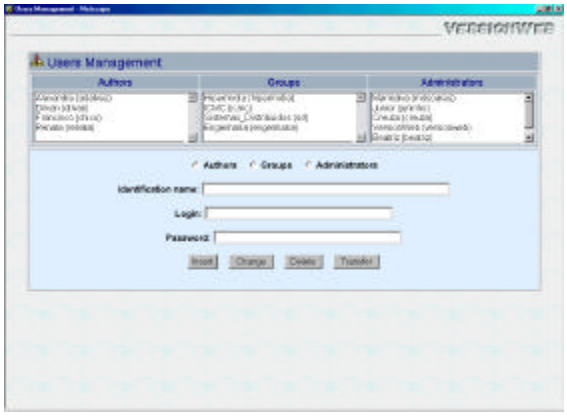

Figura 4.4 - Interface de gerenciamento de usuários

A partir da interface de gerenciamento de usuários (**Figura 4.4**), ao administrador são disponibilizadas operações para controle de informações dos usuários em cada um dos tipos definidos. Essas operações são:

- **Insert**: insere um usuário em uma das listas de usuários da ferramenta (*Authors*, *Groups* ou *Administrators*).
- **Change**: permite alterar os dados de um usuário (*name*, *login* e *senha*).
- **Delete**: permite remover um usuário de uma das listas de usuários da ferramenta.
- **Transfer**: permite transferir um usuário de uma lista para outra.

Se o usuário se identificou como um autor, ele terá acesso aos serviços relacionados com o gerenciamento de arquivos (interface ilustrada na **Figura 4.5)**. Nessa interface é exibido todo o conteúdo (arquivos e diretórios) do repositório CVS requisitado na tela de *login* pelo autor. Os arquivos gerenciados pelo CVS podem ser tanto texto como binários.

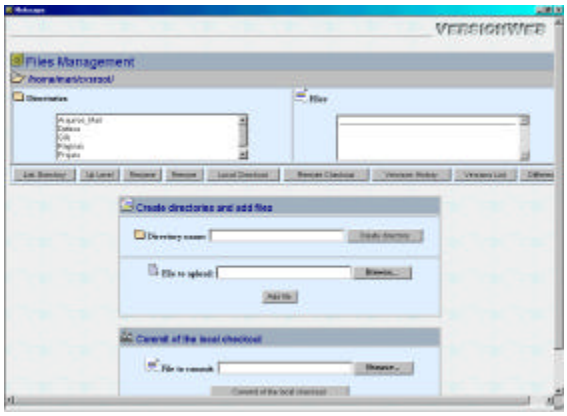

Figura 4.5 - Interface principal de gerenciamento de arquivos

A partir dessa interface (**Figura 4.5**) estão disponíveis as operações básicas de manipulação de arquivos e as operações básicas de controle de versão usuais do CVS, que são:

- **List Directory**: lista o conteúdo do diretório selecionado.
- **Up Level**: sobe um nível na árvore de diretório.
- **Rename**: renomea um diretório/arquivo selecionado.
- **Delete**: remove um diretório/arquivo selecionado.
- **Remote Checkout**: faz *checkout* remoto (da versão corrente) do arquivo selecionado no próprio servidor.
- **Local Checkout**: faz *checkout* (da versão corrente) de um arquivo/diretório para a máquina do usuário.
- **Versions History**: lista as versões de um arquivo selecionado com seus respectivos autores, data, hora e mensagem de *commit*.
- **Versions List**: abre uma nova janela com uma lista de todas as versões e *branches* do arquivo selecionado e operações permitidas sobre elas.
- **Diffs**: permite o autor visualizar as diferenças de conteúdo (localizar alterações) entre as diversas versões do arquivo selecionado.
- **Create directory**: cria um diretório no repositório;
- **Add file**: adiciona um arquivo (que está local) ao repositório (no servidor) e o coloca sob o gerenciamento do CVS.
- **Commit of the local checkout**: faz o *commit* de um arquivo e gera uma versão subseqüente àquela da qual foi feito o *checkout* local.
- **Reload this page**: faz o *reload* da página atualizando (no *browser*) o conteúdo do repositório quando um novo arquivo é adicionado.

