UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO

Sistema de Controle de Versão de Arquivos

Projeto Final

DEL
Dezembro/2010

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO

Sistema de Controle de Versão de Arquivos

Autor:
Alberto Schwengber de Alvarenga

Orientador:
Prof. Antônio Cláudio Gómez de Sousa, *M.Sc.*

Examinador:
Prof. Ricardo Rhomberg Martins, *Ph.D.*

Examinador:
Prof. Aloysio de Castro Pinto Pedroza, *Ph.D.*

DEL
Dezembro/2010

Dedicatória

Dedico este trabalho a minha mãe (Julie Ana Schwengber de Alvarenga) e a meu pai (Sergio Ribeiro Lins de Alvarenga) que sempre acreditaram em mim e investiram em minha educação e formação.

Agradecimentos

Agradeço:

- à minha mãe (Julie Ana Schwengber de Alvarenga) e a meu pai (Sergio Ribeiro Lins de Alvarenga) por terem sempre me apoiado e me dado força para atingir todos meus objetivos.
- a mais que amiga e especial Cecilia Valente Teixeira pelo apoio de sempre.
- ao meu orientador Antônio Cláudio Gómez de Sousa pela orientação, pelas conversas, pelas idéias e pela compreensão.
- aos amigos da Accenture Wiliam Mauricio Oliveira, Jose David Carvalho, Cezar Augusto de Souza, Vera Lucia Aprigio e Leia Azevedo por toda a ajuda que me deram no projeto.
- a todos meus amigos, que sempre me apoiaram e incentivaram.

Resumo:

Neste trabalho será realizada uma apresentação geral de sistemas de controle centralizado de versão (CVS). Serão mostradas as arquiteturas atuais de sistemas CVS, além de características e funções que fazem parte desses sistemas.

Além da abordagem teórica de um sistema de controle centralizado de versão, será implementado um sistema de controle de versão de arquivos no formato .PL, .SQL e .sh. Esse sistema utilizará a arquitetura Cliente/Servidor e contará com todos os recursos para um eficiente controle de versão.

O grande diferencial do projeto será o layout simples e intuitivo do sistema, permitindo que um usuário que nunca tenha utilizado um sistema de versionamento possa fazê-lo.

O sistema foi desenvolvido para o sistema operacional *Windows*, utilizando a linguagem de programação Visual Basic 6.

Índice

Capítulo 1 - Introdução

1.1 Introdução

Um sistema de controle de versão, na função prática da Engenharia de Software, é um programa com a finalidade de gerenciar diferentes versões no desenvolvimento de um documento qualquer. Esses sistemas são comumente utilizados no desenvolvimento de software para controlar as diferentes versões — histórico e desenvolvimento — dos códigos-fontes e também da documentação.

O projeto em questão visa implementar um sistema de versionamento de arquivos-texto voltado para um ambiente de desenvolvimento coletivo, utilizando a arquitetura Cliente/Servidor e recursos fundamentais para um eficiente controle de versão. Um layout simples e intuitivo permitirá que um usuário que nunca tenha utilizado esse tipo de aplicação não tenha dificuldade em fazê-lo.

1.2 Organização do Projeto

No Capítulo 2 deste projeto, abordaremos a questão do sistema a ser implementado. Será detalhado o problema identificado no processo de desenvolvimento do meu atual estágio, que serviu de motivador para elaboração desse trabalho. Serão descritas também as restrições impostas ao desenvolvimento desta aplicação.

No Capítulo 3, abordaremos a análise sobre a implementação do projeto e seu resultado, ou seja, será feita uma especificação técnica do projeto como um todo. Será apresentada a interface gráfica com o usuário (GUI) e os diagramas elaborados para a solução do projeto.

No Capítulo 4 trataremos do desenvolvimento propriamente dito. Será feita uma análise dos riscos e impactos, modelagem do banco de dados e as

interfaces do sistema. Serão abordadas também as questões referentes aos problemas e suas soluções, além da arquitetura utilizada no sistema.

No Capítulo 5, analisaremos e descreveremos todos os resultados esperados do sistema através de cenários de testes específicos, validando a etapa de análise da aplicação. Esse capítulo fará a verificação e validação da funcionalidade de todo o sistema de forma criteriosa e detalhada.

Finalmente, no Capítulo 6, concluiremos o projeto. Apontaremos os principais pontos do trabalho, explicitando suas contribuições. Apresentaremos sugestões que possibilitem agregar funcionalidades ao sistema, além de todo o conhecimento adquirido por conta do desenvolvimento desse projeto.

Capítulo 2 – Planejamento

2.1 Introdução

O objetivo deste capítulo é apresentar o panorama atual de como é feito o controle no desenvolvimento de software na empresa na qual trabalho. Na identificação do problema e, por sugestão de outros desenvolvedores, surgiu-se a idéia de criar um sistema de controle centralizado de versão, no intuito de acabar com os problemas recorrentes de perda de código e informação a medida que mais de uma pessoa trabalha em um mesmo projeto.

O desenvolvimento de códigos e scripts necessitam de um controle eficiente, que agregue informação a medida que uma nova versão é criada. O fato de não haver esse controle na criação ou alteração de objetos como PL/SQLs, Shell Scripts ou arquivos de criação de objetos de banco de dados de um modo geral, acaba gerando problemas de perda de código e informação.

Diante desse cenário, foi pensado em se criar um versionador de código, baseado em uma arquitetura cliente-servidor capaz de atender e padronizar todo o processo de criação de objetos, como os citados acima.

Na seção 2.2 apresentaremos o problema identificado no ambiente de trabalho e grande motivador da elaboração desse projeto. Na seção 2.3 apresentaremos a solução sugerida para o problema apresentado. Na seção 2.4 apresentaremos as restrições impostas ao projeto. Na seção 2.5 concluiremos o capítulo ressaltando os pontos mais importantes abordados.

2.2 Problema

A motivação para este trabalho surgiu em um ambiente de desenvolvimento de software onde algumas dezenas de desenvolvedores criavam seus códigos e, por vezes, corrigiam alguma inconsistência encontrada em algum outro sistema implementado.

Após o desenvolvimento de um projeto, seu código é entregue para ser testado e validado de acordo com a especificação funcional. Caso o comportamento do sistema não esteja de acordo com o esperado, é solicitada a correção desse erro e o processo continua até que todos os cenários de testes sejam executados.

Pelo fato da correção não ser feita necessariamente pelo mesmo profissional que desenvolveu o sistema, a existência de um repositório com o arquivo fonte e suas versões posteriores se fazia extremamente necessária para não se sobrescrever códigos ou para não haver a perda de informação.

A identificação da falta desse controle mais restrito sobre os objetos criados ou customizados foi o problema a ser solucionado e grande motivador para elaboração desse sistema de controle de versão.

2.3 Solução

A solução encontrada para garantir uma forma segura de se desenvolver ou alterar um objeto em um ambiente coletivo foi implementar um sistema de Controle de Versão centralizado de arquivos texto (.pl, .sql e .sh) utilizando uma

arquitetura Cliente/Servidor com todos os recursos para um eficiente controle de versão. Para maior clareza da solução, vamos descrever em linhas gerais as características básicas de um controle de versões.

Um sistema de controle de versão é conveniente quando diversos desenvolvedores trabalham sobre o mesmo projeto simultaneamente, pois permite resolver eventuais conflitos entre as alterações. A ramificação e mesclagem de histórico para auxiliar as tarefas sobre o arquivo se mostra também uma das maiores necessidades presentes na fase de testes e correção de erros.

Como dito anteriormente, um sistema de controle de versão centralizado nada mais é do que um programa cuja finalidade é gerenciar diferentes versões no desenvolvimento de um documento qualquer. Eles são utilizados no desenvolvimento de software para controlar as diferentes versões — histórico e desenvolvimento — dos códigos-fontes e também da documentação.

Esse tipo de sistema é muito presente em empresas e instituições de tecnologia e desenvolvimento de software. É também muito comum no desenvolvimento de software livre. É útil, em diversos aspectos, tanto para projetos pessoais pequenos e simples como também para grandes projetos comerciais.

A eficácia do controle de versão de software é comprovada por fazer parte das exigências para melhorias do processo de desenvolvimento de certificações tais como CMMI e SPICE.

Principais vantagens de um sistema CVS:

As principais vantagens de se utilizar um *sistema de controle de versão* para rastrear as alterações feitas durante o desenvolvimento de *software* ou o desenvolvimento de um documento de texto qualquer são:

- **Controle do histórico**: facilidade em *desfazer* e possibilidade de analisar o histórico do desenvolvimento, como também facilidade no resgate de versões mais antigas e estáveis. A maioria das implementações permitem analisar as alterações com detalhes, desde a primeira versão até a última.

- **Trabalho em equipe**: um *sistema de controle de versão* permite que diversas pessoas trabalhem sobre o mesmo conjunto de documentos ao mesmo tempo e minimiza o desgaste provocado por problemas com conflitos de edições. É possível que a implementação também tenha um controle sofisticado de acesso para cada usuário ou grupo de usuários.

- **Marcação e resgate de versões estáveis**: a maioria dos sistemas permite marcar onde é que o documento estava com uma versão estável, podendo ser facilmente resgatado no futuro.

- **Ramificação de projeto**: a maioria das implementações possibilita a divisão do projeto em várias linhas de desenvolvimento, que podem ser trabalhadas paralelamente, sem que uma interfira na outra.

Funcionamento básico de um sistema CVS:

A maior parte das informações - com todo o histórico - ficam guardadas em um repositório, localizado em um servidor qualquer. Geralmente o acesso é feito por um cliente pela rede (via *socket*) e pode ser feito localmente quando o cliente está na mesma máquina do servidor.

O repositório armazena a informação - um conjunto de documentos - de modo persistente num sistema de arquivos ou em um banco de dados qualquer. É possível que o armazenamento seja feito em outros dispositivos capazes de "eternizar" e resgatar facilmente a informação.

Cada servidor pode ter vários sistemas de controle de versão e cada sistema pode ter diversos repositórios, limitando-se na capacidade de gerenciamento do *software* e também no limite físico do *hardware*. Geralmente

um repositório possui um endereço lógico que permite a conexão do cliente. Esse endereço pode ser um conjunto IP/porta, uma URL ou um caminho do sistema de arquivos.

Cada desenvolvedor possui em sua máquina uma cópia local somente da última versão de cada documento. Essa cópia local geralmente é feita num sistema de arquivos simples (FAT ou NTFS). A cada alteração relevante do desenvolvedor é necessário "atualizar" as informações do servidor submetendo as alterações. O servidor então guarda a nova alteração junto de todo o histórico mais antigo. Se o desenvolvedor quer atualizar sua cópia local é necessário atualizar as informações locais, e para isso é necessário baixar as atualizações do servidor.

Após ter sido abordada toda a estrutura, aspecto e o funcionamento de um sistema genérico de versionamento de arquivos, serão apresentadas todas as funcionalidades do sistema a ser implementado nesse projeto.

As funcionalidades do sistema serão:

- Cadastro de usuários
- Deleção de usuários
- Parametrização do endereço do repositório de arquivos
- Cadastro de arquivos
- Histórico de arquivos
- Comparação de arquivos
- Controle de arquivos e usuários
- Resgate de arquivos

Cada item mencionado acima será discutido em sessões posteriores desse relatório.

2.4 Restrição

Para implementação desse projeto foi utilizada a linguagem de programação Visual Basic 6. O motivo da escolha foi a cultura da instituição onde será utilizado o sistema. A linguagem utilizada para a customização do sistema CRM na qual a empresa trabalha é o VBScript. Caso seja necessário fazer um melhoramento na ferramenta, qualquer um dos desenvolvedores não terá dificuldades técnicas em lidar com tal tecnologia.

Além da escolha desta linguagem de programação, o critério de escolha de hardware e software para os usuários são aqueles encontrados e utilizados pelos desenvolvedores da equipe. Em sendo assim, não será necessário adquirir nenhum componente ou sistema novo para utilizar o programa desenvolvido.

2.5 Conclusão

Neste capítulo apresentamos o problema que motivou a elaboração desse projeto, bem como a solução sugerida. Foram mostradas as razões pela escolha das ferramentas utilizadas no desenvolvimento do sistema e uma definição sucinta do funcionamento de um sistema de versionamento de arquivos padrão.

No próximo capítulo, será abordada toda a especificação técnica do projeto. Será apresentada a interface gráfica com o usuário (GUI), as classes utilizadas e seus atributos, além da especificação de cada caso de uso.

