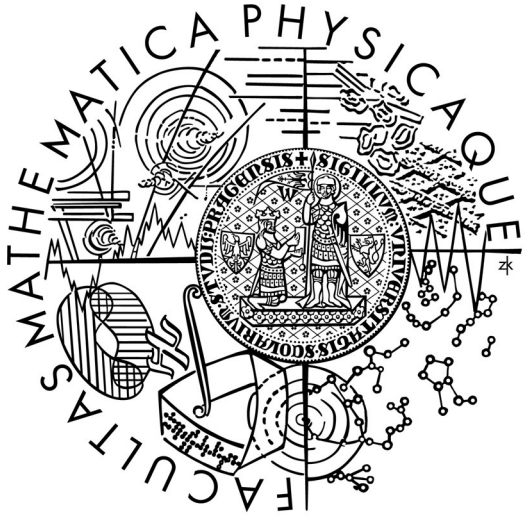
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta



BAKALÁŘSKÁ PRÁCE

Michal Foltýn

Systém pro správu versí

Katedra software a výuky informatiky

Vedoucí bakalářské práce:
RNDr. Tomáš Holan, Ph.D.

Studijní program: Informatika, Programování

2006

Na tomto místě bych rád poděkoval vedoucímu bakalářské práce, RNDr. Tomáši Holanovi, Ph.D. za podporu a věcné připomínky. Dále bych rád poděkoval svým rodičům a přátelům za neustálou podporu při psaní této práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně za použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 30.05.2006 Michal Foltýn

Obsah

Název práce: Systém pro správu versí

Autor: Michal Foltýn

Katedra (ústav): Katedra software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.

E-mail vedoucího: Tomas.Holan@mff.cuni.cz

Abstrakt: Systém pro správu versí VCS sleduje veškerou práci a změny na množině textových souborů, typicky při implementaci softwaru a dovoluje více vývojářům, kteří od sebe mohou být vzdáleni, spolupracovat. Systém zjednodušuje práci vývojovým týmům při správě více dokumentů a dovoluje jim sledovat změny, které na dokumentech proběhly. Systém je postaven na architektuře klient/server a umožňuje práci v síti i lokálně na jednom počítači. Dovoluje připojení více uživatelů zároveň. Systém je navržen pro platformu Microsoft Windows.

Klíčová slova: správa versí, verzování, CVS

Title: Version control system

Author: Michal Foltýn

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D.

Supervisor's e-mail address: Tomas.Holan@mff.cuni.cz

Abstract: Version control system VCS keeps track of all work and all changes in a set of text files, typically the implementation of a software project, and allows several, widely separated, developers to collaborate. System makes work easier to developers when working on several files and allows them to keep track of all changes that have occurred. System uses client/server architecture and allows users to connect remotely or have client and server on the same machine. Several users can work with system simultaneously. The system runs on Microsoft Windows platform.

Keywords: version control, versioning, CVS

1. Popis systému

1.1 Úvod

Systém pro správu verzí je určen především pro vývojáře, kteří potřebují jednoduchým způsobem uchovávat informace o změnách ve zdrojových kódech, které při vývoji softwaru nastaly. Takový systém uchovává jednotlivé verze zdrojových souborů a umožňuje uživateli návrat k libovolné verzi v historii.

Systém samozřejmě najde uplatnění i u lidí, kteří se vývojem softwaru nezabývají. Mohou například pracovat na dokumentech a potřebují mít přehled o změnách, které v čase provedli, popř. se k nějaké verzi vrátit. Možnosti a hlavně užitečnost takového nástroje lidé pocítí při práci v týmu (zejména nad stejnými dokumenty). Systém dokáže velmi rychle a přehledně zobrazit změny, které se provedly od poslední chvíle, kdy daný uživatel naposledy dokument uložil. Stejně jako dokáže zobrazit, kdo změny provedl, popř. komentář ke změnám. To s sebou přináší obrovské zjednodušení práce v týmu, protože není nutné vyhledávat kolegu a ptát se ho na provedené změny a jejich charakter.

Příklady systémů, které jsou dnes vývojovým týmům k dispozici, lze rozdělit do dvou skupin. Do první skupiny patří systémy, jejichž zdrojové kódy jsou volně k dispozici a jejich používání je zdarma (tzv. Open-source systémy). Do druhé skupiny patří produkty, jež jsou vlastnictvím konkrétních společností a jejich použití se již řídí licenční politikou dané firmy.

Do první skupiny lze zařadit systém CVS (Concurrent versions system) a také SubVersion. Do druhé skupiny můžeme zařadit systém SourceSafe společnosti Microsoft nebo TeamSource společnosti Borland. Samozřejmě existuje velké množství dalších systémů, zde jsou uvedeny pouze ty nejpoužívanější.