De acordo com suas funções, a maioria dessas operações prossegue abrindo outras janelas de interação com o usuário, de maneira a efetuar as tarefas necessárias à sua realização, mas serão apresentadas aqui, por questões de espaço, somente algumas dessas janelas.

Através da opção "VersionsList" da interface ilustrada na **Figura 4.5**, o autor poderá obter a lista de todas as versões e *branches* de um arquivo com suas respectivas datas e autores (veja **Figura 4.6**), fazer *checkout* remoto e/ou local sobre qualquer versão do arquivo e fazer o *commit* das alterações efetuadas. Neste caso, para o *commit*, o autor deverá especificar uma *branch* a ser gerada, pois o *checkout* não foi efetuado, normalmente, sobre a versão corrente do arquivo, e sim sobre uma versão anterior à atual.

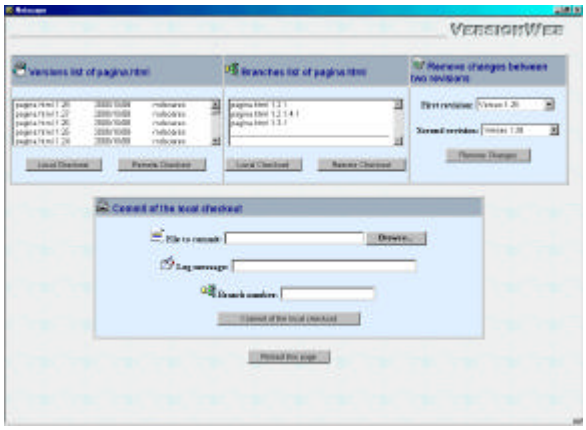

Figura 4.6 - Lista de versões e *branches* de um arquivo

Através da interface ilustrada na **Figura 4.6**, o autor poderá fazer *checkout* remoto e/ou local de qualquer versão anterior ou de uma *branch*. Quando se faz *checkout* de uma *branch*, o CVS na verdade sempre faz o *checkout* da última versão daquela *branch* e, neste caso, o *commit* gera uma versão automática para aquela *branch* e o autor não precisa especificar um número para ela. Por exemplo, suponha que existam 3 versões para a *branch* 1.3.1 (1.3.1.1, 1.3.1.2, 1.3.1.3). Ao fazer o *checkout* da *branch*, o CVS pega automaticamente sua última versão (1.3.1.3) e o *commit* irá gerar a versão 4, ou seja, 1.3.1.4. Se o *checkout* for local, o *commit* deve ser feito através da opção "Commit of the local checkout" ilustrada na mesma interface da **Figura 4.6**.

Se o usuário fizer *checkout* (remoto ou local), por exemplo, da revisão 1.2 e já existir a revisão 1.3, ele deverá especificar um número de versão (normalmente uma *branch*) a ser gerado no momento do *commit*, caso contrário irá surgir um conflito entre versões e o *commit* não poderá ser efetuado. A *branch* a ser gerada poderá ser 1.2.1, 1.2.2, 1.2.3, etc., e não 1.1.1, 1.1.2, etc (a menos que o *checkout* tenha sido feito da revisão 1.1).

O CVS, no momento do *commit*, gera uma versão automática para a *branch* especificada, ou seja, se o autor especificar o número 1.2.2 para a *branch* a ser gerada, o resultado será 1.2.2.1, onde "1" é a versão da *branch* (neste caso é a primeira versão da *branch*).

Se o *checkout* de uma das versões da lista de versões for remoto (opção "Remote Checkout" da **Figura 4.6**), o autor obterá a tela da **Figura 4.7**. Se for local, o *commit* deverá ser feito através da opção "Commit of the local checkout" da **Figura 4.6**.

