# Controle de versões - um apoio à edição colaborativa na *Web*

***Sandra Regina Quadros Moraes da Silva***

# Controle de versões - um apoio à edição colaborativa na *Web*

***Sandra Regina Quadros Moraes da Silva***

**Orientadora:** ***Profa. Dra. Renata Pontin de Mattos Fortes***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos
Junho/2005

*Embora ninguém possa voltar atrás*
*e fazer um novo começo, qualquer um*
*pode começar agora e fazer um novo fim.*
*Chico Xavier*

*Feliz aquele que transfere o que sabe*
*e aprende o que ensina.*
*Cora Coralina*

# Dedicatória

Aos meus pais e

a Maurício, companheiro em todos os momentos...

# Agradecimentos

Agradeço à minha orientadora Profa. Dra. Renata Pontin de Mattos Fortes pela paciência e oportunidade.

Agradeço ao meu marido Maurício por todo o apoio e paciência.

Agradeço aos meus pais pelo incentivo ao estudo que sempre me deram.

Agradeço a Deus pela oportunidade de ter feito este curso.

Agradeço a Sandra Freiria, Simone Kobayashi e Paulo Bodnar, pela compreensão e pela colaboração no trabalho na Assessoria de Tecnologia de Informação (ATI) da Universidade Estadual de Londrina (UEL).

Agradeço a Graziela Palombo Cremonezi e Antonio Edson Gonçalves pela licença para capacitação.

Agradeço aos funcionários da secretária de pós-graduação do ICMC-USP – Beth, Laura e Ana Paula, pelo suporte dado no decorrer do curso.

Agradeço a Jacqueline Augusto de Oliveira pela amizade, confiança e estadia em São Carlos.

Agradeço a Elizabeth Leão pela amizade, apoio e incentivo.

Agradeço a Renata Porto pela amizade e pelas caronas.

Agradeço a Flávia Bernardini pela amizade e pelo companheirismo nas aulas de HCI.

Agradeço a Ana Cláudia e Renata Góes pela amizade e pelo companheirismo nas aulas de Engenharia de Software.

Agradeço a Auri Marcelo, Ellen Francine, Silvana e Thais pela amizade.

Agradeço a Débora pela revisão e pela amizade.

Agradeço à Universidade Estadual de Londrina (UEL) pelo apoio financeiro.

E a todos aqueles que, de certa forma, contribuíram para a realização deste trabalho.

# Resumo

SILVA, S.R.Q.M. **Controle de Versões - um Apoio à Edição Colaborativa na Web. 2005**. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação (ICMC), Universidade de São Paulo (USP), São Carlos, SP, 2005.

O controle de versões é uma das principais funcionalidades do Gerenciamento de Configuração de Software (GCS) e visa, entre outras coisas, a recuperação e auditoria (quem, quando e o quê) de versões anteriores e a redução do espaço de armazenamento dos produtos em desenvolvimento. Existem ferramentas que auxiliam esse controle - o CVS é uma delas e tem sido amplamente adotado. Como apoio à edição colaborativa na *Web*, o CVS pode proporcionar benefícios no sentido de recuperar e verificar versões anteriores. Atualmente, ferramentas conhecidas como "wiki", que possibilitam edição colaborativa por meio da *Web*, têm obtido muitos adeptos. Um exemplo desse tipo de ferramenta é a CoTeia, que tem sido utilizada no ambiente acadêmico do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP) desde 2001, e vem sendo utilizada também, integrada à ferramenta DocRationale, usada para registro do *Design Rationale* (DR). Além da edição colaborativa, é possível também o armazenamento de arquivos (*uploads*) associados às páginas *Web* da wiki. A ferramenta DocRationale viabiliza o armazenamento de artefatos relacionados ao processo de desenvolvimento de software, através do *upload* de arquivos. No entanto, o controle de versões desses artefatos na CoTeia não era provido. De fato, não existe um consenso da literatura a respeito do suporte de controle de versões em áreas de *upload* nas wikis. Neste trabalho foi realizado um estudo para análise do uso de controle de versões nas páginas e nos *uploads* em um conjunto de wikis pesquisadas. Já na DocRationale, como os artefatos são alterados durante o processo de desenvolvimento de software, o controle de versões na CoTeia se torna um mecanismo importante. Com isso, foi implementado o controle de versões dos artefatos armazenados na ferramenta DocRationale, através da integração do CVS à CoTeia utilizada na DocRationale.

Palavras-chave: Controle de versões, CVS, wiki, CoTeia, CoWeb, DocRationale.

# Abstract

Versions Control is one of the main activities of Software Configuration Management (SCM) and aims, among other goals, the previous versions retrieval and auditing (who, when and what), and the reduction of storage space required by under development products. There are tools that help this control – CVS is one of these and has been widely adopted. As a support to web collaborative editing, CVS can provide benefits by retrieving and checking previous versions. Nowadays, the tools known as wiki, which allow web collaborative editions, have been gathered many adopters. An example of this kind of tool is CoTeia, that has been used in academic environment at Institute of Mathematics Science and Computing (ICMC) of University of São Paulo (USP) since 2001. CoTeia has also been used integrated to DocRationale tool, which is used to register design rationale. Besides the collaborative editing, CoTeia also permits the file uploads related to wiki webpages. DocRationale makes possible artifacts storage related to software development process, through file uploads. However, versions control of the artifacts in CoTeia was not provided. Indeed, in literature there is not a consensus about the versions control support in uploads wiki area. The present dissertation shows an analysis of versions control usage on pages and uploads areas of a set of selected wikis. On the other hand, in DocRationale, because the artifacts can be changed during all the software development process, the versions control in CoTeia becomes an important mechanism. For this reason, versions control of artifacts stored in DocRationale was implemented, through integration of CVS to CoTeia used in DocRationale.

# Lista de Figuras

# Lista de Gráficos

# Lista de Quadros

# Lista de Siglas

| | |
|---|---|
| **API** | *Application Programming Interface* |
| **ASP** | *Active Server Pages* |
| **BSD** | *Berkeley Software Distribution* |
| **CSCL** | *Computer Supported Cooperative Learning* |
| **CVS** | *Concurrent Versions System* |
| **DR** | *Design Rationale* |
| **DRL** | *Decision Rationale Language* |
| **DTD** | *Document Type Definition* |
| **GC** | Gerenciamento de Configuração |
| **GCS** | Gerenciamento de Configuração de Software |
| **GPL** | GNU *General Public License* |
| **GUI** | *Graphical User Interface* |
| **HTML** | *HyperText Markup Language* |
| **IBIS** | *Issue-Based Information System* |
| **ICMC** | Instituto de Ciências Matemáticas e de Computação |
| **KDE** | K *Desktop Environment* |
| **NFS** | *Network File System* |
| **PHI** | *Procedural Hierarchy of Issues* |
| **PHP** | *Hypertext Preprocessor* |
| **QOC** | *Questions*, *Options and Criteria* |
| **RCS** | *Revision Control System* |
| **SAFE** | *Software Engineering Available For Everyone* |
| **SEI** | *Software Engineering Institute* |
| **SPICE** | *Software Process Improvement and Capability dEtermination* |
| **TCL** | *Tool Command Language* |
| **URL** | *Uniform Resource Locator* |
| **USP** | Universidade de São Paulo |
| **XHTML** | *Extensible Hypertext Markup Language* |
| **XML** | *Extensible Markup Language* |
| **XSLT** | *Extensible Style Language Transformation* |

# SUMÁRIO

# Capítulo 1

## Introdução

O Gerenciamento de Configuração de Software (GCS) tem por objetivo colaborar para a melhoria do processo de software, sendo considerado uma das atividades de garantia de qualidade de software, e fazendo parte de modelos de referência para a melhoria do processo como o CMM (*Capability Maturity Model*) (PAULK et al., 1993) e o CMMI (*Capability Maturity Model Integration*) (CMMI, 2002), mantidos pelo *Software Engineering Institute* (SEI), da *Carnegie Mellon University*.

Como uma das atividades principais do GCS, o Controle de Versões visa controlar a evolução dos sistemas de software, possibilitando principalmente a recuperação de versões anteriores, o desenvolvimento paralelo e a auditoria do desenvolvimento (quem, quando, o quê). Uma versão é uma instância do software que difere de outras instâncias em virtude de diferentes funcionalidades e desempenhos, além da resolução de defeitos (SOMMERVILLE, 2003).

Existem várias ferramentas desenvolvidas atualmente para o controle de versões e, entre elas é possível citar o RCS, o CVS, *Arch* e *Subversion*. Dentre as ferramentas citadas, o RCS é a mais antiga, usada como base para o desenvolvimento de outras e o CVS tem sido a mais utilizada segundo Reis (2003).

Nos ambientes de trabalho de autoria colaborativa, em que várias pessoas são responsáveis pela manutenção de um mesmo conteúdo, o controle de versões é importante, pois possibilita a auditoria de quem e quando fez quais alterações. Um exemplo de ambiente de autoria colaborativa, em uso no momento, é a categoria de ferramentas conhecida como wiki, um *website* que permite que o conteúdo de suas páginas possa ser editado e atualizado por várias pessoas. Além disso, em algumas wikis é disponibilizado o recurso de *upload*, para que arquivos também possam ser armazenados e relacionados às páginas (CUNNINGHAM, 2004).

A CoTeia (ARRUDA JR; PIMENTEL, 2001; ARRUDA JR, PIMENTEL, IZEKI, 2002) é uma wiki utilizada no ambiente acadêmico do ICMC-USP, em São Carlos, desenvolvida com base na wiki CoWeb (*Georgia Tech*) (GUZDIAL et al., 2001; GUZDIAL, RICK; KEHOE, 2001; RICK et al., 2001, 2002). Além do ambiente acadêmico, foi desenvolvida uma versão da CoTeia para integração à ferramenta DocRationale (FRANCISCO, 2004), utilizada no registro de *Design Rationale* (DR). Como a DocRationale se propõe também a armazenar os artefatos relacionados ao DR, estes são

armazenados sob a forma de arquivos, usando o recurso de *upload* da CoTeia. Tais artefatos, por estarem sujeitos a alterações durante o processo de desenvolvimento de software, requerem controle de suas versões. No entanto, na bibliografia nas diversas wikis atualmente disponíveis, não existe um consenso a respeito do suporte de controle de versões no recurso de *upload*.

Neste trabalho foi realizado um estudo para análise do uso de controle de versões nas páginas e nos *uploads* em um conjunto de wikis pesquisadas. Além disso, foi implementado o controle de versões dos artefatos armazenados na ferramenta DocRationale, através da integração do CVS à versão da CoTeia utilizada na DocRationale.

## 1.1. Objetivos da Dissertação

O objetivo deste trabalho é a análise e identificação do panorama atual das wikis por meio de levantamento de dados, com o propósito de verificar a utilização de controle de versões neste tipo de ferramenta. Além disso, foi implantado o controle de versões como auxílio à edição colaborativa na versão da CoTeia que foi integrada à ferramenta DocRationale. Tendo em vista o estudo sobre controle de versões e as ferramentas construídas para esse fim (RCS, CVS, *Arch*, *Subversion*, entre outras), procurou-se selecionar uma ferramenta de controle de versões conhecida e de bastante aceitação, no caso o CVS, para o desenvolvimento do trabalho. Procurou-se também, através de levantamento, selecionar uma ferramenta para visualização gráfica do repositório CVS, no caso, a ferramenta *ViewCVS*, a qual foi integrada à CoTeia.

Pode-se dizer que o controle de versões implantado visa proporcionar o controle das alterações realizadas nos artefatos (gerados durante o processo de desenvolvimento do software) e armazenados na CoTeia por meio da DocRationale.

## 1.2. Organização da Dissertação

A organização deste documento apresenta-se conforme segue:

O Capítulo 2 apresenta a revisão bibliográfica sobre garantia de qualidade, Gerenciamento de Configuração de Software (GCS) e controle de versões. Nesse capítulo encontram-se as definições e

os benefícios da garantia de qualidade, do GCS e do controle de versões.

O Capítulo 3 contém o estado da arte no que diz respeito ao suporte ao CVS (ferramentas de visualização de repositório) e no que diz respeito ao trabalho colaborativo na Web (wikis, CoWeb e CoTeia). Neste capítulo é apresentado o conceito de *Design Rationale* (DR) e a ferramenta DocRationale, desenvolvida com o propósito de fazer o registro de DR e armazenamento dos artefatos relacionados.

O Capítulo 4 contém uma análise de dados sobre wikis, coletados em levantamento, e o desenvolvimento das implementações resultantes deste trabalho de mestrado.

O Capítulo 5 mostra as contribuições deste trabalho e os trabalhos futuros. No Apêndice A é apresentado um quadro de wikis, resultante do levantamento de dados desenvolvido apresentado no Capítulo 4.

# Capítulo 2

# Revisão Bibliográfica

Neste capítulo são apresentados os conceitos relacionados ao controle de versões, uma vez que não é um procedimento isolado, fazendo parte de um conceito maior, que é a garantia de qualidade de software, tão discutida atualmente. Assim, os modelos de referência para a melhoria de processo têm como atividade importante o Gerenciamento de Configuração de Software e, por conseguinte o controle de versões. Inicialmente, na seção 2.1 são definidos os conceitos de Processo de Software, Garantia de Qualidade e alguns modelos de referência para a melhoria do processo. Em seguida, na seção 2.2 é definido o conceito de Gerenciamento de Configuração de Software (GCS) e conceitos associados. Na seção 2.3 é definido o conceito de Controle de Versões e são citadas algumas ferramentas disponíveis atualmente. Finalmente, na seção 2.4 são feitas as considerações finais sobre este capítulo.

## 2.1. Processo de Software e Garantia de Qualidade

O processo de software pode ser definido como um arcabouço para as tarefas que são necessárias à construção de um produto de software de qualidade, com a geração de produtos de trabalho (artefatos), por exemplo, programas de computador, documentos e dados. O processo de software define a maneira na qual o desenvolvimento de software é organizado, gerenciado, medido, executado e melhorado. Embora possam apresentar diferentes níveis de sofisticação na administração de seus processos, todas as organizações envolvidas com desenvolvimento de software seguem algum tipo de processo (implícito ou explícito, reproduzível, instrumentado, adaptável, etc). O processo de software consiste de dois processos inter-relacionados: (i) o processo de produção, que lida com a produção e manutenção do produto e (ii) o processo de gerenciamento que, além de fornecer os recursos necessários para o processo de produção, também o controla (MONTANGERO et al., 1999).

Em uma organização ou domínio de aplicação, os processos de diferentes projetos tendem a seguir padrões. O estudo dos processos de produção de software tem levado ao desenvolvimento de várias abordagens de ciclo de vida de software (cascata, evolucionário, espiral, etc), cada uma com suas características. As funções primárias do ciclo de vida de um software são: (i) determinar as

dependências entre as fases envolvidas no desenvolvimento e evolução do software, e (ii) estabelecer critérios para passar de uma fase a outra, o que inclui os critérios de conclusão da fase atual e os critérios de passagem para a próxima fase. Esses modelos de ciclo de vida podem ajudar engenheiros de software e gerentes a compreender o processo de software, e determinar a ordem das atividades envolvidas na produção de software (MONTANGERO et al., 1999). Martín-Vivaldi e Isacsson (1998) colocam que quanto mais crítica a função do software, maior será o ganho com o foco no processo e na compreensão do mesmo.

O uso de modelos de processos depende fortemente da consciência que a organização tem em relação ao que representa. O processo é parte da memória da organização e por meio disso, uma base para o aprendizado da organização. O processo também é um veículo para prover os produtos com um nível esperado ou mesmo previsto de confiabilidade (MARTÍN-VIVALDI; ISACSSON, 1998).

Um problema fundamental das organizações é a falta de habilidade em gerenciar o processo de software, uma vez que os projetos freqüentemente atrasam e ficam fora do orçamento (PAULK et al., 1993). As organizações têm descoberto que o caminho para a entrega bem-sucedida do software (no tempo certo, dentro do orçamento e com a qualidade esperada) tem como base o gerenciamento efetivo do processo de software. O foco principal é a introdução de passos de garantia de qualidade no processo de desenvolvimento o mais cedo possível, visando colaborar com o teste final (MARTÍN-VIVALDI; ISACSSON, 1998).

Pressman (2002) define qualidade de software como a “conformidade com requisitos funcionais e de desempenho explicitamente declarados, padrões de desenvolvimento explicitamente documentados e características implícitas, que são esperadas em todo software desenvolvido profissionalmente...” e define garantia de qualidade como um padrão planejado e sistemático de ações, necessárias para se obter a qualidade do software.

A qualidade não está relacionada somente ao software, mas também à organização e ao processo de produção utilizado (MONTANGERO et al., 1999). Melhorar a qualidade do software e do processo possibilita que a organização possa se tornar competitiva na indústria de software. A qualidade também contribui para melhorar a produtividade nas organizações de software, porque pode reduzir drasticamente o custo com trabalho refeito e defeitos causados pela baixa qualidade. Existe uma estreita relação entre a qualidade do software e os processos que são usados, por isso o ponto principal é verificar como o software está sendo desenvolvido e mantido, a fim de que seja possível estabelecer os planos para melhorar a qualidade do produto de software como resultado da melhoria dos processos de software em uso (GRADY, 1997 apud VISCONTI; VILLARROEL, 2002).

A adesão às práticas de gerenciamento de garantia de qualidade (que devem ser aplicadas na organização) requer a definição e melhoria dos processos de desenvolvimento, essenciais para alcançar produtos com qualidade. Várias abordagens têm sido desenvolvidas para melhorar a qualidade do software e a capacidade de processo de uma organização - a melhoria do processo de software tem levado as organizações a avaliarem os processos de software, a fim de encontrar formas para melhorar a capacidade dos mesmos (HWANG, 2004).

A fim de garantir que o que foi gerado pelo processo de software está correto, existem mecanismos de avaliação que permitem às organizações determinar a maturidade do processo (PRESSMAN, 2002) - uma avaliação geralmente é baseada em um modelo de referência para a melhoria do processo, entre os quais é possível destacar o *Capability Maturity Model* – CMM (PAULK et al., 1993), o *Capability Maturity Model Integration* - CMMI (CMMI, 2002) e a ISO/IEC 15504 (projeto SPICE - *Software Process Improvement and Capability dEtermination*) (ISO/IEC 15504, 2005). Estes padrões e modelos são apresentados para identificar as melhores práticas de qualidade para definir, melhorar e avaliar os processos de desenvolvimento de software (MOURA et al., 2003).

Os modelos de referência para a melhoria do processo, a seguir destacados – CMM e CMMI, foram estudados por representarem as alternativas que têm sido mais adotadas para que se obtenha garantia de qualidade no processo de desenvolvimento de software.

### 2.1.1. CMM

No modelo *Capability Maturity Model* - CMM (PAULK et al., 1993) são descritos princípios e práticas básicas da maturidade do processo de software. É apresentado um conjunto de práticas recomendadas em áreas-chave de processo, com o objetivo de melhorar a capacidade do processo de software, com base no conhecimento adquirido das avaliações do processo de software e do *feedback* obtido na indústria e no governo. O modelo CMM fornece às organizações de software um guia para obtenção de controle nos processos de desenvolvimento e manutenção de software e como evoluir para uma cultura de excelência em engenharia e gerenciamento de software. O CMM foi projetado para guiar as organizações na seleção de estratégias de melhoria de processo, através da determinação do processo de maturidade atual e da identificação das questões mais críticas para a qualidade de software e melhoria do processo.

No CMM existe diferença entre organizações de software maduras e imaturas. Na organização imatura

os processos de software são geralmente improvisados pelos profissionais e pela gerência no decorrer do projeto, e mesmo que um processo de software tenha sido especificado, não é rigorosamente seguido ou aplicado. Os gerentes focam em resolver crises imediatas. Cronogramas e orçamentos são excedidos com freqüência porque não foram baseados em estimativas reais. Quando prazos finais são duramente impostos, a qualidade e funcionalidade do produto são freqüentemente comprometidas para cumprir o cronograma. Na organização madura, existe a habilidade para gerência dos processos de desenvolvimento e manutenção de software. O processo de software é precisamente comunicado ao pessoal existente e a novos empregados, e as atividades são desenvolvidas de acordo com o processo planejado. Os gerentes monitoram a qualidade dos produtos de software e dos processos responsáveis por sua produção. Orçamentos e cronogramas são baseados na performance histórica e são realistas. Os resultados esperados para custo, cronograma, funcionalidade e qualidade do produto são normalmente alcançados.

No CMM existem alguns conceitos fundamentais que envolvem a maturidade do processo:

- **Processo de software**: pode ser definido como um conjunto de atividades, métodos, práticas e transformações, usados para desenvolver e manter o software e os produtos associados (por exemplo, planos e documentos do projeto, código-fonte, casos de teste e manuais do usuário). À medida que uma organização amadurece, o processo de software é melhor definido e é implementado de forma mais consistente na organização;
- **Capacidade do processo de software**: descreve os resultados esperados que podem ser alcançados quando um processo de software é seguido. A capacidade do processo de software possibilita que a organização tenha possibilidade de prever os resultados esperados para o próximo projeto de software;
- **Desempenho do processo de software**: representa os resultados reais atingidos quando um processo de software é seguido;
- **Maturidade do processo de software**: um processo específico é explicitamente definido, gerenciado, medido, controlado e eficaz. Implica em um potencial para crescimento na capacidade e indica a riqueza do processo de software de uma organização e a consistência com a qual é aplicado nos projetos da organização. À medida que a organização progride na maturidade do processo de software, ela institucionaliza o processo de software por meio de políticas, padrões e estruturas organizacionais;
- **Nível de maturidade**: é utilizada uma escala (de 1 a 5) para medir a maturidade do processo de software e para avaliar a capacidade do processo de software. Os níveis também ajudam a organização a priorizar os esforços de melhoria. Cada nível contém um conjunto de metas que, uma vez satisfeitas, fixam um importante componente do processo de software, resultando em um aumento na capacidade do processo da organização. Os níveis de maturidade 2 a 5 podem ser caracterizados por meio das atividades feitas pela organização para estabelecer ou melhorar o

processo de software, pelas atividades feitas em cada projeto e pela capacidade do processo resultante entre os processos. Uma caracterização comportamental do nível 1 é incluída para estabelecer uma base de comparação para melhorias de processo em níveis maiores de maturidade: **nível 1 – Inicial** (a organização não provê um ambiente estável para o desenvolvimento e manutenção de software, uma vez que o sucesso depende da competência dos funcionários e não pode ser repetido, a menos que as mesmas pessoas sejam atribuídas ao próximo projeto); **nível 2 – Repetível** (estabelecimento de políticas para gerenciar um projeto de software e de procedimentos para implementá-las, sendo que o planejamento e gerência de novos projetos são baseados na experiência em projetos similares); **nível 3 – Definido** (o processo padrão para desenvolvimento e manutenção de software na organização é documentado, incluindo processos de engenharia de software e gerenciamento, os quais são integrados de uma forma coerente); **nível 4 – Gerenciado** (neste nível a organização define os objetivos de qualidade para os produtos e processos de software, sendo que produtividade e qualidade são medidas para atividades do processo de software importantes em todos os projetos, como parte de um programa de medida organizacional) e **nível 5 - em Otimização** (a organização inteira foca na melhoria contínua do processo, pois tem os meios para identificar fraquezas e fortalecer o processo, a fim de prevenir a ocorrência de defeitos). A tentativa de pular níveis não é indicada, uma vez que cada nível de maturidade funciona como fundação necessária para alcançar o próximo nível.

Cada nível de maturidade foi decomposto em partes (com exceção do nível 1) - as áreas-chave de processo, onde cada uma identifica um conjunto de atividades relacionadas que, quando executadas, atingem um conjunto de objetivos considerados importantes no alcance da capacidade do processo. O caminho para atingir os objetivos de uma área-chave de processo pode diferir entre projetos, baseado nas diferenças em domínios de aplicação ou ambientes, entretanto, todas as metas de uma área-chave de processo devem ser atingidas pela organização para que a área-chave de processo possa ser satisfeita. As áreas-chave de processo podem ser consideradas como requisitos para atingir o nível de maturidade, assim, todas devem ser satisfeitas para cada nível.

No nível 2 do CMM, as áreas-chave de processo focam nos interesses do projeto de software, relacionados ao estabelecimento de controles básicos de gerenciamento do projeto. Uma dessas áreas-chave é o Gerenciamento de Configuração de Software (GCS), que visa estabelecer e manter a integridade dos produtos do projeto durante o ciclo de vida do projeto de software.

### 2.1.2. CMMI

No modelo *Capability Maturity Model Integration* - CMMI são descritos os princípios e práticas básicas da maturidade do processo de software (CMMI, 2002). O CMMI foi liberado pelo *Software Engineering Institute* (SEI) em janeiro de 2002, e tem por objetivo fornecer um guia para melhorar os processos da organização e a habilidade para gerenciar o desenvolvimento, aquisição e manutenção de seus produtos de software. Além do mais, o modelo CMMI pode ser útil para avaliar a maturidade organizacional ou capacidade de área de processo, estabelecendo prioridades para melhoria e implementando essas melhorias (WESZKA, 2000 apud CHANG; CHEN, 2004).

A meta é um dos componentes do modelo CMMI, e representa um estado final desejado, a conquista de algo que indica que um certo grau de controle do projeto e do processo foi atingido. Quando uma meta é exclusiva de uma única área de processo, é chamada "meta específica". Quando uma meta pode ser aplicada em outras áreas de processo, é chamada "meta genérica".