Capítulo 3 - Análise

3.1 Introdução

Neste capítulo abordaremos as soluções para a implementação do projeto proposto. Será discutida e mostrada a especificação técnica elaborada para o projeto. Por ter sido utilizada orientação a objeto, será apresentada a estrutura de classes do sistema, além da interface gráfica com o usuário. Em paralelo a isso, será detalhado o comportamento esperado do sistema através de um diagrama de caso de uso e da especificação de cada um dos casos de uso.

3.2 Diagrama de classes

Nesta seção apresentaremos o diagrama de classes do sistema. Embora esse projeto tenha sido desenvolvido em Visual Basic, uma linguagem predominantemente orientada a eventos, a codificação foi feita utilizando os conceitos da orientação a objeto. A figura 3.1 ilustra o diagrama de classes utilizado na especificação do sistema de controle de versão.

Esse diagrama de classes é uma representação da estrutura e relações das classes que servem de modelo para objetos.

É uma modelagem que define todas as classes que o sistema necessita e é a base para a construção dos diagramas de comunicação, sequência e estados.

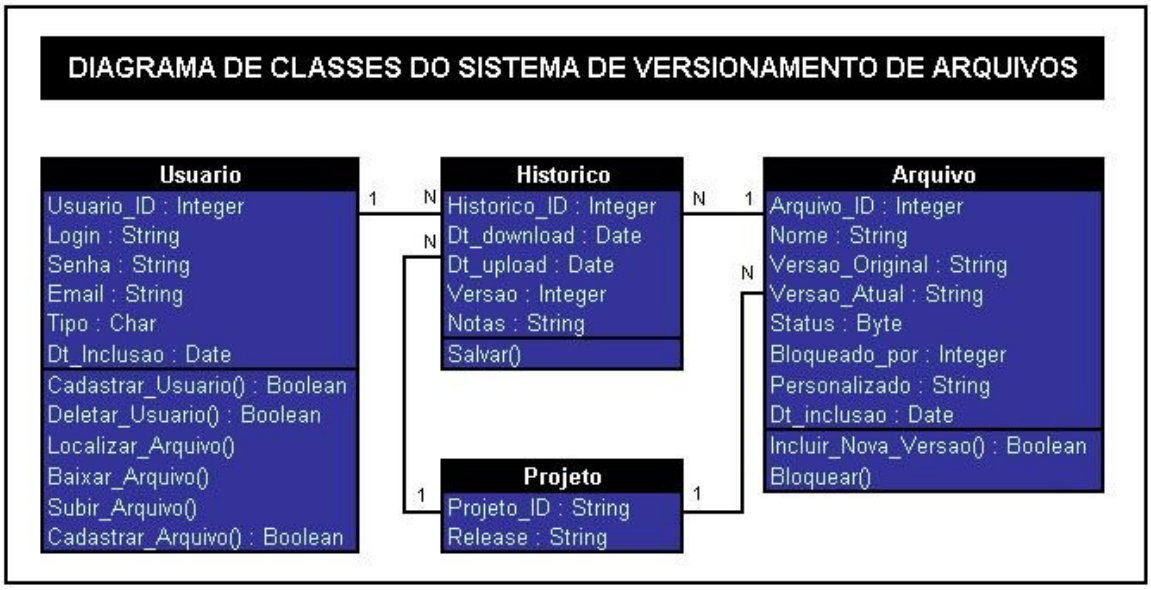


Figura 3.1 – Diagrama de Classes (UML)

3.3 Casos de uso

Nesta seção apresentaremos o diagrama de casos de uso do sistema e a especificação de cada um deles. Um caso de uso é uma modelagem usada para descrever o que um novo sistema deve fazer. Ele é construído através de um processo interativo no qual as discussões entre o cliente e os desenvolvedores do sistema conduzem a uma especificação do sistema com a qual todos estão de acordo.

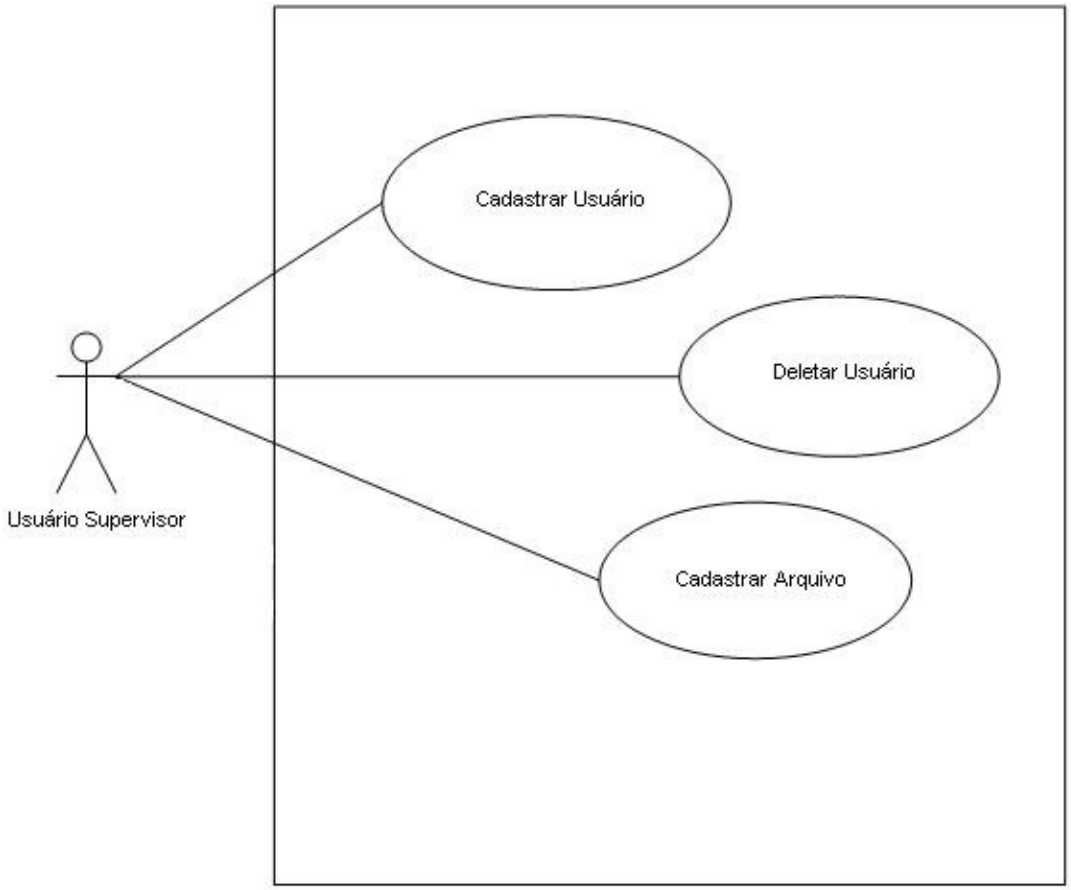


Figura 3.2 – Casos de Uso

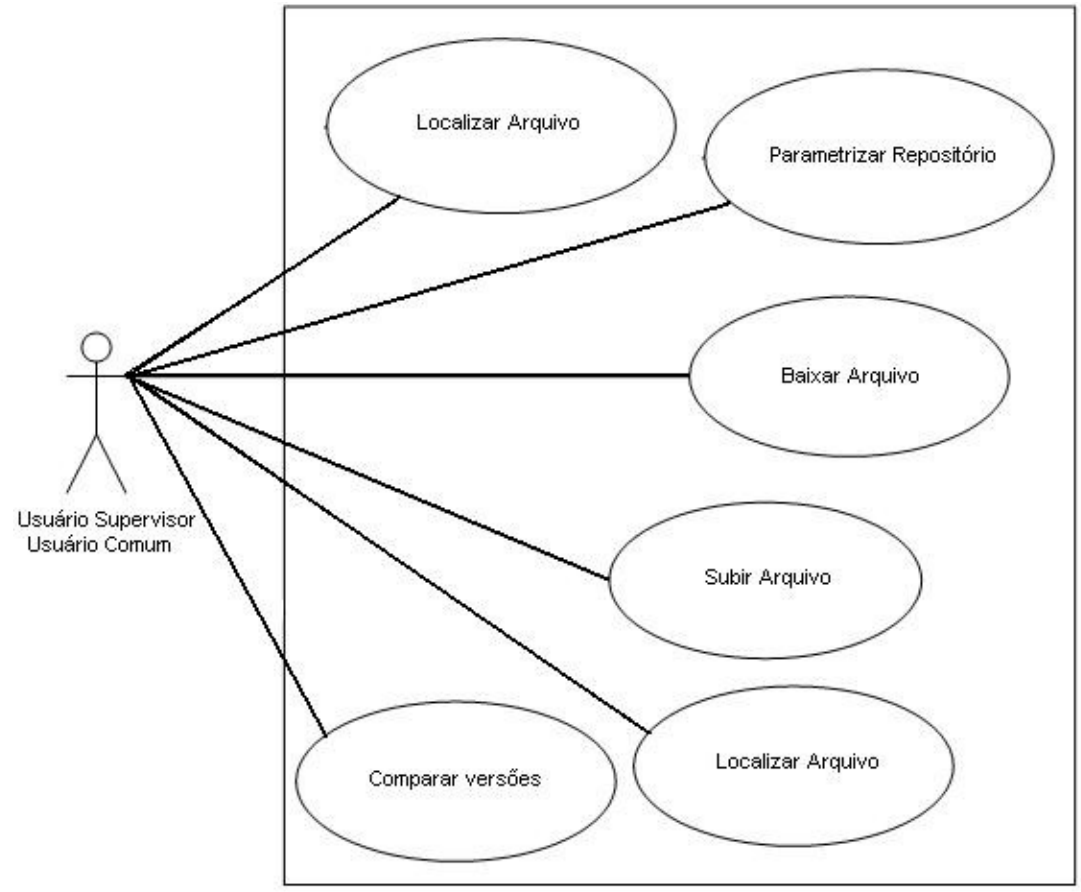


Figura 3.3 – Casos de Uso

Localizar Arquivo

Atores

Usuário supervisor / Usuário comum

Pré-condição:

O usuário deve ter feito "log-in" e obtido autorização do sistema

Fluxo de Eventos (caminho básico):

1. O caso de uso começa quando o usuário seleciona "Localizar Arquivo".
2. O usuário fornece o nome do arquivo.
3. O sistema retorna o nome do arquivo, versão atual, status do arquivo e projeto relacionado.
4. O caso de uso é encerrado.

Pós-condição:

N/A.

Cadastrar Arquivo

Atores

Usuário supervisor

Pré-condição:

O usuário deve ter feito "log-in" e obtido autorização do sistema

Fluxo de Eventos Primário (caminho básico):

1. O caso de uso começa quando o usuário seleciona "Cadastrar Arquivo".
2. O usuário seleciona o arquivo.
3. O usuário fornece o nome do Projeto.
4. O usuário fornece o nome da Release.
5. O usuário clica em salvar.
6. O Sistema cadastra o arquivo.
7. O caso de uso é encerrado.

Pós-condição:

O arquivo foi cadastrado.

Cadastrar Usuário

Atores

Usuário supervisor

Pré-condição:

O usuário deve ter feito "log-in" e obtido autorização do sistema

Fluxo de Eventos Primário (caminho básico):

1. O caso de uso começa quando o usuário seleciona "Cadastrar Usuário".
2. O usuário escreve o e-mail do usuário a ser cadastrado.
3. O usuário escreve o login do usuário a ser cadastrado.
4. O usuário escreve a senha do usuário a ser cadastrado.
5. O usuário escolhe o tipo do usuário a ser cadastrado.
6. O usuário clica em “Salvar”.
7. O caso de uso é encerrado.

Fluxo de Secundário (caminho alternativo):

Se o novo usuário tiver um login ou e-mail já existente no sistema, o usuário não será cadastrado.

Pós-condição:

O usuário foi cadastrado.

Deletar Usuário

Atores

Usuário supervisor

Pré-condição:

O usuário deve ter feito "log-in" e obtido autorização do sistema

Fluxo de Eventos Primário (caminho básico):

1. O caso de uso começa quando o usuário seleciona "Deletar Usuário".
2. O usuário seleciona o usuário a ser deletado.
3. O usuário clica em “Deletar”
4. O Sistema deleta o usuário.
5. O caso de uso é encerrado.

Fluxo de Secundário (caminho alternativo):

Se no passo 2, o usuário selecionado for “Admin”, o sistema informa que o usuário “Admin” não pode ser deletado.

Pós-condição:

O usuário foi deletado.

Parametrizar Repositório

Atores

Usuário supervisor / Usuário comum

Pré-condição:

O usuário deve ter feito "log-in" e obtido autorização do sistema.

Fluxo de Eventos (caminho básico):

1. O caso de uso começa quando o usuário se loga no sistema pela primeira vez.
2. O sistema solicita o endereço do repositório de arquivos.
3. O caso de uso é encerrado.