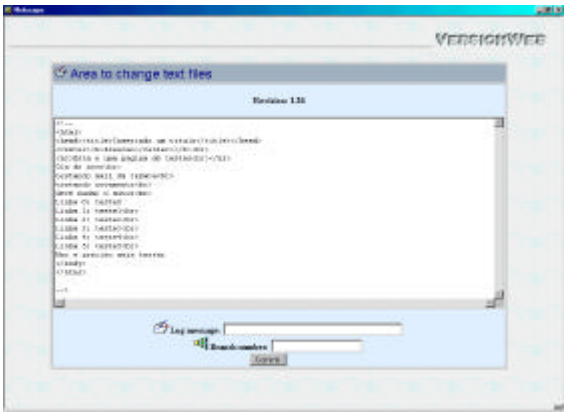

Figura 4.7 - Área de alteração do conteúdo de um arquivo texto com opção para gerar *branches*

Ainda na interface da **Figura 4.6**, o autor poderá remover alterações feitas entre duas versões diferentes do arquivo. Para isso, ele deve selecionar as duas versões do arquivo (a primeira versão selecionada deve ser maior que a segunda) e clicar sobre o botão "Remove Changes". Por exemplo, se o autor selecionar as versões 1.5 e 1.2, o CVS remove todas as alterações que foram feitas entre a versão 1.2 e 1.5 e gera uma nova versão com a remoção efetuada, ou seja, essa nova versão irá conter, na verdade, o conteúdo da versão 1.2. Isso pode ser útil quando se deseja recuperar uma versão e trabalhar sobre ela na linha principal de desenvolvimento, e não em uma *branch* separada.

Um dos objetivos da *VersionWeb* foi o de permitir que os internautas, durante a navegação, tivessem acesso às informações que alguma vez estiveram disponíveis, mas que, geralmente, como conseqüência das constantes atualizações das páginas não estão mais, naquele momento, sendo apresentadas. Por meio da recuperação de versões anteriores dessas páginas e localização de alterações, o usuário pode rever informações disponibilizadas anteriormente.

Para atender esse objetivo, o administrador pode escolher entre permitir que todos os internautas ou um grupo específico destes tenham acesso às versões da página. Se o administrador optar por deixar que apenas grupos específicos de internautas tenham esse acesso, a *VersionWeb* requer que o internauta faça parte de algum grupo já cadastrado. Para isso, a página que ele estiver visitando, se estiver sob controle de versão, conterá um *link* para a ferramenta *VersionWeb*. Se o acesso às versões for restrito a um grupo específico, o usuário deve então passar pelo processo de autenticação de usuários (através da interface ilustrada na **Figura 4.3)**. Se a autenticação estiver correta, o usuário obterá a tela mostrada na **Figura 4.8** a seguir.

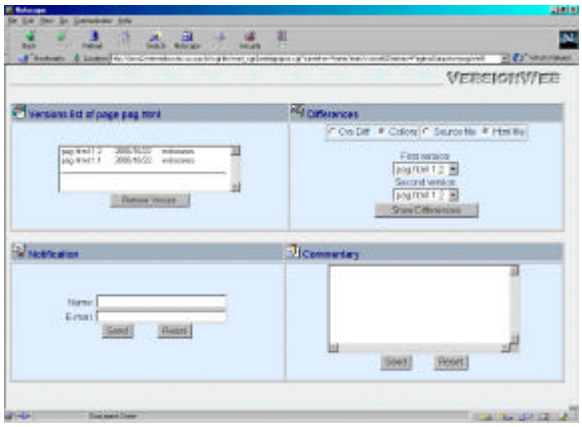

Figura 4.8 - Interface principal de recuperação de versões pelos internautas

Se o administrador permitir que todos os internautas tenham acesso às versões da página, eles terão acesso direto à interface ilustrada na **Figura 4.8** quando acionarem o *link* para a ferramenta, ou seja, eles não terão de passar pela tela de autenticação de usuários.

A partir da interface ilustrada na **Figura 4.8**, o internauta poderá realizar quatro operações:

- recuperar uma versão para visualização;
- visualizar as diferenças de conteúdo entre duas versões através de cores ou do próprio formato exibido pelo CVS;
- pedir para ser notificado de novas versões disponíveis da página;
- e enviar comentários ou críticas sobre a página e/ou ferramenta para os autores.