Nos modelos CMMI, as áreas de processo descrevem aspectos-chave dos processos como gerenciamento de requisitos, gerenciamento de configuração, verificação, validação e outros. Uma área de processo fornece uma lista de práticas requeridas para atingir as metas pretendidas, mas não descreve como um processo efetivo é executado (por exemplo, critérios de entrada e saída, funções de participantes, recursos).

Existem dois tipos de representações do modelo CMMI: ***staged***, que usa conjuntos pré-definidos de áreas de processo para definir um caminho de melhoria para a organização, descrito pelo nível de maturidade (um platô evolucionário para atingir a melhoria dos processos organizacionais); e ***continuous***, que permite que uma organização selecione uma área de processo específica para que seja melhorada, usando níveis de capacidade para caracterizar a melhoria relativa a uma área de processo individual. É necessário analisar as vantagens e desvantagens de cada representação para decidir qual modelo CMMI melhor se enquadra nas necessidades de melhoria do processo da organização (DIKENELLLI, 1998; HOYLE, 2001 apud CHANG; CHEN, 2004).

Para que uma organização seja bem sucedida na melhoria de processo, a integridade dos itens de configuração no processo de gerenciamento de configuração deve ser assegurada através da identificação dos itens de configuração e controle das mudanças nos itens. Com o processo de GCS definido, espera-se reduzir o custo e o tempo de entrega, objetivos básicos no desenvolvimento de software (HWANG, 2004).

## 2.2. Gerenciamento de Configuração de Software

O Gerenciamento de Configuração de Software (GCS) é considerado uma atividade de garantia de qualidade de software, aplicada ao longo de todo o processo de software (PRESSMAN, 2002) e pode ser encontrado nos modelos de referência para a melhoria do processo, como parte dos requisitos para a obtenção da certificação (MIDHA, 1997): no CMM, o GCS é uma das áreas-chave de processo para ir do nível 1 (inicial) para o nível 2 (repetível) (ASKLUND et al., 2001; CONRADI; WESTFECHTEL, 1998); no CMMI, é uma das áreas de processo de Suporte (CMMI, 2002) e na ISO/IEC 15504 é um dos processos da categoria de Suporte (SALVIANO, 2001).

O aumento da complexidade dos produtos de software e dos ambientes de desenvolvimento ressaltou a necessidade de adoção do GCS nos projetos de desenvolvimento de software (ASKLUND et al., 2001; MIDHA, 1997), pois à medida que muitas versões do software são criadas (devido às modificações realizadas) surge a necessidade de gerenciamento das novas versões que são geradas. É possível ter várias versões em desenvolvimento e em uso ao mesmo tempo, e por isso faz-se necessário manter o controle das mudanças implementadas e de como essas mudanças foram incluídas no software (TICHY, 1985). É possível também ter várias configurações do produto de software, devido a diferentes computadores, diferentes sistemas operacionais, funções específicas para o cliente, etc (SOMMERVILLE, 2003). Além disso, é cada vez mais comum uma equipe de desenvolvedores dispersa geograficamente, trabalhar em um mesmo projeto de software (ASKLUND et al., 2001). A perda de controle tem várias conseqüências, entre elas, problemas de interface na integração de módulos, reaparecimento de erros removidos anteriormente ou desaparecimento de funcionalidades acrescentadas após as modificações, dificuldade em identificar a última versão do produto e perda dos arquivos-fonte que compõem o sistema distribuído ao cliente. Isso leva a altos custos de desenvolvimento, baixa qualidade do produto e insatisfação do cliente ou usuário final.

O GCS é um elemento-chave do processo de software e propõe-se a minimizar os problemas surgidos durante o ciclo de vida do software através de controle sistemático sobre as modificações, a fim de evitar ou minimizar as suas decorrências negativas (DART, 1990; FEILER, 1991). Segundo Babich (1986 apud PRESSMAN, 2002), o gerenciamento de configuração de software é "a arte de coordenar o desenvolvimento de software para minimizar a confusão". É uma disciplina de natureza técnica e gerencial, que visa evitar as alterações descontroladas em um produto de software, através de um conjunto de atividades desenvolvidas, com a finalidade de administrar as alterações durante o ciclo de vida do software, com ênfase na fase de desenvolvimento (ASKLUND et al., 2001; MIDHA, 1997).

O GCS envolve um processo e um produto: o processo representa a sequência de tarefas necessárias (plano) para efetuar o GCS, definindo o que precisa ser feito, quem faz e como será feito; o produto é o resultado do processo de GCS. Um sistema de GCS deve prover funcionalidade para os dois aspectos. O GCS pode ser uma combinação de procedimentos manuais e automatizados, sendo possível sua implantação sem qualquer tipo de assistência automatizada, embora a meta seja automatizá-lo o máximo possível (DART, 1990). Os custos e benefícios do investimento em GCS devem ser avaliados criteriosamente, uma vez que o valor do investimento depende do tipo e valor do software a ser controlado e principalmente, de quanto a função do software é crítica.

Entre os benefícios proporcionados pelo GCS, os principais são: (i) facilidade para acomodar mudanças; (ii) possibilidade de maior controle sobre os produtos; (iii) economia de tempo no desenvolvimento; (iv) facilidade na geração de versões diferentes de um mesmo produto de software; (v) manutenção do histórico do software e (vi) facilidade de recuperar versões anteriores.

No que diz respeito aos custos, são necessários dois tipos de investimentos (OLIVEIRA et al., 2001): (i) a adoção, que envolve a aquisição de informações, consultorias e treinamento do pessoal envolvido com a adoção de GCS e (ii) a implementação, que engloba os custos com o pessoal envolvido na implementação do GCS, os recursos computacionais que serão utilizados e ferramentas de software para automação do processo.

São vários os impedimentos na adoção do GCS (OLIVEIRA et al., 2001), entre eles: (i) o custo; (ii) a resistência das pessoas às mudanças; (iii) a falta de costume em seguir regras ou padrões; (iv) o aumento na formalidade e burocracia devido aos procedimentos que devem ser seguidos; (v) a falta de conhecimento sobre o assunto e (vi) a ausência de comprometimento ou o comprometimento insuficiente da gerência superior da empresa.

O GCS tem origem em dois grupos, com diferentes necessidades - a gerência e o desenvolvimento. A gerência foi a abordagem que primeiro foi notada, pois queria ter mais controle de medições em cima do desenvolvimento, e em particular, sobre os *releases* dos produtos. Nesse caso, as rotinas eram manuais e frequentemente gerenciadas por um bibliotecário. No desenvolvimento, existia a necessidade de automação com ferramentas sofisticadas, permitindo que os desenvolvedores pudessem ser mais eficientes e mais seguros do que estivesse acontecendo em um projeto (ASKLUND et al., 2001). Sob a perspectiva de gerência, o GCS direciona e controla o desenvolvimento de um produto através da identificação dos componentes do produto e controle das contínuas mudanças. O objetivo é documentar e disponibilizar a composição de um determinado produto e seus componentes, com a garantia de que a base correta esteja sendo usada e que a composição certa do produto esteja sendo feita.

Asklund et al. (2001) coloca que os padrões de GCS e muitos livros que abordam o assunto definem GCS como consistindo de quatro atividades:

- **Identificação da configuração**: determinação da estrutura do produto, seleção dos itens de configuração, documentação das características físicas e funcionais dos itens, incluindo interfaces e mudanças subseqüentes e alocação de caracteres ou número de identificação para os itens de configuração e seus documentos;
- **Controle da configuração**: controle de mudanças de um item de configuração depois do estabelecimento formal de seus documentos. Inclui a avaliação, coordenação, aprovação/desaprovação e implementação de mudanças;
- **Relato do *status* de configuração**: registro formalizado e relatório dos documentos de configuração estabelecidos, o *status* das mudanças propostas e o *status* de implementação de mudanças aprovadas;
- **Auditoria de configuração**: determina se um item de configuração está de acordo com os documentos de configuração, garantindo que a modificação foi adequadamente implementada (PRESSMAN, 2002). Pode ser: (i) funcional, que verifica se um item de configuração atingiu as características de performance e funções definidas no documento de configuração, ou (ii) física, que faz uma avaliação formal para verificar a conformidade entre o item de configuração produzido de fato e a configuração, de acordo com os documentos de configuração (ASKLUND et al., 2001; ROZANTE, 2003).

Os procedimentos de GCS são constituídos pelo registro e processamento das mudanças propostas para o sistema e como relacioná-las aos componentes do sistema e aos métodos utilizados para identificar diferentes versões do sistema. Esses procedimentos são baseados em padrões, por exemplo, o IEEE 828-1998, que é um padrão para planos de gerenciamento de configuração (SOMMERVILLE, 2003). Na organização esses padrões são publicados em um manual de GCS ou como parte de um manual de qualidade. Um plano de gerenciamento de configuração descreve padrões e procedimentos que devem ser utilizados para um efetivo GCS, sendo que o ponto de partida para desenvolver o plano é um conjunto de padrões gerais de Gerenciamento de Configuração (GC) para toda a empresa, os quais devem ser adaptados, conforme necessário, a cada projeto específico. A parte mais importante do plano é a definição de responsabilidades: quem será responsável pela entrega de cada documento ou componente do produto para o gerenciamento da garantia de qualidade e configuração e quem serão os revisores de cada documento. O plano deve ser apresentado em capítulos e apresentar as seguintes informações:

- Identificação dos itens de configuração, definindo quais entidades serão gerenciadas (planos de projeto, especificação do projeto, programas, conjunto de dados de teste e outros documentos que possam ser úteis à futura manutenção do sistema);

- Declaração de quem assume a responsabilidade pelos procedimentos de GCS e pela submissão de entidades controladas à equipe de GCS;
- Política de GCS para o controle de qualidade e o controle de versões;
- Descrição dos registros de procedimentos de GCS que deverão ser mantidos;
- Descrição das ferramentas usadas pelo GCS e o processo que deverá ser aplicado quando essas ferramentas forem usadas;
- Definição do banco de dados de configuração, usado para registrar as informações sobre a configuração. As principais funções do banco de dados segundo Sommerville (2003), seriam ajudar na avaliação do impacto das mudanças ocorridas no sistema e fornecer informações gerenciais sobre o processo de GCS. Tem que ser capaz de fornecer respostas a consultas do tipo "Que hardware e que configuração de sistema operacional são necessárias para executar uma versão específica do sistema?"ou "Quantas versões do sistema foram criadas e quais foram as datas de sua criação?". O ideal seria que esse banco de dados fosse integrado ao sistema de controle de versões, possibilitando vincular diretamente as mudanças aos documentos e aos componentes afetados por elas. Os itens de configuração podem ser alterados sem passar pelo banco de dados de configuração;
- Outras informações de GCS, como software de fornecedores externos e de procedimentos de auditoria para o processo de GCS.

Sob a perspectiva do desenvolvedor, o GCS mantém os componentes atuais dos produtos, armazena sua história, oferece um ambiente de desenvolvimento estável e coordena mudanças simultâneas no produto. Os aspectos relevantes para o desenvolvedor são, de acordo com Asklund et al. (2001) e Midha (1997):

1) **Controle de versões**: usado no controle da evolução dos artefatos (será visto com mais detalhes na seção 2.3);
2) **Seleção de configuração**: em situações em que é grande o número de versões de arquivos, nem sempre é óbvio qual versão escolher, pois o número de combinações pode ser grande. É preciso assegurar que haja uma seleção e configuração consistentes, em termos de inclusão de versões com modificações interligadas. A disponibilidade de um mecanismo de seleção baseado em regras é uma técnica útil para a especificação de uma configuração suportada por vários sistemas. Uma configuração parcialmente definida é um sistema construído com base em uma regra, que especifica a "última configuração", pois as versões exatas que são incluídas irão variar no tempo. Uma configuração definida é um sistema construído sem tal regra, adequado para entregas do software, pois as versões de todos os arquivos incluídos são fixas, garantindo que o sistema possa ser recriado. Através dessa configuração é possível formar uma *baseline* ou um *release*;

3) **Desenvolvimento concorrente**: um sistema de GCS permite que as equipes trabalhem em paralelo, seja trabalhando no mesmo arquivo, corrigindo vários *bugs* ou um desenvolvedor trabalhando no último *release*, enquanto outro corrige um *bug* em outro *release*. Isso é possibilitado através da seleção de versões, construindo-se configurações específicas para diferentes necessidades, através de um modelo de sincronização de mudanças concorrentes. Quando o desenvolvimento é distribuído ou remoto, onde os desenvolvedores estão geograficamente dispersos, embora trabalhando no mesmo sistema, é feito uso de repositórios replicados, que são cópias sincronizadas do mesmo repositório;

4) **Gerenciamento de construção**: envolve um procedimento de construção de um conjunto inteiro ou subconjunto de uma versão de um produto de software, a partir da configuração selecionada dos componentes do produto, facilitando o armazenamento do registro de uma versão construída, o ambiente de construção e as versões dos componentes selecionados para uma configuração (MIDHA, 1997). Essa construção é feita através da busca do código-fonte para um *release* em particular e do uso de ferramentas de construção, como por exemplo, a ferramenta Make (FELDMAN; 1979), utilizadas para criar uma configuração automaticamente. A ferramenta Make verifica as dependências entre códigos-fonte no momento da construção e assegura que o código-fonte dependente seja construído na ordem correta;

5) **Gerenciamento de *releases***: O gerenciamento de *releases* consiste na identificação e organização de tudo o que tem que ser entregue (documentos, executáveis, bibliotecas, etc) em um *release* de produto. O suporte ao *release* torna isso possível, a fim de controlar quais usuários têm quais versões de quais componentes e conseqüentemente, assegurar quais serão afetados por uma mudança em particular. A criação de kits de instalação é apropriada, pois facilita a tarefa do gerente de construção, responsável por providenciar o produto embalado com as características e configurações corretas. Existem softwares específicos para a criação de kits de instalação, como por exemplo, o InstallShield[1] ou o Windows Installer[2] da Microsoft;

6) **Gerenciamento da área de trabalho**: implantar o GCS em uma organização é incômodo sem o suporte eficiente de ferramentas. A mudança da cultura existente requer educação massificada, suporte e motivação através do suporte para ferramentas integradas ao ambiente de desenvolvimento. Através do gerenciamento da área de trabalho, os desenvolvedores podem trabalhar de forma transparente, sob o controle do GCS na área de trabalho, mesmo que estejam trabalhando em algum programa em particular. Quando vários programadores trabalham

[1] http://www.installshield.com.
[2] http://www.microsoft.com/downloads/details.aspx?FamilyID=4b6140f9-2d36-4977-fa1-f8a0f5dca8f&displaylang=en.

concorrentemente em suas áreas de trabalho, é necessário o controle entre diferentes cópias do mesmo objeto;

7) **Gerenciamento de mudanças**: as razões para as mudanças podem ser múltiplas, complexas e procedentes de várias fontes. O gerenciamento de mudanças trata de todas as mudanças requeridas em um sistema, em razão de erros, melhoria ou adição de funcionalidade. Os procedimentos do gerenciamento de mudanças foram concebidos para assegurar que os custos e benefícios das mudanças sejam adequadamente analisados, de forma que as mesmas possam ser feitas de maneira controlada em um sistema. Esse gerenciamento é possibilitado através do uso de ferramentas separadas, integradas à uma ferramenta principal de GCS (SOMMERVILLE, 2003). Dois principais objetivos devem ser atingidos com o gerenciamento de mudanças: (i) o fornecimento de um processo no qual as solicitações de mudanças são priorizadas e as decisões para implementar ou rejeitar são feitas e (ii) a possibilidade de consultar todas as solicitações ativas e implementadas e rastrear todas as mudanças realmente feitas.

Em Kandt (2002) são identificados alguns princípios fundamentais para uma gerência bem-sucedida da configuração do software desenvolvido e práticas que melhor os incluem. As práticas são resultado de exame próprio e através do exame explícito das melhores práticas definidas (PAULK et al., 1994; BABICH, 1986b; JONES, 2000 apud KANDT, 2002) e outras práticas definidas implicitamente por processos e procedimentos de GCS (BERSOFF, 1980; WHITGIFT, 1991; BERLACK, 1992; BUCKLEY, 1996 apud KANDT, 2002) e avaliações de ferramentas (RIGG, BURROWS; INGRAM, 1995 apud KANDT, 2002). Conseqüentemente, muitas destas práticas são bem conhecidas, enquanto outras são novas ou não são freqüentes. Além disso, Kandt (2002) cita que existem três propósitos secundários para identificar os princípios e práticas de gerenciamento de configuração: (i) as organizações podem usar estes princípios e práticas para treinar especialistas em GCS e avaliar as práticas de GCS utilizadas; (ii) estes princípios e práticas podem ser usados para definir um conjunto central de práticas de GCS como *baseline* para um esforço de melhoria do processo de software e (iii) a identificação destes princípios e práticas pode servir como motivação para explorar oportunidades de automação dessas práticas, assim como também para a automação de sua verificação.

Os dez princípios básicos que auxiliam as atividades de GCS são (KANDT, 2002):

- **Princípio 1 - Proteger dados críticos e outros recursos:** os artefatos e suas revisões, produzidos durante o processo de desenvolvimento de software, devem ser mantidos de forma confiável, sempre acessíveis aos usuários e de fácil recuperação em caso de falha, pois a perda dos mesmos pode prejudicar a organização (por exemplo, prejuízo financeiro);

- **Princípio 2 - Monitorar e controlar os processos e procedimentos de desenvolvimento de**

**software:** a organização deve monitorar os usuários para assegurar que sigam os processos e procedimentos de desenvolvimento de software que usa na produção de artefatos;

- **Princípio 3 - Automatizar processos e procedimentos quando o custo for efetivo:** a automação de processos e procedimentos tem dois benefícios primários: (i) garante que uma organização os aplique de forma consistente, indicando ser mais apta a produzir produtos de qualidade; e (ii) a automação melhora a produtividade das pessoas que devem executar os processos e procedimentos, uma vez que reduz as tarefas que devem ser feitas e possibilita a execução de mais trabalhos;

- **Princípio 4 - Conceder valor aos clientes:** três questões acabam afetando o sucesso de um produto. A primeira é que um produto deve prover as necessidades de seus clientes com a funcionalidade desejada, e de uma forma consistente e confiável. Segundo, um produto deve ser de fácil uso. Terceiro, uma organização deve atender aos interesses e questões dos usuários de forma adequada. Essas três questões envolvem o valor do cliente e uma ferramenta de GCS poderia automatizar as práticas que forneçam um valor maior para seus usuários;

- **Princípio 5 - Artefatos de software devem ter alta qualidade:** existem várias medidas da qualidade do produto (que devem ser adotadas continuamente): (i) adaptabilidade - fácil adição de novas características ou aumento de outras; (ii) eficiência, onde produtos eficientes executam de forma mais rápida e consomem os recursos estritamente necessários; (iii) generalidade - produtos gerais resolvem classes maiores de problemas; (iv) manutenibilidade - produtos manuteníveis são mais fáceis de arrumar; (v) confiabilidade - produtos confiáveis executam as tarefas pretendidas de uma forma consistente com as expectativas do usuário; (vi) reusabilidade – produtos reusáveis preenchem as necessidades de muitas tarefas; (vii) simplicidade - produtos simples são mais fáceis de entender, adaptar e manter, e também são mais elegantes e eficientes; e (viii) compreensão - produtos compreensíveis permitem que os desenvolvedores possam alterá-los mais facilmente, uma vez que produtos difíceis de entender tendem a ser mal projetados, implementados e documentados, podendo se tornar menos “confiáveis”. Conseqüentemente, várias medidas de qualidade de artefatos de software devem ser adotadas continuamente;

- **Princípio 6 - Sistemas de software devem ser confiáveis:** os sistemas de software devem funcionar como seus usuários esperam que funcionem. Também não devem ter defeitos significantes, que ocasionem perda de dados ou dano aos mesmos. Assim, esses sistemas devem ser altamente acessíveis e requerer pouca manutenção;

- **Princípio 7 - Produtos devem fornecer somente características necessárias, ou aquelas que**

**têm alto valor:** os produtos devem fornecer somente as características e capacidades desejadas pelos usuários. O acréscimo de características e capacidades não essenciais, que forneçam pouco ou nenhum valor aos usuários, tende a diminuir a qualidade do produto;

- **Princípio 8: Sistemas de software devem ser manuteníveis:** sistemas de software manuteníveis são geralmente simples, altamente modulares, bem projetados e documentados. Também tendem a ter baixo acoplamento. Uma vez que a maioria dos softwares é usada por muito tempo, os custos de manutenção para grandes sistemas de software geralmente excedem os custos iniciais de desenvolvimento;

- **Princípio 9: Usar recursos críticos eficientemente:** vários recursos são usados ou consumidos para desenvolver software, bem como pelo próprio software, que geralmente escassos, fazem com que a organização se preocupe em usá-los da forma mais eficiente possível;

- **Princípio 10: Minimizar o esforço de desenvolvimento:** o esforço humano é um recurso crítico. A principal motivação para usar os recursos humanos de forma eficiente é minimizar os custos de desenvolvimento.

Um sistema de gerenciamento de configuração estabelece e mantém a integridade de artefatos de software durante o ciclo de vida do software (PAULK et al., 1994 apud KANDT, 2002), identifica os artefatos gerenciados e suas configurações em vários pontos no tempo, assim como controla as mudanças nas configurações de software. Além disso, um sistema de GCS mantém a seqüência de derivações de cada configuração e artefatos individuais através do ciclo de vida de desenvolvimento do software (KANDT, 2002). O sistema de GCS suporta os desenvolvedores através do armazenamento da composição de produtos de software versionados, derivados em muitas versões e variantes, com a manutenção da consistência entre componentes de software interdependentes, reconstruindo configurações de software armazenadas e construindo produtos a partir das fontes apropriadas. Para alcançar esses objetivos, um sistema de GCS deve ajudar os desenvolvedores a construir configurações consistentes das versões dos seus produtos de software. Em uma configuração consistente de versões, as versões selecionadas dos componentes de configuração são consistentes entre si, uma propriedade importante, uma vez que mudanças nos componentes podem ser interdependentes (MUNSON; NGUYEN; BOYLAND, 2003).

Midha (1997) destaca algumas características desejáveis em um sistema de GCS, entre elas: (i) a facilidade de uso, para que o usuário possa fazer uso da funcionalidade como parte da execução de seu trabalho, e também para que ele possa sentir-se confortável com a interface; (ii) a facilidade de administração, pois os administradores de um sistema de GCS devem fazer auditoria dos repositórios

de dados e executar ações corretivas quando necessário; (iii) suporte ao desenvolvimento distribuído, para que o sistema possa ser usado com base em uma rede local, ou com equipes de desenvolvimento distribuídas remotamente, ou em bancos de dados distribuídos com sincronização periódica e (iv) integração da funcionalidade de GCS com outras ferramentas.

Um sistema de GCS é composto por um repositório, que é um local sob controle de acesso (banco de dados) onde os itens de configuração são armazenados, possibilitando o controle de versões. Uma vez que o item de configuração esteja armazenado no repositório, só pode ser alterado após uma solicitação de alteração formalmente aprovada pelo gerente de configuração. Além de armazenar o histórico dos itens, permite também armazenar a razão da mudança, quem submeteu a nova versão e quando (DART, 1990). Um item de configuração pode ser um produto de software (programas, procedimentos, documentação relacionada e informações relacionadas para entrega a um cliente ou usuário final) ou um produto de desenvolvimento de software (descrições de procedimentos, planos, procedimentos, programas de computador, documentos relacionados, os quais podem ou não ter a finalidade de entrega a um cliente ou usuário final). Através de estudo feito por Rozante (2003), a documentação relacionada pode ser selecionada para o controle de versões por meio dos seguintes critérios: (i) os documentos mais usados no ciclo de vida, (ii) os documentos mais genéricos, (iii) os documentos alterados por mais de um desenvolvedor e (iv) os documentos de maior importância para a segurança do sistema.

Pressman (2002) coloca que, além dos artefatos produzidos pelo processo de software, algumas organizações colocam também ferramentas de software sob controle de configuração (versões específicas de editores, compiladores e outras ferramentas), pois uma vez que foram usadas para produzir documentação, código-fonte e dados, precisam estar disponíveis quando modificações na configuração de software forem necessárias (apesar de raro, podem surgir problemas quando uma versão diferente da ferramenta é usada para fazer as modificações necessárias nos artefatos). Ao conjunto de itens de configuração de software inter-relacionados dá-se o nome de **configuração de software**.

***Baseline***, **linha-base** (PRESSMAN, 2002) ou **linha de referência** é um conceito relacionado ao GCS que ajuda no controle das modificações, estabelecendo um marco no processo de desenvolvimento de software. Quando um item de configuração é formalmente revisto e aprovado, as alterações nesse item não são permitidas se não forem formalmente solicitadas.

Para acesso ao repositório, é necessário o uso de algumas operações, cujos nomes aqui apresentados são conceituais, uma vez que o nome e a forma das operações podem variar conforme a ferramenta. O ***check out*** é a operação realizada quando uma cópia do item de configuração é colocada numa área de

trabalho do desenvolvedor. Após as modificações necessárias, a operação usada é o ***check in***, que permite que um item de configuração seja recolocado no repositório, com a liberação do bloqueio (*lock*) no arquivo. A operação de ***get***, citada por OLIVEIRA et al. (2001), permite que seja feita uma cópia para leitura do item de configuração, sem impedimento do acesso ao item por outros usuários.