Pós-condição:

Diretório do repositório definido.

Baixar Arquivo

Atores

Usuário supervisor / Usuário comum

Pré-condição:

O usuário deve ter feito "log-in" e obtido autorização do sistema.

Fluxo de Eventos Primário (caminho básico):

1. O caso de uso começa quando o usuário seleciona “Baixar Arquivo (Check out)”.
2. O usuário fornece o nome do arquivo.
3. O sistema devolve o nome do arquivo, versão atual, status do arquivo e projeto relacionado.
4. O usuário seleciona o arquivo.
5. O usuário clica em “Baixar Arquivo”.
6. O sistema apresenta mensagem questionando o usuário se ele deseja bloquear ou não o arquivo selecionado.
7. Se o usuário bloquear o arquivo
    - O usuário baixa o arquivo para sua estação de trabalho
    - O sistema atualiza o status do arquivo para bloqueado
    - O caso de uso é encerrado
8. Se o usuário não bloquear o arquivo
    - O usuário baixa o arquivo para sua estação de trabalho
    - O caso de uso é encerrado

Fluxo de Secundário (caminho alternativo):

Se no passo 5, o arquivo tiver o status bloqueado, o sistema mostrará uma mensagem alertando ao usuário que o arquivo está bloqueado por outro usuário e só permitirá ser feita a cópia do arquivo para a estação de trabalho do usuário. O caso de uso é encerrado

Pós-condição:

O arquivo foi baixado.

Subir arquivo

Atores

Usuário supervisor / Usuário comum

Pré-condição:

Usuário ter arquivo bloqueado para seu login

Fluxo de Eventos Primário (caminho básico):

1. O caso de uso começa quando o usuário seleciona “Subir Arquivo (Check in)”.
2. O nome do arquivo é selecionado
3. O arquivo é selecionado na máquina local do usuário
4. O usuário descreve as alterações feitas
5. O usuário clica no botão salvar e atualiza o arquivo no repositório
6. O caso de uso é encerrado

Fluxo de Secundário (caminho alternativo):

Se no passo 1, o usuário não possuir arquivo bloqueado para seu login no repositório, o sistema mostrará uma mensagem alertando ao usuário que não há arquivo bloqueado para ele e que não será possível realizar a tarefa. O caso de uso é encerrado

Pós-condição:

O arquivo foi atualizado.

Ver Histórico

Atores

Usuário supervisor / Usuário comum

Pré-condição:

Ter arquivo no repositório

Fluxo de Eventos Primário (caminho básico):

1. O caso de uso começa quando o usuário seleciona “Ver Histórico”.

2. O usuário seleciona o arquivo
3. O usuário clica no botão “Ver Histórico”
4. O sistema retorna as diferentes versões geradas do arquivo
5. O caso de uso é encerrado

Pós-condição:
N/A

Comparar Versões

Atores

Usuário supervisor / Usuário comum

Pré-condição:

Ter mais de uma versão do mesmo arquivo no repositório

Fluxo de Eventos Primário (caminho básico):

1. O caso de uso começa quando o usuário seleciona “Comparar Versões”.
2. O usuário seleciona as versões que deseja comparar
3. O usuário clica em “Comparar”
4. O sistema retorna as diferenças existentes entre as versões do arquivo

Pós-condição:
N/A

3.4 Interface gráfica (GUI)

Nesta seção apresentaremos a interface gráfica desenvolvida para o sistema. Como toda a aplicação, a interface gráfica foi elaborada em Visual Basic 6.0. O objetivo de desenvolver uma interface gráfica é de tornar a aplicação mais amigável para o usuário e mais acessível a usuários de diferentes níveis de conhecimento de computação.

Seguem as telas criadas para a interface gráfica do programa:

Figura 3.4 – Tela de inicialização do Servidor

Figura 3.5 – Tela de login

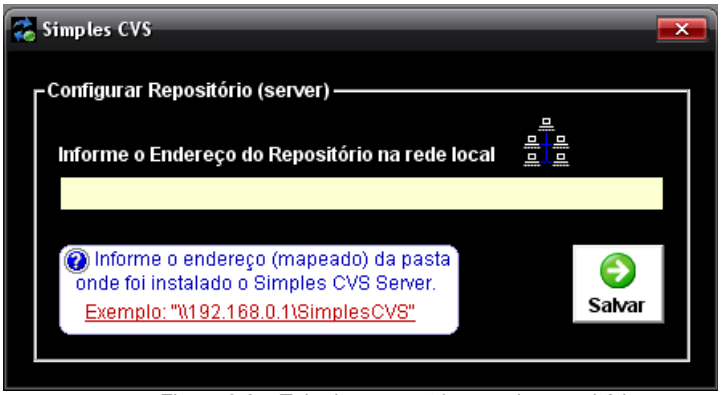


Figura 3.6 – Tela de parametrização do repositório

Figura 3.7 – Tela de menu de usuário - Supervisor

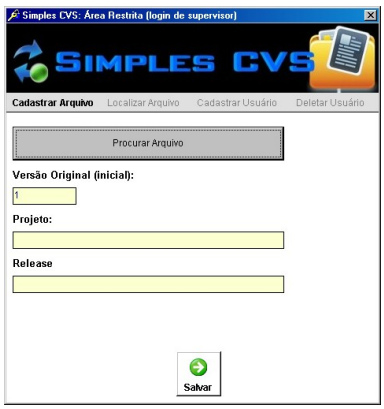


Figura 3.8 – Tela de cadastro de arquivos

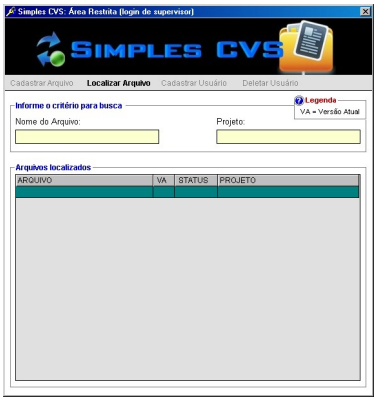


Figura 3.9 – Tela de localização de arquivos

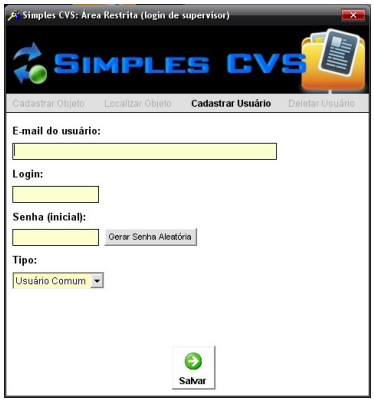


Figura 3.10 – Tela de cadastro de Usuário

Figura 3.11 – Tela de deleção de Usuário

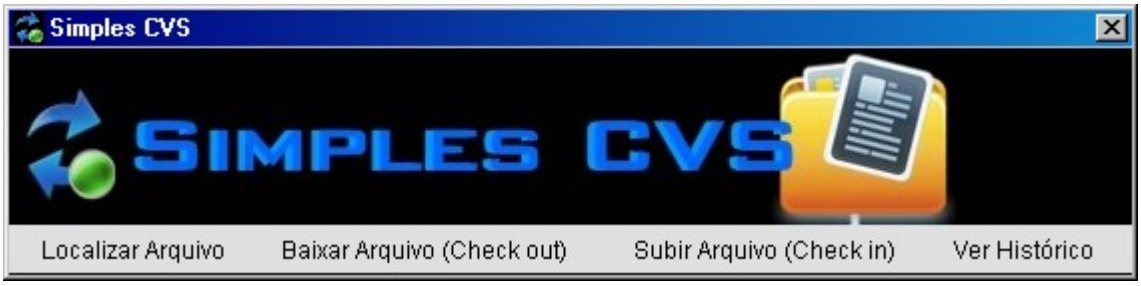


Figura 3.12 – Tela de menu de Usuário

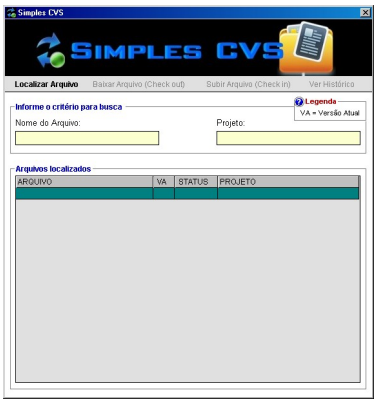


Figura 3.13 – Tela de localização de arquivos

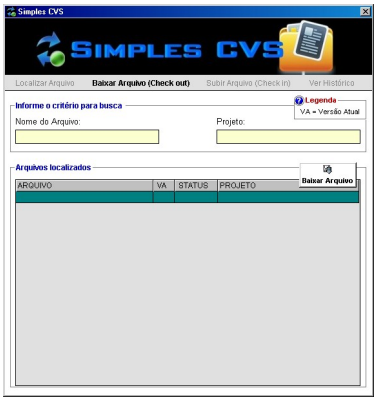


Figura 3.14 – Tela para baixar arquivos

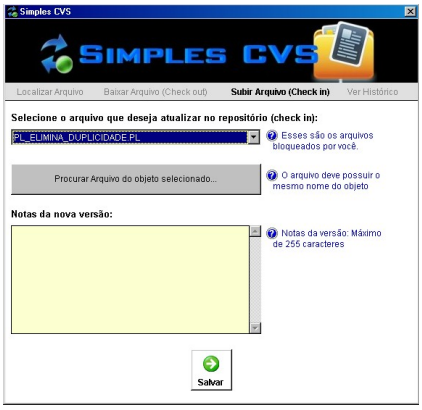


Figura 3.15 – Tela para subir arquivos

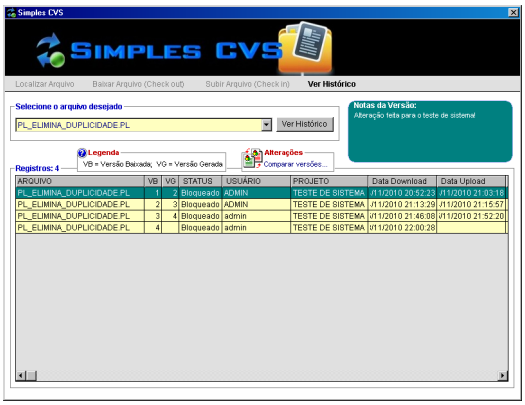


Figura 3.16 – Tela de histórico de arquivos

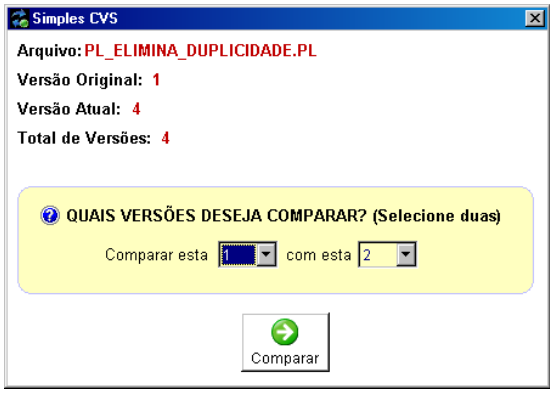


Figura 3.17 – Tela de comparação de versões

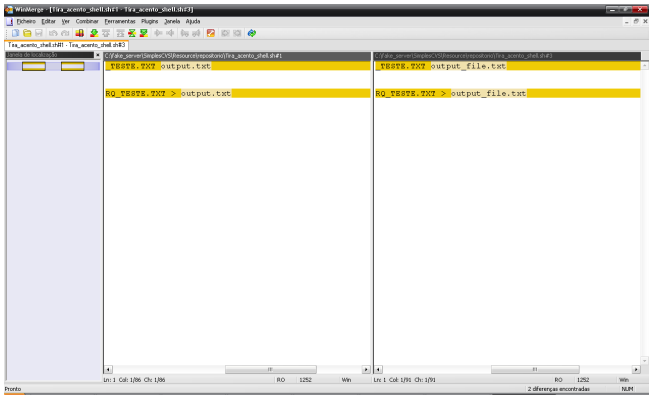


Figura 3.18 – Tela de comparação de versões

3.5 Conclusão

Neste capítulo, foi abordada a especificação do sistema proposto. Foi feito um detalhamento no que diz respeito às classes utilizadas, além da interface gráfica com o usuário a ser usada no programa. Em paralelo a isso, foi detalhado todo o comportamento esperado do sistema através de um diagrama de caso de uso e a especificação de cada um deles.