Como pode ser observado na **Figura 4.8**, a lista de versões para recuperação é exibida juntamente com a data e o autor da geração de cada versão. Se o usuário desejar recuperar uma versão da página para visualização, ele deverá selecionar a versão desejada e clicar sobre o botão "Retrieve Version". Feito isso, a versão da página é recuperada em uma nova janela do *browser* para visualização como uma página normal, ou seja, como se ele estivesse especificado diretamente na URL uma página para ser mostrada no *browser*.

Geralmente, quando as alterações efetuadas na página são pequenas, fica difícil para o usuário identificar o que realmente mudou naquela página apenas através da visualização de uma versão de cada vez. Dessa forma, a *VersionWeb* possibilita que o internauta visualize as diferenças entre duas versões quaisquer da página, ou seja, o que realmente mudou de uma versão para outra.

Para isso, o usuário deverá informar se quer que essas diferenças sejam exibidas através de cores ou no formato mostrado pelo CVS (o que não é muito aconselhável para quem não está familiarizado com o CVS), e se deseja ver as diferenças no fonte da página ou da forma como ela é exibida pelo *browser*. Feito isso, ele deverá então especificar as versões para as quais as diferenças deverão ser exibidas. A interface ilustrada na **Figura 4.9** é um exemplo que mostra as diferenças através de cores do fonte do arquivo HTML.

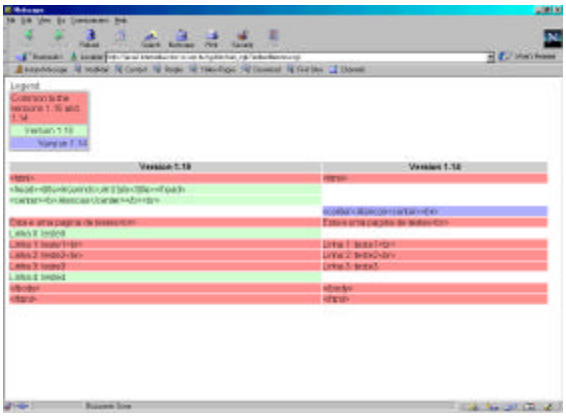

Figura 4.9 - Visualização das diferenças entre duas versões de um arquivo através de cores

Ainda através da interface ilustrada na **Figura 4.8**, o usuário poderá pedir para ser notificado de novas versões disponíveis da página. Essa notificação é feita automaticamente pelo CVS quando uma nova versão daquela página é gerada. Porém, o autor pode não disponibilizar essa nova versão, o que vai causar uma certa decepção no usuário, pois ele vai receber uma notificação de que uma nova versão da página foi gerada, mas que pode não estar disponível. Além disso, o usuário pode enviar sugestões ou críticas sobre as alterações efetuadas na página. Todos os autores da *VersionWeb* terão acesso aos comentários enviados pelo usuário.

5 Trabalhos relacionados

Atualmente, as ferramentas existentes para controle de versões de arquivos e gerenciamento de mudanças, como o RCS e o SCCS por exemplo, têm ajudado desenvolvedores de *software* no controle das alterações. Os desenvolvedores, com o uso de tais ferramentas, economizam tempo e espaço físico requeridos para as diversas cópias criadas durante o processo de desenvolvimento. Em ambientes de autoria, especialmente falando da *Web*, foram encontradas algumas ferramentas tais como V-Web e AIDE.

O V-Web é um simples sistema de versionamento de páginas (Sommerville et al., 1998). A principal dificuldade nesse sistema foi a necessidade de manter o acesso à página através da sua URL original da página não versionada (que não estava sob o controle de versão). No modelo de versão adotado em V-Web, uma página original não versionada é substituída por uma página V-Web. Esta contém o conteúdo da página original e uma referência para uma lista de versões da

página que está armazenada em um diretório juntamente com informações sobre cada uma de suas versões.