1.2 Cíl práce

Úkolem této bakalářské práce je navrhnout a implementovat systém pro správu verzí textových dokumentů. Při návrhu takového systému se bude zejména počítat s možným budoucím vývojem a tedy bude kladen důraz na jednoduchou rozšiřitelnost systému. Systém bude již od začátku navrhován tak, aby pracoval s podporou síťové komunikace, tedy na architektuře klient/server. Součástí úlohy je implementace serverové části, která bude poskytovat všechny potřebné prostředky pro správu verzí a klientské části, která bude využívat těchto prostředků a nabídne uživateli příjemné rozhraní pro práci s tímto systémem. Serverová část bude navržena a implementována tak, aby bylo možné implementovat vlastní klientskou část, čímž bude docíleno nezávislosti na konkrétní implementaci klientské části.

Systém bude navržen pro použití na platformě Microsoft Windows. Návrh bude směrován k použití především jednotlivci a malých až středně velkých týmů. Systém by měl administrátorům poskytnout jednoduchou a přehlednou správu

a uživatelům příjemné a intuitivní ovládání. Cílem je vytvořit takový software, který by byl bez jakékoliv složité administrace ihned k dispozici a byl přístupný i uživatelům, kteří s počítačem běžně pracují, nicméně nemají zkušenosti na to, aby prováděli náročnou instalaci. Předpokládá se poskytnutí takových možnosti, jaké jsou od systému tohoto typu očekávány a v neposlední řadě také poskytnout nástroje, které nejsou v konkurenčních systémech implementovány, popř. jsou implementovány v jednom, ale již ne v druhém.

Pro implementaci serverové a klientské části bude použita technologie .NET společnosti Microsoft a programovací jazyk C#.

1.3 Terminologie

Pro srozumitelnost dalšího textu se již předpokládá znalost některých termínů, které se při správě verzí používají. Uvedené termíny jsou ponechány bez překladu v anglickém jazyce, protože tyto termíny jsou již velmi zavedené a definování českých ekvivalentů, by pouze přineslo nedorozumění mezi stávajícími uživateli systémů pro správu verzí. Tabulka 1.1 obsahuje výčet termínů a jejich popis.

Repository	úložiště, kde systém jednotlivé verze dokumentů uchovává
Add	přidání dokumentu (složky, projektu) do repository
Get	uložení kopie dokumentu z repository na disk
Get latest version	uložení aktuální verze dokumentu z repository na disk
CheckOut	uložení aktuální verze dokumentu z repository na disk a současně její zablokování výhradně pro daného uživatele; ostatní uživatelé dokument nemohou upravovat
Commit	uložení dokumentu z disku na repository
CheckIn	uložení dokumentu do repository a jeho uvolnění pro další uživatele
Update	Aktualizace pracovní kopie na disku poslední verzí z repository
Branch	vytvoření větve projektu, tak aby se mohlo nad dokumenty pracovat nezávisle na sobě
Diff	algoritmus (program) pro porovnání obsahu dvou dokumentů
Compare	Funkce pro porovnání souboru na disku se souborem v repository popř. porovnání dvou verzí v repository

Tabulka 1.1: Terminologie

1.4 Analýza možných implementací

1.4.1 Serverová část

Serverová část systému VCS hraje v celém návrhu nejdůležitější roli, protože právě tato část bude implementovat logiku systému. Jelikož celý systém bude pracovat na architektuře klient/server, je třeba určit, jakým způsobem bude klient se serverem komunikovat. Existují dva možné způsoby implementace serverové části:

1. Webová aplikace – serverová část bude umístěna na webovém serveru a klienti k ní budou přistupovat pomocí protokolu http nebo https. Výhodou je jednoduchá implementace bez nutnosti řešit vzájemnou komunikaci serverové a klientské části. Nevýhodou takového řešení je značné omezení v implementaci, bez možnosti vlastního přizpůsobení.
2. Win32 aplikace (Windows service) – výhodou tohoto řešení je nezávislost při implementaci a úplná kontrola nad chováním aplikace včetně její komunikace s klienty. Nevýhodou je náročnější implementace, vlastní řešení zabezpečení a nutnost zajistit rozhraní pro komunikaci s klientem.

Implementace systémů, které se dnes běžně používají, vycházejí především z druhého způsobu, kdy základ systému tvoří aplikace napsaná jako Win32 aplikace, popř. aplikace pro platformu Unix. Teprve pro tyto aplikace vzniká rozhraní, které umožňuje přístup přes webový server. Příkladem takového systému je SubVersion, který jako webový server používá Apache.