Capítulo 4 – Desenvolvimento

4.1 Introdução

Neste capítulo, será abordada a etapa de desenvolvimento da aplicação. Serão consideradas todas as etapas de implementação desse projeto, incluindo modelagem de banco de dados, interfaces e casos de uso. Os problemas encontrados na fase de construção e suas soluções também serão evidenciados neste capítulo, juntamente com a arquitetura utilizada para a criação de um sistema de versionamento de arquivos de alto nível.

Na implementação não foi incluída a classe Projeto, mostrada no modelo conceitual, para simplificar o sistema. Esta classe foi substituída pelo atributo chamado Projeto na classe arquivo.

4.2 Banco de dados

Como a linguagem utilizada não tem persistência de dados, foi necessário utilizar um banco de dados para dar persistência às classes definidas na especificação. O diagrama MER abaixo mostra o modelo do banco de dados, que suporta a persistência das classes definidas para o sistema.

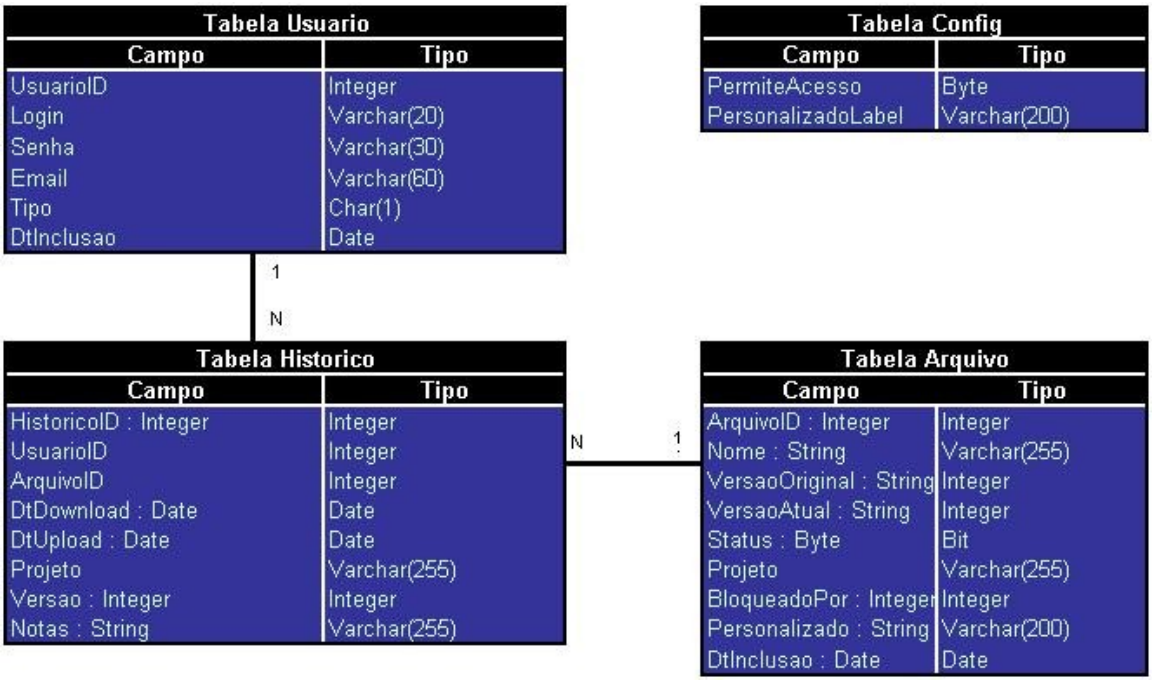


Figura 4.1 (MER)

O programa implementado foi divido em duas partes: Servidor e Cliente. Abaixo, segue uma descrição mais detalhada a respeito de cada campo das tabelas relacionadas a cada um dos módulos do projeto.

Módulo cliente:

DETALHAMENTO DAS TABELAS DO SISTEMA DE VERSIONAMENTO DE ARQUIVOS

Tabela Usuario

Campo		Tipo	Descrição
UsuarioID	PK	Integer	Id do usuario
Login	Unique	Varchar(20)	Login do usuário
Senha		Varchar(10)	Senha do usuário
Email	Unique	Varchar(60)	E-mail do usuário
Tipo		Char(1)	Tipo do usuário (Supervisor ou comum)
DtInclusao		Date	Data de inclusão do usuário

Tabela Historico

Campo		Tipo	Descrição
HistoricoID	PK	Integer	Id do histórico
UsuarioID	FK	Integer	Id do usuário
ArquivoID	FK	Integer	Id do arquivo
DtDownload		Date	Data de download
DtUpload		Date	Data de upload
Projeto		Varchar(255)	Nome do projeto
Versao		Integer	Versão do arquivo
Notas		Varchar(255)	Notas do arquivo alterado

Tabela Arquivo

Campo		Tipo	Descrição
ArquivoID	PK	Integer	Id do arquivo
Nome		Varchar(255)	Nome do arquivo
VersaoOriginal		Integer	Versão original
VersaoAtual		Integer	Versão atual
Status		Bit	Status (Livre ou Bloqueado)
Projeto		Varchar(255)	Nome do projeto
BloqueadoPor		Integer	Id do usuário com o arquivo bloqueado
Personalizado		Varchar(200)	Notas do arquivo
DtInclusao		Date	Data de cadastro do arquivo

Figura 4.2 – Detalhamento das tabelas (Cliente)

Módulo Servidor:

Tabela Config

Campo		Tipo	Descrição
PermiteAcesso		Byte	Servidor liberado (1ou 0)
PersonalizadoLabel		Varchar(200)	Texto padrão

Figura 4.3 – Detalhamento das tabelas (Servidor)

4.3 Interfaces

Além da interface gráfica criada, destaca-se a interface existente entre o módulo servidor e o sistema responsável pela comparação dos arquivos. Essa interface é chamada quando o usuário queira realizar uma comparação entre as versões do objeto dentro do repositório.

O sistema CVS passa os parâmetros necessários via linha de comando e o sistema responsável pela comparação de arquivos, que deve estar instalado na máquina do usuário, retorna a informação desejada.

4.4 Casos de Uso

O Diagrama de Casos de Uso tem o objetivo de auxiliar a comunicação entre os analistas e o cliente. Ele descreve cenários que mostram as funcionalidades do sistema do ponto de vista do usuário.

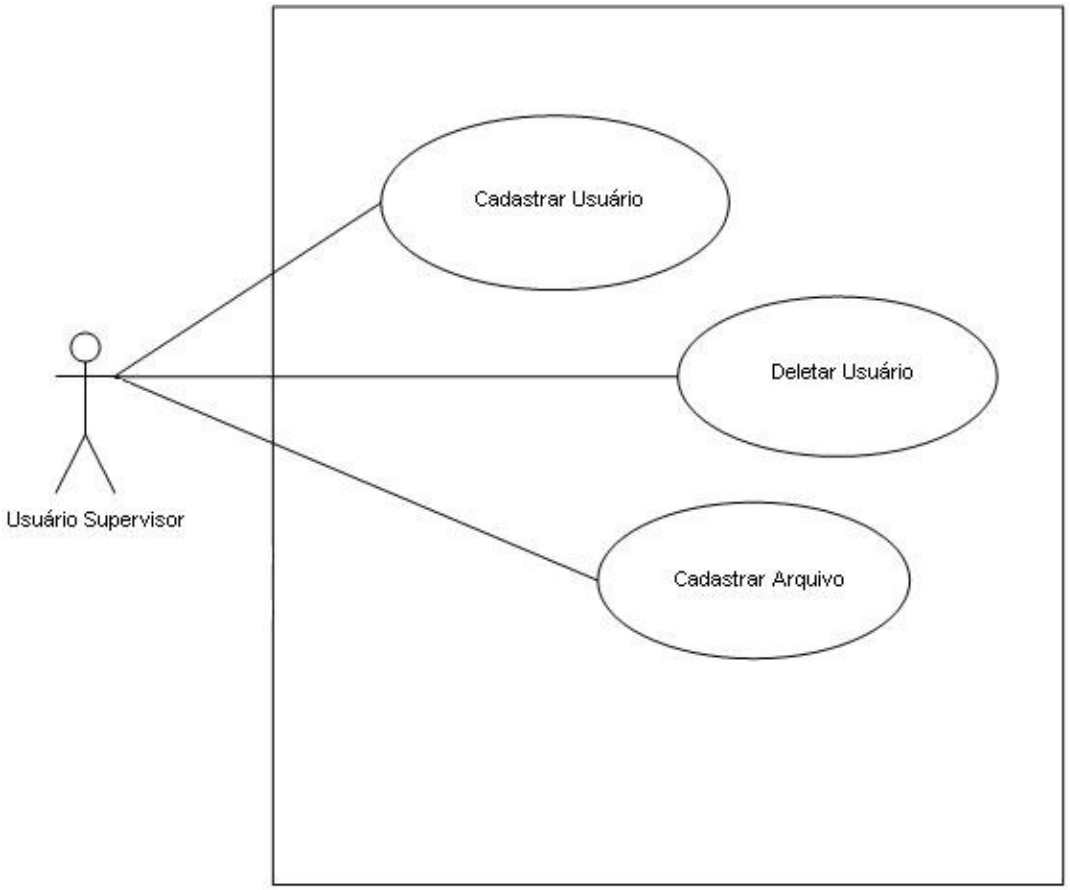


Figura 4.4 – Casos de Uso

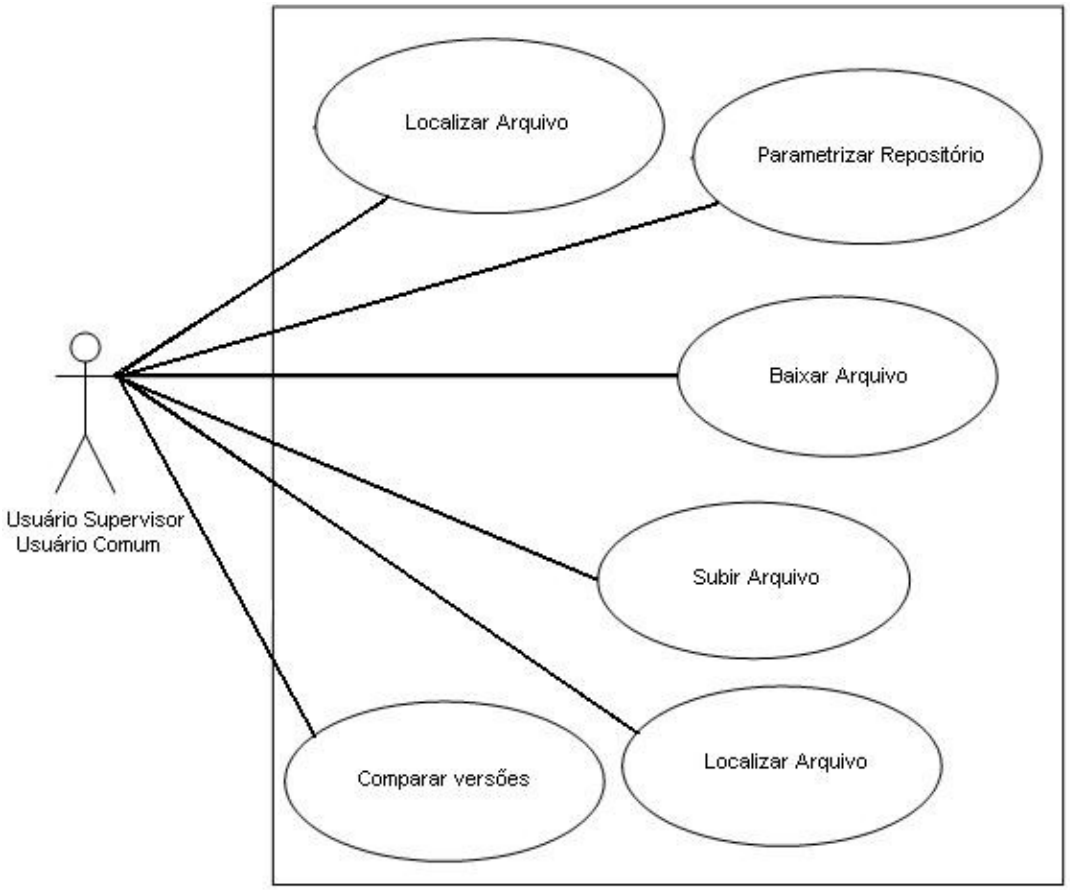


Figura 4.5 – Casos de Uso

Os casos de uso listados para esse sistema e descritos no capítulo anterior são:

1. Localizar Arquivo
2. Cadastrar Arquivo
3. Deletar Usuário
4. Parametrizar Repositório
5. Baixar Arquivo
6. Subir Arquivo
7. Ver Histórico
8. Comparar Versões

4.5 Problemas / Soluções

Desde o momento da idéia inicial de se fazer um versionador de arquivos, a etapa de comparação entre dois arquivos era o grande desafio a ser superado. Por ser uma aplicação que utilizava de dois módulos distintos (cliente e servidor), a comparação entre os arquivos e o controle sobre eles eram processos críticos para que o sistema fosse seguro e confiável.