Esse sistema é muito utilizado em situações onde a informação evolui naturalmente através de uma série de versões, onde existe a necessidade de acessar versões anteriores da mesma forma que a versão atual e quando a informação é produzida com um esforço colaborativo por equipes de pessoas responsáveis pela criação da informação.

Neste sistema, a página original (não versionada) é substituída por uma página V-Web. A página V-Web possui basicamente 3 *frames* HTML: um *frame* exibe o conteúdo da página original, outro *frame* mostra o conjunto de versões da página e o último *frame* mostra um conjunto de opções que permite os autores terem acesso às funcionalidades do V-Web como: adicionar uma versão à lista de versões, remover uma versão do conjunto de versões e criar uma página V-Web de uma página não versionada. Essas operações são feitas por *scripts* CGIs no servidor.

V-Web dá suporte para a autoria por grupos de autores. Os autores devem estar registrados em algum grupo para terem acesso à edição das páginas. Cada grupo tem um conjunto de páginas que lhe é específico. Se um autor fizer *login* como um membro do grupo, ele poderá ver o que os membros desse grupo estão fazendo, mas não poderá editar as páginas. Cada membro do grupo tem acesso a um conjunto de páginas que lhe é específico e pode editar somente esse conjunto.

Cada membro deve estar consciente do trabalho cooperativo e saber o que o outro está fazendo para facilitar a união de seus trabalhos em uma cópia final. Quando o grupo termina o trabalho e junta as cópias em uma única versão, essa versão é então disponibilizada para acesso público. Para isso, é necessário comunicação constante entre eles para notificar uns aos outros das atividades que estão sendo feitas para não haver inconsistências.

AT&T Internet Difference Engine (AIDE) é um sistema que fornece algum suporte de versão usando o sistema RCS (Douglis et al., 1998). O objetivo principal do AIDE é permitir que os usuários vejam as diferenças entre páginas *Web* quando elas são atualizadas. Esse sistema permite que os usuários armazenem as páginas no sistema RCS e fornece facilidades para mostrar as diferenças entre as versões passadas e a versão atual da página. Os usuários devem requisitar explicitamente ao sistema para mostrar essas diferenças de acordo com a data. O sistema também fornece uma facilidade automática para detectar quando páginas de interesse foram atualizadas e permite recuperar as versões mais recentes dessas páginas.

Neste sistema, os usuários devem: **a)** especificar a URL para localização das mudanças, incluindo o grau de recursividade de *links* para outras páginas **b)** especificar que a nova versão da URL que está sendo localizada deve ser salva **c)** ver de forma textual as diferenças entre duas versões **d)** ou ver um grafo mostrando a estrutura de uma página, incluindo o estado das páginas a que ela se refere.

O sistema AIDE consiste de alguns componentes, incluindo um *Web-crawler* que detecta alterações nos arquivos, um arquivo contendo as versões da página, uma ferramenta chamada *HtmlDiff* que mostra as alterações entre duas versões de uma página, e uma interface gráfica para visualizar o relacionamento entre as páginas (Douglis et al., 1998). Os usuários interagem com o sistema via formulários HTML, os quais passam os pedidos para um *script* CGI e este realiza operações como salvar e recuperar versões de uma página.

Em relação ao sistema V-Web, a *VersionWeb* apresenta algumas vantagens como: exibe as diferenças entre versões de uma página, permite a edição simultânea dos arquivos por autores e fornece vários recursos relacionados ao controle de versão.

Com relação ao sistema AIDE, a *VersionWeb* tem a vantagem de usar o CVS, o qual possui vantagens e características adicionais ao RCS e dá suporte ao desenvolvimento paralelo entre autores. O AIDE não oferece esse recurso. Além disso, no AIDE, o usuário deve especificar a URL a ser localizada para exibir as diferenças e as versões. Na *VersionWeb*, porém, isso é feito no momento em que o internauta está navegando pela página e ele pode ver as diferenças entre as versões e recuperar uma delas acionando o *link* que dá acesso à ferramenta. Esse *link* pode estar em qualquer lugar da página. Porém, essa página deve estar sob o gerenciamento da *VersionWeb*, ao contrário do que é feito no AIDE.