A fim de minimizar o espaço de armazenamento das versões no repositório, são usados ***delta scripts***, ou simplesmente **delta**, no qual somente uma versão é armazenada integralmente – as demais versões têm apenas as diferenças armazenadas, consistindo em uma seqüência de comandos *edit* que transformam uma versão de um documento em outra (REICHENBERGER, 1991). Apesar de economia no consumo de espaço proporcionada pelo uso de deltas, o tempo de acesso pode aumentar, dependendo da versão desejada (BIELIKOVÁ; NÁVRAT, 1996). É excelente para arquivos do tipo texto, contudo, não é eficiente para arquivos binários, armazenados na íntegra no repositório. Porém, a maioria das ferramentas oferece armazenamento apenas para arquivos do tipo texto e como os arquivos podem de outros tipos (arquivos gerados em processadores de texto, programas em uma linguagem intermediária, imagens), deve ser previsto o armazenamento do delta para esses arquivos. Por isso, Reichenberger (1991) propõe uma técnica para encontrar o delta entre dois arquivos de tipos arbitrários, sem que se saiba a estrutura do arquivo. Para geração de delta entre arquivos do tipo texto existem várias abordagens, incluindo o algoritmo proposto por Tichy (1985).

O **delta negativo**, também chamado de **delta reverso**, foi primeiro proposto por Tichy (BIELIKOVÁ; NÁVRAT, 1996) e armazena integralmente a versão mais recente e as diferenças até então, disponibilizando de forma mais rápida a última versão - as versões anteriores exigem processamento adicional para serem obtidas. O **delta positivo** armazena integralmente a versão mais antiga e as diferenças (deltas) desde então. Além do uso de delta, muitas ferramentas fazem uso de técnicas de compressão de arquivos, com o intuito de otimizar a quantidade de espaço utilizado para o armazenamento no repositório.

Segundo Asklund (2001), existem muitas ferramentas de GCS, com várias funções, diferentes complexidades e diferentes preços. Cabe ressaltar que não há a melhor ferramenta e sim a mais apropriada, dependendo dos objetivos da organização. É possível encontrar muitas ferramentas simples, com funções básicas como controle de versões e gerenciamento de configuração, suficientes para pequenas equipes de desenvolvimento e ferramentas mais avançadas, requeridas por grandes empresas, contendo recursos para desenvolvimento distribuído, gerenciamento de mudanças e construção sofisticada. Algumas ferramentas de GCS podem ser citadas, entre elas, ClearCase[3],

[3] http://www-306.ibm.com/software/awdtools/clearcase.

Continuus/CM[4], Visual SourceSafe[5], TeamWare[6], QVCS[7], FreeVCS[8] e PVCS[9].

Mesmo que os recursos estejam disponíveis, usar uma ferramenta de GCS não é suficiente, pois o sucesso na utilização do GCS depende de suporte contínuo ao processo. Mesmo as ferramentas mais sofisticadas devem ser integradas ao processo e administradas, juntamente com outras ferramentas.
As ferramentas de GCS podem ser classificadas em grupos, de acordo com suas características funcionais, de forma a auxiliar na aquisição da ferramenta (DART, 1996 apud OLIVEIRA et al., 2001):

- Ferramentas de controle de versões;
- Ferramentas orientadas ao desenvolvedor: além do suporte ao controle de versões, possui suporte ao trabalho em equipe, facilitando o desenvolvimento concorrente e possibilitando que os desenvolvedores possam trabalhar em um mesmo conjunto de arquivos ao mesmo tempo;
- Ferramentas orientadas a processo: suporte ao controle de versões, parte das funcionalidades das ferramentas orientadas ao desenvolvedor e a automação do gerenciamento do ciclo de vida dos objetos envolvidos no desenvolvimento.

## 2.3. Controle de versões

O controle de versões (Figura 1) é muitas vezes confundido com o GCS, pois para muitas pessoas os dois conceitos têm o mesmo significado, no entanto, o controle de versões é a parte central de um GCS e a funcionalidade principal em muitas ferramentas (ASKLUND et al., 2001; MIDHA, 1997). Surgiu para controlar a evolução dos sistemas de software (MIDHA, 1997), combinando procedimentos e ferramentas para administrar diferentes versões de itens de configurações, criadas durante o processo de software (PRESSMAN, 2002). Tem como objetivos:

- Automação do rastreamento de arquivos;
- Prevenção de conflitos entre os desenvolvedores;
- Recuperação de versões anteriores;
- Desenvolvimento paralelo;
- Auditoria do desenvolvimento: quem, quando, o quê;
- Estabelecimento de agregações de arquivos: *baselines* e distribuições;

[4] http://www.telelogic.com/continuus.cfm.
[5] http://msdn.microsoft.com/vstudio/previous/ssafe.
[6] http://www.e-business.com/products/teamware.htm.
[7] http://www.qumasoft.com.
[8] http://www.thensle.de.

- Redução do espaço de armazenamento.

Os gerentes de versão são responsáveis pela garantia de que diferentes versões de um sistema possam ser acessadas quando necessário e que não sejam modificadas acidentalmente. As novas versões devem ser criadas sempre pela equipe de GCS e não pelos desenvolvedores do sistema, mesmo quando não se destinem à liberação externa. Além disso, somente a equipe de GCS pode modificar as informações sobre a versão, a fim de que a consistência do banco de dados de configuração seja mantida (SOMMERVILLE, 2003).

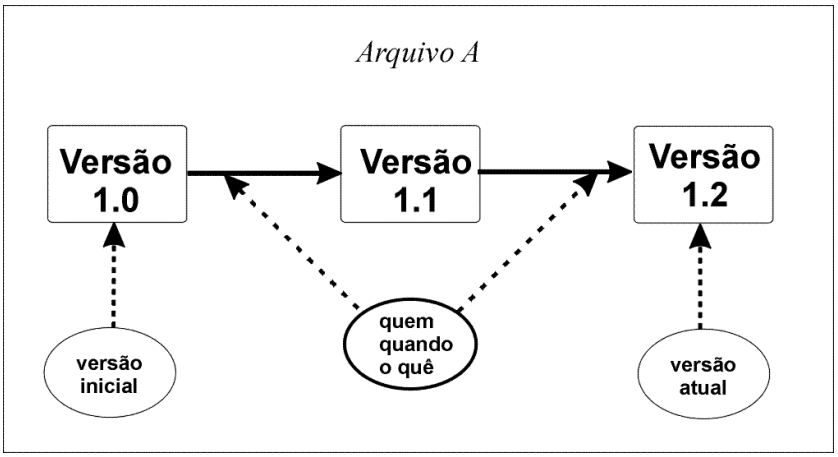


**Figura 1 -** Controle de versões.

Uma **versão** (Figura 2) pode ser definida como uma instância de um sistema, que difere, de muitas maneiras, de outras instâncias, em virtude de diferentes funcionalidades, diferentes desempenhos ou resolução de defeitos (SOMMERVILLE, 2003). A propriedade mais importante de uma versão é a imutabilidade, pois quando uma versão é congelada, seu conteúdo não pode ser modificado, tornando necessária a criação de uma nova versão (ASKLUND et al., 2001). As versões são gerenciadas em termos de arquivos e diretórios, que nem sempre correspondem a módulos no código-fonte. A modularização é usada com o intuito de reduzir a complexidade do software, através da decomposição do programa em funções, procedimentos ou classes, representando os módulos (LIN; REISS, 1995).

Um produto de software pode ter várias versões: quando a versão envolve a alteração de um arquivo é chamada de **revisão** e quando envolve a alteração de vários arquivos é chamada de ***release***. O *release* é a versão distribuída para os clientes, sendo que há sempre mais versões do que *releases*, pois uma versão pode ser criada para desenvolvimento interno ou testes e não ser liberada para os clientes. Um *release* não consiste apenas no código executável do sistema, e pode incluir o arquivo de configuração (como o *release* deve ser criado para instalações específicas), arquivo de dados, programa de

[9] http://www.pvcs.synergex.com

instalação, documentação eletrônica/em papel e embalagem e publicidade associada. Novos *releases* não podem depender da existência de *releases* anteriores (SOMMERVILLE, 2003).

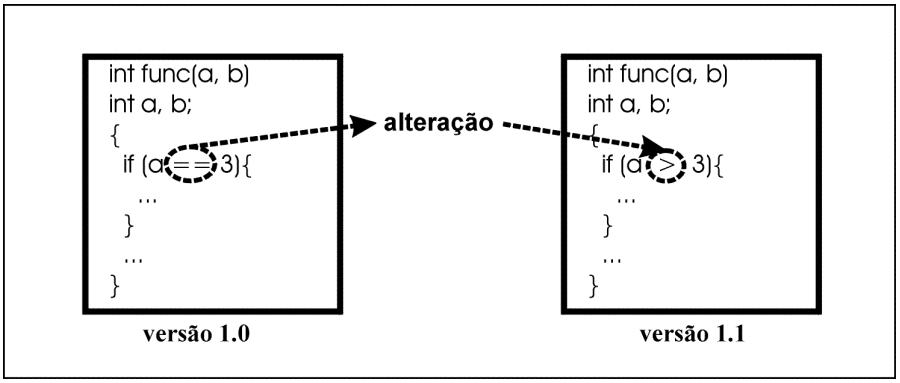


**Figura 2 –** Geração de uma nova versão.

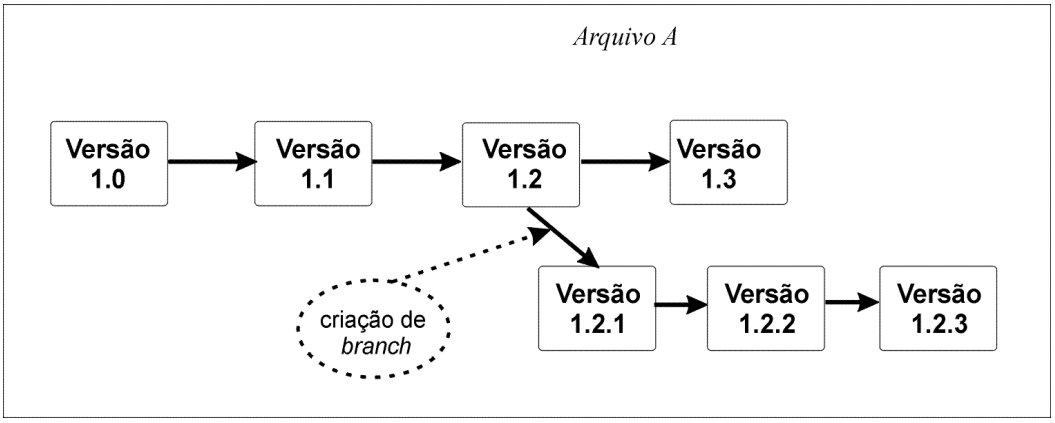


**Figura 3** – Criação de *branches*.

As versões de um componente de software são frequentemente organizadas em uma árvore ancestral, composta por uma versão raiz, representando a primeira versão de um componente de software e por ramificações laterais (*branches*), que surgem à medida que o desenvolvimento prossegue (BIELIKOVÁ; NÁVRAT, 1996). Um *branch* é uma forma de organização de versão, caracterizada pelo desenvolvimento paralelo (Figura 3), tendo como ponto inicial uma determinada versão, que evolui de forma independente. Pode ser usado para: (i) representar um caminho independente de desenvolvimento; (ii) representar diferentes variantes do componente, para uso em diferentes plataformas, interface com diferentes sistemas, etc; (iii) possibilitar modificações em paralelo no mesmo arquivo, executadas por diferentes desenvolvedores, permitindo que os produtos de trabalho sejam juntados posteriormente, gerando uma única versão e (iv) desenvolvimento distribuído em vários lugares (BIELIKOVÁ; NÁVRAT, 1996). O *branch* pode ainda ser definido como permanente, onde é feito o ajuste do arquivo de acordo com necessidades divergentes (por exemplo, no caso de diferentes sistemas operacionais) ou temporário, no caso de desenvolvimento paralelo (ASKLUND et al., 2001).

Uma variante é um caso especial de versão, onde há várias versões alternativas desenvolvidas a partir da mesma versão original. O uso de variantes ocorre quando produtos inteiros ou configurações são ajustados, de acordo com demandas divergentes (por exemplo, no caso de sistemas operacionais diferentes ou adaptações diferentes do software para o cliente), resultando em pequenas diferenças entre as versões. A criação de variantes pode ser feita de quatro formas (ASKLUND et al., 2001):

- Com *branches* permanentes dos arquivos incluídos;
- Com compilação condicional (diretivas de compilação);
- Com descrições de instalação, esclarecendo quais funcionalidades deveriam ser incluídas em uma determinada variante, sendo que funcionalidades dependentes são implementadas em diferentes arquivos, um para cada variante;
- Verificação em tempo de execução (*run-time check*).

Em Sommerville (2003) são citadas três técnicas básicas para a identificação de versões:

1. **Numeração de versões:** formato bastante definido, onde cada versão tem um número. A derivação da versão não é necessariamente linear, pois qualquer versão existente pode ser usada como ponto de partida para uma nova versão do sistema;
2. **Baseada em atributos:** um atributo pode ser o cliente, a linguagem de desenvolvimento, o *status* do desenvolvimento, a plataforma de hardware ou a data de criação;
3. **Identificação orientada a mudanças:** é usada para sistemas e não para componentes. Cada mudança proposta do sistema tem um conjunto de mudanças associado. A equipe interage com o sistema de controle de versões indiretamente, por meio do sistema de gerenciamento de mudanças.

Existem duas técnicas básicas de versionamento: (i) **explícita** (*extensional*), onde o conjunto das versões de um componente é definido explicitamente através da enumeração dos componentes-membro - uma nova versão é criada após as alterações de uma versão existente e recuperada, e (ii) **implícita** (*intensional*), onde um conjunto de versões de componentes é definido implicitamente através de predicados ou restrições, usados para formar as regras de seleção de versão, que selecionam uma determinada versão ou variante de um componente a ser incluído na configuração de um produto de software (MUNSON; NGUYEN; BOYLAND, 2003).

Muitos modelos para gerenciamento de configuração, assim como algumas ferramentas disponíveis, por exemplo, ClearCase e CVS, fazem uma clara separação na forma de tratamento de entidades atômicas (objetos versionados, módulos, etc) e compostas (configurações, bibliotecas, sistemas) (ASKLUND et al., 1999): nas entidades atômicas, o controle de versões é feito de forma individual, enquanto as configurações são formadas através de mecanismos de seleção. Se todas as entidades atômicas forem consideradas ao mesmo tempo, o número de combinações possíveis de suas versões e variantes é enorme - o uso de regras como "a última versão" é uma tentativa de automatizar o processo

de seleção. No Modelo de Versionamento Explícito Unificado proposto por Asklund et al. (1999), as entidades atômicas e as configurações têm o controle de versões feito de forma explícita, baseando-se em um mecanismo (concentração de versão) para reduzir o número de combinações que precisam ser consideradas.

### 2.3.1. Ferramentas para Controle de Versões

As ferramentas de controle de versões têm a função de armazenar versões de componentes do sistema, de maneira que seja possível a construção dos sistemas a partir desses componentes e o acompanhamento dos *releases* do software para os clientes (SOMMERVILLE, 2003).

A seguir são apresentadas as principais ferramentas de controle de versões utilizadas nos sistemas de GCS, a saber: RCS e CVS. Outras ferramentas de controle de versões mais recentes têm sido desenvolvidas, como por exemplo, *Arch* e *Subversion*, e tendem a superar os problemas existentes no RCS e no CVS, as duas ferramentas mais conhecidas nos meios acadêmicos e empresariais de desenvolvimento de software.

#### 2.3.1.1. *Revision Control System* (RCS)

O *Revision Control System* (RCS) (TICHY, 1985) é uma das ferramentas mais antigas para controle de versões e é baseada em um conjunto de comandos UNIX. É simples, com funções aplicadas a arquivos, sendo usada também como ferramenta básica para muitas outras ferramentas de controle de versões. Os princípios introduzidos pelo RCS ainda existem em muitas ferramentas avançadas de controle de versões (ASKLUND et al., 2001).

Sua estrutura é orientada a arquivos, na qual um arquivo versionado contém uma ou mais versões de um arquivo, armazenado em um repositório, que consiste em um diretório que contém arquivos versionados. Contém funções de diferença entre arquivos (*diff*) e junção de arquivos (*merge*), comandos de *check out/check in* e *baselines* (ASKLUND et al., 2001).

A função primária do RCS é o gerenciamento de grupos de revisão, onde as revisões são formadas por um conjunto de documentos do tipo texto. As revisões são organizadas em um árvore "ancestral", onde

a raiz da árvore á a revisão inicial. Além de gerenciar grupos individuais de revisões, tem funções de seleção para compor configurações, podendo ser combinada com a ferramenta Make (FELDMAN, 1979).

Com o intuito de conservar o espaço utilizado para o armazenamento das revisões, usa o conceito de deltas. No RCS um delta é baseado em linhas, permitindo os comandos de inserção e exclusão de linhas - quando um caracter em uma linha muda, é considerado que a linha inteira mudou.

#### 2.3.1.2. *Concurrent Versions System* (CVS)

O CVS (CEDERQVIST, 1993) é um sistema para controle de versões, muito usado em ambientes de software, obtendo papel importante em projetos *open source*, normalmente executados por equipes distribuídas (BRUCKER; RITTINGER; WOLFF, 2002) Possibilita armazenar o histórico dos arquivos-fonte com economia de espaço, através do armazenamento somente das diferenças entre as versões. A base do CVS é o RCS, mas ao contrário deste, gerencia também a estrutura de diretórios. A ferramenta tem origem em um conjunto de *scripts shell* escritos por Dick Grune. Em 1989, Grian Berliner projetou e codificou o CVS, com ajuda posterior de Jeff Polk no desenvolvimento do suporte a *branches*. Os arquivos sob controle de versões no CVS são armazenados em um repositório central (ASKLUND; MAGNUSSON, 1996), que contém todas as mudanças feitas e informações sobre o quê e quando foi mudado e quem fez as modificações. Os arquivos armazenados no repositório não podem ser acessados diretamente, somente através do uso de comandos CVS, que permitem que uma cópia dos arquivos seja feita em uma área de trabalho. O arquivo alterado só vai para o repositório após o término das alterações. O repositório não é um subdiretório do diretório de trabalho ou vice-versa - ambos devem estar em lugares separados. O acesso ao repositório pode ser feito através de um computador local, do outro lado da sala ou do outro lado do mundo. A área de trabalho é disponibilizada pelo CVS e consiste em um espaço destinado ao desenvolvedor, para que possa trabalhar de forma individual quando faz parte de um grupo de pessoas trabalhando no mesmo projeto. Após o término do trabalho, existe a possibilidade de junção (*merge*) dos trabalhos dos desenvolvedores.

Algumas características que não são encontradas no CVS:

- Não é um sistema de construção de configuração, apenas armazenando os arquivos para posterior recuperação;
- Não pode ser considerado um substituto para a gerência;

- Não tem controle de mudanças;
- Não é uma ferramenta de teste automatizada;
- Não tem um modelo de processo embutido.

Juntamente com os diretórios estão armazenados os arquivos de histórico (*history files*), que contêm informação suficiente para recriar qualquer revisão do arquivo, um *log* de todas as mensagens de *commit* e o nome do usuário que fez o *commit* da revisão. Esses arquivos também são conhecidos como arquivos RCS, pois esse foi o primeiro programa que armazenou arquivos nesse formato. Para cada arquivo sob controle de versões, existe um arquivo de histórico, identificado pelo nome do arquivo correspondente, acrescentado de ".v" no final.

No CVS, cada versão de um arquivo tem um único número de revisão, atribuído automaticamente, sendo a primeira revisão identificada como 1.1 (*default*) e as revisões subseqüentes têm o número mais à direita incrementado, por exemplo, 1.2, 1.3, etc.

No caso de vários desenvolvedores tentarem a edição simultânea de um mesmo arquivo, uma das soluções é o *lock* do arquivo (ou *check out* reservado), onde somente uma pessoa pode editar um determinado arquivo por vez, consistindo na única solução para alguns sistemas de controle de versões (RCS). No CVS, o *default* é o *check out* não reservado, no qual os desenvolvedores podem editar sua própria cópia de trabalho de um arquivo simultaneamente, sendo que o primeiro que fizer o *check in* não tem uma forma automática de saber que outro desenvolvedor está editando o mesmo arquivo - este só será avisado com mensagem de erro quando também tentar fazer o *check in*. Nesse caso, devem usar comandos CVS para atualizar a cópia de trabalho com a revisão do repositório. Cada forma de *check out* tem seus prós e contras: no caso do *check out* reservado, é considerado contraproducente quando duas pessoas desejam editar diferentes partes de um arquivo, pois não há razão para impedimento; no caso do *check out* não reservado, se não existir uma ferramenta de *merge* para o tipo de arquivo que está sendo gerenciado, a resolução de conflitos pode ser desagradável.

O armazenamento de arquivos do tipo texto é mais comum para o CVS, para os quais é possível mostrar as diferenças entre revisões, juntar revisões (com a sinalização de conflito quando as mudanças conflitam) e outras operações. No caso de arquivos binários, o CVS pode fazer o armazenamento de arquivos desse tipo, sem entretanto usufruir das mesmas funcionalidades. Para obter a diferença entre versões, seria preciso compará-las com o uso de ferramentas externas ao CVS ou então seguir as mudanças através de outros mecanismos, como por exemplo, escrever boas mensagens de *log* e esperar que as mudanças feitas sejam as que realmente deveriam ser feitas. No *merge* de arquivos binários, o melhor que o CVS pode fazer é mostrar as duas cópias e deixar o usuário decidir, podendo usar uma ferramenta de *merge* externa que suporte o formato do arquivo.

O armazenamento de um arquivo binário com o *check in* deve ser feito com a opção correspondente a esse tipo de arquivo, senão o formato do arquivo não é preservado. Existem duas questões que devem ser levadas em consideração em se tratando de arquivos binários:

1. É padrão do CVS converter os finais de linha entre a forma canônica, em que são armazenados no repositório (somente *linefeed*) e a forma apropriada ao sistema operacional em uso no cliente (por exemplo, *carriage return* seguido por *linefeed* no Windows NT);
2. Um arquivo binário pode conter dados que se parecem com uma palavra-chave, assim a expansão da palavra-chave deve ser desligada.

#### 2.3.1.3. *Subversion*

A ferramenta *Subversion* foi desenvolvida por Karl Fogel e Ben Collins-Sussman, com a colaboração de outras pessoas, sendo disponibilizada em 2001. Os arquivos ficam armazenados em um repositório central, que pode ser acessado via rede, possibilitando que usuários em diferentes computadores possam acessá-lo (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2004).

A ferramenta implementa a maior parte das funcionalidades do CVS, com uma interface bastante semelhante. Conforme levantado por Junqueira (2004), *Subversion* possui algumas características que a diferenciam do CVS:

- Além de controlar versões do conteúdo dos arquivos e de sua existência, possibilita o controle de versões dos diretórios, das cópias, e das renomeações;
- Os *commits* são sempre atômicos, ou seja, um *commit* só tem efeito a partir do momento que é completamente concluído, não gerando erros caso ocorra, por exemplo, a queda do servidor durante um *commit*;
- Manipula de forma eficiente arquivos texto e binários, devido ao algoritmo otimizado de geração de diferenças que ele utiliza.

#### 2.3.1.4. *Arch*

*Arch* é uma ferramenta para controle de versões e foi idealizada em 2001 por Thomas Lord, com o objetivo de atender às suas necessidades de controle de versões (KRAUSE, 2002).

Controle de versões de árvores de diretórios (e não apenas de arquivos), repositórios distribuídos, capacidade avançada de *merge*, suporte a mudança de nome de arquivos e diretórios são algumas das características encontradas na ferramenta (LORD, 2003; JUNQUEIRA, 2004).

### 2.3.2. Modelos de versão

Segundo Conradi e Westfechtel (1998), um modelo de versão define os objetos que serão versionados, a identificação e organização de versões, assim como operações para recuperação de versões existentes e construção de novas versões. Os objetos de software e seus relacionamentos constituem o espaço do produto, e as versões são organizadas no espaço de versão. Uma base de objetos versionados combina espaço de produto e de versão, armazenando todas as versões de um produto de software, baseando-se, por exemplo, em deltas intercalados, e é aumentada com a base de regras de configuração armazenadas. O *configurator* é a ferramenta que constrói a versão com base na avaliação de uma consulta em uma base de objetos versionados e uma base de regras.