A solução para a questão da comparação entre dois arquivos foi encontrada em um tópico abordado na disciplina Sistemas Operacionais chamado Comunicação entre Processos. Este é um tópico de comunicação, como o nome diz, e não de comparação de arquivos. A comparação pode usar essa comunicação, mas é algo diferente.

O sistema operacional fornece mecanismos para facilitar a comunicação e compartilhamento de dados entre aplicativos. As atividades iniciadas por estes mecanismos são chamadas de comunicações entre processos (IPC). Algumas formas de IPC facilitam a divisão do trabalho entre os vários processos e outras facilitam a divisão do trabalho entre computadores em uma rede.

Normalmente, os aplicativos podem usar IPC classificados como clientes ou servidores. Um cliente é um aplicativo ou um processo que solicita um serviço de algum outro aplicativo ou processo. Um servidor é um aplicativo ou um processo que responde a uma solicitação do cliente. Muitos aplicativos atuam tanto como um cliente ou servidor, dependendo da situação.

Dessa forma, o sistema interage com um aplicativo chamado WinMerge via de linha de comando e esta aplicação externa faz a comparação entre os arquivos. A solução faz uso da ferramenta livre chamada WinMerge, versão 2.12.4, para chegar a um objetivo definido.

A solução para a questão da comparação entre dois arquivos utilizou um programa externo, chamado pela aplicação desenvolvida via linha de comando.

Para se obter informações sobre esse sistema, pode-se acessar o site http://winmerge.org/ [6].

4.6 Arquitetura

Como já foi mencionado, o projeto é dividido em dois módulos: o *módulo cliente* e o *módulo servidor*. O Primeiro será utilizado nas estações de trabalho (máquinas locais) para se conectar ao servidor (módulo servidor). O segundo será responsável por manter e controlar o repositório de arquivos.

A utilização dessa arquitetura para a implementação do sistema atende muito bem a uma equipe de desenvolvimento de software. Baseia-se em um modelo onde há um repositório central (servidor) e cópias de trabalho para os desenvolvedores (clientes), de acordo com a figura abaixo.

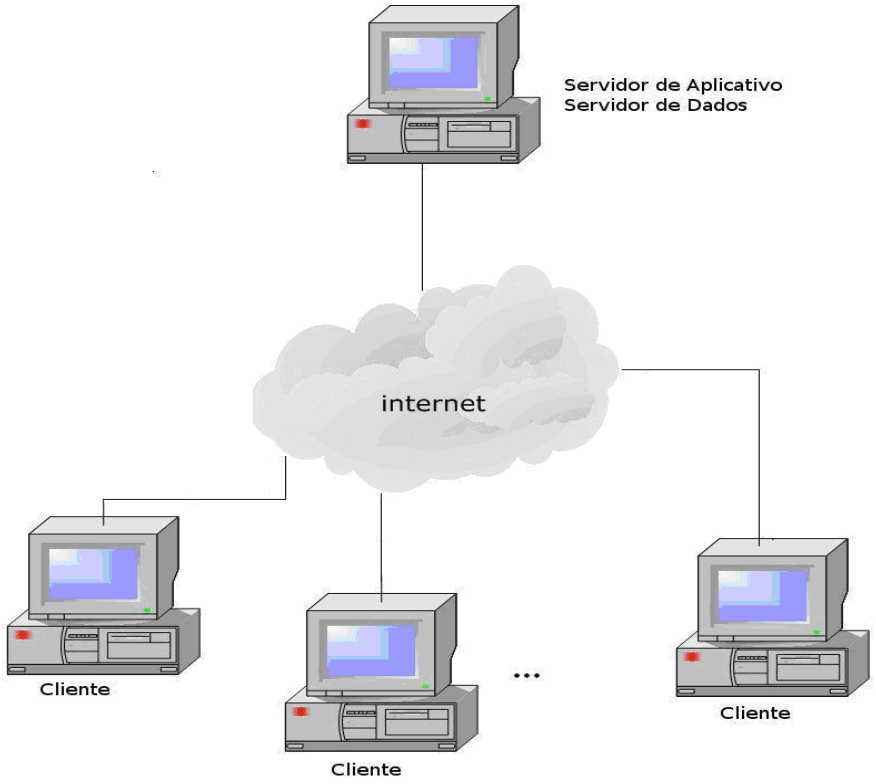


Figura 4.6 - Modelo cliente-servidor

Cliente-servidor é um modelo computacional que separa clientes e servidores, sendo interligados entre si geralmente utilizando-se uma rede de computadores. Cada instância de um cliente pode enviar requisições de dado para algum dos servidores conectados e esperar pela resposta. Por sua vez, algum dos servidores disponíveis pode aceitar tais requisições, processá-las e retornar o resultado para o cliente.

Muitas vezes os clientes e servidores se comunicam através de uma rede de computador com hardwares separados, mas o cliente e servidor podem residir no mesmo sistema. A máquina servidora é um host que está executando um ou mais programas de servidor que partilham os seus recursos com os clientes.

Um cliente não compartilha de seus recursos, mas solicita o conteúdo de um servidor ou função de serviço. Os clientes, portanto, iniciam sessões de comunicação com os servidores que esperam as solicitações de entrada.

4.7 Conclusão

Neste capítulo, foi abordada a etapa de desenvolvimento da aplicação. As etapas de implementação desse projeto, como modelagem de banco de dados, interfaces e casos de uso foram descritos. A questão dos problemas encontrados na fase de construção e suas soluções também foram expostos.

Com a finalização da parte de implementação do trabalho, serão abordados, no próximo capítulo, vários aspectos da implementação, especialmente o que se refere à verificação e validação de todas as funcionalidades que um sistema de versionamento de arquivos deve possuir.

Capítulo 5 – Resultados

5.1 Introdução

Este capítulo é dedicado aos resultados da implementação desse projeto. Após termos abordado todo o processo que levou à elaboração desse sistema, desde a identificação do problema até sua solução, e contemplando todo o detalhamento técnico da solução, apresentaremos a estrutura de funcionamento da aplicação, explicando a função de cada módulo do sistema.

Na seção 5.2, apresentaremos a estrutura do funcionamento da aplicação, explicando a função de cada módulo. Na seção 5.3, apresentaremos o resultado dos testes de sistema, ou seja, uma validação feita contra a análise realizada previamente. Serão evidenciados todos os resultados dos cenários de testes, contemplando todos os resultados esperados para um sistema de controle de versão centralizado, como proposto neste projeto.

5.2 Estrutura do Funcionamento da Aplicação

Nesta seção, será explicado o funcionamento da aplicação desenvolvida. Pode-se dizer que a aplicação é dividida em duas partes: Módulo Servidor e Módulo Cliente. Esses dois módulos da aplicação serão detalhados de uma forma funcional, de acordo com a aplicação desenvolvida.

O módulo Servidor servirá de repositório de arquivos onde os mesmos serão armazenados para serem manipulados pelos usuários. Esse módulo é responsável por toda a inteligência envolvida nesse sistema de controle de versões, pois é através dele que é feito o controle de histórico, versão e resgate de todos os arquivos. O módulo cliente, por sua vez, é utilizado nas estações de trabalho dos usuários para se conectar ao servidor (módulo servidor).

Com a arquitetura utilizada, o módulo servidor armazena a(s) versão(ões) atuais do projeto e seu histórico, e os clientes se conectam a esse servidor para obter uma cópia completa do projeto, trabalhar nessa cópia e então devolver suas modificações. Nesse sistema, cliente e servidor estão conectados por uma rede local, mas o cliente e o servidor podem estar na mesma máquina se a configuração do sistema for feita de maneira a dar acesso a versões e histórico do projeto apenas a usuários locais. Tanto o servidor quanto o cliente dessa aplicação utilizam e foram desenvolvidos para o sistema operacional Windows.

Vários clientes podem editar cópias do mesmo projeto de maneira concorrente. Quando eles confirmam suas alterações, o servidor gera uma nova versão do arquivo. Somente os usuários com arquivos bloqueados para seus respectivos usuários podem alterar o arquivo e confirmar sua atualização dentro do repositório. Os usuários que não tenham um determinado arquivo bloqueado têm acesso a ele somente para leitura. Com a validação da alteração bem sucedida, o número de versão do arquivo é incrementado, e o servidor escreve uma linha de observação (fornecida pelo usuário), a data e o autor das alterações na tela de histórico do sistema.

No programa implementado neste projeto, clientes podem comparar diferentes versões de um arquivo, pedir um histórico completo das alterações, ou baixar uma determinada versão do projeto, ou de uma data específica, não necessariamente a versão mais atual.

Clientes também podem usar o comando "update" para manter suas cópias locais atualizadas com a última versão do servidor. Isso elimina a necessidade de se fazer diversos downloads de todo o projeto.

O CVS também pode manter diferentes "estados" do projeto. Por exemplo, uma versão do software pode ser um desses estados, usado para correção de bugs, enquanto outra versão, que está realmente sob desenvolvimento, sofrendo alterações e tendo novas funcionalidades implementadas, forma o outro estado.

Abaixo é ilustrada de forma detalhada a estrutura de funcionamento da aplicação.

Ao executar o programa em seu lado Servidor, o administrador do sistema verá uma janela como mostrado na Figura 5.1. Essa tela indica que o repositório está configurado e preparado para gerenciar os arquivos a serem alterados pelos usuários.

Figura 5.1 – Janela inicial da aplicação (Servisor).

Ao executar o programa, o supervisor verá uma janela como mostrado na Figura 5.2. Antes de digitar o Login e a Senha, o supervisor deve clicar em *Configurar Repositório (server)*. Com isso uma janela de configuração vista na Figura 5.3 irá se abrir. Nesta janela o supervisor deve parametrizar o caminho do repositório no servidor. Essa informação é guardada em um arquivo de configuração do sistema chamado Config.ini.

Figura 5.2 – Janela inicial da aplicação (Cliente).

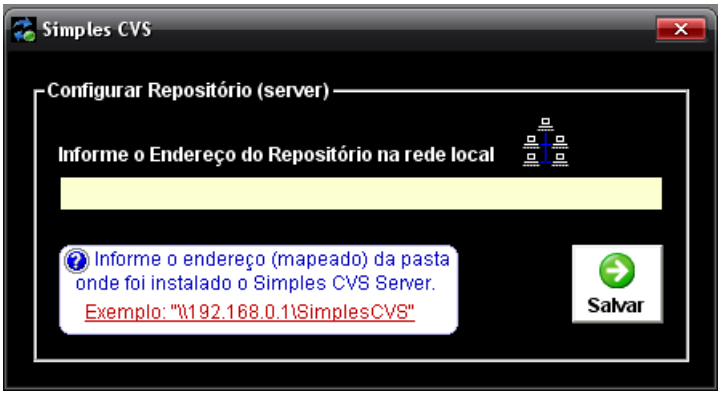


Figura 5.3 – Janela de configuração, na qual o usuário define o caminho do repositório no servidor.

Após um usuário com status supervisor se logar no sistema, a interface mostrada na Figura 5.4 será apresentada. O Menu ilustra as funções permitidas a esse tipo de usuário, além da possibilidade de ser acessado o menu do usuário não-supervisor clicando-se no checkbox conforme figura abaixo. Serão ilustradas em detalhes as telas para cada item do menu.

Figura 5.4 – Menu de usuário - Supervisor.

O usuário com status de Supervisor cadastra os arquivos no repositório para que os usuários com status não-supervisor possam customizá-los caso seja necessário. Após o usuário clicar em *Cadastrar Arquivo* será apresentada a seguinte tela, conforme Figura 5.5.

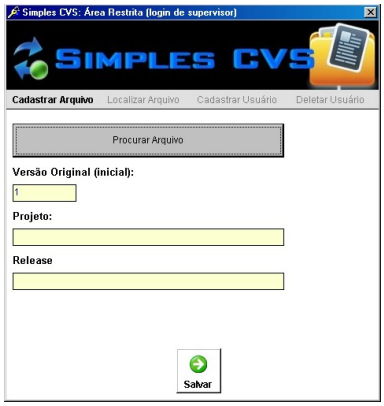


Figura 5.5 – Cadastro de arquivos.

O usuário com status de Supervisor tem a possibilidade de verificar os arquivos que estão no repositório e obter informações como versão corrente, status e projeto relacionado aos arquivos. Após o usuário clicar em *Localizar Arquivo* será apresentada a seguinte tela, conforme Figura 5.6.