1.4.2 Umístění repository

Při návrhu systému pro správu verzí je třeba rozhodnout, kde se budou ukládat jednotlivé verze dokumentů. Pro umístění úložiště (repository) se používají dva způsoby, které lze používat zároveň.

1. Verze dokumentů se ukládají do databáze. Výhodou takového řešení je zabezpečení repository, kdy o bezpečnost se stará databáze. Dále je zajištěna integrita a konzistence dat, transakční zpracování, zajištění při přístupu více klientů. Nevýhodou je závislost na databázi a její licenční politice, dále také značná různorodost databázových systémů a s tím spojená zvýšená režie na obecnost implementace nad databázi.
2. Verze dokumentů se ukládají do vlastní struktury na disku – výhodou je vlastní řešení a tedy nezávislost při implementaci a také kontrola nad celou repository. Nevýhodou je náročná implementace, kdy je třeba zajistit všechny funkce, které databáze již poskytují, tzn. zabezpečení, více přístupů do repository, zajištení integrity, apod.

Starší implementace podobných systémů využívají výhradně druhý způsob, jako příklad lze uvést CVS. Novější implementace, jako např. SubVersion, již správcům

dávají na výběr, zda-li použít klasické umístění anebo použít databázi. V případě SubVersion je použita databáze Berkeley DB, která je vydávána s licencí GNU General Public Licence. Takové řešení je závislé na použité databázi a nelze změnit databázový server. Obecné řešení, které by umožňovalo zvolení vlastní databáze by nutně vyžadovalo použití nejmenší společné množiny funkcí, které by fungovaly nad všemi databázemi. V důsledku toho by došlo také k rozšíření implementace o funkce, které již některé databáze implementují a některé ne.

1.4.3 Způsob ukládání do repository

S problémem, kde umístit repository je úzce spojená otázka, jakým způsobem jednotlivé verze ukládat. V praxi se používají dva možné způsoby:

1. Do repository se bude ukládat celá kopie souboru. Tato možnost předpokládá, že při požadavku pro uložení nové verze souboru, se do repository uloží celá kopie souboru. To znamená, že pro každou verzi budeme mít v repository přesnou kopii souboru v dané verzí. Výhodou tohoto řešení je jednoduchá implementace a rychlost přístupu k libovolné verzi v repository bez jakýkoliv další operací. Nevýhodou je náročnost na kapacitu úložného média a z tohoto důvodu se v praxi takřka nepoužívá.
2. Tento způsob stojí na myšlence, že by bylo výhodné si ukládat pouze změny, které nastaly a pomocí těchto změn si zpětně vytvořit libovolnou verzi. Zmíněný způsob vyžaduje přítomnost algoritmu, který dokáže porovnat dva soubory a vrátit seznam kroků, které je třeba vykonat, abychom z jednoho souboru dokázali vytvořit soubor druhý - samozřejmě bez přítomnosti druhého souboru. Výhodou takového použití jsou nižší náklady na kapacitu úložného média, nevýhodou je složitější implementace a pomalejší přístup ke starším verzím.

I přes zmíněné nevýhody se v praxi používá druhý způsob. Vzhledem k tomu, že se předpokládá minimální přístup k jiným verzím než aktuálním, není druhá nevýhoda zásadní. V případě požadavku na získání starší verze tedy není vyšší časová náročnost na takovou operaci kritickou překážkou.

Pro tento způsob, jak již bylo zmíněno, je zásadní existence algoritmu pro porovnávání dvou souborů (dále jen Diff algoritmus). Aplikace využívající Diff algoritmus velmi závisí na tom, jak je algoritmus implementován, jelikož jakékoliv chybné porovnání znamená zcela chybné uložení dané verze, což a v důsledku způsobí porušení všech předešlých verzí. Špatná, ale funkční implementace může na druhou stranu způsobit zbytečné nároky na prostor. Implementace Diff algoritmu musí zajistit, že pro každou dvojici souborů vždy vrátí seznam změn, pomocí kterých lze vytvořit druhý soubor. Pokud existuje taková dvojice souborů, pro které následující požadavek neplatí, je implementace nepoužitelná, z důvodů, které jsem uvedl výše. V neposlední řadě algoritmus musí být dostatečně rychlý, aby zbytečně neomezoval uživatele.

Jak již bylo zmíněno, Diff algoritmus vrací seznám změn popř. kroků, které je nutné vykonat pro přechod mezi jednotlivými verzemi. V angličtině se takový seznam označuje jako ‘Diff’, ‘Patch’ nebo ‘Script’ a v každé implementaci se líší.