6 Conclusões

A disponibilização de uma ferramenta para o gerenciamento de versões de arquivos na *Web* com a arquitetura adotada na *VersionWeb* e, além disso, usando o CVS para permitir o acesso simultâneo sobre os arquivos (esses arquivos podem ser páginas *Web*, fontes de programas, documentação de projetos, textos, e outros) visa estimular a colaboração entre os autores que podem estar localizados em diferentes lugares, pois eles não precisam se preocupar tanto em comunicar uns aos outros que vão trabalhar em um determinado arquivo e não precisam necessariamente ter os arquivos locais em suas máquinas.

A abordagem adotada pela *VersionWeb*, de possuir o repositório centralizado no servidor, facilita ainda mais esse processo, pois as versões se tornam mais prontamente disponíveis aos autores à medida que cada

um vai modificando os arquivos e fazendo o *commit*. O fato do CVS não bloquear os arquivos para edição por mais de um autor ao mesmo tempo gera uma produtividade maior entre os integrantes da equipe, pois a espera de liberação de um arquivo para escrita não é necessária e isso com certeza retardaria a obtenção do produto final.

Os recursos de localização das alterações feitas pelo autor, juntamente com data e comentários da versão gerada, dão um certo conforto e agilidade ao processo de desenvolvimento no sentido de que os autores não precisam estar se comunicando e analisando cada linha de código para ver o que foi alterado e quem alterou, pois isso é feito automaticamente. Além disso, os autores dos *sites* não precisam colocá-los "indisponíveis" (fora do ar) enquanto estiverem fazendo modificações ou adaptações. Isso certamente causa menos constrangimento aos navegadores que estão visitando essas páginas, pois têm sempre uma versão disponível.

A princípio pode-se observar três contribuições iniciais deste trabalho: **a)** a viabilização de uma ferramenta de controle de versões na *Web*, através de um ambiente não orientado a linhas de comando (como são a maioria das ferramentas de controle de versões atualmente), indicando claramente que o controle de versões de arquivos através do CVS pode ser feito na *Web*; **b)** o uso de scripts CGI, tecnologia amplamente difundida, conhecida e fácil de usar mostrou ser eficiente e prático no estabelecimento da comunicação e interação dos usuários com o CVS; **c)** o fato dos autores não precisarem "memorizar" os comandos do CVS para manipulação dos arquivos é, com certeza, um outro aspecto positivo para aceitação da ferramenta para o trabalho colaborativo. Em geral, os desenvolvedores de páginas *Web* alegam que seu objetivo principal não é "aprender os comandos CVS" e que o conhecimento dos recursos de desenvolvimento na *Web*, por apresentarem freqüentes inovações, já lhes demandam um alto nível de aprendizado.

Ainda neste contexto, pode-se concluir que, entre as vantagens do controle de versão encontradas em ambientes de engenharia de *software* e ambientes de autoria, uma das mais significativas está relacionada com a reconstrução de versões anteriores de seu trabalho. Outras vantagens como localização das alterações, seus autores, data de alteração, etc., são também muito relevantes.

Com o objetivo de se obter algumas impressões sobre a interação com a ferramenta, foram feitos alguns testes de usabilidade (dado um conjunto pré-definido de tarefas de controle de versão em um formulário para serem realizadas na *VersionWeb*) com 20 usuários do próprio Departamento de Computação do ICMC-USP. Esses usuários, em geral, possuíam nível de conhecimento computacional alto, mas com diferentes níveis de conhecimento sobre controle de versões. Esses usuários são estudantes de Computação, sendo 70% em nível de Mestrado, 15% em nível de Graduação e 15% em nível de Doutorado.