O espaço do produto descreve a estrutura de um produto de software sem levar em conta o versionamento e pode ser representado por um grafo de produto, no qual os nós correspondem aos objetos do software e os vértices correspondem ao relacionamento entre os nós. Diferentes modelos de versão variam em suas definições em relação ao espaço do produto: os tipos de objetos de software e relacionamentos, granulosidade das representações do objeto, e os níveis semânticos de modelagem. Um objeto de software tem um identificador único e registra o resultado de uma atividade de desenvolvimento ou manutenção, sendo que um sistema de GCS tem que gerenciar todos os tipos de objetos de software criados durante o ciclo de vida do software, incluindo a especificação de requisitos, projetos, documentações, códigos-fonte, planos e casos de teste, manuais do usuário, planos de projeto, etc. Os relacionamentos representam a conexão entre os objetos de software de vários tipos e podem ser definidos como:

- **Relacionamentos de composição**: usados para organizar os objetos de software com respeito à sua granulosidade. Objetos que são decompostos são chamados de objetos ou configurações decompostos e objetos que ficam nas folhas da hierarquia de composição são chamados objetos atômicos. A raiz de uma hierarquia de composição é chamada o produto (de software);
- **Relacionamentos de dependência**: estabelecem conexões direcionadas entre os objetos, que são ortogonais aos relacionamentos de composição. Incluem: dependência de ciclo de vida entre especificações de requisitos, projetos e implementação de módulos, dependência de importação ou

inclusão entre módulos e dependência de construção entre o código compilado e o código-fonte. A fonte de uma dependência é considerada um objeto dependente e o destino (*target*) um objeto mestre. Uma dependência implica que o conteúdo do objeto dependente seja mantido consistente com o conteúdo do objeto mestre, com necessidade de modificação toda vez que o mestre for alterado. Uma outra classificação dos objetos de software, dependente do suporte da ferramenta de GCS, é o objeto fonte, criado por um desenvolvedor e apoiado por ferramentas interativas, como editores de texto ou gráficos e o objeto derivado, criado automaticamente por uma ferramenta, por exemplo, um compilador. Um objeto pode ser parcialmente derivado e parcialmente construído de forma manual, onde o esqueleto de um módulo é criado automaticamente por uma ferramenta e depois preenchido por um programador. Na construção do sistema, as ações a serem tomadas são determinadas pelas regras de construção, que devem assegurar que os passos de construção sejam executados na ordem correta, levando as dependências em consideração.

O espaço de versão organiza as versões e envolve vários conceitos relacionados a uma versão. Uma versão representa um estado de um item em desenvolvimento, que pode ser qualquer coisa que possa ser colocada sob controle de versões, incluindo, arquivos e diretórios em sistemas baseados em arquivos, objetos armazenados em bancos de dados orientados a objeto, entidades, relacionamentos, e atributos em bancos de dados relacionais, etc. Um item versionado é um item colocado sob controle de versões e cada versão deve ser identificável unicamente através de um identificador de versão. A diferença entre duas versões, que pode não ser necessariamente pequena (a parte comum entre as versões pode ser vazia), é representada pelo delta, podendo ser definido de duas maneiras:

- **Delta simétrico**: contém as propriedades específicas entre as versões (intersecção);
- **Delta dirigido**: também chamado de mudança, corresponde a uma sequência de operações de mudança elementares que quando aplicada a uma versão produz outra.

O versionamento pode ser aplicado em qualquer nível de granulosidade, de um produto de software até linhas de texto, e é feito com intenção de: (i) revisão, onde a nova versão substitui a anterior por várias razões (corrigir *bugs*, acréscimo ou expansão de funcionalidades, adaptação a mudanças em bibliotecas básicas, etc), preservando a versão antiga para suporte à manutenção do software entregue aos clientes, para recuperação de atualizações errôneas, etc e (ii) variante, onde a nova versão pretende coexistir com outras, por exemplo, no caso de produtos de software desenvolvidos para diferentes sistemas operacionais. O versionamento pode ser definido como: (i) explícito, definido pela enumeração de seus membros, ou (ii) implícito, aplicado quando é necessária a construção automática e flexível de versões consistentes em um grande espaço de versão (CONRADI; WESTFECHTEL, 1998; MUNSON; NGUYEN; BOYLAND, 2003).

O versionamento explícito suporta recuperação de versões construídas previamente (necessário em

qualquer modelo de versão) e que tenha sido feito *check in* delas pelo menos uma vez. Cada versão é identificada tipicamente por um número único. Uma versão recuperada, alterada e submetida é considerada uma nova versão e é imutável (em alguns sistemas, a partir do momento em que é feito o *check in* e em outros (ROCHKIND, 1975 apud CONRADI; WESTFECHTEL, 1998) operações explícitas são feitas para congelar uma versão mutável (WESTFECHTEL, 1996 apud CONRADI; WESTFECHTEL, 1998). No versionamento implícito, ao invés de enumerar os membros, o conjunto de versão é definido por um predicado. As versões são implícitas e muitas novas combinações são construídas sob demanda, em resposta a alguma consulta. Esse tipo de versionamento é muito importante para grandes espaços de versões, onde um produto de software evolui em muitas revisões e variantes e muitas mudanças têm que ser combinadas.

De acordo com Conradi e Westfechtel (1998) e Munson (2003), as regras de configuração são usadas para configurar versões consistentes de uma base de objetos versionada e são requeridas para tratar do problema de explosão combinatória que ocorre quando o número de versões potenciais explode combinatoriamente, sendo que somente algumas são consistentes ou relevantes. A regra de configuração guia ou restringe a seleção de versões para uma determinada parte de um produto de software e consiste na parte do produto (determina o escopo no espaço de produto) e na parte da versão (faz uma seleção no espaço de versão).

As regras de seleção são avaliadas sob demanda, quando é necessário visualizar, editar ou traduzir uma entidade atômica, por exemplo, um arquivo. Entretanto, se for necessário saber quais arquivos estão incluídos em uma configuração e quais versões, é necessária a construção da configuração para análise do resultado. Além disso, não é possível a comparação de configurações com o objetivo de verificação das diferenças através das regras de seleção, sendo necessária a construção das configurações e a posterior comparação dos resultados. Consistência é uma coisa que não pode ser garantida, uma vez que imperfeições ou erros nas regras podem ficar despercebidos por um longo período de tempo, até que uma configuração errada seja gerada, e também, por incluírem facilidades genéricas como a seleção da última versão, que podem resultar em muitas configurações diferentes ao longo do tempo (ASKLUND et al., 1999).

Conforme observado em Feiler (1991), através da análise de vários sistemas de GCS, é possível definir quatro modelos de GCS: *Check out/Check in*, Composição (*Composition Model*), Longa Transação (*Long Transaction Model*) e Conjunto de mudanças (*Change-based Model*).

O **Modelo *Check out/Check in***, modelo mais básico (ASKLUND et al., 2001), possibilita o controle de versões de forma individual dos componentes do produto de software, possibilitando também o controle de modificações concorrentes. Baseia-se na existência de um repositório para o

armazenamento e controle das versões dos arquivos que constituem uma determinada configuração de software. Para que o conteúdo de um arquivo seja acessado, é necessário fazer uma cópia deste (*check out*) na área de trabalho do desenvolvedor. A área de trabalho real é disponibilizada através do sistema de arquivos, assim, quando é feito o *check out*, a versão de trabalho fica em um diretório no sistema de arquivos, e portanto, fora do controle do sistema de GCS. Diferentes áreas de trabalho podem ser representadas por diferentes diretórios. Quando o *check out* é feito com o objetivo de modificação, existem mecanismos de controle de concorrência no repositório que coordenam possíveis modificações concorrentes, isto é, quando dois ou mais usuários fazem modificações no mesmo arquivo, permite que somente um deles possa fazer o *check out* do arquivo para modificação. Uma vez alterado, o arquivo modificado é armazenado novamente no repositório (*check in*), gerando uma nova versão para o arquivo.

Uma mudança lógica em um produto de software pode originar alterações em vários arquivos. Nesse modelo o repositório não tem condições de saber quais modificações fazem parte de uma mudança lógica, a menos que as versões tenham a mesma identificação (por exemplo, o número da solicitação de mudança).

O **Modelo de Composição** é baseado no modelo *Check out/Check in*, considerando o conceito de repositório e área de trabalho, assim como o controle de concorrência através de bloqueio (*lock*) no componente recuperado para alteração. O objetivo desse modelo é melhorar o suporte para a criação de configurações, a fim de manter o histórico de versões de um sistema e seus componentes com a seleção de versões de componentes que resultem em uma configuração consistente, onde a versão selecionada de um componente é consistente com a versão selecionada de outros componentes. Nesse modelo, os desenvolvedores definem o sistema em termos de seus componentes e, em um passo separado, selecionam uma versão apropriada para cada componente.

Uma configuração é composta por: (i) modelo de sistema, que contém todos os componentes do sistema, e (ii) regras de seleção de versão, que indicam qual versão de um componente (do modelo de sistema) deve ser escolhida para compor uma configuração.

Os sistemas de GCS que usam esse modelo permitem ao usuário indicar regras de seleção de versão, ao invés de pegar manualmente as versões componentes no momento da recuperação de componentes do repositório. Essas regras de seleção podem identificar unicamente a versão de cada componente (configuração definida) ou a aplicação da regra de seleção em diferentes momentos pode resultar em uma configuração diferente (configuração parcialmente definida).

Na configuração parcialmente definida, as regras de seleção são constituídas por predicados na lógica

de primeira ordem e as versões componentes, que satisfazem o predicado, são parte de uma seleção de configuração. É possível que alguns componentes tenham versões múltiplas que satisfaçam o predicado e, por isso, o predicado deve ser reforçado de forma a resultar em uma configuração totalmente definida, e ao mesmo tempo, deve ser bem especificado para que não possa ser satisfeito por alguns componentes.

No que diz respeito ao controle de concorrência, são usados os mecanismos de bloqueio (*lock*) e ramificação (*branch*) nos componentes do modelo *Check out/Check in*. Apesar do bloqueio, pode ser feito o compartilhamento de componentes modificados, antes de sua liberação no repositório, em uma área de trabalho compartilhada. Com o intuito de evitar que os componentes possam ser mudados, podem ser usados direitos de acesso apropriados ou ainda, de maneira alternativa, cada desenvolvedor pode ter sua própria área de trabalho, trabalhando de forma independente, mas ainda compartilhando modificações.

O **Modelo de Transação Longa** enfoca o suporte à evolução de um sistema inteiro como uma série de mudanças aparentemente atômicas, assim como a coordenação da modificação dos sistemas pelas equipes de desenvolvedores. Nesse modelo, a modificação é feita em uma transação: uma configuração específica é escolhida como ponto inicial e as modificações não são visíveis fora da transação até que a mesma seja encerrada. O controle de concorrência garante que não haja perda das modificações através da coordenação das várias transações. Uma nova versão de configuração é o resultado do encerramento de uma transação. A transação desse modelo difere de uma transação tradicional de banco de dados no que diz respeito à duração, uma vez que pode durar horas, meses e, além disso, deve existir além da duração de uma sessão de *login* do desenvolvedor.

Dois conceitos envolvem a transação longa: o espaço de trabalho (*workspace*) e um esquema de controle de concorrência. O espaço de trabalho representa o contexto de trabalho, provendo memória local e substituindo o uso do sistema de arquivos como área de trabalho, permitindo com isso, que o sistema de GCS possa gerenciar o repositório e suportar desenvolvedores durante as modificações no sistema, em suas áreas de trabalho. O espaço de trabalho tem origem em uma configuração definida no repositório ou em um espaço de trabalho anexo, e consiste em uma configuração de trabalho, que representa a configuração na qual os componentes e a estrutura do sistema podem ser modificados, e uma série de configurações preservadas. Para que as mudanças sejam visíveis fora do espaço de trabalho, são efetivadas no repositório ou no espaço de trabalho anexo, criando uma versão sucessora à versão da configuração anterior.

Os desenvolvedores entram em um espaço de trabalho, ao invés de explicitamente recuperar componentes do repositório para o sistema de arquivos. Existem duas alternativas para isso: (i) os

espaços de trabalho e suas configurações aparecem como diretórios e os desenvolvedores navegam até o espaço de trabalho apropriado, sendo a seleção do espaço de trabalho e a versão da configuração desejada embutidas no caminho e visíveis ao desenvolvedor e (ii) o desenvolvedor solicita ao sistema de GCS o mapeamento da configuração desejada para um ponto base no sistema de arquivos. O acesso à configuração na forma de diretórios e arquivos é fornecido de maneira similar ao *Network File System* (NFS).

Enquanto estiver operando em um espaço de trabalho, o desenvolvedor é isolado das alterações efetuadas em outros espaços de trabalho, assim como das alterações efetivadas por outros desenvolvedores no repositório. Um espaço de trabalho pode ser aninhado, ou seja, ao invés de ser originado do repositório, é originado de uma configuração preservada de outro espaço de trabalho.

No que diz respeito ao controle de concorrência, existem três categorias de suporte ao desenvolvimento concorrente:

1) Concorrência com um espaço de trabalho, onde vários desenvolvedores estão ativos no mesmo espaço de trabalho ou quando um desenvolvedor está ativo em um espaço de trabalho várias vezes (várias janelas), sendo que as modificações devem ser sincronizadas através de exclusão mútua;
2) Concorrência entre espaços de trabalho requerendo coordenação, que ocorre quando diferentes desenvolvedores trabalham em espaços de trabalho estáveis e separados e a coleção de suas mudanças expande um sistema. Esquemas para controlar a concorrência nessa situação caem em duas categorias: conservador, que requer bloqueio sobre os espaços de trabalho, e otimista, que permite que as modificações ocorram concorrentemente, e os conflitos sejam detectados no momento em que forem efetivados. Um conflito é considerado quando um componente foi modificado em mais de um espaço de trabalho. Em caso de ocorrência de conflito, a primeira efetivação é bem-sucedida enquanto as outras são abortadas. Quando um conflito ocorre, o espaço de trabalho é atualizado com as mudanças no espaço de trabalho anexo, e o desenvolvedor é questionado sobre juntá-las com a modificação no espaço de trabalho. Eficazmente a transação é abortada e refeita. Uma vez que o *merge* é completado, a efetivação das mudanças (*commit*) pode ser tentada novamente. O efeito desse esquema é forçar a integração de mudanças concorrentes antes que fiquem disponíveis para outros;
3) Desenvolvimento concorrente independente, que assume que um sistema evolui em caminhos de desenvolvimento independentes, enquanto as duas primeiras tratam de mudanças concorrentes em uma configuração do sistema. As mudanças podem ser depois serializadas através de propagação e *merge* se desejado.

O **Modelo Conjunto de Mudanças** enfoca o suporte ao gerenciamento de mudanças lógicas nas configurações do sistema. Um conjunto de mudanças se refere a um conjunto de modificações em

diferentes componentes, compondo uma mudança lógica. É o registro de uma mudança lógica que persiste depois que a atividade de criação da mudança foi completada. Nesse modelo, as configurações podem ser descritas consistindo de uma *baseline* e um conjunto de conjuntos de mudanças. As mudanças são propagadas a outras configurações através da inclusão do conjunto de mudanças respectivo. Os desenvolvedores podem localizar mudanças lógicas e determinar se são parte de uma determinada configuração. A visão de gerenciamento orientado a mudanças do modelo difere da visão orientada a versão dos modelos anteriores, que focam no versionamento de componentes e configurações. O conjunto de mudanças provê uma ligação natural para as solicitações de mudanças, representando as modificações reais. Uma solicitação de mudança é uma ferramenta de gerenciamento para o controle da iniciação, avaliação, autorização e aprovação de mudanças.

Esse modelo por si só não provê mecanismos de bloqueio para o controle de concorrência. Consequentemente, os sistemas de GCS suportando conjuntos de mudanças também suportam o modelo *Check out/Check in* para esse propósito.

## 2.4. Considerações Finais

Neste capítulo foram apresentados alguns conceitos fundamentais, na forma de revisão bibliográfica, com o intuito de contextualizar o trabalho desenvolvido. O controle de versões é a base deste trabalho e o seu objetivo – controlar a evolução, é parte de um conceito maior - o GCS, que tem sido uma das principais atividades usadas no esforço para garantia da qualidade do software.

Dentre as ferramentas de controle de versões levantadas (RCS, CVS, *Arch* e *Subversion*), optou-se por focar no CVS, que é uma ferramenta bastante utilizada atualmente no controle de versões, com destaque para os projetos de Software Livre (REIS, 2003). Assim, no capítulo 3 é apresentado o estado da arte no que diz respeito aos suportes existentes para o CVS.

Além disso, no capítulo a seguir, é apresentado um estudo sobre o estado da arte em relação à edição colaborativa na Web; em especial, a ferramenta de edição colaborativa CoTeia é descrita, por ter sido alvo do desenvolvimento realizado neste trabalho, no qual foi considerado o suporte de CVS para o controle de versões na CoTeia.

# Capítulo 3

# Estado da Arte sobre Edição Colaborativa e o Suporte de CVS

Neste capítulo é apresentado o estudo sobre o estado da arte das ferramentas disponíveis atualmente para suporte ao CVS, bem como o estudo do suporte à edição colaborativa. Inicialmente, na seção 3.1 é mostrado um levantamento de ferramentas disponíveis para visualização de repositório CVS. Em seguida, na seção 3.2, como ferramenta para a edição colaborativa, a **wiki** é caracterizada e são detalhadas as ferramentas CoWeb (seção 3.4) e CoTeia (seção 3.5). Como um dos objetivos deste trabalho foi o de investigar também o tipo de auxílio ao registro das alterações, ocorridas nas versões, que viabilizam um conjunto de informações que explique as razões de evolução de projetos de software, na seção 3.5 é descrito o conceito de *Design Rationale* (DR) e a ferramenta DocRationale é abordada como forma de registro do DR. Cabe ressaltar que a ferramenta DocRationale é integrada a uma versão da CoTeia. Finalmente, na seção 3.6, são feitas as considerações finais sobre este capítulo.

## 3.1. Ferramentas para Visualização de Repositório CVS

A interface tradicional disponível para o CVS é a de linha de comando, entretanto, existem também algumas aplicações-cliente GUI (CEDERQVIST, 1993; VENUGOPALAN, 2002), que disponibilizam uma interface gráfica para acesso ao repositório CVS. Essas interfaces estão disponíveis para a maioria das plataformas populares (Windows, Mac e Linux). Dentre essas interfaces, existe um conjunto de ferramentas que possibilitam apenas a visualização do repositório e operações como *diff* e *annotate*, não alterando o conteúdo do repositório (por exemplo, *ViewCVS* e CVSweb). Outras ferramentas possibilitam também a manutenção do repositório, com comandos como *check out* e *check in*, disponibilizados por meio da interface gráfica (por exemplo, WinCVS e jCVS).

Com o intuito de selecionar uma ferramenta de visualização do repositório CVS que pudesse ser integrada à CoTeia, foi feito um levantamento de ferramentas disponíveis nos *websites* Sourceforge[10] e

[10] http:/sourceforge.net.

Freshmeat[11], com características identificadas como relevantes, e sintetizadas nos Quadros 1 e 2. No Quadro 1 são apresentadas as características técnicas das ferramentas:

(i) identificação e data da última versão;
(ii) Interface;
(iii) Linguagem de programação utilizada no desenvolvimento da ferramenta;
(iv) Idioma em que a ferramenta está disponível;
(v) Licença de software livre;
(vi) *Status* de desenvolvimento[12] ;
(vii) Sistema operacional para instalação da ferramenta.

No Quadro 2 podem ser vistas as características relacionadas ao suporte às versões:

(i) Visualização do repositório;
(ii) Visualização de arquivos;
(iii) Histórico de revisões;
(iv) *Diff*;
(v) *Diff* colorido;
(vi) *Annotate*.

Dentre as ferramentas verificadas, optou-se por usar a ferramenta *ViewCVS* para integração à CoTeia, por fornecer as características desejadas e também por estar sendo amplamente utilizada - a seção 3.1.2 descreve a ferramenta *ViewCVS*.

[11] http:/fresmeat.net
[12] Segundo os *websites* Sourceforge e Freshmeat, o status pode ser: 1 – Planejamento, 2 – Pré-Alfa, 3 – Alfa, 4 – Beta, 5 – Produção/Estável, 6 – Maduro e 7 – Inativo.

**Quadro 1 -** Ferramentas de visualização de repositórios CVS.

| | | | | | | | Sistema Operacional | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ferramenta** | **Versão/ Data** | **Interface** | **Ling. Programação** | **Idioma** | **Licença** | ***Status* de Desenvolvimento** | **POSIX (Linux /BSD/ Unix-like Oses)** | **Windows 32 bits (95/98/NT/ 200/XP)** | **Independente de S.O. (escrito em linguagem interpretada)** | **Linux** | **SUN Solaris** | **OS X** |
| Bonsai[13] | 1.33 20/12/2004 | Web | Perl | Inglês | GPL | | | | | | | |
| Cervisia[14] | 1.4.1 17/05/2001 | K Desktop Environment (KDE) | C++ | Inglês | QPL | 5 | X | | | | | |
| CVS Monitor [15] | 0.6.3 08/03/2004 | Web | Perl | Inglês | GPL | 5 | X | | | | | |
| CVS4IIS[16] | 2.0.1 11/09/2003 | Web | ASP Visual Basic VB6.0/COM+/ASP/IIS/Javascript. | Inglês | GPL | 5 | | X | | | | |
| CVSweb[17] | 3.0.4 06/11/2004 | Web | Perl | Inglês | BSD | | | | X | | | |
| Gruntspud[18] | 0.4.6-beta 11/04/2004 | - | Java | Inglês | GPL | 4 | | | X | | | |
| JviewCVS[19] | 0.5 20/09/2003 | Web | Java JavaScript Unix Shell | Inglês | GPL | 4 | X | | | X | | X |

[13] http://www.mozilla.org/projects/bonsai/ e em http://directory.fsf.org/bonsai.html
[14] http://sourceforge.net/projects/cervisia/ e em http://www.kde.org/apps/cervisia/
[15] http://ali.as/devel/cvsmonitor/index.html e em http://sourceforge.net/projects/cvsmonitor/
[16] http://sourceforge.net/docman/display_doc.php?docid=18719&group_id=88210
[17] http://freshmeat.net/projects/cvsweb/; http://www.freebsd.org/projects/cvsweb.html; http://www.icewalkers.com/Linux/Software/517020/FreeBSD-CVSweb.html
[18] http://sourceforge.net/projects/gruntspud/
[19] http://jviewcvs.sourceforge.net/

| | | | | | | | Sistema Operacional | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ferramenta** | **Versão/ Data** | **Interface** | **Ling. Programação** | **Idioma** | **Licença** | ***Status* de Desenvolvimento** | **POSIX (Linux /BSD/ Unix-like Oses)** | **Windows 32 bits (95/98/NT/ 200/XP)** | **Independente de S.O. (escrito em linguagem interpretada)** | **Linux** | **SUN Solaris** | **OS X** |
| LinCVS[20] | 1.3.2 09/05/2004 | Cocoa (MacOS X) Win32 (MS Windows) X Window System (X11) | C++ | Inglês | GPL | 5 | X | X | | | | X |
| PhpCVSView[21] | 0.1.5 07/05/2004 | Web | PHP (classe p/ ser usada em outro código) | Inglês | GPL | 2 | X | X | X | X | | |
| TkCVS [22] | 7.2.2 21/11/2004 | Win32 (MS Windows) X Window System (X11) | Tcl | Inglês | GPL | 5 | | | X | | | |
| TortoiseCVS[23] | 1.8.11 30/12/2004 | Win32 (MS Windows) | C++ | Português Catalão Chinês Inglês Francês Alemão Koreano Norueguês | GPL | 5 | | X | | | | |
| ViewCVS[24] | 2.2 15/01/2002 | Web | Phyton | Inglês | BSD | 4 | X | | | | X | |

[20] http://sourceforge.net/projects/lincvs/ e em http://www.lincvs.org/
[21] http://phpcvsview.sourceforge.net/
[22] http://sourceforge.net/projects/tkcvs/ e em http://www.twobarleycorns.net/tkcvs.html
[23] http://sourceforge.net/projects/tortoisecvs/
[24] http://viewcvs.sourceforge.net/

| | | | | | | | Sistema Operacional | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ferramenta** | **Versão/ Data** | **Interface** | **Ling. Programação** | **Idioma** | **Licença** | ***Status* de Desenvolvimento** | **POSIX (Linux /BSD/ Unix-like Oses)** | **Windows 32 bits (95/98/NT/ 200/XP)** | **Independente de S.O. (escrito em linguagem interpretada)** | **Linux** | **SUN Solaris** | **OS X** |
| WinCVS[25] | 1.3b23 09/01/2005 | Win32 (MS Windows) | Python Tcl | Inglês | GPL | 4 5 | | X | | | | |

[25] http://www.wincvs.org/

**Quadro 2 -** Características das ferramentas de visualização

| Ferramenta | *Visualização* de diretórios | Visualização de arquivos | Histórico de Revisões | Diff | Diff Colorido | Annotate |
|---|---|---|---|---|---|---|
| Bonsai | SIM | SIM | SIM | SIM | SIM | SIM |
| Cervisia | SIM | SIM | SIM | SIM | SIM | SIM |
| CVS Monitor | SIM | SIM | SIM | SIM | NÃO | SIM |
| CVS4IIS | SIM | SIM | - | SIM | SIM | SIM |
| CVSweb | SIM | SIM | SIM | SIM | SIM | SIM |
| Gruntspud | SIM | SIM | SIM | SIM | SIM | SIM |
| JviewCVS | SIM | SIM | SIM | SIM | SIM | SIM |
| LinCVS | SIM | SIM | SIM | SIM | SIM | SIM |
| PhpCVSView | SIM | SIM | SIM | - | - | SIM |
| TkCVS | SIM | SIM | SIM | SIM | SIM | SIM |
| TortoiseCVS | SIM (Ferramenta é integrada ao Windows Explorer) | SIM (Ferramenta é integrada ao Windows Explorer) | SIM | SIM (através de ferramenta externa) | Depende da ferramenta externa usada | NÃO |
| ViewCVS | SIM | SIM | SIM | SIM | SIM | SIM |
| WinCVS | SIM | SIM | SIM | SIM | SIM | SIM |

### 3.1.2. *ViewCVS*

A ferramenta *ViewCVS* mostra o conteúdo de repositórios CVS e Subversion, e tem sido usada em vários projetos, inclusive no projeto SAFE (*Software Engineering Available For Everyone*) do ICMC-USP. O desenvolvimento desta ferramenta foi baseado no *script* CVSweb (escrito em Perl) e a conversão para Python foi feita por Greg Stein, com algumas atualizações e mudanças (SILVA; IGNACIO JR; FORTES, 2004).