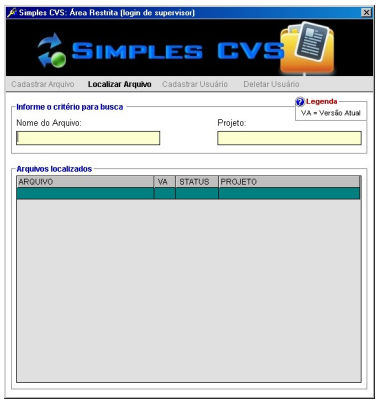


Figura 5.6 – Localização de arquivos.

O usuário com status de Supervisor tem a possibilidade de cadastrar novos usuários para o sistema. Há a possibilidade de ser cadastrado um usuário com status Supervisor ou não-supervisor, conforme as informações obrigatórias no cadastro. Após o usuário clicar em *Cadastrar Usuário* será apresentada a seguinte tela, conforme Figura 5.7.

Figura 5.7 – Cadastro de Usuário.

O usuário com status de Supervisor tem a possibilidade de Deletar usuários do sistema. Todos os usuários podem ser deletados, com exceção do usuário Admin. Após o usuário clicar em *Deletar Usuário* será apresentada a seguinte tela, conforme Figura 5.8.

Figura 5.8 – Deleção de Usuário.

Após um usuário com status não-supervisor se logar no sistema, a interface mostrada na Figura 5.9 será apresentada. O Menu ilustra as funções permitidas a esse tipo de usuário. Serão ilustradas em detalhes as telas para cada item do menu.

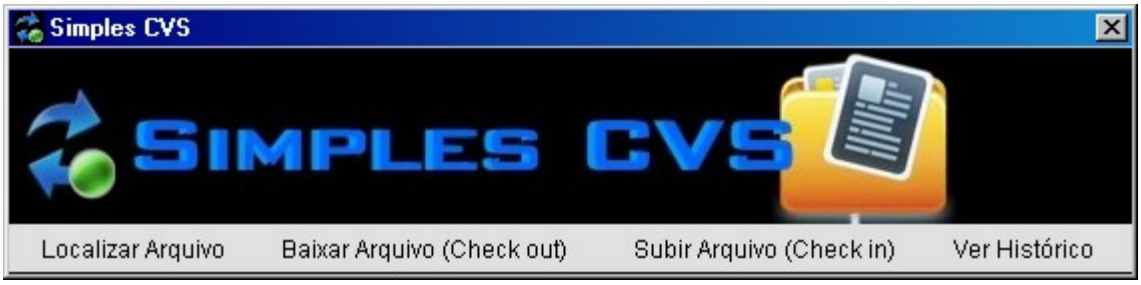


Figura 5.9 – Menu de Usuário.

O usuário do sistema tem a possibilidade de Localizar os arquivos que estão no repositório e verificar a versão corrente, alem do status e do projeto relacionado a eles. Após o usuário clicar em *Localizar Arquivo* será apresentada a seguinte tela, conforme Figura 5.10.

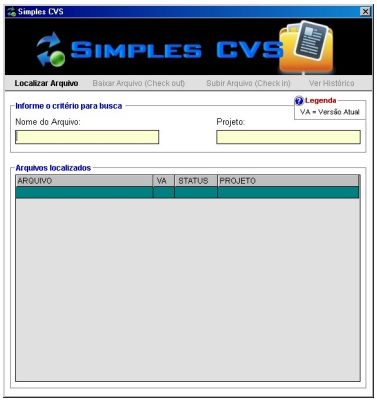


Figura 5.10 – Localização de arquivos.

Como um aplicativo versionador de código, o usuário do sistema tem a possibilidade de baixar os arquivos cadastrados previamente por algum supervisor do sistema. Ao baixar um arquivo, o usuário pode locá-lo no repositório ou simplesmente baixar uma copia do arquivo de interesse para sua estação de trabalho. Caso o arquivo já esteja locado por algum outro usuário, não será possível locar o arquivo. Após o usuário clicar em *Baixar Arquivo (Check out)* será apresentada a seguinte tela, conforme Figura 5.11.

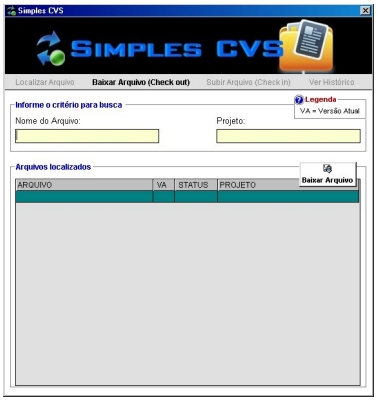


Figura 5.11 – Baixar arquivos.

Como um aplicativo versionador de código, após o usuário do sistema baixar os arquivos e alterá-los, se faz necessário atualizar as alterações dos arquivos no servidor. Para esse fim, o usuário precisa atualizar os arquivos por ele locado e alterado no repositório. Após fazer essa atualização, o usuário libera o arquivo no repositório para que outro usuário possa alterá-lo caso seja necessário. Após o usuário clicar em *Subir Arquivo (Check in)* será apresentada a seguinte tela, conforme Figura 5.12.

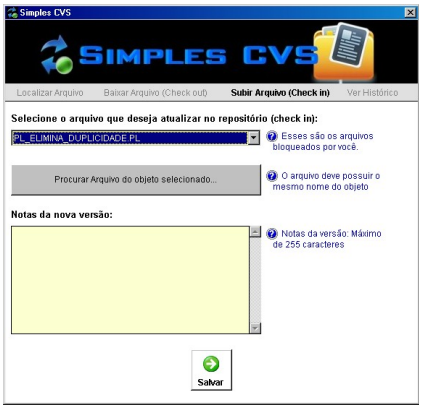


Figura 5.12 – Subir arquivos.

O usuário do sistema tem a possibilidade de analisar o histórico dos arquivos cadastrados no sistema. Essa função permite mostrar um resumo das alterações por arquivo, seu status, além de um comparativo entre as versões alteradas por usuários diferentes. Após o usuário clicar em *Ver Histórico* será apresentada a tela referente ao histórico, conforme a figuras 5.13. As figuras

5.14 e 5.15 são referentes às funções de comparação entre os arquivos de versões diferentes.

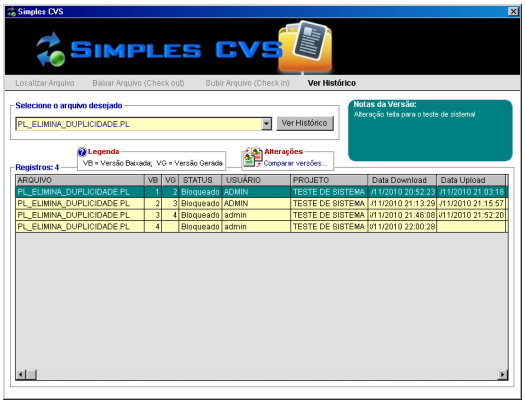


Figura 5.13 – Histórico de arquivos.

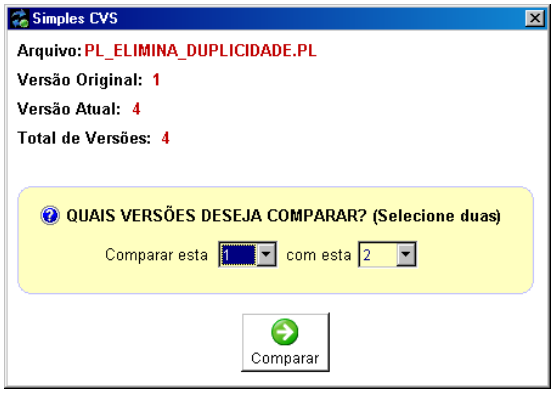


Figura 5.14 – Comparação de versões.

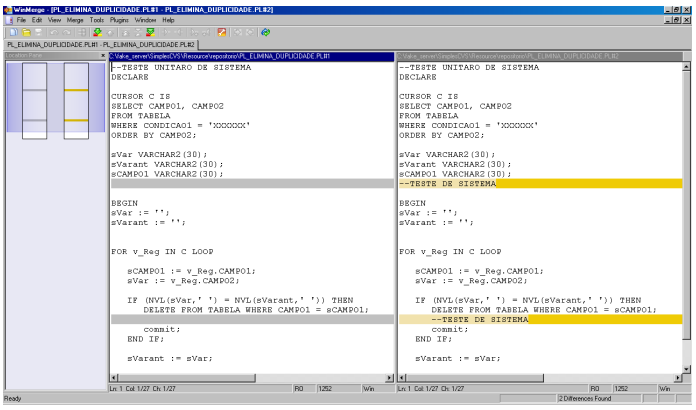


Figura 5.15 – Comparação de versões.

5.3 Teste de Sistema

Após abordada toda a estrutura do funcionamento da aplicação, um teste de sistema foi elaborado para garantir que todas as funcionalidades estejam implementadas de acordo com o esperado.

Nesta seção, será apresentado o resultado de um teste de sistema elaborado para o projeto, onde todas as funcionalidades do sistema estão contempladas. Foram anexadas as evidências de teste para cada cenário, como pode ser verificado nas informações abaixo.

Os cenários mapeados para o teste de sistema foram:

1. Inicialização do Servidor
2. Cadastro de usuário comum
3. Cadastro de usuário supervisor
4. Log in de usuário com servidor nao-inicializado
5. Log in de usuário supervisor com servidor inicializado
6. Log in de usuário comum com servidor inicializado
7. Deleção de usuário
8. Cadastro de arquivo
9. Cadastro de arquivo com extensão diferente de .sql,.pl e .sh
10. Cadastro de arquivo já existente no repositório
11. Localização de arquivo
12. Download de arquivo
13. Upload de arquivo
14. Upload de arquivo sem bloqueio
15. Visualização de histórico
16. Comparação de versões
17. Comparação de versões de um arquivo com apenas uma versão no repositório

Cenário 1	Inicialização do Servidor

Ator	Usuário supervisor
Descrição	Usuário inicializa o módulo Servidor da aplicação
Resultado	Servidor é inicializado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

	personalizado_label	permiteAcesso
▶	Release	1
*		

Figura 5.16 – Evidências de teste de sistema.

Cenário 2	Cadastro de usuário
Ator	Usuário supervisor
Descrição	Usuário cadastra um usuário do tipo não-supervisor no sistema
Resultado	Usuário cadastrado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

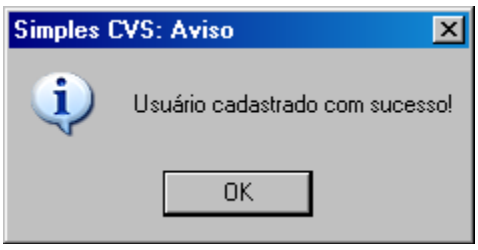
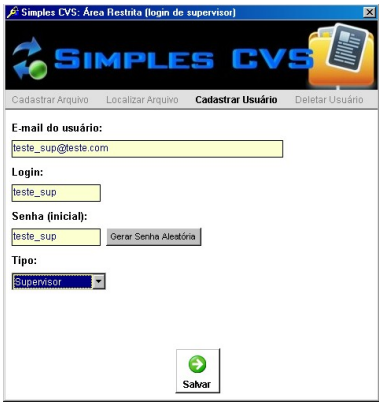


Figura 5.17 – Evidências de teste de sistema.

Cenário 3	Cadastro de usuário
Ator	Usuário supervisor
Descrição	Usuário cadastra um usuário do tipo Supervisor no sistema
Resultado	Usuário cadastrado com sucesso.
Comentários	Cenário executado com sucesso

Evidências:



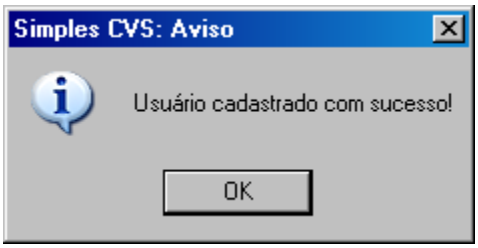


Figura 5.18 – Evidências de teste de sistema.

Cenário 4	Cadastro de usuário
Ator	Usuário supervisor
Descrição	Usuário cadastra um usuário que já existe no sistema
Resultado	Mensagem de erro
Comentários	Cenário executado com sucesso

Evidências:

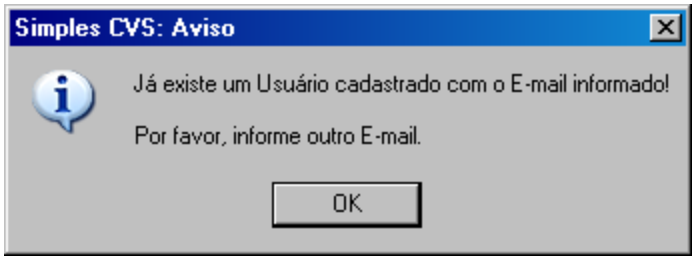


Figura 5.19 – Evidências de teste de sistema.