Esse teste de usabilidade, embora tenha sido aplicado a um número relativamente pequeno de usuários, foi muito importante, pois apresentou uma perspectiva de necessidades que não tinham sido consideradas até esta etapa de desenvolvimento da ferramenta. Por outro lado, mostrou também que as funcionalidades estão atendendo às tarefas propostas de forma satisfatória, uma vez que as respostas às solicitações de opinião sobre a ferramenta, em sua maioria, foram relativas à aparência e apresentação das telas. Um aspecto também importante observado foi de que, nas solicitações de opiniões (que o usuário forneceria conforme sua conveniência), 100% dos entrevistados forneceram respostas livres, variadas e indicando aspectos de melhoria da interface (o que de certa forma, representou que a ferramenta tenha despertado interesse).

Como resultado deste trabalho, obteve-se uma publicação nacional no IV Workshop de Teses em Engenharia de Software (ocorrido juntamente com o XIII SBES - Simpósio Brasileiro de Engenharia de Software) em outubro de 1999 (Soares e Fortes, 1999), e uma publicação internacional no IMSA (*International Conference on Internet Multimedia Systems and Applications*) em novembro de 2000 (Soares et al., 2000).

A *VersionWeb* está atualmente disponível para *download* em http://versionweb.sourceforge.net .

6 Referências Bibliográficas

B. Berliner, "CVS II: parallelizing software development", Proceedings of the Winter 1990 USENIX Conference. Washington, DC, January, 1990.

M. Bieliková, "Space-efficient version storage", disponível *on-line* em: wysiwyg://63/http://www.dcs.elf.stuba.sk/~bielik/scm/delta.htm. Visitado em agosto de 1999.

P. Cederqvist, "Version Management with CVS", disponível *on-line* em: ftp://java.icmc.sc.usp.br/library/books/cvs.pdf . Visitado em agosto de 1999.

R. Conradi, B. Westfechtel, "Version Models for Software Configuration Management", *ACM Computing Surveys*, vol. 30, Nº. 2, 1998.

CVS, "Concurrent Versions Systems", disponível *on-line* em: http://www.cyclic.com, http://www.loria.fr/~molli/cvs-index.html. Visitado em agosto de 1999.

F. Douglis, T. Ball, Y.-F. Chen, E. Koutsofios, " The AT&T Internet Difference Engine: Tracking and viewing changes on the web", *World Wide Web*, Volume 1, Nº. 1,27-44, 1998.

T. Hungs, P. Kunz, "UNIX Code Management and Distribution", *Conference on Computing in High Energy Physics*, Annecy, France, September 21-25, 1992.

T. Kilpi, "Product Management Requeriments for SCM Discipline", *Lecture Notes in Computer Science***,** 1235, Springer, 186-200, 1997.

B. Munch, "Versioning", disponível *on-line* em: http://www.idt.unit.no/~bjounmu/thesis/node45.html. Visitado em agosto de 1999.

R. S. Pressman, *Software Engineering*, MacGraw Hill, 3ª edition, 1995.

R. S. Pressman, *Software Engineering*, MacGraw Hill, 4ª edition, 1997.

M. J. Rochkind, "The source code control system", *IEEE Transactions Software Engineering***.** Vol.1, Nº 4, 364-370, 1975.

M. D. Soares, R. P. M. Fortes, "Gerenciamento de Controle de Versões de Páginas Web", XIII Simpósio Brasileiro de Engenharia de Software, IV Workshop de Teses em Engenharia de Software, Florianópolis, Santa Catarina, Brasil, 33-37, outubro de 1999.

M. D. Soares, R. P. M. Fortes, D. A Moreira, "*VersionWeb*: A Tool for Helping Web Pages Version Control". *In: International Conference on Internet Multimedia Systems and Applications (IMSA 2000)*, Proceedings. Las Vegas USA, November 2000. pp.275-280.

I. Sommerville, T. Rodden, P. Rayson, A. Kirby, A. Dix, "Supporting information evolution on the WWW", *World Wide Web*, Vol. 1, Nº 1, 45-54, 1998.

W. F. Tichy, "RCS – A system for version control", *Software Practice and Experience*, Vol. 15, Nº 7, 637-654, 1985.