Como a ferramenta *ViewCVS* não tem controle de acesso, pode ser acessada por qualquer pessoa. Para usá-la, é necessário que o usuário tenha um *browser* (Internet Explorer, Netscape, Mozilla, Opera, etc) instalado no computador que será usado para fazer o acesso ao conteúdo do repositório CVS (SILVA; IGNACIO JR; FORTES, 2004).

O conteúdo do repositório (arquivos e diretórios) pode ser visualizado, como ilustra a Figura 4, com vários tipos de ordenação:

- Por nome do arquivo que está no repositório;
- Pelo número da revisão (mais recente) do arquivo;
- Pelo autor da revisão;
- Pelo tempo de permanência da revisão no repositório;

- Pela mensagem de *log* fornecida para a revisão mais recente. Como a revisão inicialmente mostrada para cada arquivo é a mais recente, é necessário que o usuário clique no nome do arquivo para acesso ao histórico de revisões do mesmo.

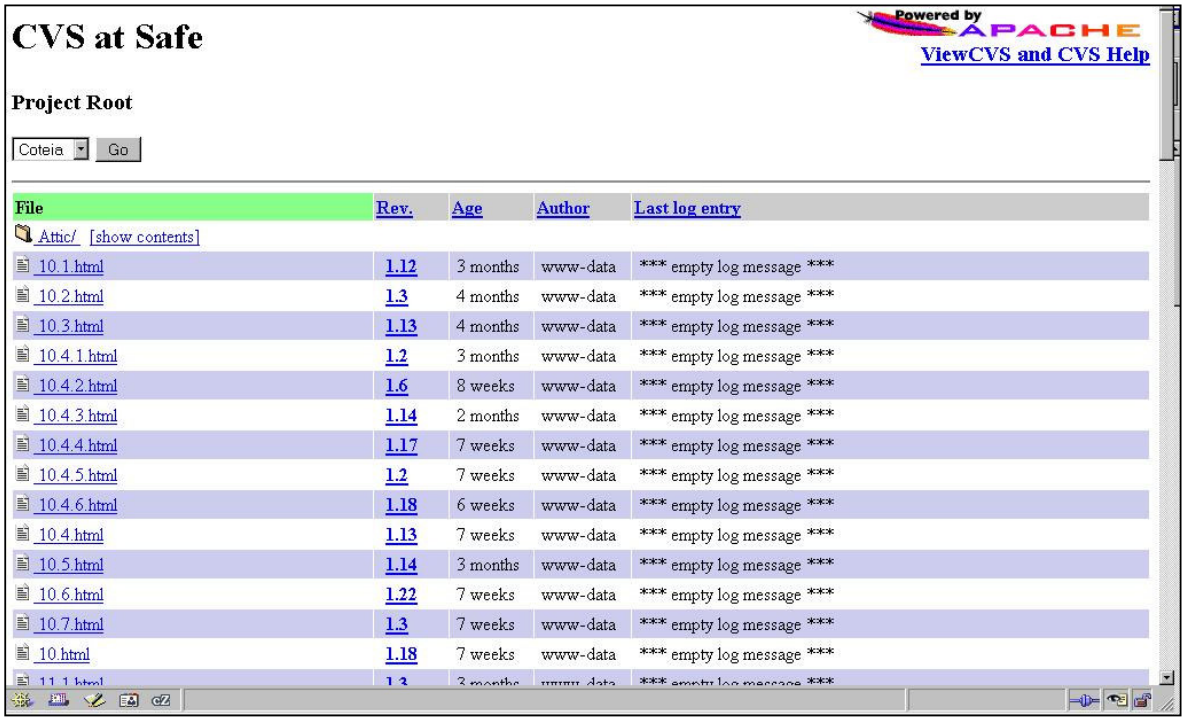


**Figura 4 -** Tela inicial da ferramenta *ViewCVS*.

É possível também, através da ferramenta (SILVA; IGNACIO JR; FORTES, 2004):

- Visualização do conteúdo da revisão do arquivo;
- Comparação das revisões do arquivo (*diff*), mostrando as diferenças de várias formas (*Colored diff*, *Long colored diff*, *Unidiff*, *Context diff* e *Side by side*);
- Visualização das alterações feitas associadas à revisão e ao autor das modificações (*annotate*).

## 3.2. Wiki – Ferramenta de Edição Colaborativa na Web

A palavra Wiki é a abreviação de Wiki Wiki, que no Hawaí significa "muito rápido", e foi usada por Ward Cunningham em 1995 (CUNNINGHAM, 2004) para dar nome a um sistema – Wiki Wiki Web - que permitisse colaboração de forma rápida na Web, através do desenvolvimento de páginas de livre edição. Assim, uma Wiki é um *website* que, além de ser visualizado, pesquisado e ter conteúdos adicionados, pode ser editado diretamente por qualquer pessoa (RUPLEY, 2003). Segundo Cuningham (2004), "Wiki é um sistema de composição, um meio de discussão; é um repositório, é

um sistema de correio; é uma ferramenta para colaboração...".

Desde a estréia da Wiki Wiki Web original, muitas implementações de wiki surgiram, variando em características, facilidade de instalação, sintaxe e semântica, sendo que a maioria são públicas (*open source*) (CUNNIGHAM, 2004). Existem implementações de wikis que permitem associar arquivos às páginas (*upload*) para que possam ser usados posteriormente, diretamente da wiki. Outra característica comum é o controle de versões das páginas, possibilitando que os usuários mantenham versões anteriores das páginas, com o objetivo de desfazer possíveis danos às páginas. A maioria das implementações de wiki também apresenta mecanismo de busca em suas páginas (FISH, 2004). Em uma wiki é possível o uso de *hiperlinks* e referências cruzadas entre páginas internas de forma rápida (CUNNINGHAM, 2004). É possível também utilizar uma linguagem de marcação nas páginas de uma wiki, geralmente diferente do HTML, possibilitando uma edição mais fácil, ou ainda, um subconjunto do HTML (FISH, 2004). É possível indicar porções de texto como *links* para URLs, incluir imagens baseadas em URLs, destacar texto com negrito, sublinhado ou itálico, colocar listas numeradas e não numeradas, etc (FISH, 2004).

Cunningham (2004) relaciona alguns princípios de projeto para uma wiki:

- **Aberta**: qualquer um pode editar e arrumar uma página que possa estar incompleta ou mal organizada,;
- **Incremental**: as páginas podem citar outras páginas, mesmo que ainda não tenham sido criadas;
- **Construtivo**: a estrutura e o conteúdo do texto na página são abertos à edição e evolução;
- **Mundano**: um pequeno número de (irregulares) convenções de texto permite acesso à marcação de página mais utilizada;
- **Universal**: como os mecanismos de edição e organização são os mesmos, qualquer pessoa é automaticamente um editor e organizador;
- **Evidente**: a saída formatada (e impressa) irá sugerir a entrada requerida para reproduzi-la;
- **Unificada**: nenhum contexto adicional será requerido para interpretar os nomes das páginas;
- **Preciso**: os nomes das páginas serão precisos o suficiente para evitar conflitos de nome;
- **Tolerante**: comportamento interpretável (mesmo quando indesejado) é preferido a mensagens de erro;
- **Observável**: atividades na página podem ser observadas e revisadas por qualquer visitante da mesma;
- **Convergente**: a duplicação pode ser desencorajada ou removida através da localização e citação de conteúdo similar ou relacionado.

O Apêndice A contém um quadro, no qual estão relacionadas implementações de wiki que foram

estudadas. As wikis constantes no quadro foram extraídas a partir de uma lista de wikis disponível em Cunnigham (2004). A CoTeia é uma dessas wikis, baseada na CoWeb (desenvolvida e utilizada na *Georgia Tech*). A CoTeia é utilizada por muitos professores e alunos, atualmente no Instituto de Ciências Matemáticas e Computação (ICMC) da Universidade de São Paulo (USP), campus de São Carlos. A seção 3.3 descreve a CoWeb e a seção 3.4, a CoTeia.

## 3.3. CoWeb

CoWeb (***Co****llaborative* ***Web****site*) (GUZDIAL, 1999; GUZDIAL et al., 2001; GUZDIAL; RICK; KEHOE, 2001; RICK et al., 2001, 2002) foi o nome usado para a ferramenta *open source* que possibilita a criação de *websites* colaborativos (*open authoring*). A ferramenta permite a criação e a modificação de páginas na *Web* através de um *Web browser*, sem a necessidade de *software* especializado no lado do cliente.

Cada página CoWeb se parece e se comporta como uma página *Web* normal, exceto que cada página tem um conjunto de botões no topo que permite ao usuário fazer várias coisas, tais como editar uma página, bloquear/desbloquear uma página, ou visualizar o histórico de alterações da página. O *link* "Edit" abre uma página com uma área de texto para renomear a página e uma área de texto grande para o conteúdo da página. Ao clicar no *link* "*Save*" o conteúdo da página é armazenado pelo autor. Uma página CoWeb pode conter qualquer tipo de mídia ou formato que qualquer outra página *Web* suporta e que os *Web browsers* possam apresentar. As imagens podem também ser incorporadas em uma página CoWeb usando a técnica usada na criação de *links*. Em versões mais recentes da CoWeb, as imagens podem ser carregadas (*upload*) diretamente na CoWeb e depois referenciadas de forma simplificada (por exemplo, *!myimage.gif!*). Uma página de "Mudanças Recentes" também está disponível, mostrando as páginas da CoWeb por título e por dia em que foram alteradas, em ordem cronológica decrescente (por exemplo, as alterações feitas hoje estão no topo da lista). Funciona como mecanismo de alerta aos usuários, a fim de indicar quando uma página existente foi alterada ou uma nova página foi criada.

A idéia básica é que qualquer usuário possa editar qualquer página e que qualquer usuário possa criar novas páginas, com *links* de e para qualquer outra página. O conceito deste tipo de autoria colaborativa baseada na *Web* foi desenvolvido por Ward Cunningham na WikiWikiWeb (CUNNINGHAM; LEUF, 2001 apud GUZDIAL et al., 2001; RICK et al., 2002). A CoWeb foi desenvolvida como um tipo de WikiWikiWeb e tem sido adotada por várias instituições, tanto acadêmicas como profissionais

(GUZDIAL et al., 2001).

A CoWeb foi implementada por Mark Guzdial em *Squeak* (INGALLS, 1997 apud GUZDIAL, 2001; GUZDIAL, 1999), uma linguagem de programação *open source* e altamente portável da linguagem de programação SmallTalk - assim, a versão inicial da CoWeb era conhecida como Swiki (de *Squeak* + Wiki). O servidor *Web* utilizado para executar a CoWeb foi o Comanche (*open source*), e que torna possível desenvolver aplicações *Web* inteiramente em *Squeak*, sem a necessidade de rodar um servidor *Web* externo.

A CoWeb oferece muito pouco de segurança, apesar da possibilidade de restauração de uma página a qualquer momento. Mesmo que exista a opção de bloqueio (*lock*), para evitar a edição por pessoas não autorizadas, na prática são poucas as páginas bloqueadas, pois as pessoas normalmente não destroem a contribuição de outras. Na Wiki original, existiam pessoas com a função de "*housekeeper*", que asseguravam a restauração de um conteúdo importante apagado, dando uma natureza colaborativa à tarefa de proteção.

Existem três grandes categorias usadas para descrever os tipos de atividades inventadas para a CoWeb (GUZDIAL et al., 2001):

- **Distribuição de informação**: a CoWeb é o lugar de armazenamento da informação para informação coletada. Funciona basicamente como uma lousa onde todos podem escrever e deixar informações para outras pessoas;
- **Criação colaborativa de artefatos**: a CoWeb é o veículo de criação e o instrumento para suportar a criação de um ou mais artefatos;
- **Discussão e revisão**: a CoWeb é o meio para a discussão e revisão de tópicos, artefatos ou idéias.

Segundo Arruda Jr e Pimentel (2001), a ferramenta CoWeb apresenta algumas limitações na sua infra-estrutura, tanto no aspecto de autoria colaborativa quanto no aspecto de ferramenta de *Computer Supported Cooperative Learning* (CSCL). As principais limitações identificadas em uma investigação do uso da CoWeb no ICMC são (MACEDO; BULCÃO NETO; PIMENTEL, 2001 apud ARRUDA JR; PIMENTEL, 2001):

- **Inexistência de controle de acesso**: a CoWeb faz uso da infra-estrutura da Internet, ficando vulnerável à ação de pessoas mal intencionadas, uma vez que todos os usuários Web possuem os mesmos direitos de acesso sobre as páginas;
- **Deficiência no controle de concorrência**: não existem mecanismos para resolução ou prevenção de eventuais colisões durante a edição simultânea de uma página por usuários diferentes - o usuário que salvar sua edição por último determina a versão atual da página, podendo acarretar a perda de informações;

- **Inexistência de base de dados**: não é possível o reuso da informação, uma vez que a referência a páginas em diferentes repositórios é feita do mesmo modo que uma ligação externa qualquer – o suporte de um banco de dados poderia minimizar esta limitação;
- **Funcionalidades**: é possível citar algumas funcionalidades desejáveis para a CoWeb, entre elas, permitir que o sistema de busca obtenha dados com maior precisão e atue entre as várias *swikis* da CoWeb, suportar interfaces alternativas para a operação de *upload* e implementação de um sistema de gerenciamento para as diferentes versões das páginas mantidas em um repositório;
- **Limitações de recursos de navegação**: os usuários que não utilizam recursos navegacionais do *browser* (*backward*, *forward*, *bookmark*s, *hot lists*, barra de endereço) sentem dificuldade no ambiente, uma vez que não existem recursos suficientes de navegação na CoWeb. O usuário pode experimentar a sensação de desorientação comum a usuários que navegam em hiperdocumentos (CONKLIN, 1987 apud ARRUDA JR; PIMENTEL; IZEKI, 2002) ao não conseguir visualizar sua posição atual dentre os hiperdocumentos navegados, o que pode ser resolvido, por exemplo, com um mapa do repositório.

## 3.4. CoTeia

A CoTeia é uma wiki desenvolvida com base na CoWeb e vem sendo utilizada no ICMC-USP desde janeiro de 2001, sendo aprimorada desde então, com a inclusão de novas funcionalidade, de acordo com as necessidades do usuários. Foi implementada sobre plataforma Linux, com um servidor Web Apache estendido com um interpretador PHP e um servidor de banco de dados relacional MySQL (ARRUDA JR; PIMENTEL, 2001; ARRUDA JR; PIMENTEL; IZEKI, 2002). A implementação foi construída totalmente independente da versão original da CoWeb, baseada em *Squeak*. Na implementação também foram usados XML, DTD, folha de estilo XSLT, XHTML, processador Java XT e linguagem ECMAScript (ARRUDA JR; PIMENTEL, 2001).

O modelo conceitual da CoTeia, definido em Arruda Jr e Pimentel (2001), e expresso por um Modelo Entidade-Relacionamento (MER) (Figura 5), indica que:

- Cada repositório de páginas é referenciado por uma entidade denominada *Swiki*, cadastrada por um administrador, representado pela entidade *Admin*. O administrador tem como atributos *login*, *email* e *password* e pode cadastrar várias *Swikis,* como indicado pelo relacionamento *Cadastra* de cardinalidade 1:N. Embora uma *Swiki* seja cadastrada por um único administrador, o mesmo pode cadastrar várias *Swikis*;

- A entidade *Páginas* é uma abstração de controle de hiperdocumentos, cujos atributos são *identificador*, *palavras-chave*, *autor*, *título*, *conteúdo*, *data*, *ip* da máquina criadora e *link* para reuso de páginas. A entidade *Páginas* se relaciona com a entidade *Swiki* através do relacionamento *Tem* de cardinalidade N:M, visto que uma *Swiki* pode ter várias páginas e uma página pode pertencer a várias *Swikis*;
- A transferência de arquivos é abstraída pela entidade *Uploads*, tendo como atributos um *identificador* e o *caminho do arquivo*. Esta entidade se relaciona com a entidade *Swiki* através do relacionamento *Referencia* de cardinalidade 1:N, pois um arquivo deve estar em uma única *Swiki*, e uma *Swiki* pode referenciar vários arquivos. Na implementação, esta entidade é armazenada no sistema de arquivos.

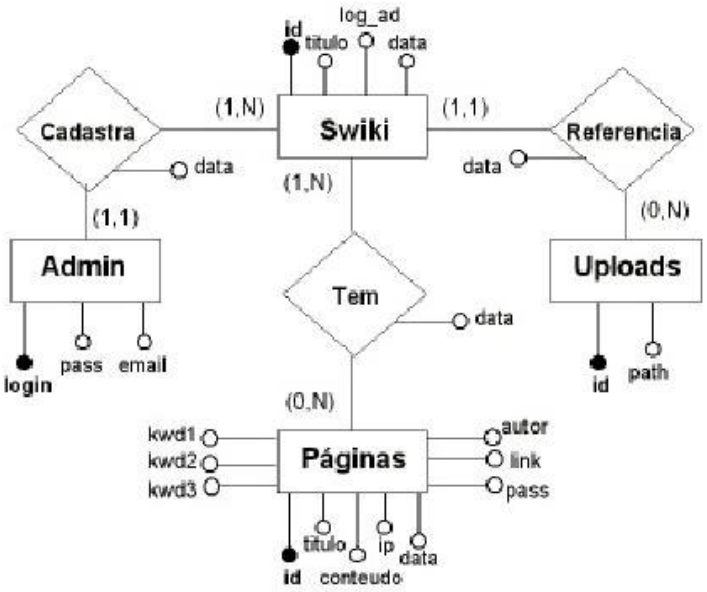


**Figura 5 -** Modelo Conceitual da CoTeia (ARRUDA JR; PIMENTEL, 2001).

As funcionalidades da CoTeia são (ARRUDA JR; PIMENTEL, 2001; ARRUDA JR; PIMENTEL; IZEKI, 2002):

- Criação e edição de páginas *Web* no próprio *browser*, por meio de formulários HTML Não é necessário conhecimento anterior dessa linguagem, e sendo possível o uso de *tags* pré-definidas pela CoTeia;
- Gerenciamento de repositórios *swiki* (como criação, remoção e modificação do nome de *swikis)* através de *login*, de uso exclusivo do administrador para cadastro do responsável pela *swiki*, com a definição do tipo de acesso ao repositório (público ou restrito) e a possibilidade do controle de edição concorrente de páginas;
- Lista das últimas modificações, onde é mostrada a última versão de cada página modificada em uma *swiki*;

- Histórico, onde os usuários podem acessar o conteúdo das últimas versões de cada página em uma *swiki*;
- Pesquisa (*search*), onde o usuário pode fazer busca por título, palavras-chave e/ou conteúdo dos hiperdocumentos em uma determinada *swiki*, ou em todas as *swikis* cadastradas na CoTeia;
- *Upload*, onde é possível fazer submissão de arquivos de qualquer formato ao servidor CoTeia, e também solicitá-los através de uma lista;
- Ajuda (*help*), que propicia auxílio aos usuários, principalmente quanto à sintaxe utilizada na criação e edição de páginas;
- Recursos de navegação, representado por um mapa do repositório, no qual o usuário pode navegar pelos hiperdocumentos existentes na ferramenta, diminuindo a sensação de desorientação, comum a usuários que navegam em hiperdocumentos (CONKLIN, 1987 apud ARRUDA JR; PIMENTEL; IZEKI, 2002);
- Edição concorrente de documentos, onde o controle de concorrência é possibilitado através de um conjunto de extensões para o protocolo HTTP, denominado WebDAV (GOLAND et al., 1999 apud ARRUDA JR; PIMENTEL; IZEKI, 2002);
- Serviço de *chat*, que possibilita aos usuários a troca de informações em tempo real sobre assuntos abordados em sala de aula ou reuniões;
- Serviço de anotações, onde os usuários podem trocar e compartilhar informações relacionadas ao conteúdo da página corrente, estendendo o espaço colaborativo de informações (anotações textuais em hiperdocumentos) (PIMENTEL et al., 2001b apud ARRUDA JR; PIMENTEL; IZEKI, 2002).

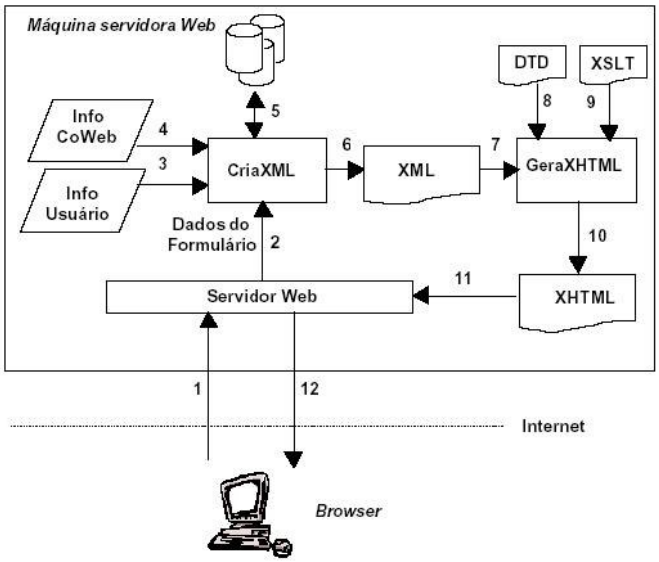


**Figura 6 -** Aplicação CoTeia (ARRUDA JR; PIMENTEL, 2001).

A criação de páginas colaborativas corresponde à base da ferramenta CoTeia, possibilitada através de um mecanismo de criação, transparente ao usuário, que permite que essas páginas sejam criadas, armazenadas, interpretadas, e finalmente visualizadas (ARRUDA JR; PIMENTEL, 2001). Isso é possível através da inserção dos dados pelo usuário em um formulário HTML, com opções de entrada para título, autor, palavras-chave e conteúdo, que são submetidos através do método *POST* (passo 1 da Figura 6) ao processo `CriaXML`, que faz:

- A integração com outras informações (dados dos usuários - como *login* ou *ip* da máquina; e informações do ambiente - como data de criação do documento ou título da *swiki*);
- A padronização de caixa baixa das *tags* para ficarem em conformidade com o DTD;
- A conversão de entidades (caracteres de acentuação e símbolos especiais) em entidades HTML, e posteriormente validadas pela gramática definida pelo DTD;
- A criação de ligações definidas pelos usuários através de diretrizes próprias, como *tags* <lnk>, inseridas no formulário. Consequentemente, realiza-se o seguinte procedimento:
  - extração do conteúdo *X* da *tag* `<lnk>`
  - busca ao identificador *Y* da *swiki* através da consulta à base de dados (passo 5)
  - comparação de *X* com os títulos das páginas cadastradas em *Y*
  - **Se** *X* já foi cadastrado:
    - retorno do índice que aponta para o hiperdocumento indicado pelo *link*
    - geração da âncora HTML correspondente ao hiperdocumento
  - **Senão**:
    - retorno de um novo índice para o documento cadastrado
    - direcionamento do *link* para a interface de criação de hiperdocumentos

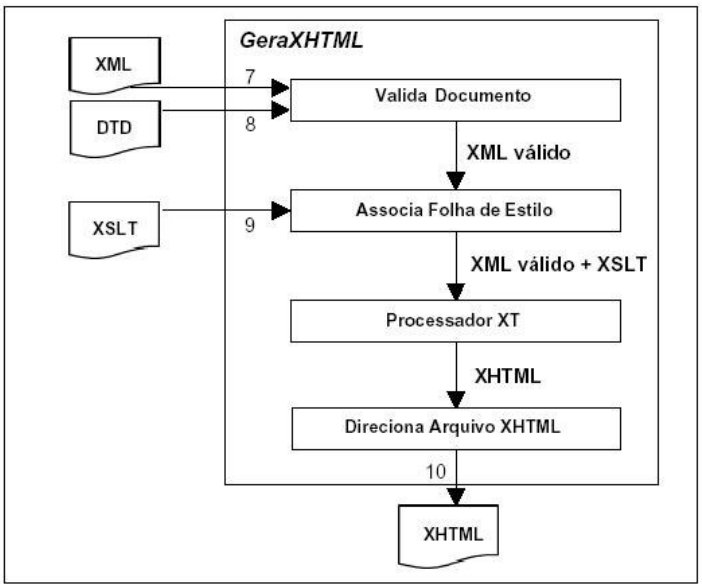


**Figura 7 -** Processo GeraXHTML (ARRUDA JR; PIMENTEL, 2001).

Ao final do processo *CriaXML*, é ativado (passo 7) o processo *GeraXHTML*, detalhado na Figura 6, que tem como sub-processos:

- Validação do documento: associa o documento XML a uma gramática (*DTD*) para validá-lo (passo 8);
- Associação de folha de estilo (passo 9): especifica e associa uma folha de estilo *XSLT* ao documento XML validado para o tratamento da apresentação;
- Processamento: envia os arquivos *XML* validado e folha de estilo *XSLT* ao processador *Java XT*, obtendo como saída (passo 10) um arquivo físico *XHTML*. A principal vantagem em utilizar o processador XT no servidor consiste em tornar a CoTeia uma aplicação independente de *browser*, uma vez que são enviados documentos XHTML ao usuário, formato reconhecido por todos os *browsers* do mercado. Como os processadores XML/XSLT das ferramentas de navegação atuais ainda não fazem o processamento de modo eficiente, é um diferencial importante a permissão de que os usuários não tenham qualquer tipo de restrição de plataforma.