Cenário 5	Log in de usuário
Ator	Usuário comum
Descrição	Usuário se loga no sistema com o módulo servidor não-inicializado
Resultado	Mensagem de erro
Comentários	Cenário executado com sucesso

Evidências:

	personalizado_label	permiteAcesso
	Release	0
▶		

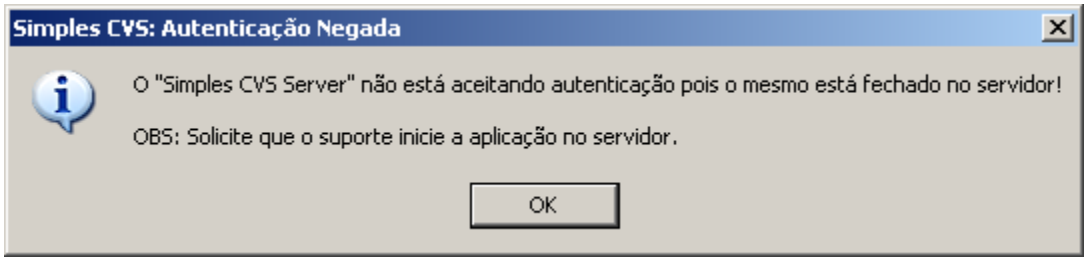


Figura 5.20 – Evidências de teste de sistema.

Cenário 6	Log in de usuário
Ator	Usuário supervisor
Descrição	Usuário se loga no sistema com o módulo servidor inicializado
Resultado	Usuário logado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

Figura 5.21 – Evidências de teste de sistema.

Cenário 7	Log in de usuário
Ator	Usuário comum
Descrição	Usuário se loga no sistema com o módulo servidor inicializado
Resultado	Usuário logado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

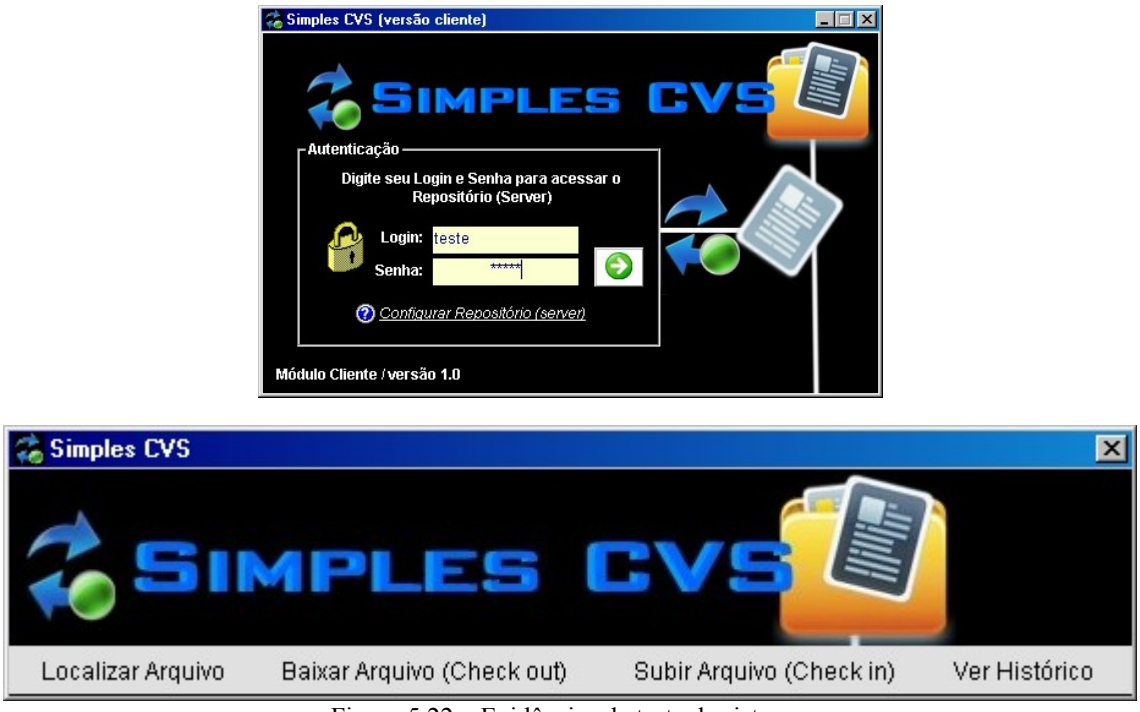


Figura 5.22 – Evidências de teste de sistema.

Cenário 8	Deleção de usuário

Ator	Usuário supervisor
Descrição	Usuário deleta um usuário no sistema
Resultado	Usuário deletado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

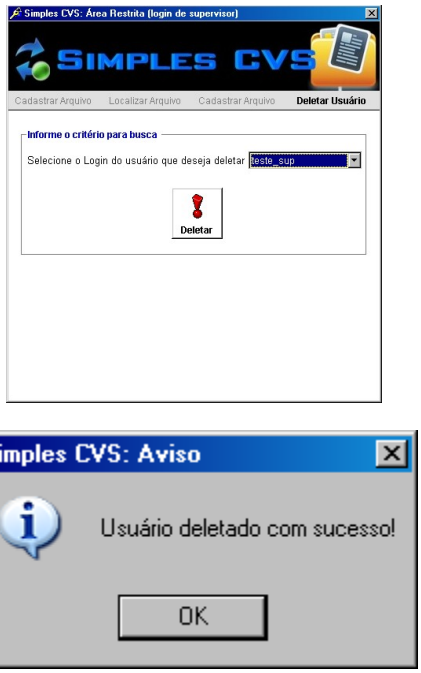


Figura 5.23 – Evidências de teste de sistema.

Cenário 9	Cadastro de arquivo
Ator	Usuário supervisor
Descrição	Usuário cadastra um arquivo no sistema
Resultado	Arquivo cadastrado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

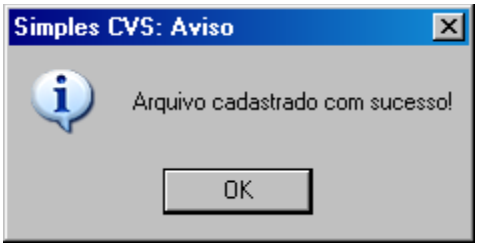


Figura 5.24 – Evidências de teste de sistema.

Cenário 10	Cadastro de arquivo
Ator	Usuário supervisor
Descrição	Usuário cadastra um arquivo com extensão diferente de .sql, .pl ou .sh no sistema
Resultado	Arquivo cadastrado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

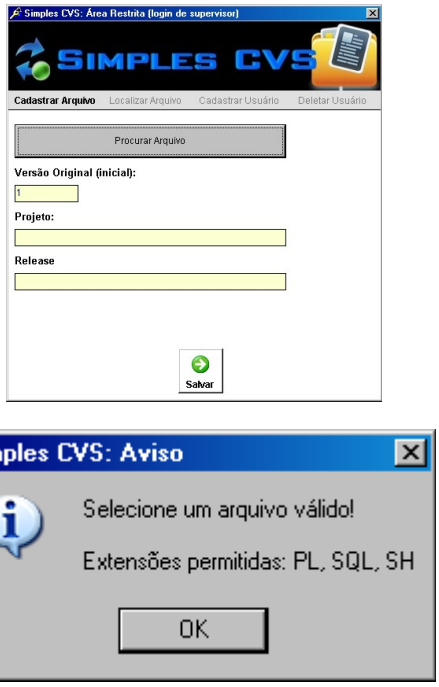


Figura 5.25 – Evidências de teste de sistema.

Cenário 11	Cadastro de arquivo
Ator	Usuário supervisor
Descrição	Usuário cadastra um arquivo já existente no repositório do sistema
Resultado	Mensagem de erro
Comentários	Cenário executado com sucesso

Evidências:

Figura 5.26 – Evidências de teste de sistema.

Cenário 12	Localização de arquivo
Ator	Usuário supervisor / Usuário comum
Descrição	Usuário localiza um arquivo no sistema
Resultado	Arquivo localizado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

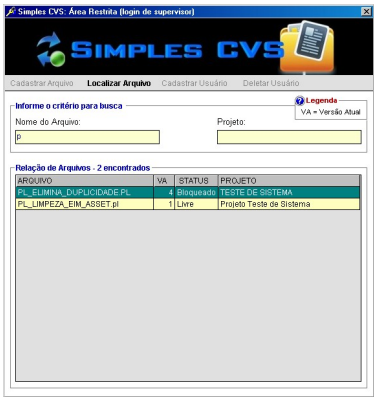


Figura 5.27 – Evidências de teste de sistema.

Cenário 13	Baixar arquivo
Ator	Usuário supervisor / Usuário comum
Descrição	Usuário baixa um arquivo disponível no repositório para sua máquina local
Resultado	Arquivo baixado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

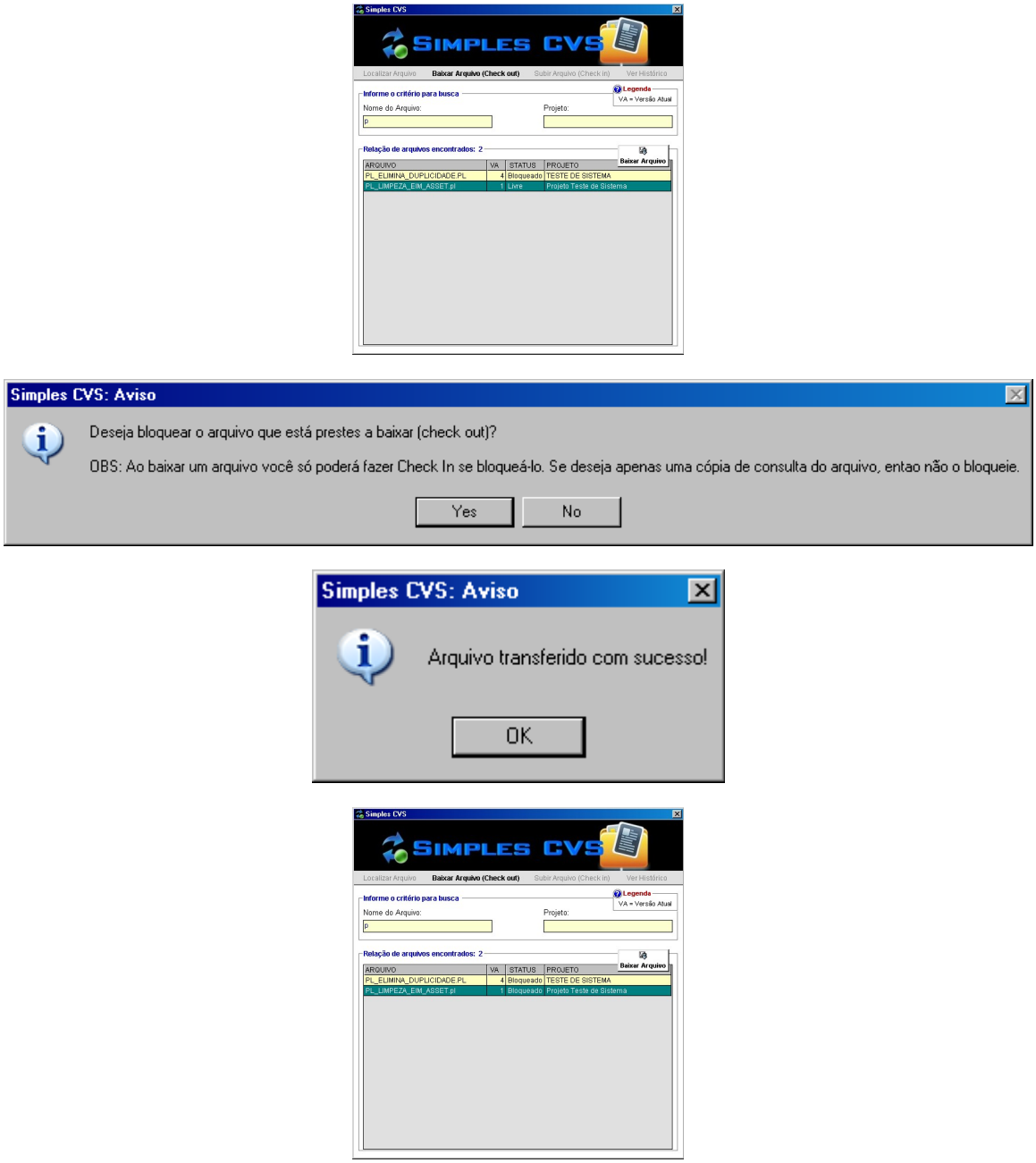


Figura 5.28 – Evidências de teste de sistema.