Základ nicméně tvoří operace ‘přidat (Add)’, ‘odebrat (Delete)’ a také operace ‘nahradit (Replace)’. Operace Add značí, že do nové verze souboru přibyl řádek, operace Delete naopak značí, že v nové verzi souboru se daný řádek již nevyskytuje. Replace operace je v některých implementacích vynechána a nahrazena ekvivalentni posloupností operací Add a Delete, a značí, že řádek byl nahrazen.
Jako příklad konkrétní implementace Diff algoritmu lze uvést dodnes hojně používaný GNU Diff, který používá CVS.

Změny, které získává Diff algoritmus, musí být v repository uloženy ke každému souboru, aby bylo umožněno se kdykoliv vrátit k libovolné verzi. Záleží na návrhu již samotného systému, jakým způsobem se změny budou ukládat. Řešení, které používá např. CVS, ukládá tyto změny na konec každého souboru. Při získávání souboru z repository se tyto změny ořízno a uživatel je již nevidí. Dalším řešením je mít vyhrazený speciální soubor, kde se budou tyto změny ukládat a to jak centrálně (pro všechny soubory jeden soubor se změnami) anebo jednotlivě (tzn. pro každý soubor speciální soubor se změnami).

1.4.4 Způsob verzování dokumentů

Existují dva možné způsoby, jak přiřadit číslo verze dokumentům v rámci jednoho projektu.

1. Každý dokument bude mít vlastní číslo verze – tento způsob předpokládá, že pokud dojde ke změně v daném dokumentu, zvýší se (např. o 1) číslo verze oproti předchozímu číslu verze. Tento způsob je výhodný v tom, že každý soubor má vlastní verzování a v podstatě nezávislost na projektu. Nevýhodou je značný zmatek při návratu k libovolné verzi projektu.
2. Tento způsob předpokládá přidělování čísla verze dokumentům podle aktuální verze projektu. To znamená, že dokumenty uložené v jedné verzi budou mít společné číslo verze. Výhodou tohoto řešení je přehledný způsob verzování, jednoduchý přístup k libovolné verzi v minulosti, snadná orientace, ve které verzi došlo ke změně. Nevýhodou je centralizované řešení, závislost dokumentů na projektu a s tím zvýšená režie.

1.4.5 Práce více uživatelů nad jedním dokumentem

Systém pro správu verzí musí zajistit spolupráci uživatelů nad jedním dokumentem. Při návrhu lze vycházet ze dvou způsobů:

1. Lock-Modify-Unlock – tento způsob používá zamykání dokumentů pro uživatele, který chce soubor upravovat. Pokud je dokument uzamčen, žadný jiný uživatel, kromě uživatele, který ho uzamknul, nemůže soubor uzamknout nebo ho upravovat. Uvolnění souboru se provede operací „CheckIn“ nebo „UndoCheckOut“.
2. Copy-Modify-Merge – tento způsob neblokuje soubory výhradně pro jednoho uživatele. Pokud se klient pokusí uložit změny do repository, server provede operaci „merge“.

První způsob je vhodný pro situace, kdy se nepředpokládá, že dva uživatelé budou chtít upravovat jeden soubor nebo když tento konflikt nebude velmi častý. Pro větší vývojové týmy je tento způsob značně neefektivní, kdy nastává často, že jeden uživatel blokuje druhého. Výhodou je snadnější implementace.

Druhý způsob nepoužívá blokování souborů a řeší kolize, které při používání tohoto způsobu mohou nastat. Operace „merge“, kterou tento způsob využívá nemusí vždy uspět. Pokud neuspěje, např. proto, že dva klienti se pokusili změnit stejnou řádku v jednom souboru, server neprovede uložení a informuje klienta o vzniku konfliktu. Uživatel již sám musí provést nápravu. Nevýhodou je těžší implementace a existence algoritmu „merge“.

Oba dva způsoby lze kombinovat dohromady a v praxi se tak často děje. Někdy uživatel vyžaduje, aby mě výhradní právo na soubor a nechce, aby někdo jiný soubor upravoval, mezitím co ho upravuje. Potom se dostává na řadu způsob Lock-Modify-Unlock, kdy uživatel řekne systému, že ho chce uzamknout.

1.4.6 Zabezpečení

Systém, který nějakým způsobem ukládá data musí již od samého počátku myslet, jakým způsobem data zabezpečit tak, aby se k nim neoprávněný člověk nedostal. Zabezpečení systému lze rozdělit na dvě části: zabezpečení repository a šifrování komunikace.

V systémech pro správu verzí se zabezpečení repository řeší většinou nastavením přístupu k repository a obecně se předpokládá, že k počítači s repository nemá neoprávněný uživatel přístup. V případě implementace repository v databázi je ve většině databázových systémech toto již zajištěno samo, ale opět je zde předpoklad, že neoprávněný uživatel by neměl získat přístup k počítači, popř. získat přihlašovací údaje do databáze.