Com o processamento terminado (passo 11), o arquivo *XHTML* é devolvido ao *browser* (passo 12), sendo visualizado pelo usuário. Ainda, a requisição do arquivo físico para visualização influencia positivamente no desempenho da aplicação, uma vez que o processamento indicado na Figura 6 e na Figura 7 realiza-se somente a cada edição e/ou criação de novos hiperdocumentos.

## 3.5. *Design Rationale* e Ferramenta DocRationale

O *Design Rationale* (DR) refere-se a explicações de "como" e "porquê" um artefato foi projetado de uma determinada forma, ou seja, é o conjunto de informações que descreve o raciocínio que justitfica o projeto resultante (GRUBBER; RUSSEL, 1991; MORAN; CAROL, 1996 apud FRANCISCO, 2003). Lee (1997) considera que DR abrange não apenas as decisões de um projeto, mas também as justificativas, alternativas e argumentações que levaram às decisões.

Os sistemas de DR visam a fornecer suporte por meio de algum auxílio ao processo, capturando o conhecimento, as suposições e as razões das decisões de projeto, de modo que essas informações possam ser úteis aos desenvolvedores no futuro. Segundo Hu et al. (2000 apud FRANCISCO, 2004), um sistema de DR tem por objetivo fazer os desenvolvedores pensarem sobre o projeto e discutirem dentro de uma determinada estrutura da representação de conhecimento, sendo que as funcionalidades esperadas para este tipo de sistema são:

- **Captura**: como obter e converter em um formato lógico os tipos de informação sobre as decisões

de projeto. Decisões e razões consideradas durante o processo de tomada de decisão podem ser armazenadas pelos desenvolvedores ou capturadas por um módulo de monitoração do sistema;

- **Representação**: refere-se ao uso de um esquema para representar o DR a ser armazenado para que possa ser utilizado posteriormente;
- **Recuperação do DR capturado**: responsável pelo fornecimento do DR adequado ao contexto do projeto.

A escolha do esquema de representação utilizado pode ser feita observando-se a perspectiva de DR. São três as diferentes perspectivas encontradas, segundo Shipman e McCall (1997 apud FRANCISCO, 2004):

- **Argumentação**: sob essa perspectiva, DR é tido como a argumentação (os diversos pontos discutidos, os argumentos apresentados e como foram selecionados) que os projetistas usam na formulação e solução de problemas. Tem por finalidade expressar o raciocínio dos desenvolvedores, particularmente para melhorar os argumentos que eles usam. A representação da perspectiva de argumentação é feita por meio de nós (elementos da "discussão") e *links* (relacionamento entre os elementos) (BACELO; BECKER, 1996 apud FRANCISCO, 2004), que possibilitam uma maior flexibilidade. Porém, essa última forma de representação acaba acarretando um certo *overhead*, devido à necessidade de classificação dos nós e *links* em abstrações e relacionamentos, respectivamente, pré-estabelecidos do modelo de esquema de representação;
- **Comunicação**: sob essa perspectiva, DR é a captura e recuperação de toda e qualquer comunicação ocorrida entre os desenvolvedores, por meio das mais variadas mídias – áudio, vídeo, *e-mail*, anotações, desenhos, etc. Tem por finalidade armazenar toda comunicação possível entre os desenvolvedores, capturando as "discussões" e a evolução do projeto conforme ocorrem;
- **Documentação**: sob essa perspectiva, DR é o registro da informação sobre decisões de projeto – que decisões são feitas, quando elas foram feitas, quem as fez e porquê. Tem por finalidade armazenar, de maneira clara, as decisões que foram tomadas durante a "reunião", para que a documentação gerada seja usada tanto para compreensão do projeto por pessoas externas à equipe de desenvolvimento quanto para obtenção e defesa de patentes. Usa um esquema linear de representação - os documentos.

Mesmo sendo diferentes, as três perspectivas não são contraditórias, nem totalmente independentes. A captura das informações é feita de forma ordenada, estruturada nas perspectivas de documentação e argumentação, facilitando o armazenamento e a recuperação do DR, ao passo que na perspectiva de comunicação, há dificuldades na recuperação das informações, pois não são capturadas de forma ordenada e o armazenamento não é feito de acordo com as necessidades esperadas para uma recuperação adequada. No que diz respeito ao volume de informação armazenado, a perspectiva de

documentação é a que armazena menos informações, pois consiste em um resumo da "reunião" dos desenvolvedores, registrando apenas as questões que foram discutidas, suas soluções, quando foram feitas, quem as fez, sem todos os detalhes. A perspectiva de comunicação é a que tem um volume maior de armazenamento, pois além das informações contidas na documentação, possui informações sobre os argumentos de cada questão levantada, e pontos a favor e contra às argumentações, o que pode ser útil na formação de melhores argumentos, além de possibilitar a detecção e correção de deficiências do projeto. As perspectivas de argumentação e comunicação capturam o DR no momento da discussão entre os desenvolvedores, o que pode garantir a fidelidade do conteúdo armazenado ao DR real, enquanto a perspectiva de documentação faz a captura do DR após a reunião dos desenvolvedores, de forma resumida, o que pode ocasionar o armazenamento de alguma informação incorreta.

Quanto mais estruturado estiver o DR armazenado no repositório de dados, melhor será sua recuperação. Hu et al. (2000 apud FRANCISCO, 2004) considera que os modelos de esquemas de representação mais conhecidos da perspectiva de argumentação, além de considerá-los muito parecidos (diferenciando-se nos tipos de nós e *links*), são:

- *Issue-Based Information System* (IBIS);
- *Questions*, *Options and Criteria* (QOC);
- *Decision Rationale Language* (DRL);
- *Procedural Hierarchy of Issues* (PHI)

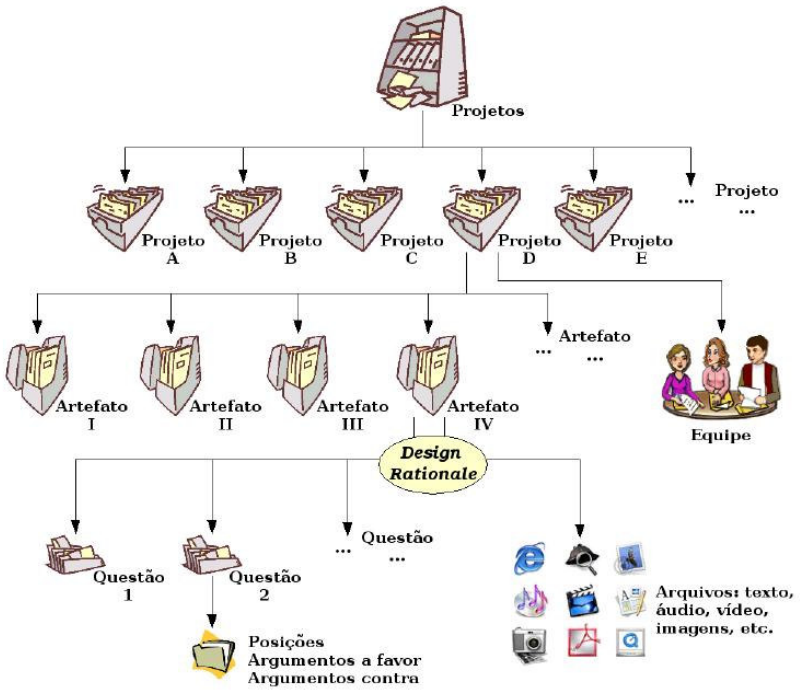


**Figura 8 -** Informações a serem manipuladas na ferramenta DocRationale (FRANCISCO, 2004).

A ferramenta DocRationale, desenvolvida por Francisco (2004), tem por finalidade permitir a captura, armazenamento e recuperação (por meio de navegação simples) de DR relacionado aos artefatos de software. A representação de DR utilizada foi simples, a fim de facilitar o uso e não acarretar muito esforço aos desenvolvedores. Não são usadas técnicas de Inteligência Artificial, pois o objetivo é a documentação de informações de um projeto visando o reuso das mesmas em outros projetos. Para que o DR seja obtido e registrado na ferramenta, é necessária a colaboração entre os membros da equipe de desenvolvimento. Também faz uso de anotações (estruturadas por uma representação simplificada do modelo PHI), consideradas por Souza et al. (1998 apud FRANCISCO, 2004), um bom meio de captura de DR.

Para a construção da DocRationale foi usada a abordagem orientada por processo, a fim de que fosse possível a inserção de projetos sem a necessidade de saber quais DRs seriam capturados e recuperados, possibilitando ao usuário a inserção/exclusão de fases, atividades e artefatos durante todo o desenvolvimento.

Na construção da DocRationale também foi usada uma combinação das perspectivas de comunicação e de argumentação, com o objetivo de que a captura e o armazenamento proporcionassem uma recuperação satisfatória do DR armazenado. A perspectiva de comunicação possibilita que todas as formas de comunicação digital (arquivos de áudio, vídeo, *e-mail*, etc) sejam anexadas, de forma organizada, complementando a documentação do processo. Para a perspectiva de argumentação foram utilizadas as características do esquema de representação PHI (SHUM, 1991 apud FRANCISCO, 2004), devido à simplicidade na representação de DR e à estruturação no armazenamento do DR, capturado de forma hierárquica e ordenado cronologicamente. O esquema utilizado na DocRationale foi simplificado para utilizar apenas três tipos de nós - questões, posições (respostas) e argumentos – para que houvesse uma captura com intervenção do usuário. A navegação foi o esquema usado para a recuperação. Para que a DocRationale fornecesse suporte ao DR de artefatos de software, para cada projeto, diversos artefatos deveriam ser elaborados, os quais seriam determinados por uma série de questões, posições e argumentos, além de outros arquivos que representem o DR.

Francisco (2004) constatou que cada projeto possui uma equipe composta por desenvolvedores e um gerente responsável pelo desenvolvimento e com base nisso, a estruturação das informações na DocRationale pode ser visualizada na Figura 8. Para evitar que qualquer pessoa tenha acesso às informações armazenadas na DocRationale, a ferramenta tem controle de acesso implementado, o qual tem também restrição de acesso a informações, dependendo das permissões dadas ao usuário, que variam de acordo com a função exercida dentro da equipe. São quatro os tipos de usuários definidos dentro da ferramenta:

- **Administrador de usuários**: responsável pelo gerenciamento dos usuários da DocRationale;

- **Administrador de Projetos**: responsável pelo gerenciamento de projetos, artefatos e questões;
- **Membro do Projeto**: pode inserir e atualizar artefatos, inserir questões e também, navegar sobre projetos, artefatos e questões;
- **Visitante**: apenas pode navegar sobre projetos, artefatos e questões.


**Figura 9 -** Casos de uso da Ferramenta DocRationale (FRANCISCO, 2004).

A ferramenta foi desenvolvida para o ambiente Web, em linguagem PHP, usando o servidor de banco de dados MySQL, a fim de que os usuários pudessem acessá-la remotamente, assim todos podem usá-la mesmo que não estejam no mesmo espaço físico. Além disso, a DocRationale foi integrada com a CoTeia (ARRUDA JR; PIMENTEL; IZEKI, 2002) e com o GroupNote (PIMENTEL; IZEKI; ARRUDA JR, 2001a apud FRANCISCO, 2004), o que possibilitou o apoio ao suporte a DR, principal funcionalidade da DocRationale. Dentre as funcionalidades da CoTeia, as mais interessantes para a DocRationale são: (i) criação e edição de páginas Web, que são feitas no próprio *browser* via formulário HTML; (ii) histórico, que permite o acesso ao conteúdo das últimas versões de cada hiperdocumento; e (iii) *upload*, que possibilita a submissão de arquivos de qualquer formato ao servidor CoTeia.

O GroupNote é um serviço aberto de anotações colaborativas na Web, implementado como uma API, sendo que as características interessantes para a DocRationale são: (i) compartilhamento de anotações

por grupos de usuários; e (ii) fornecimento da funcionalidade de hierarquia de nós. Uma anotação é formada por um título, um tipo (posição ou argumento), palavras-chave, formato (texto/html), o documento anotado, a porção do documento anotado, permissões de acesso e outras informações (quem - usuário e grupo a criou e quando). Uma anotação pode conter outra, formando um *thread* de discussão. Para que o GroupNote pudesse ser usado na DocRationale, para prover discussões sobre cada questão levantada pelo usuário-desenvolvedor, foram feitas pequenas modicações em sua API, trocando o banco de dados de usuários do GroupNote pelo banco de dados de usuários da DocRationale.

Assim, com a integração da DocRationale com a CoTeia, é possível a autoria colaborativa e inclusão de anexos (DR), e com a integração da DocRationale com o GroupNote, obtem-se a qualificação das anotações como questão, posição e argumento contra ou a favor.

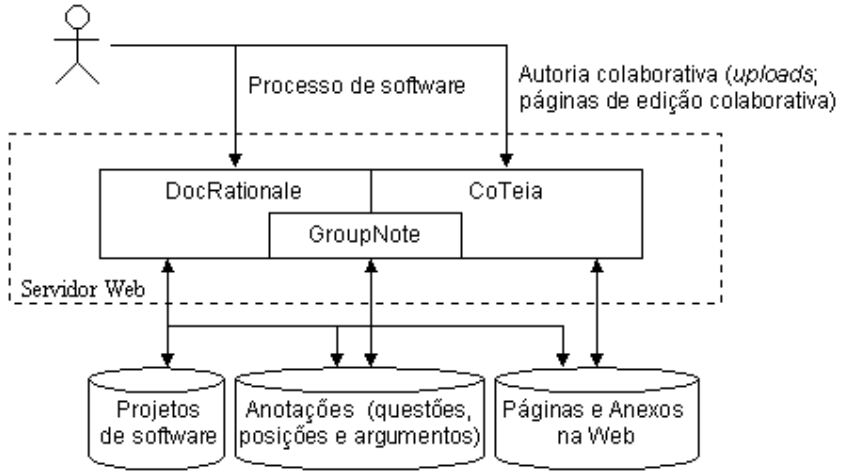


**Figura 10 -** Integração da DocRationale com a CoTeia e GroupNote (FRANCISCO, 2004).

O diagrama de casos de uso da DocRationale (**Erro! A origem da referência não foi encontrada.** mostra as funcionalidades da ferramenta e também as relações da mesma com as entidades externas. Uma visão geral da arquitetura da DocRationale é mostrada na Figura 10, que denota a integração da DocRationale com a CoTeia e o GroupNote por meio dos blocos unidos. É possível observar também que tanto a DocRationale quanto a CoTeia fazem uso das anotações do GroupNote, assim como a DocRationale faz uso das páginas e anexos da CoTeia.

As relações entre a DocRationale, a CoTeia e o GroupNote foram estabelecidas antes da implementação da DocRationale, devido à necessidade de integração da DocRationale para utilização dos recursos da CoTeia e do GroupNote. Na Figura 11 é possível observar que a CoTeia e a DocRationale possuem telas correspondentes para projetos, artefatos e questões, possibilitando o acesso às informações tanto pela DocRationale quanto pela CoTeia.

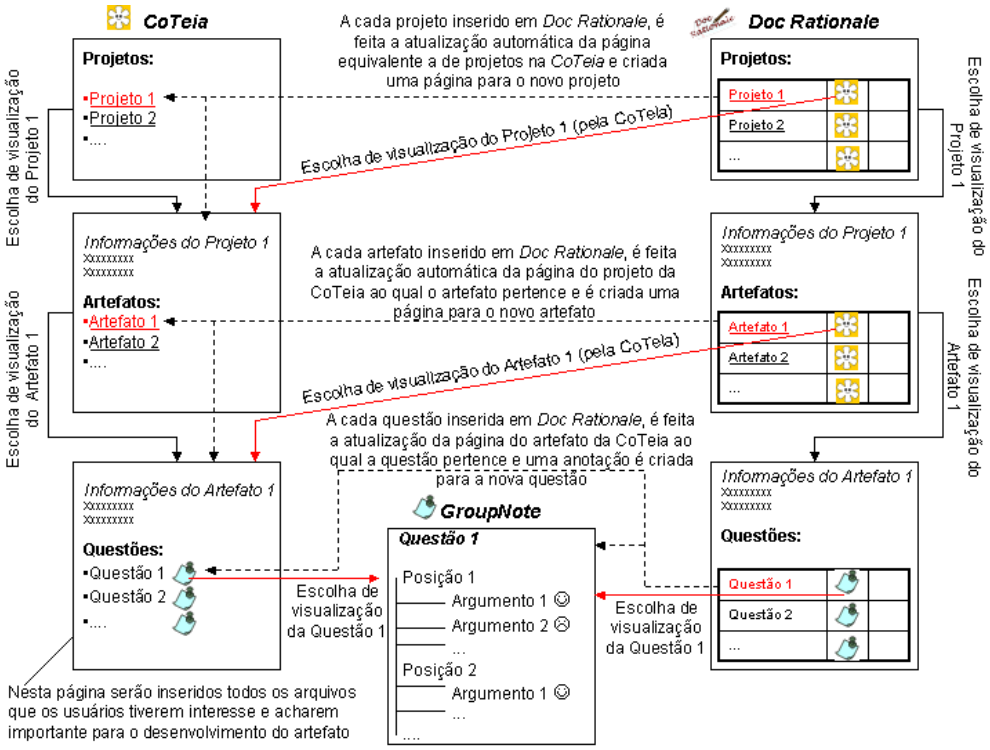


**Figura 11 –** Ligações entre DocRationale, CoTeia e GroupNote (FRANCISCO, 2004).

**Quadro 3** - Modificações na CoTeia e GroupNote para integração com a DocRationale.

| CoTeia | GroupNote |
|---|---|
| • Opção de anotação do menu foi desabilitada para todas as páginas;<br>• Opção de *upload* na página relacionada aos projetos foi desabilitada;<br>• O código do *upload* foi alterado para não inserir diretamente o arquivo na página do artefato (página não editável) | • Substituição da base de dados de usuários pela base de dados de usuários da DocRationale (para discussões sobre cada questão levantada pelo usuário desenvolvedor);<br>• Troca dos tipos de nós, com dois níveis de discussão: no primeiro nível, uma posição é colocada em relação à questão levantada; no segundo nível, usuários argumentam contrariamente ou a favor da posição |
| Inserção de recursos de autenticação, sessão e permissão para a CoTeia e GroupNote para que apenas usuários autorizados tivessem acesso às informações armazenadas. | |

A DocRationale é responsável pela criação e atualização das páginas encontradas na CoTeia, através de funções que transmitem os dados para funções da CoTeia e GroupNote (por exemplo, criação de

*swikis* e páginas). Com isso, a CoTeia é atualizada a cada inserção de um projeto ou artefato na base de dados da DocRationale, com a criação de um *link* na DocRationale para a página correspondente criada na CoTeia. Da mesma forma, para cada inserção de uma questão na base de dados da DocRationale, uma anotação é criada no GroupNote, além de ser feita a atualização da página CoTeia - as anotações podem ser visualizadas por *links*, tanto na DocRationale quanto na CoTeia. Os arquivos de DR e artefatos são armazenados na CoTeia.

Quadro 3 contém as modificações que foram necessárias para que a integração da CoTeia e do GroupNote fosse viabilizada.

## 3.6. Considerações Finais

Neste capítulo, os tópicos referentes ao estado da arte no que diz respeito à edição colaborativa, à visualização de repositórios CVS e à DR foram apresentados.

As ferramentas para visualização de repositório CVS têm facilitado a visualização do conteúdo dos repositórios através da interface gráfica em que são disponibilizadas, ao contrário da interface *default* do CVS. Além disso, oferecem diferentes formas de visualização, assim como recursos para comparação das versões. Algumas delas têm sido desenvolvidas para uso na *Web*, facilitando a distribuição e o uso da ferramenta – a instalação é feita em um servidor e o requisito para que os usuários possam utilizá-la é a instalação de um *Web browser* na máquina do usuário. *Websites* como Sourceforge e Freshmeat fazem uso deste tipo de ferramenta: o primeiro usa a ferramenta *ViewCVS* e o segundo a ferramenta CVSweb. Percebeu-se, durante o estudo realizado, que a ferramenta *ViewCVS* tem tido bastante aceitação nos meios acadêmicos e empresariais, o que levou à decisão de sua utilização no desenvolvimento do trabalho.

Uma categoria de ferramenta bastante utilizada na edição colaborativa é a wiki. Criada inicialmente por Ward Cunningham, este tipo de ferramenta se propõe a permitir a colaboração por meio de *websites* construídos especificamente para isso. A partir da Wiki de Cunningham, várias implementações foram desenvolvidas – a CoTeia é uma delas. Desenvolvida a partir da CoWeb, uma wiki desenvolvida na *Georgia Tech*, a CoTeia passou a ser usada no ambiente acadêmico do ICMC e também na ferramenta DocRationale, que auxilia o registro de DR.

# Capítulo 4

# Análise do Controle de Versões como apoio à Edição Colaborativa

Neste capítulo é apresentado, na seção 4.1, um levantamento de dados realizado durante o desenvolvimento do trabalho. O objetivo deste levantamento é a obtenção de um panorama das wikis existentes atualmente no que diz respeito à possibilidade de *uploads* e ao uso de controle de versões (para as páginas e para os *uploads*). Em seguida, na seção 4.2 é relatado o desenvolvimento deste trabalho – a associação do controle de versões à edição colaborativa. Finalmente, na seção 4.3 são apresentadas as considerações finais relacionadas a este capítulo.

## 4.1. Levantamento de Dados

A fim de se estabelecer um panorama das wikis em desenvolvimento/em uso atualmente, no que diz respeito ao controle de versões e ao uso do recurso de *upload*, foi feito um levantamento em uma amostra de 79 wikis, obtida a partir de uma lista de wikis disponível em Cunningham (2004).

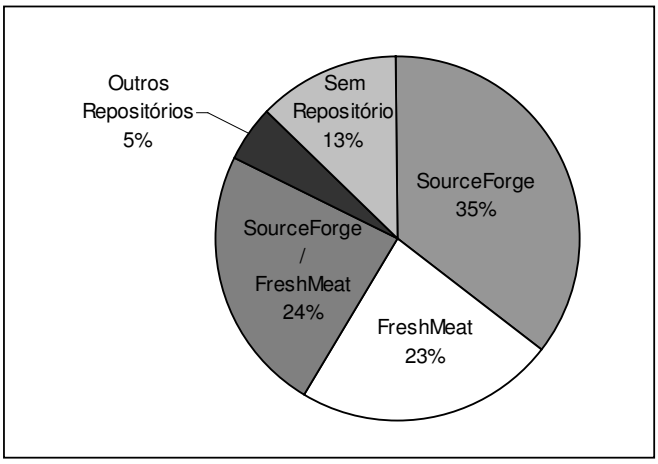


**Gráfico 1 -** Distribuição de wikis por repositório.

Devido à lista inicial ser muito extensa, a pesquisa foi restrita a wikis que possuísem licença de software livre, constatado por meio de pesquisa nos *websites* SourceForge e Freshmeat (28 wikis foram encontradas no Sourceforge, 18 wikis foram encontradas no Freshmeat, 19 wikis foram

encontradas em ambos os repositórios). Quando a wiki não era encontrada nesses *websites*, era então pesquisada usando a ferramenta de busca Google[26]: algumas foram encontradas em outros *websites* que funcionam como repositórios de software livre (4 wikis) e outras só foram encontradas em *websites* próprios (10 wikis). No Quadro 4 encontra-se um resumo das wikis pesquisadas e o Gráfico 1 mostra a distribuição das wikis por *website* de software livre.