Cenário 14	Subir arquivo editado
Ator	Usuário supervisor / Usuário comum
Descrição	Usuário sobe um arquivo alterado e bloqueado para o servidor
Resultado	Arquivo atualizado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

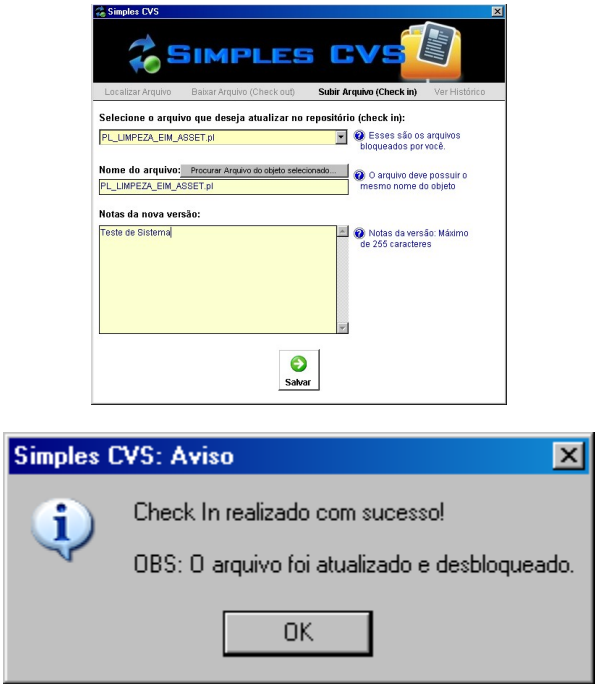


Figura 5.29 – Evidências de teste de sistema.

Cenário 15	Subir arquivo
Ator	Usuário supervisor / Usuário comum
Descrição	Usuário sobe um arquivo para o servidor sem ter arquivos bloqueados para ele
Resultado	Mensagem de erro
Comentários	Cenário executado com sucesso

Evidências:

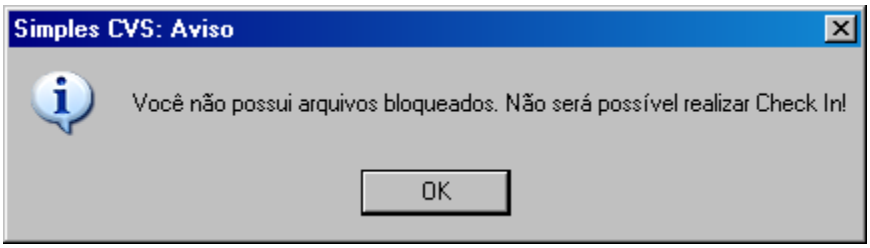


Figura 5.30 – Evidências de teste de sistema.

Cenário 16	Visualização de histórico de arquivos
Ator	Usuário supervisor / Usuário comum
Descrição	Usuário visualiza histórico do arquivo
Resultado	Histórico visualizado com sucesso
Comentários	Cenário executado com sucesso

Evidências:

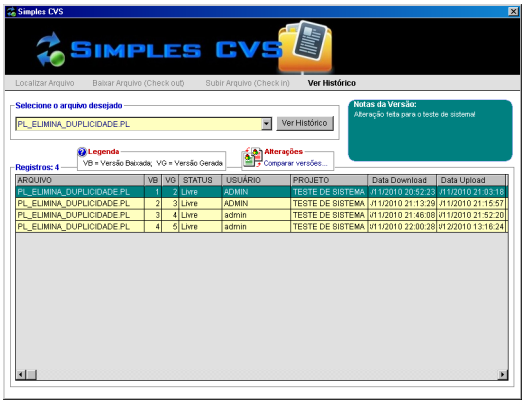
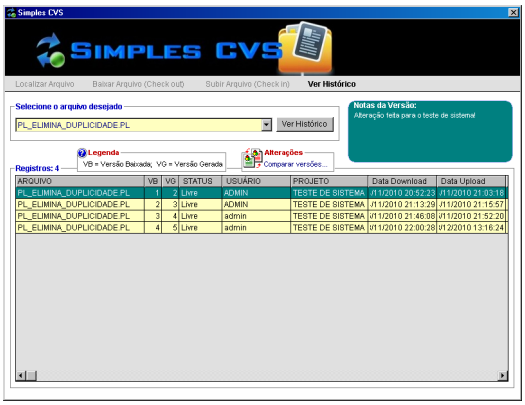


Figura 5.31 – Evidências de teste de sistema.

Cenário 17	Comparação de versão de arquivos
Ator	Usuário supervisor / Usuário comum
Descrição	Usuário compara versões do mesmo arquivo
Resultado	Versões comparadas com sucesso
Comentários	Cenário executado com sucesso

Evidências:



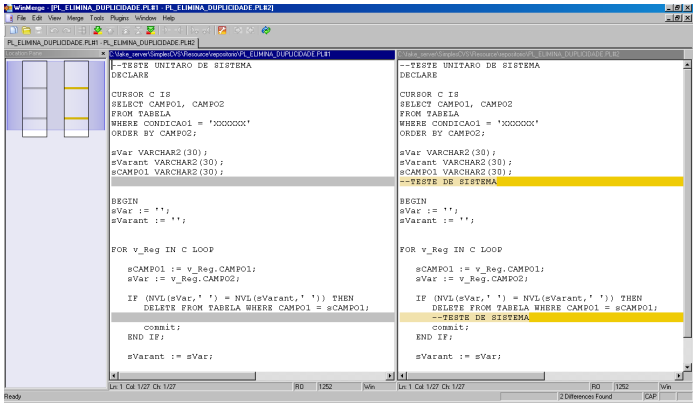
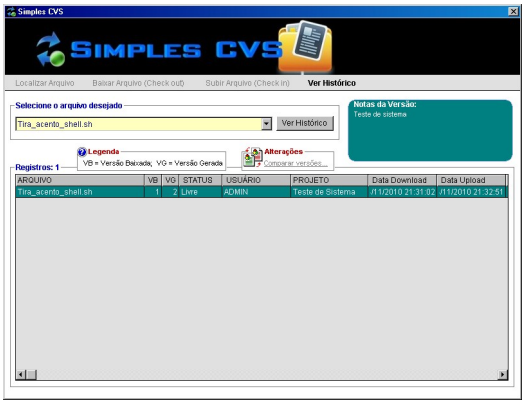


Figura 5.32 – Evidências de teste de sistema.

Cenário 18	Comparação de versão de arquivos
Ator	Usuário supervisor / Usuário comum
Descrição	Usuário compara versões de um arquivo com apenas uma versão no repositório
Resultado	Mensagem de erro
Comentários	Cenário executado com sucesso

Evidências:



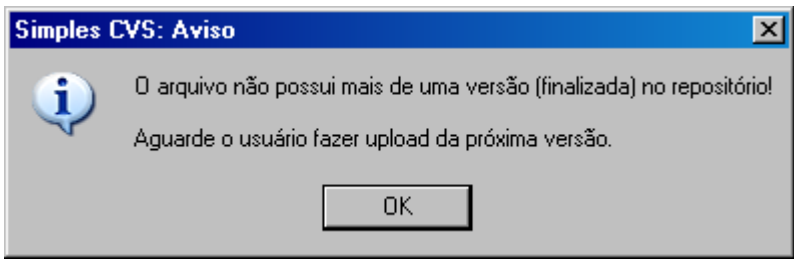


Figura 5.33 – Evidências de teste de sistema.

5.4 Conclusão

Neste capítulo descrevemos o resultado da implementação desse projeto. Foram abordados os resultados e apresentada a estrutura do funcionamento da aplicação, explicando a função de cada módulo. Foi elaborado e evidenciado o resultado dos testes de sistema, ou seja, uma validação feita contra a análise realizada previamente.

No próximo capítulo iremos concluir o projeto, destacando os principais pontos do projeto e trabalhos futuros que enriqueceriam o que já foi implementado até agora.

Capítulo 6 – Resumo, Avaliação e Conclusão

6.1 Resumo

Neste item será feito um breve resumo do que foi abordado no trabalho, dando ênfase aos tópicos mais relevantes. Serão apontados, também, possíveis trabalhos futuros que contribuam para o aperfeiçoamento do sistema desenvolvido, além de uma avaliação das ferramentas utilizadas e o conhecimento agregado com a elaboração desse projeto.

O projeto teve como objetivo abordar a conjuntura atual dos sistemas de versionamento de arquivos, seus principais componentes e criar um sistema que contribua para a resolução do problema apresentado em um ambiente de desenvolvimento coletivo de sistemas.

Para propor um sistema de versionamento de arquivos, foi necessário, primeiramente, avaliar as ferramentas que seriam utilizadas para sua construção. Para isto, dedicou-se o Capítulo 2, onde foi evidenciado o problema atual de versionamento e apresentada uma descrição breve da solução. Junto a

isso, foi apresentada uma restrição ao projeto que foi o uso da linguagem Visual Basic, por ser cultura da instituição.

No Capítulo 3, foi abordada a etapa de análise do projeto e seus resultados esperados. Foi elaborado um Diagrama de Classes e uma apresentação da interface gráfica do sistema (GUI). Neste capítulo é feita a especificação de cada caso de uso do projeto através do diagrama de casos de uso.

Todo o detalhamento de como a aplicação foi desenvolvida foi mostrado no Capítulo 4. Foi feita uma análise no que se refere a modelagem de banco de dados, interfaces do sistema e casos de uso. Foram apresentados também os problemas e soluções encontrados no decorrer da implementação do projeto, além da arquitetura da aplicação.

O capítulo 5 deste relatório foi dedicado aos resultados gerados pela aplicação. Os casos de usos foram executados criteriosamente e todos os resultados esperados foram verificados e validados de acordo com a análise sistêmica feita previamente.

6.2 Avaliação de ferramentas

A utilização da linguagem Visual Basic foi escolhida por ser a linguagem utilizada no dia-a-dia pelos desenvolvedores que farão uso do sistema desenvolvido. A experiência dos recursos nessa tecnologia permitirá que qualquer customização feita no sistema possa ser implementada por qualquer desenvolvedor.

Além disso, a utilização dessa linguagem se deu pelo fato do Visual Basic possuir um ambiente de desenvolvimento integrado (IDE - Integrated Development Environment) totalmente gráfico, facilitando a construção da

interface da aplicação e possuir tecnologias como DAO, RDO, e ADO que permitem fácil acesso a bases de dados.

6.3 Conclusão

Desde a elaboração da proposta deste projeto, a questão da comparação entre dois arquivos se mostrou o grande desafio a ser superado. Depois de algumas semanas tentando buscar uma solução, a idéia de utilizar um sistema externo para executar tal função se mostrou a mais viável. Foi nos conceitos de Comunicação entre Processos que encontrei a solução para esse problema. O projeto me permitiu estudar Sistemas Operacionais, Bancos de Dados, comparação entre programas e agregar um conhecimento bastante satisfatório.

O sistema resultante deste trabalho está em uso na empresa em que trabalho.

6.4 Trabalhos Futuros

Sistemas CVS é um assunto muito vasto e relativamente recente, devido a isto, são inúmeras as possíveis continuações e para este trabalho. Dentre elas, podemos destacar as seguintes propostas:

- Versionamento de todos os tipos de arquivos e extensões.
- Emissão de relatórios via e-mail.
- Fazer com que a aplicação suporte troca de mensagem entre os usuários com winsock.
- Incluir no sistema o projeto a que pertence cada arquivo, para permitir recuperar os arquivos de um projeto e estabelecer os direitos de acesso por projeto.

Referências Bibliográficas

[1] Cristiano Caetano. CVS: Controle de Versões e Desenvolvimento Colaborativo de Software, Novatec, 2004.

[2] Marcelo Malheiros. Sistema de controle de Versão, *V seminário de desenvolvimento em software livre* (UNIVATES).

[3] Marden Neubert, CVS: Guia de Consulta Rápida, Novatec, 2004.

[4] Jennifer Vesperman, Essencial CVS, O'Reilly Media, 2003.

[5] Ronald L. Brown e Mitchell C. Kerman, “Computer Programming Fundamentals With Applications in Visual Basic 6”, Addison-Wesley, 1999.

[6] The WinMerge Development Team. Disponível em <http: //winmerge.org> Acesso em 01/Nov/2010