Přenos dat mezi klientem a repository se řeší šifrováním komunikace, tak aby se nikdo nemohl k utajovaným dokumentům dostat během komunikace mezi serverem a klientem. Toto je možno řešit buď přes protokol SSH, kdy dochází k výměně klíčů mezi serverem a klientem a následným šifrování a dešifrováním. Anebo také způsobem, kdy si šifrování a dešifrování zařídí systém sám již existujícími klíči, jak na straně serveru, tak i klienta. Výhodou použití SSH je značná obecnost a již známá implementace. Nevýhodou je nutnost výměny klíčů před posíláním dat, tedy jisté omezení rychlosti. Druhý způsob je implementačně náročnější, je třeba zajistit distribuci klíčů, ale již není nutná výměna klíčů před šifrováním. V obou dvou případech se jeví jako vhodné použít asymetrické šifrování, např. dnes nejpoužívanější RSA.

1.5 Implementace

Pro implementaci byla vybrána technologie .NET společnosti Microsoft a programovací jazyk C#. Jelikož zadání práce stanovilo, že systém bude určen výhradně pro platformu Windows, bylo na výběr několik jazyků, z nichž jako vhodný se jevil právě C#, z důvodu orientace na platformu Windows a také, že se jedná o moderní jazyk.

Samotná implementace se dá rozdělit na dvě části; na část serverovou a klientskou. Serverová část implementuje logiku správy dokumentů, práci s repository a zabezpečení a poskytuje prostředky pro práci se systémem klientským částem. Klientská část implementuje grafické rozhraní pro snadnější práci s repository.

1.5.1 Serverová část

Jak již bylo zmíněno, serverová část zastřešuje veškerou práci s repository. Také zajišťuje autentizaci a autorizaci uživatelů a bezpečnost komunikace. Serverová část poskytuje klientům rozhraní, které mohou využívat pro práci s repository. Serverová část může být od klientské části oddělena a umístěna na jiném počítači anebo se může nacházet na stejném počítači, pokud o správu dokumentů má zájem jednotlivec nebo velmi omezený počet lidí. Vzájemná komunikace mezi serverovou částí a klientskou částí probíhá pomocí technologie .NET Remoting na protokolu TCP.

Projekt serverové části byl rozdělen do logických částí, tak aby samotná logika systému byla oddělena od datových operací jako např. načítání a ukládání dokumentu. Toto rozdělení má výhodu v tom, že je velmi snadno rozšiřitelné a jednoduše lze změnit ukládání ze souborového systému do databáze a naopak. Logika systému je umístěna ve třídách označených jako BRO (Business rule object) a datové operace ve třídách DTO (Data-tier object). Metody, které se poskytují klientům, volají BRO objekty, ve kterých je celá výkonná logika systému a tyto objekty, v případě potřeby dat z repository, volají objekty DTO.

Repository je umístěno na souborovém systému, tzn. že k ukládání se nepoužívá žádná databáze. Toto řešení má výhodu v nezávislosti na databázi a nevýhodou je těžší implementace, kdy všechny funkce, které databáze mají již implementované, je nutné vytvořit. Díky návrhu projektu je ale velmi snadné umístit repository do databáze, kdy by se pouze změnily třídy DTO, které poskytují přístup k datům.

Pro repository musí být na disku vyhrazen jeden adresář. Cesta k tomuto adresáři musí být uvedena při instalaci a umístěna v konfiguračním souboru systému. Repository se skládá z projektů, větví, složek a souborů. Projekty, větve a složky jsou na disku reprezentovány pomocí adresářů a soubory jsou samostatné jednotky. Repository dále obsahuje interní soubory nutné pro správu dokumentů. Tyto soubory

obsahují informace, které objekty se v repository nacházejí, historii těchto objektů a také informace o verzích a uživatelích.

Projekt neboli modul je základní jednotkou repository. Pouze do projektu a jeho podsložek mohou být umístěny dokumenty. Různé projekty jsou od sebe logicky odděleny a každý projekt si sám spravuje zabezpečení, verzování, historii a dokumenty. Toto řešení má za důsledek jasné rozdělení repository a také samostatné verzování pro každý projekt.
Každá složka v repository obsahuje interní soubor, který říká, jaké objekty se v dané složce nacházejí. V tomto souboru jsou informace:

- Typ objektu (projekt, branch, složka, soubor)
- Číslo poslední verze souboru
- Datum a čas poslední změny
- Login uživatele, který změnu provedl
- Login uživatele, který provedl CheckOut souboru
- Cesta do adresáře na disku, kam se provedl CheckOut souboru
- Velikost souboru
- Kontrolní součet souboru