**Quadro 4 -** Wikis selecionadas para estudo.

| | | |
|---|---|---|
| AspWiki | AsWiki | AtisWiki |
| AwkiAwki | CfWiki | Chiki Wiki |
| Chiq_Chaq | Cliki | Cloud Wiki |
| CocanWiki | CoTeia | CoWiki |
| CVSTrac | CyberPublishing | DevWiki |
| DiamondWiki | DidiWiki | DokuWiki |
| DotWiki *(.Wiki no SF)* | EditThisPagePhp | ErfurtWiki |
| FireWiki | FitNesse | FlexWiki |
| Friki | fuwiki – the community wiki | GeboGebo Wiki |
| GikiWiki | GracefulTavi | InstikiWiki |
| IoWiki | JASSWiki | JaWiki |
| JspWiki | Kwik Wiki | Kwiki Kwiki |
| Mase Wiki | MediaWiki *(usada pela Wikipedia)* | MiniRubyWiki |
| MiniWiki | MoinMoin | MoniWiki |
| NewLISP-Wiki | OddMuseWiki | PerSpective |
| PhearWiki | PhpWiki | PmWiki |
| PodWiki | PurpleWiki | PWP Wiki Processor |
| PyleWiki | PyWiki | QwikiWiki |
| Radeox | RuWiki | ScrumWiki |
| SeedWiki | SnipSnap | SpinnerWiki |
| TikiWiki CMS/Groupware | TikiWikiLite | TipiWiki |
| TwikiClone *(TWiki)* | UniWakka | UseModjWiki |
| UseModWiki | Very Quick Wiki | Wiclear |
| WikiNehesa | WiKit | Wikiwig |
| WikiX | WikkaWiki | WikkiTikkiTavi |
| Wiski | Xwiki | Yawiki |
| ZwiKi | | |

Após a seleção das wikis pesquisadas, foram definidas as características que seriam investigadas nas wikis (um quadro, contendo as wikis pesquisadas e suas respectivas características, pode ser encontrado no Apêndice A):

(i) Linguagem de desenvolvimento,

(ii) Recurso de *upload*,

(iii) Existência de controle de versões (tanto de páginas quanto de *uploads*);

(iv) Ferramenta de controle de versões utilizada (quando cabível).

[26] http:/www.google.com

Essas características (exceto a (i)) permitem uma análise dos recursos para controle das informações que são editadas (nas páginas das wikis) e disponibilizadas (por meio de *uploads*). A linguagem de desenvolvimento foi também coletada como informação que pode representar qual a tendência mais recente de desenvolvimento pela comunidade envolvida com projetos de software livre.

A primeira característica foi verificada nos *websites* de software livre (Sourceforge, Freshmeat, etc) ou na página da wiki. As demais características das wikis foram localizadas por meio de exploração do funcionamento de cada uma, possibilitado por meio do uso do *sandbox* (em algumas foi encontrado com o nome de *playground*), um recurso disponível nas wikis, consistindo em uma área pública para teste do funcionamento da wiki e disponibilizado com o intuito de evitar a "poluição" do conteúdo das páginas da wiki com testes: em cada wiki era verificada a possibilidade de *upload* e a existência de controle de versões nas páginas e nos *uploads*. Para as wikis que possuíam controle de versões nos *uploads*, a documentação disponível (manual, arquivos do tipo *Readme*, código-fonte, etc) foi analisada, com o intuito de descobrir qual a ferramenta de controle de versões utilizada (CVS, RCS, Subversion, etc).

É importante ressaltar que existem wikis, que embora fossem software livre, não foram pesquisadas por não possuírem o recurso de *sandbox* ou pelo mesmo estar disponível apenas para acesso registrado. O acesso registrado tem sido usado como tentativa de prevenção a ataques freqüentes de *spam*[27] nas wikis, embora seja contrário à idéia colaborativa da wiki. Um dos tipos de *spam* mais comuns é a inclusão de URLs no conteúdo das wikis para aumentar a classificação pelo Google do *website* referenciado (CUNNINGHAM, 2004).

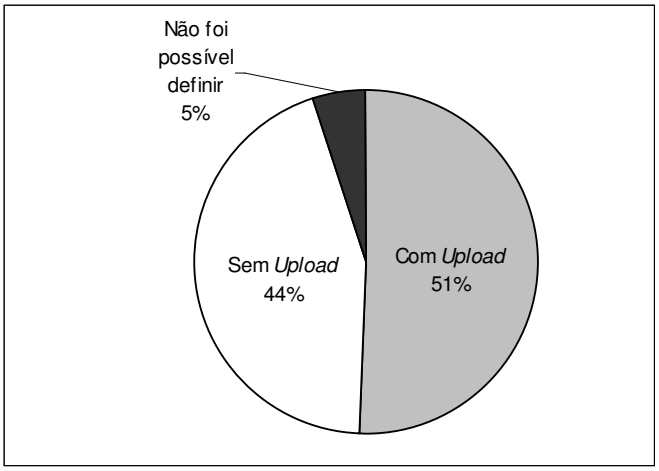


**Gráfico 2 -** Recurso de *upload* nas wikis.

De acordo com os dados coletados (Gráfico 2), o recurso de *upload* de arquivos não é consenso entre as wikis da amostra – 40 wikis (51%) possuíam este recurso, 35 (44%) não possuíam e nas demais (5%) não foi possível apurar. Quanto à existência de controle de versões (Gráfico 3), foi apurado que 76 (96%) wikis possuíam algum tipo de controle de versões nas páginas. Apenas 5 wikis usam o controle de versões também para os *uploads* (Gráfico 4): neste caso, foi detectado o uso da ferramenta RCS em 2 wikis, sendo que não foi encontrada wiki na amostra que usasse o CVS.

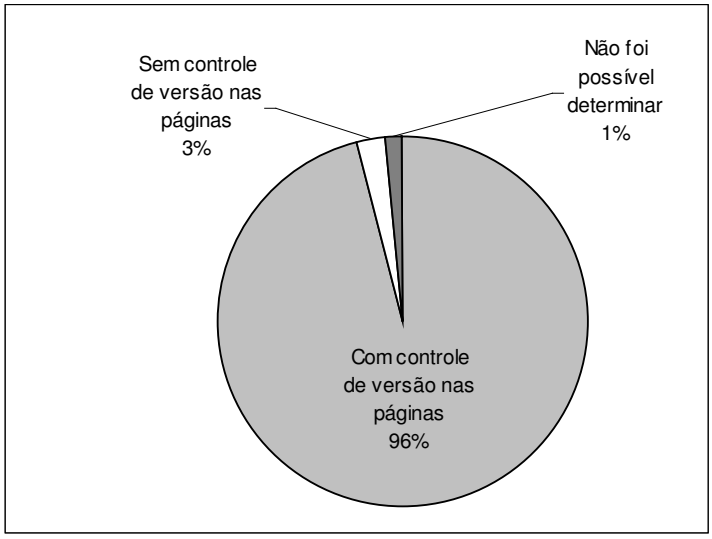


**Gráfico 3 -** Uso de controle de versões nas páginas.

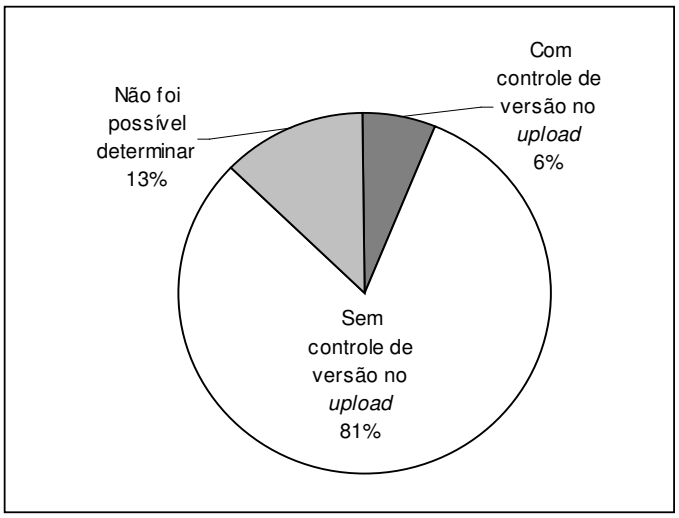


**Gráfico 4 -** Uso de controle de versões no *upload*.

Um aspecto observado nas wikis, no que diz respeito ao *upload*, é a restrição no tipo de arquivo submetido: possibilidade apenas de imagens (GIF, JPEG, etc) ou apenas de alguns tipos de arquivos.

[27] O termo "*spam*" denota o uso abusivo de comunicações eletrônicas.

O Quadro 5 é uma síntese do levantamento realizado, mostrando as wikis de acordo com os seus objetivos: edição colaborativa geral ou especializada. Cabe ressaltar que as wikis pesquisadas eram de caráter geral, dedicado à edição colaborativa. No caso da wiki especializada, enquadra-se a CoTeia usada na DocRationale.

**Quadro 5 -** Wiki geral e wiki especializada.

| Wiki (objetivos) | Controle de versões nas páginas | Upload | Controle de versões no *upload* |
|---|---|---|---|
| **Edição colaborativa geral** | **Sim – 96%** | **Sim – 51%** | **Sim – 6%** |
| | | | **Não – 81%** |
| | | **Não – 44%** | |
| | **Não – 3%** | | Indeterminado – 13% |
| | Indeterminado – 1% | Indeterminado – 5% | |
| **Especializada (DR)** | **Sim** | **Sim** | **Sim** |

Analisando o Quadro 5, tem-se que uma característica comum aos dois propósitos de wiki é o uso de controle de versões nas páginas. Já as características de *upload* e controle de versões do *upload*, que devem estar presentes na wiki especializada, não são necessariamente uma regra para as demais wikis.

Nota-se que nas wikis em geral, o recurso de *upload* não é consenso. O controle de versões no *upload* também tem tido uma utilização mínima. Mas dependendo do propósito da wiki, o controle de versões dos *uploads* pode ser um dos principais recursos. O Quadro 5 mostra o grau de uso de *upload* e controle de versões das páginas e dos artefatos em relação a wikis de uso geral e a presença dessas características na wiki especializada.

## 4.2. Implementação realizada

Foram duas as alterações realizadas na CoTeia: uma na versão da CoTeia que é usada na DocRationale, e a outra na CoTeia de uso acadêmico no ICMC. Apesar das duas versões manterem a natureza colaborativa, a visão das operações possíveis envolvendo o *upload* de arquivos é diferente. Como a DocRationale visa registrar o DR dos projetos durante o processo de desenvolvimento de software, juntamente com os artefatos relacionados, é importante o registro das alterações efetuadas nos mesmos – o registro das alterações do conteúdo das páginas já estava sendo feito, armazenado no CVS. Como o artefato é parte do DR, não é possível que seja removido, assim, a CoTeia da DocRationale seria acrescida do controle de versões dos artefatos, mas continuaria não sendo possível a remoção dos mesmos. Já a CoTeia em uso no ICMC, usada principalmente como repositório didático para as disciplinas ministradas, não tem a intenção de armazenar as alterações feitas nos *uploads*, mas necessita de um mecanismo de remoção, uma vez que não existe controle para submissão dos *uploads*. Em síntese, foram disponibilizadas duas novas funções na CoTeia, mas cada uma em versões diferentes: uma foi dotada de controle de versões nos artefatos e a outra foi dotada da possibilidade de remoção dos *uploads*.

### 4.2.1. CoTeia Especializada

A ferramenta DocRationale é integrada à uma versão da CoTeia, específica para uso no registro do DR e seus artefatos. Como o objetivo do DR é o registro das razões de um projeto, e na maioria das vezes ocorrem várias alterações durante o desenvolvimento do mesmo, é necessário o armazenamento dessas alterações. As alterações podem resultar em modificação dos artefatos e por isso é necessário o armazenamento também da evolução desses artefatos. Quando a DocRationale foi desenvolvida, as alterações nos artefatos podiam ser armazenadas, mas sem qualquer histórico e relacionamento entre elas.

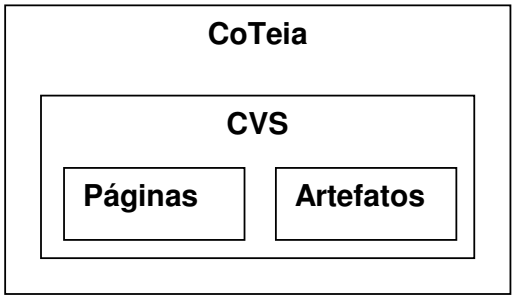


**Figura 12 -** Integração do CVS com a CoTeia para o armazenamento de páginas e artefatos.

Na CoTeia da DocRationale, foram alterados:

(i) Módulo de *upload*;

(ii) Módulo de edição das páginas da wiki;

(iii) Módulo de criação das páginas da wiki.

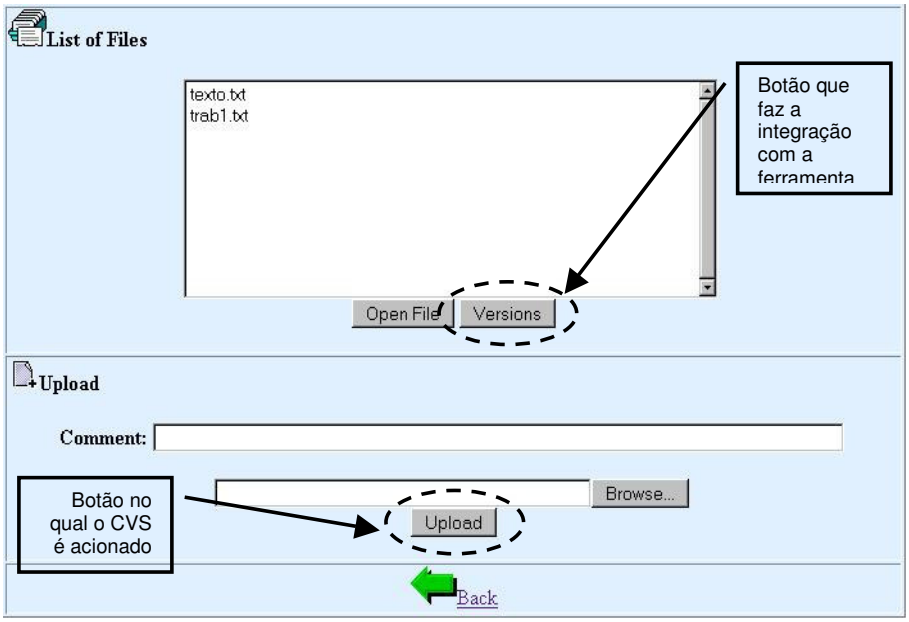


**Figura 13 -** CoTeia especializada: tela de *upload* com controle de versões.


**Figura 14 -** Interface de visualização do histórico de versões na ferramenta *ViewCVS*.

Cabe ressaltar que os dois últimos módulos ((ii) e (iii)) foram alterados porque, como o CVS já estava sendo utilizado para o controle de versões das páginas, foi usada a mesma biblioteca de funções, com as adaptações necessárias para uso com o *upload* de arquivos. Foram acrescentadas algumas

validações na tela de *upload*, a fim de garantir que os campos necessários para o armazenamento da versão do *upload* fossem todos preenchidos. Como o CVS possibilita o registro do *log* para cada versão, foi acrescentado também o campo de comentário, preenchido no momento do *upload* (Figura 13). Com isso, foi necessária a atualização de cinco aplicações escritas em PHP, a biblioteca de funções para interface com o CVS e também uma aplicação Javascript. A visualização do histórico de versões para cada artefato foi disponibilizada no módulo de *upload* por meio da integração da CoTeia com a ferramenta *ViewCVS* (Figura 14), que possibilita a visualização de repositórios CVS.

### 4.2.2. CoTeia Geral

Na versão da CoTeia integrada à DocRationale não é possível a remoção dos artefatos, uma vez que este recurso poderia prejudicar o registro da "memória" do DR. Entretanto, a remoção foi uma necessidade levantada entre os usuários da CoTeia em uso no ICMC: como cada *upload* de arquivos é feito com um nome de arquivo diferente, ou seja, mesmo que o arquivo submetido seja uma alteração de um arquivo já presente nos *uploads*, não é permitido uma nova operação de *upload* com o mesmo nome de arquivo. Isso faz com que, em algumas situações, existam várias cópias do mesmo arquivo na área de *uploads*, embora com nomes diferentes. Dentre as wikis pesquisadas, algumas fazem a sobreposição de arquivos, caso o arquivo submetido para *upload* seja encontrado no repositório. Entretanto, devido à natureza colaborativa da CoTeia e a inexistência de controle de acesso, optou-se pelo recurso de *upload* não ter o comportamento de sobreposição. Para que a remoção não seja usada de forma prejudicial, optou-se pelo uso de controle de acesso, sendo possível somente ao administrador da *swiki* a remoção dos arquivos.

Na CoTeia do ICMC, foi alterado o módulo de *upload* (Figura 15), além da criação de um módulo de autenticação (*login*) para concretização da operação de remoção. Como a CoTeia é colaborativa, colocar o recurso de remoção simplesmente poderia vir a trazer prejuízos ao conteúdo das wikis, uma vez que qualquer usuário poderia remover arquivos, a exemplo do que ocorre atualmente com os *uploads*. Assim, ficou definido que a operação de remoção só estaria disponível para o administrador de cada *swiki*, mediante *login* para validação do acesso.

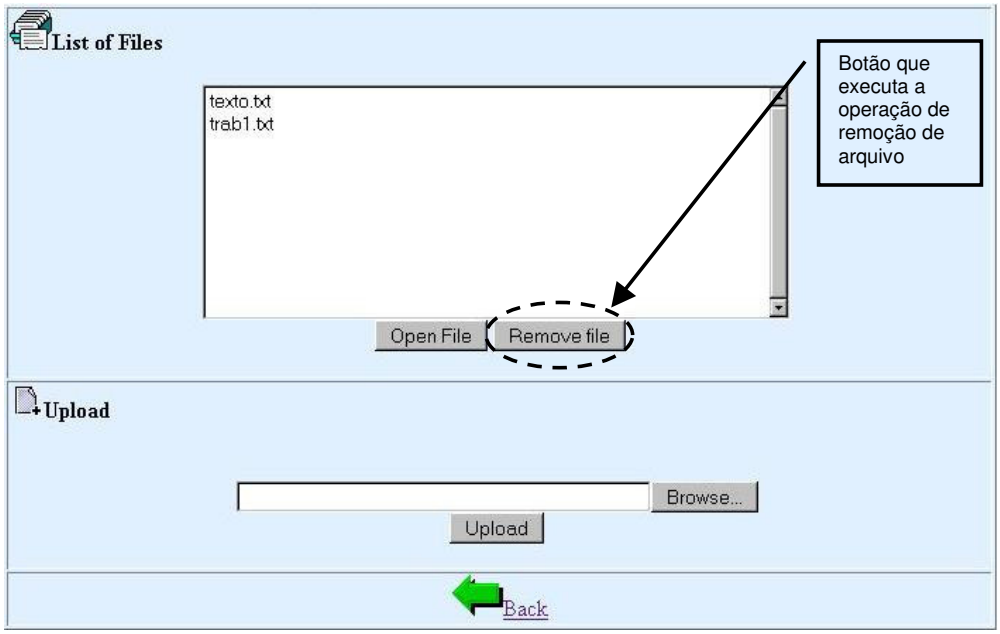


**Figura 15 -** CoTeia geral: tela de *upload* com o recurso de remoção.

## 4.3. Considerações Finais

Neste capítulo foram abordados o levantamento de dados sobre wikis e as implementações realizadas. O levantamento mostrou que o recurso de *upload* não é consenso entre as wikis, embora seja um recurso importante, dependendo de sua utilização. A ferramenta DocRationale usa a CoTeia para armazenamento dos artefatos do projeto, e isso não seria possível sem o recurso de *upload*.

Neste caso, além do *upload* ter sua relevância, foi notada a necessidade de um controle maior em cima das alterações realizadas nos artefatos. O uso de uma ferramenta de controle de versões integrada a esta CoTeia, possibilita saber as modificações efetuadas e quem as executou, sem contar que a inserção do artefato no controle de versões é automática, integrada ao processo de *upload*, para evitar sobrecarga nas tarefas do usuário. Futuramente, poderá ser possível a ligação entre as alterações efetuadas nos artefatos e o DR armazenado.

# Capítulo 5

# Conclusão

Como uma das atividades principais de GCS, o controle de versões tem obtido contínua atenção dos desenvolvedores, presente no esforço para garantia da qualidade de software. O controle de versões foi a base dos estudos deste trabalho e o seu objetivo – controlar a evolução dos produtos em desenvolvimento, foi investigado como potencial apoio às ferramentas atuais de edição colaborativa por grupos, via web.

Em ambientes de trabalho de autoria colaborativa, que contam com diversas pessoas responsáveis pela manutenção de um mesmo conteúdo, o controle de versões se torna um mecanismo importante, principalmente por viabilizar auditoria do que, quem e quando foram realizadas alterações. As ferramentas conhecidas como wikis são um exemplo atual de ambiente de autoria colaborativa na *Web*, pois se apresentam como um *website* que permite que o conteúdo de suas páginas possa ser editado e atualizado por várias pessoas. Além disso, em algumas wikis é disponibilizado o recurso de *upload*, para que arquivos também possam ser armazenados e relacionados às páginas (CUNNINGHAM, 2004).

Uma versão da CoTeia, wiki utilizada e desenvolvida no ambiente acadêmico do ICMC-USP, foi integrada à ferramenta DocRationale, a qual é utilizada no registro de *Design Rationale* (DR). Dessa forma, essa versão da CoTeia foi estudada como uma wiki especializada, pois o suporte de controle de versões dos produtos (artefatos) em desenvolvimento se apresenta como requisito imprescindível.

Na literatura, no entanto, não existe um consenso sobre a adoção de controle de versões em todos os recursos processados nas wikis (como *uploads*, por exemplo). Neste trabalho foi então realizado um levantamento de wikis e realizada uma análise para identificação do panorama dos desenvolvimentos e tendências das wikis atuais.

Na versão original da CoTeia verificou-se que, no momento, não haveria a necessidade de armazenamento das alterações efetuadas nos *uploads*. Entretanto, outra necessidade levantada foi o recurso de remoção de *uploads.* A remoção seria restrita apenas ao administrador da *swiki*, uma vez que a CoTeia não possui controle de acesso e a remoção indiscriminada poderia afetar o conteúdo das *swikis*. Por outro lado, a implementação da remoção na versão da CoTeia utilizada na DocRationale mostrou-se desnecessária, uma vez que o objetivo do DR é preservar a memória do processo de

desenvolvimento de software. Além disso, na Coteia especializada para utilização na DocRationale, foi realizada a integração com a ferramenta *ViewCVS*, para visualização das versões armazenadas dos artefatos.

Neste trabalho, foi então realizada a análise referente ao aperfeiçoamento da wiki – CoTeia – usada em duas instâncias diferentes: uma especializada, como parte da ferramenta DocRationale (captura, armazenamento e recuperação de DR na Web) e a outra (usada no ICMC) como apoio ao trabalho de edição por grupos, de maneira colaborativa na *Web*. Para isso, foram realizados estudos sobre controle de versões, ferramentas para controle de versões, ferramentas para visualização de repositórios CVS e estudo do panorama das wikis em desenvolvimento e/ou em uso, com a finalidade de selecionar ferramentas para a realização do trabalho e também, para verificar a necessidade das modificações propostas.

Finalmente, sumarizando as contribuições deste trabalho, temos:

- Estudo sobre Gerenciamento de Configuração de Software (GCS) e os principais conceitos relacionados;
- Estudo sobre Controle de Versões e de ferramentas que cumprem esta missão (CVS, RCS, *Arch* e *Subversion*);
- Levantamento de ferramentas (Quadro 1) para visualização de repositórios CVS;
- Estudo sobre wikis e coleta de dados (Apêndice A), estabelecendo um panorama das wikis em uso e em desenvolvimento atualmente;
- Integração do CVS à CoTeia, para controle de versões dos artefatos (*uploads*) armazenados na ferramenta DocRationale;
- Implementação do recurso de remoção com controle de acesso na CoTeia em uso no ICMC;
- Estudo da ferramenta *ViewCVS* (SILVA; IGNACIO JR; FORTES, 2004).

## 5.1. Trabalhos Futuros

Entre os trabalhos futuros que podem vir a ser desenvolvidos como prosseguimento desta pesquisa, pode-se destacar:

- Identificação de outras wikis especializadas, por meio de experimentos, e com a obtenção de dados reais, que comprovem as tendências de uso dos recursos de controle de versões ou outros

mecanismos particulares, em wikis. Atualmente, são discutidos vários domínios de aplicações de uso de wikis em trabalho colaborativo, como por exemplo: em jornalismo, com a elaboração colaborativa de textos de notícias; em ambientes corporativos, como incentivo à produção de documentos colaborativos, etc;

- Desenvolvimento de ferramenta para rastreio dos históricos de versões dos artefatos e associação ao DR registrado, a fim de auxiliar na compreensão do raciocínio das tomadas de decisão no processo de desenvolvimento de software;
- Estudo de mecanismos para detecção/prevenção de *spams* nas wikis: ataques a wikis têm sido frequentes e prejudicam a autoria colaborativa. Isto faz com que algumas wikis tenham controle de acesso (apenas um grupo de pessoas pode colaborar) ou tenham restrição ao *upload* de arquivos.

## 5.2. Considerações Finais

Neste trabalho, a pesquisa científica realizada foi uma análise da adoção do controle de versões como apoio à edição colaborativa na *Web*. De fato, foi observado da literatura e dos esforços para melhoria de qualidade durante o desenvolvimento de software, que a atividade de controle de versões é importante e apresenta benefícios incontestáveis, quando criteriosamente adotada, considerando-se sua adequação ao suporte de GCS.

Os estudos apresentados por este trabalho visam contribuir com as pesquisas relacionadas ao suporte atual de wikis como forma colaborativa de edição. Nesse contexto, foi possível identificar que o controle de versões também traz benefícios, mas sua adoção deve ser criteriosamente considerada.