Všechny tyto informace hrají důležitou roli, jelikož velmi urychlují práci s obsahem repository, kdy k základním informacím není třeba načítat jiné soubory. Tento soubor lze samozřejmě velmi jednoduše rozšířit a přidat další informace, které by uživatel potřeboval vědět. Důležitou roli hrají dva poslední údaje, velikost a kontrolní součet. Tyto hodnoty výrazně urychlují práci se systémem, protože klient na základě těchto hodnot dokáže poznat, zda-li v pracovní kopii na disku došlo ke změně či nikoliv. A tuto informaci může klient využít při optimalizaci posílání dat na server, kdy např. u commit operace není třeba posílat na server soubory, ve kterých nedošlo ke změně. Jelikož soubory v repository jsou uloženy jako přesná kopie poslední verze, je velmi snadné získat velikost souboru. Kontrolní součet je reprezentován hash hodnotou, která se generuje algoritmem MD5. Při každé operaci, která vyžaduje obsah libovolného adresáře v repository, se pracuje s tímto interním souborem. Možným řešením by místo jednoho souboru pro každou složku bylo, mít jeden centrální soubor pro všechny složky a v něm potom vyhledávat informace. Toto řešení by ale bylo značně neefektivní při větším objemu dat, kdy všechny soubory a složky byly popsány v tomto souboru, což by mělo za následek neúměrnou velikost souboru, která by vedla k dlouhému načítání a ukládání.

Každý dokument v repository je reprezentován dvěma soubory. První soubor je přesná kopie poslední verze dokumentu a druhý soubor obsahuje informace o změnách provedených v jednotlivých verzích. Díky existenci přesné kopie poslední verze, jsou velmi rychlé nejpoužívanější operace Get, CheckOut. Díky druhému souboru se lze vrátit k libovolné verzi v historii a sem se zapisují kroky, které je nutné vykonat pro přechod z jedné verze do druhé. Tato implementace je výhodná v oddělení historie změn od kopie souboru, kdy zvlášť při velkém množství změn jsou operace Get, CheckOut stejně rychlé jako po vložení souboru do projektu. Soubor se změnami obsahuje následující informace:

- Číslo verze
- Typ změny (přidání, smazání, změna)
- Číslo a hodnota řádku z prvního souboru

- Číslo a hodnota řádku z druhého souboru

Každý projekt k výše uvedeným interním souborům obsahuje ještě jeden soubor s informacemi souvisejícími s projektem. Mezi tyto informace patří seznam verzí, kdo a kdy ji vytvořil, seznam souborů, které byly ve verzích změněny apod. Tyto informace jsou společné pro celý projekt a přistupuje se k nim, např. když je vyžádána historie projektu nebo když se provádí operace commit a tím se v důsledku vytváří nová verze. Tento soubor hraje důležitou roli, při vytváření nové verze, kdy se pouze v tomto souboru nalézá informace o čísle poslední verze. Díky tomuto souboru tedy lze zjistit číslo příští verze. Umístění tohoto souboru uvnitř projektu má výhodu v oddělení informací o projektu od dalších projektů a od repository samotné. V případě, že by budoucí vývojář chtěl implementovat přesun projektu např. do jiné složky, stačilo by mu vzít celou složku projektu a zkopírovat na jiné místo. Žádné další úpravy na projektu by nebyly třeba. Díky tomuto návrhu je tedy celá struktura repository značně flexibilní.

Jak již bylo zmíněno, serverová část zajišťuje zabezpečení jednotlivých projektů. Toto je zabezpečení spočívá v zamezení přístupu neautorizovaným uživatelům. Informace o tom, kdo může a kdo nemůže k projektu přistupovat, lze najít v interním souboru projektu. V tomto souboru je seznam uživatelů, kteří smí k danému projektu přistupovat, popř. jaké práva mají na daný projekt. Mezi tyto práva patří právo zápisu nebo právo pouze číst. Tato funkčnost je odchýlením od podobných produktů, kdy konkurenční systémy používají zabezpečení pouze v rámci celé repository. Tato funkce umožňuje správcům důslednější kontrolu nad přístupy k projektům a větší zabezpečení dokumentů.

Správa dokumentů funguje na principu Lock-Modify-Unlock. Tento způsob z moderních systémů používá pouze SourceSafe, u ostatních produktů to lze explicitně vyžádat. Tento způsob, jak již název říká, funguje na principu uzamkni a potom upravuj. Znamená to tedy, že pokud uživatel chce upravovat nějaký dokument, musí nejprve provést operaci check-out a potom teprve může upravovat soubor. Až uživatel skončí práci na dokumentu, provede operaci Commit a tím se změny uloží do repository. Tento způsob je vhodný, pokud se systémem pracuje menší množství lidí a navzájem si neblokují soubory . Pro velké vývojové týmy je ovšem tento způsob nevyhovující, kdy více uživatelů nemůže pracovat nad stejným dokumentem.

Když uživatel provádí operaci commit, serverová část musí nejprve zjistit, jaké číslo se nové verzi přiřadí. Toto číslo, jak již bylo řečeno, se zjistí z interního souboru projektu. Každá operace commit dostane unikátní číslo verze. Toto číslo se přiřadí všem dokumentům, nad kterými se provádí commit. V případě, že je commit prováděn pouze nad jedním dokumentem, tak se číslo verze nastaví pouze tomuto dokumentu a do historie se zapíšou informace o verzi. Výhodou tohoto způsobu, na rozdíl od odděleného verzování, je lepší orientace ve verzích. Uživatel, který se chce vrátit k libovolné verzi v historii stačí znát číslo verze, a dostane aktuální stav projektu k dané verzi.

Komunikace mezi serverovou částí je zajištěna technologií .NET Remoting přes protokol TCP. Serverová část deklaruje na určitém portu kanál, na který se klienti mohou připojit a vzdáleně volat metody ze serverové části. Tato implementace je značně transparentní a není třeba definovat vlastní způsob volání metod

a implementovat vlastní komunikaci. Pro klienta je díky tomuto řešení volání metod ze serverové části velmi jednoduché a v důsledku ani nepozná, že vlastně komunikuje s částí, která se nalézá na jiném počítači.

1.5.2 Klientská část

Klientská část v systému VCS je příkladem jak implementovat uživatelské rozhraní pro danou serverovou část. Klientská část přes protokol TCP komunikuje se serverovou částí a volá metody serverové části a tímto způsobem pracuje s repository.

Stejně jako serverová část je i projekt klientské části rozdělen do několika prvků. Objekty grafického rozhraní volají metody logiky aplikace, která je implementována v objektech BRO (Business rule object). Tyto objekty implementují výkonnou část aplikace a také volají vzdálené metody na serverové části. Data získaná ze serverové části, popř. generovaná na klientovi, se ukládají do objektů BDO (Business data object), jež jsou v jazyce C# definovány jako datasety. Toto rozdělení umožňuje jednoduchou orientaci v kódu a také oddělení logiky aplikace od grafického rozhraní.

Klientská část je napsána způsobem, aby se co nejméně volaly metody na serverové části a tak docházelo co nejméně ke komunikaci po síti. K optimalizaci se také využívají informace již zaslané ze serveru, např. velikost souboru v repository. Této informace se využívá při operacích „Update" resp. "Commit", kdy se na klientovi nejprve kontroluje, zda-li se soubor změnil a až poté se vyžádá soubor z repository, resp. pošle soubor na server. Tyto optimalizace výrazně redukují drahou síťovou komunikaci a tím zrychlují práci se systémem.

Jak již bylo zmíněno, serverová část komunikuje s klientskou pomocí technologie .NET Remoting přes protokol TCP. Při přihlášení do programu se kromě loginu a hesla zadává také adresa serveru. Tento server se na zadaném portu kontaktuje a pokusí se o přihlášení. Pokud zadané údaje byly správné, klient zaregistruje kanál serveru, aktivuje vzdálenou třídu a od této chvíle může vzdáleně volat metody na serverové části. Tato funkčnost je zapouzdřena ve speciální třídě BRO, přes kterou se provádí veškeré volání metod. Tento způsob je velmi flexibilní a pro vývojáře přímý, jelikož nepozná, že komunikuje se serverem. Autentizace je oddělená od implementace repository, takže lze velmi jednoduše implementovat vlastní autentizační proces a začlenit ho do systému pomocí konfiguračního souboru. Implementovaný způsob využívá administrační prostředí (VCS.Admin) pro nastavování práv a správu uživatelů.

1.5.3 Administrační prostředí

Administrační prostředí slouží ke správě uživatelů, kteří mohou k serverové části přistupovat. Toto prostředí musí být nasazeno na stejném počítači jako serverová část. Uživatelským účtům lze nastavit dobu vypršení platnosti účtu, kdy po vypršení daný uživatel, již nemá možnost se k serverové části připojit.

Prostředí ukládá informace o uživatelských účtech do XML souboru. Hesla jsou uloženy hash hodnotou a k heslům jsou navíc připojovány náhodné řetězce (salt), které způsobí, že dva uživatelé se stejným heslem mají různé hash hodnoty. Tato informace je uchovávána spolu s dalšími informacemi o účtu.