Como resultado prático deste trabalho, duas versões da wiki CoTeia foram implementadas:

(i) a primeira, usada no ICMC, foi aprimorada, acrescentando a possibilidade de remoção dos *uploads*, necessidade sentida pelos administradores das *swikis*. Além disso, a fim de evitar remoções mal-intencionadas ou acidentais, a operação de remoção foi associada a um controle de acesso, disponibilizado apenas ao administrador da *swik*i. Com isso, espera-se que a manutenção do espaço reservado para o armazenamento dos *uploads* seja mais facilitada;

(ii) a segunda, usada na ferramenta DoRationale, foi aprimorada, por meio da integração de CVS ao recurso de *uploads.* Com isso, espera-se que a memória dos artefatos e respectivos *design rationales*, armazenados, auxiliem a tarefa dos desenvolvedores de software.

# Referências Bibliográficas


ARRUDA JR., C. R. E.; PIMENTEL, M. G. C.; IZEKI, C. A. **CoTeia: Uma ferramenta colaborativa de edição baseada na web**. In: WORKSHOP DE FERRAMENTAS E APLICAÇÕES DO VIII SBMIDIA, 2002, Fortaleza, CE, p.371–375.

ARRUDA JR, C. B.; PIMENTEL, M. G. C. **Projeto e Implementação de um Sistema Colaborativo de Edição**. Revista Eletrônica de Iniciação Científica da Sociedade Brasileira de Computação (REIC-SBC), ano 1, 12 p., nov. 2001.

ASKLUND, U.; MAGNUSSON, B. **Fine Grained Version Control of Configuration in COOP/Orm**. In: PROCEEDINGS OF SCM SYMPOSIUM ON CONFIGURATION MANAGEMENT, 6, mar.1996, p.31-48. Disponível em: <http://citeseer.nj.nec.com/magnusson96fine.html.>. Acesso em: 10 fev. 2004.

ASKLUND, U. et al. **The Unified Extensional Versioning Model**. In: SYSTEM CONFIGURATION MANAGEMENT: 9TH INTERNATIONAL SYMPOSIUM, SCM-9, Toulouse, France, sept.1999, p.100-122. Disponível em: <http://citeseer.nj.nec.com/asklund99unified.html>. Acesso em: 10 fev. 2004.

ASKLUND, U. et al. **Product Data Management and Software Configuration Management - Similarities and Diferences**. The Association of Swedish Engineering Industries, 2001.

BABICH, W.A. **Software Configuration Management**. Addison-Wesley, 1986a.

BABICH, W.A. **Software configuration management: coordination for team productivity**. Addison-Wesley, 1986b.

BACELO, A; BECKER, K. **Uma ferramenta de apoio à discussão e deliberação em gru**po. In: III WORKSHOP EM SISTEMAS MULTIMÍDIA E HIPERMÍDIA, 1996, São Carlos, SP. Anais... 1996, p.119–130.

BERLACK, H. R. **Software configuration management**. Wiley, 1992.

BERSOFF, E. H. **Software configuration management, an investment in product integrity**. Prentice-Hall, 1980.

BIELIKOVÁ, M.; NÁVRAT, P. **Space-efficient Techniques for Storing Versions of Software Components**. In: PROC. OF SCIENTIFIC CONF. ELEKTRONIC COMPUTERS AND INFORMATICS, 1996, Kosice - Herlany, Slovakia. Sept.1996, p. 56-61. Disponível em: <www.fiit.stuba.sk/~bielik/publ/abstracts/1996/delta.ps>. Acesso em: 20 abr. 2005.

BRUCKER, A. D.; RITTINGER, F.; WOLFF, B. **The CVS-Server Case Study**. In: FM-TOOLS, Augsburg, jul.2002, p.47-52. Disponível em: <http://citeseer.nj.nec.com/524877.html>. Acesso em: 20 abr. 2004.

BUCKLEY, F. J. **Implementing configuration management: hardware, software, and firmware**. 2nd ed. IEEE Computer Society Press, 1996.

CEDERQVIST, P. **Version Management with CVS**. 1993. Disponível em: <http://www.cvshome.org/docs/manual>. Acesso em: jan. 2004.

CHANG, W.; CHEN, C. **Integrity-Enhanced Verification Scheme for Software-Intensive Organizations**. In: ATVA 2004. LNCS, n.3299, 2004, p.402–414.

**CMMI – Capability Maturity Model Integration**. *Software Engineering Institute*, *Carnegie Mellon University*, 2002. Disponível em: <http://www.sei.cmu.edu/cmmi>. Acesso em: 16 maio 2005.

COLLINS-SUSSMAN, B.; FITZPATRICK, B.W.; PILATO, C.M. **Version Control with Subversion: For Subversion 1.0**. 2004. Disponível em: <http://svnbook.red-bean.com>. Acesso em: 05 maio 2005.

CONKLIN, J. **Hypertext: An Introduction and Surve**y. IEEE Computer Society Press Los Alamitos, CA, USA, v.20, n.9, p. 17-41, 1987.

CONRADI, R.; WESTFECHTEL, B. **Version Models for Software Configuration Management**. ACM Computing Surveys, v.30, n.2, p.232-282, june 1998.

CUNNINGHAM, W.; LEUF, B. **The Wiki Way**. Reading, MA: Addison-Wesley, 2001.

CUNINGHAM, W. **WikiWikiWeb**. Disponível em: <http://c2.com/cgi/wiki?WikiWikiWeb>. Acesso em: 11 nov. 2004.

DART, S. **Spectrum of Functionality in Configuration Management Systems**. Software Engineering Institute, dec.1990. Disponível em: <http://www.sei.cmu.edu/legacy/scm/abstracts/abscm_spectrum_of_func_TR11_90.html>. Acesso em: 20 abr. 2005.

DART, S. **Not All Tools are Equal**. 1996.

DIKENELLLI, O. Process Improvement towards ISO 9001 Certification in a Small Software Organization. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - PROCEEDINGS OF THE 20TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, Kyoto, Japan, 1998, p.435 - 438.

FEILER, P.H. **Configuration Management Models in Commercial Environments**. Software Engineering Institute, mar.1991. Disponível em: <http://www.sei.cmu.edu/activities/legacy/scm/abstracts/abscm_models_TR07_91.html>. Acesso em: jan. 2004.

FELDMAN, S. I. **Make - A Program for Maintaining Computer Programs**. Software - Practice & Experience, v.9, n.3, p.255-265, mar.1979.

FISH, S. **Which Open Source Wiki Works for You?**. O'Reilly ONlamp.com, 11 abr. 2004. Disponível em: <http://www.onlamp.com/lpt/a/5290>. Acesso em: 11 nov. 2004.

FRANCISCO, S.D. et al. **DocRationale: Um Auxílio a Design Rationale na Documentação de Software**. X Sessão de Ferramentas, XVII Simpósio Brasileiro de Engenharia de Software (SBES2003), Manaus - AM, 2003, p.61 - 66.

FRANCISCO, Simone D. **DocRationale: uma ferramenta para suporte a design rationale de artefatos de software**. 2004. 123 f. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, 2004.

GOLAND, Y. et al. **HTTP Extensions for Distributed Authoring – WebDAV**. Internet Proposed Standard – RFC 2518, 1999.

GRADY, R. **Successful Software Process Improvement**. Hewlett-Packard Professional Books, 1997.

GRUBBER, T.R.; RUSSEL, D.M. **Design konwledge and design rationale: A framework for representation, capture and use**. Technical Report KSL 90-45, Knowledge Systems Laboratory, Standford, California, 1991, 40p.

GUZDIAL, M. **Teacher and Student Authoring on the Web for Shifting Agency**. In: ANNUAL MEETING OF THE AMERICAN EDUCATIONAL RESEARCH ASSOCIATION as part of the session *How can CSCL (Computer Supported Collaborative Learning) change classroom culture and patterns of interaction among participants?*, 1999.

_____. **Use of Collaborative Multimedia in Computer Science Classes**. In: PROCEEDINGS OF THE 6TH ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 2001, Atlanta, USA. GVU Center and EduTech Institute, College of Computing, Georgia Institute of Technology, 2001, p.17–20. Disponível em: <http://coweb.cc.gatech.edu:8888/csl/uploads/24/ITICSE-CoWeb-final.pdf>. Acesso em: 23 nov. 2004.

GUZDIAL, M.; RICK, J.; KEHOE, C. **Beyond Adoption to Invention: Teacher-Created Collaborative Activities in Higher Education**. Journal of the Learning Sciences, v.10, n.3, p.265-279, july 2001. Disponível em: <http://coweb.cc.gatech.edu:8888/csl/uploads/24/CoWeb-final-Jan01.pdf>. Acesso em: 23 nov. 2004.

GUZDIAL, M. et al. **The challenge of collaborative learning in engineering and math**. Reno, NV: IEEE/ASEE Frontiers in Education, 2001. Disponível em: <http://coweb.cc.gatech.edu:8888/csl/uploads/24/CMCI-Mellon.pdf>. Acesso em: 23 nov.2004.

HOYLE, D. **ISO 9001 Quality Systems Handbook**. 4th ed., Woburn, MA: Butterworth-Heinemann, 2001.

HU, X. et al. **A survey on design rationale: Representation, capture and retrieval**. Engineering with Computers: An Int'l Journal for Simulation-Based Engineering, v.16, p.209–235, 2000.

HWANG, S. **A Design of Configuration Management Practices and CMPET in Common Criteria Based on Software Process Improvement Activity**. In: ICCSA, 2004. LNCS n.3043, 2004, p.481–490.

INGALLS, D. et al. **Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself**. In: OOPSLA'97 CONFERENCE PROCEEDINGS, 1997, Atlanta, GA. ACM, 1997, p.318-326.

**ISO/IEC 15504: Information Technology — Process Assessment**. 2005.

JONES, C. **Software Assessments, Benchmarks, and Best Practices**. Addison-Wesley, 2000.

JUNQUEIRA, D.C. **Desenvolvimento do módulo de acesso a SubVersion por VersionWeb.** 2004. 37p. Monografia (Graduação em Ciência da Computação). Instituto de Ciências Matemáticas e de Computação de São Carlos, Universidade de São Paulo.

KANDT, R. K. **Software Configuration Management Principles and Best Practices**. In: PROFES, 2002. LNCS n.2559, 2002, p.300-313.

KRAUSE, R. **An Introduction to the arch Version Control System**. Linux Journal, 10 may 2002. Disponível em: <http://www.linuxjournal.com/article/5928>. Acesso em: 05 maio 2005.

LEE, J. **Design Rationale Systems: Understanding the issues**. IEEE Expert, 1997, p.78-84.

LIN, Y.; REISS, S. P. **Configuration Management in terms of Modules**. In: PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON SOFTWARE CONFIGURATION MANAGEMENT, apr.1995, p.17-26.

LORD, T. **GNU Arch**. 2003 Disponível em: <http://gnuarch.org/>. Acesso em: 05 maio 2005.

MACEDO, A. A.; BULCÃO NETO, R. F.; PIMENTEL, M. G. C. **Autoria colaborativa na Web: experiências e reflexões sobre a CoWeb**. Revista Brasileira de Informática na Educação (RBIE), n.9, set.2001.

MARTÍN-VIVALDI, N.; ISACSSON, P. **Controlling Your Design through Your Software Process**. In: SAFECOMP'98, 1998, Heidelberg, Germany. LNCS n.1516, p.77-88.

MIDHA, A. K. **Software configuration management for the 21st century**. Bell Labs Technical Journal, 2(1), winter 1997. Disponível em: <http://citeseer.nj.nec.com/midha97software.html>. Acesso em: 10 fev. 2004.

MONTANGERO, C. et al. **The Software Process: Modelling and Technology**, Software Process, 1999. LNCS n.1500, p.1-13.

MORAN, T.P.; CARROL, J.M. **Design Rationale: Concepts, Techniques, and Use Computers, Cognition, and Work**. Lawrence Erlbaum Associates, New Jersey, 1996, 659p.

MOURA, C. A. T.; LAHOZ, C. H. N.; ABDALA, M. A. D. **A Practical Approach To Quality Assurance in Critical Systems**. In: LADC 2003, 2003. LNCS n.2847, p.369-370.

MUNSON, E. V.; NGUYEN, T.; BOYLAND, J. T. **State-based Product versioning in the Fluid/SC Configuration Management System**. Submitted to the 2003 Automated Software Engineering Conference (ASE'03), 2003. Department of EECS, University of Wisconsin-Milwaukee.

OLIVEIRA, A. A. C. P. et al. **Gerência de Configuração de Software - Evolução de Software sem Controle**. São Paulo, SP: Instituto Nacional de Tecnologia da Informação, 2001. Disponível em:

<http://www.psphome.hpg.ig.com.br/downloads/Apostila_curso_GCS.pdf. Acesso em: 06 fev. 2004.

PAULK, M.C. et al. **The Capability Maturity Model for Software**. Technical Report CMU/SEI-93-TR-024, feb.1993. Disponível em: <http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.024.html>. Acesso em: 16 maio 2005.

PAULK, M. C. et al. **The Capability Maturity Model: Guidelines for Improving the Software Process**. Addison-Wesley, 1994.

PIMENTEL, M. G. C.; IZEKI, C. A.; ARRUDA JR, C. R. E. **An XML-based Infrastructure Supporting Collaborative Annotations as First-class Hyperdocuments**. In: PROCEEDINGS OF VII BRAZILIAN SYMPOSIUM OF MULTIMEDIA AND HYPERMEDIA SYSTEMS, 2001, Florianópolis, SC. 2001a, p.173–186.

_____. **An XML-based Infrastructure Supporting Collaborative Annotations as First-Class Hyperdocuments.** In: VII SIMPÓSIO BRASILEIRO DE HIPERMÍDIA E MULTIMÍDIA – SBMIDIA, 2001b.

PRESSMAN, R. S. **Engenharia de Software**. 5. ed., Rio de Janeiro: McGraw-Hill, 2002.

REICHENBERGER, C. **Delta Storage for Arbitrary Non-Text Files**. ACM, p.144-152, 1991.

REIS, C.R. **Caracterização de um Processo de Software para Projetos de Software Livre**. 2003. 195p. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional). Instituto de Ciências Matemáticas e de Computação de São Carlos, Universidade de São Paulo, São Carlos, SP.

RICK, J. et al. **Collaborative Web-sites for English Composition**. 2001. Disponível em: <http://coweb.cc.gatech.edu:8888/csl/uploads/24/CoWeb-Mellon-chapter.pdf>. Acesso em: 23 nov. 2004.

RICK, J. et al. **Collaborative Learning at Low Cost: CoWeb Use in English Composition**. In: PROCEEDINGS OF THE COMPUTER SUPPORTED COLLABORATIVE LEARNING 2002, Boulder, CO, USA. Disponível em: <http://coweb.cc.gatech.edu:8888/csl/uploads/24/CoWebInEnglish-CSCL2002.pdf>. Acesso em: 23 nov. 2004.

RIGG, W.; BURROWS, C.; INGRAM, P. **Ovum Evaluates: Configuration Management Tools**. Ovum Limited, 1995.

ROCHKIND, M.J. **The source code control system**. IEEE Trans. Soft.Eng. 1, 4 (Dec.), p.364-370.

ROZANTE, T. A. A. **Implantação do reuso de componentes no processo de desenvolvimento de software**. 2003. 184p. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, 2003.

RUPLEY, S. **What's a wiki?**. PC Magazine, 05 set. 2003. Disponível em: <http://www.pcmag.com/article2/0,4149,1071705,00.asp>.

SALVIANO, C. F. **Futura Norma ISO/IEC 15504 (SPICE)**. In: Qualidade e Produtividade em Software, 4. ed., Makron Books, 2001, p.19-25.

SHIPMAN, F.; MCCALL, R. **Integrating diferent perspectives on design rationale: Supporting the emergence of design rationale from design communication**. Artificial Intelligence in Engineering Design, Analysis, and Manufacturing, v.11, n.2, p. 141–154, 1997.

SHUM, S. **A cognitive analysis of design rationale representation**. Tese de Doutoramento, University of York, 1991.

SILVA, S. R. Q. M.; IGNÁCIO JUNIOR, J. C. V.; FORTES, R. P. M. **Manual de uso ViewCVS**. Relatório técnico. São Carlos, SP: USP/ICMC, n.245, dez.2004. 41 f.

SOMMERVILLE, I. **Engenharia de Software**. 6. ed., São Paulo: Addison-Wesley, 2003.

SOUZA, C. R. B.et al. **A model and tool for semi-automatic recording of design rationale in software diagram**s. In: PROCEEDINGS OF THE 6TH STRING PROCESSING AND INFORMATION RETRIEVAL SYMPOSIUM & 5TH INTERNATIONAL WORKSHOP ON GROUPWARE, Cancun, Mexico, 1998, p.306–313.

TICHY, W. F. **RCS - A System for Version Control**. Software - Practice and Experience, v.15, n.7, p.637-654, 1985. Disponível em: <http://citeseer.ist.psu.edu/tichy91rcs.html>. Acesso em: jan. 2004.

VENUGOPALAN, V. **CVS Best Practices**. Sept. 2002. Disponível em: <http://www.tldp.org/REF/CVS-BestPractices/CVS-BestPractices.pdf>. Acesso em: 20 abr. 2005.

VISCONTI, M.; VILLARROEL, R. **Managing the Improvement of SCM Process**. M. Oivo and S. Komi-Sirviö (Eds.): PROFES 2002, LNCS 2559, p. 35-48, 2002.

WESTFECHTEL, B. **A graph-based system for managing configurations of engineering design documents**. Int. J. Softw. Eng. Knowledge Eng., 1996.

WESZKA, J. **CMMI: Evolutionary Path to Enterprise Process Improvement**. Crosstalk, p.8-10, july 2000.

WHITGIFT, D. **Methods and Tools for Software Configuration Management**. Wiley, 1991.

# APÊNDICE A – Implementações de wiki

| | | | Versionamento | | | |
|---|---|---|---|---|---|---|
| | **Wikis** | **Linguagem** | **Páginas** | **Arquivos (uploads)** | **CVS** | **Upload** |
| 1 | Asp Wiki | ASP | SIM | NÃO | – | NÃO |
| 2 | AsWiki | Rubby | SIM | NÃO | NÃO (RCS) | NÃO |
| 3 | AtisWiki | Perl | SIM | – | SIM | – |
| 4 | AwkiAwki | Awk | SIM | NÃO | NÃO (RCS) | NÃO |
| 5 | CfWiki | ColdFusion | SIM | NÃO | – | NÃO |
| 6 | ChikiWiki | Java | SIM | NÃO | – | NÃO |
| 7 | ChiqChaq | Perl | SIM | NÃO | NÃO | NÃO |
| 8 | Cliki | CommonLisp | SIM | NÃO | – | NÃO |
| 9 | Cloud Wiki | Python | SIM | – | NÃO | – |
| 10 | CocanWiki | ObjectiveCaml | SIM | NÃO | NÃO | SIM |
| 11 | CoTeia | PHP | SIM | NÃO | SIM | SIM |
| 12 | CoWiki | PHP | SIM | NÃO | NÃO | NÃO |
| 13 | CVSTrac | Cee | SIM | NÃO | NÃO | NÃO |
| 14 | CyberPublishing | Python | NÃO | NÃO | NÃO | NÃO |
| 15 | DevWiki | Java | SIM | NÃO | SIM | NÃO |
| 16 | DiamondWiki | Python | SIM | NÃO | NÃO | NÃO |
| 17 | DidiWiki | Cee | SIM | NÃO | NÃO | NÃO |
| 18 | DokuWiki | PHP | SIM | NÃO | NÃO | SIM |
| 19 | DotWiki | VisualBasic | SIM | NÃO | NÃO | NÃO |
| 20 | EditThisPagePhp | PHP | SIM | NÃO | NÃO | SIM (imagens) |
| 21 | ErfurtWiki | PHP | SIM | NÃO | NÃO | SIM (imagens) |
| 22 | FireWiki | PHP | SIM | NÃO | NÃO | NÃO |
| 23 | FitNesse | Java | SIM | NÃO | – | NÃO |
| 24 | FlexWiki | Csharp | SIM | NÃO | – | NÃO |
| 25 | Friki | Java | – | NÃO | – | NÃO |
| 26 | fuwiki | PHP | SIM | NÃO | NÃO | SIM (imagens) |
| 27 | GeboGebo Wiki | EASY (incluída no banco de dados Tdbengine) | SIM | – | NÃO | SIM |
| 28 | GikiWiki | ObjectiveCaml | SIM | NÃO | RCS/ Perforce | NÃO |
| 29 | GracefulTavi | PHP | SIM | NÃO | NÃO | NÃO |
| 30 | InstikiWiki | Ruby | SIM | NÃO | NÃO | NÃO |
| 31 | IoWiki | Io | SIM | NÃO | – | NÃO |
| 32 | JASSWiki | ASP | SIM | NÃO | NÃO | NÃO |
| 33 | JaWiki | Java | SIM | NÃO | NÃO | NÃO |

| | | | Versionamento | | | |
|---|---|---|---|---|---|---|
| | **Wikis** | **Linguagem** | **Páginas** | **Arquivos (uploads)** | **CVS** | **Upload** |
| 34 | JspWiki | Java | SIM | SIM | NÃO (RCS) | SIM |
| 35 | Kwiki Kwiki | Perl | SIM | NÃO | NÃO | NÃO |
| 36 | Kwik Wiki | Java | SIM | – | NÃO | – |
| 37 | MaseWiki | Java | SIM | SIM | – | SIM |
| 38 | MediaWiki | PHP | SIM | SIM | NÃO | SIM |
| 39 | MiniRubyWiki | Ruby | SIM | NÃO | – | NÃO |
| 40 | MiniWiki | Perl | SIM | NÃO | NÃO (RCS) | NÃO |
| 41 | MoinMoin | Python | SIM | NÃO (sobreposição) | NÃO | SIM |
| 42 | MoniWiki | PHP | SIM | NÃO (opção de sobreposição ou renomear) | NÃO (RCS) | SIM |
| 43 | newLISP-Wiki | NewLISP | SIM | NÃO | NÃO | NÃO |
| 44 | OddMuseWiki | Perl | SIM | NÃO | NÃO | SIM |
| 45 | PerSpective | Csharp | SIM | NÃO | – | SIM |
| 46 | PhearWiki | Perl | SIM | NÃO | – | NÃO |
| 47 | PhpWiki | PHP | SIM | NÃO | NÃO | SIM |
| 48 | PmWiki | PHP | SIM | NÃO (sobreposição) | NÃO | SIM |
| 49 | PodWiki | Perl | SIM | – | – | SIM |
| 50 | PurpleWiki | Perl | SIM | NÃO | NÃO | NÃO |
| 51 | PWP Wiki Processor | PHP | SIM | SIM | NÃO | SIM |
| 52 | PyleWiki (opção de usar ViewCVS para visualização das diferenças) | Python | SIM | NÃO | SIM (páginas) | SIM |
| 53 | PyWiki | Python | SIM | NÃO | NÃO (RCS) | NÃO |
| 54 | QwikWiki | Perl | SIM | NÃO | NÃO | SIM |
| 55 | Radeox | Java | SIM | NÃO | NÃO | SIM |
| 56 | RuWiki | Ruby | SIM | NÃO | NÃO | NÃO |
| 57 | ScrumWiki | Perl | SIM | - | NÃO | SIM |
| 58 | SeedWiki | ColdFusion | SIM | NÃO | NÃO (sobreposição) | SIM |
| 59 | SnipSnap | Java | SIM | NÃO | NÃO | SIM |
| 60 | SpinnerWiki | Perl | SIM | NÃO (sobreposição de arquivos) | NÃO | SIM |
| 61 | TikiWiki CMS/Groupware | PHP | SIM | – | NÃO | SIM |
| 62 | TikiWikiLite | PHP | SIM | NÃO | NÃO | – |
| 63 | TipiWiki | PHP | SIM | NÃO | NÃO | NÃO |
| 64 | TwikiClone (TWiki) | Perl | SIM | SIM | NÃO (RCS) | SIM |
| 65 | UniWakka | PHP | SIM | – | NÃO | SIM |
| 66 | UseModjWiki | Java | SIM | NÃO (sobreposição de arquivos) | NÃO | SIM |
| 67 | UseModWiki | Perl | SIM | NÃO | NÃO | NÃO |

| | | | Versionamento | | | |
|---|---|---|---|---|---|---|
| | **Wikis** | **Linguagem** | **Páginas** | **Arquivos (uploads)** | **CVS** | **Upload** |
| 68 | VeryQuickWiki | Java | SIM | – | NÃO | SIM |
| 69 | Wiclear | PHP | SIM | NÃO | NÃO | NÃO |
| 70 | WikiNehesa | Python | NÃO | NÃO | – | SIM |
| 71 | WiKit | TCL | SIM | NÃO | – | NÃO |
| 72 | Wikiwig | JavaScript/PHP | SIM | NÃO | – | SIM |
| 73 | WikiX | PHP | SIM | NÃO | NÃO | SIM |
| 74 | WikkaWiki | PHP | SIM | NÃO | NÃO | SIM |
| 75 | WikkiTikkiTavi | PHP | SIM | NÃO | NÃO | NÃO |
| 76 | Wiski | PHP | SIM | NÃO (sobreposição de arquivos) | – | SIM |
| 77 | XwikiWiki | Java | SIM | NÃO | NÃO | SIM |
| 78 | Yawiki | Java | SIM | – | NÃO | SIM |
| 79 | ZwiKi | Python | SIM | NÃO | – | SIM |