1.5.4 Implementace Diff algoritmu

Pro vytvoření vlastní implementace algoritmu Diff bylo několik důvodů. Jedním z nejdůležitějších argumentů byla neexistence vhodného algoritmu v jazyce C#. Cílem mimo jiné bylo vytvořit jednotnou aplikaci, napsanou v jednom programovacím jazyce, která by bylo snadno rozšiřitelná a spravovatelná. Použití jednoho jazyka výrazně zjednodušuje práci při dalším vývoji systému i za cenu mnohem složitější implementace. Dalším důvodem byla možnost si algoritmus i jeho výstup upravit dle svých představ a nebýt závislý na cizí implementaci. A v neposlední řadě také touha poznat, jak se podobný algoritmus vytváří a co vše to obnáší.

Základem implementace Diff algoritmu, je nalezení nejdelší společné sekvence (LMS – The longest matching sequence). LMS je seznam dvojic indexů řádků (v původním a novém souboru), které se shodují. Nalezení změn mezi původním a novým souborem je poté pouze zpracování tohoto seznamu a vyplnění prázdných polí. Tato prázdná pole jsou právě změny, které je nutné vykonat pro přechod z původní verze do nové. Proto implementace Diff algoritmu je v podstatě otázkou nalezení, co možná nejdelší, posloupnosti řádků, které se shodují.

Pro nalezení LMS je použit rekurzivní způsob a několik optimalizací, které výrazně zredukují počet nutných porovnání. Rekurzivní volání zajistí, že se jednotlivé části, které se porovnávají, budou rozpadat na menší a menší části, stejně jako v případě algoritmu QuickSort.

Pozn.: Pro popis implementace je použit pseudo-kód, který je snadno čitelný i bez znalosti programovacího jazyka. Kompletní implementaci v jazyku C# najdete ve zdrojových kódech aplikace na přiloženém CD.

```
Pro každý řádek v Destination souboru proveď
Begin
        #if  (nemůžu najít delší sekvenci)
           Konec;
        End if;
        Pro daný řádek v Destination najdi LMS v Source;
        if  (existuje sekvence)
          Pokud tato sekvence je nejdelší, tak ji ulož.
          #Přeskoč všechny řádky v Destination, které jsou zahrnuty do této sekvence.
        End if;
end;

if  (máme LMS v daném rozsahu řádků v Destination souboru )
Begin
        Ulož tuto LMS do konečného výsledku;
        If  (existují položky nad LMS (jak v Source tak Destination))
                Znovu zavolej tuto metodu – rekurze
        End if
        If  (existují položky pod LMS (jak v Source tak Destination))
                Znovu zavolej tuto metodu – rekurze
        End if
End if;
```

Řádky začínající znakem '#' značí operace, které slouží jako optimalizace. Tyto operace značně redukují počet nutných porovnání a výrazně zrychlují činnost algoritmu. Hledání LMS v Source probíhá stejným způsobem jako první část výše uvedeného algoritmu.

První optimalizace spočívá v přeskočení výpočtu, kdy již nemůžeme teoreticky najít delší sekvenci.

```
maxPossibleDestLength = (destEnd – destIndex) + 1;
if (maxPossibleDestLength <= curBestLength)
  přeskoč;
```

Druhá optimalizace spočívá v přeskočení položek, které jsou zahrnuty v nejdelší, již nalezené, sekvenci. Tato optimalizace výrazně zrychluje algoritmus.

Při dvou naprosto odlišných souborech dochází k porovnání každého řádku z prvního souboru s každým řádkem z druhého souboru. Ovšem při menším počtu změn algoritmus dokáže najít LMS velmi rychle. Algoritmus, zde nastíněný, pracuje spolehlivě a vždy vrátí seznam kroků, pomocí kterých se lze vrátit k původnímu souboru. Bylo by ovšem vhodné algoritmus otestovat při větším používání při větším počtu uživatelů, aby šlo skutečně říci, že tento algoritmus je vhodný pro běžné používání.

1.5.5 Bezpečnost

Zabezpečení komunikace je v systému řešena pomocí šifrování algoritmem RSA. Toto šifrování se používá při přihlašování, aby nedošlo k vyzrazení přihlašovacích informací. Server i klient dostanou při přihlášení své klíče, které se v procesu používají. Tyto klíče lze změnit v administračním prostředí a pro maximální bezpečnost je to i doporučováno. Klientská část pomocí veřejného klíče zašifruje přihlašovací informace a pošle je na server. Server je pomocí privátního klíče dešifruje a ověří, zda-li uživatel má právo k přístupu či nikoliv. Pokud ano, odešle informace o přihlášeném uživateli zpět klientovi a tyto data podepíše privátním klíčem. Klientská část přijme data, veřejným klíčem ověří správnost podpisu a použije je dále k přístupu do systému. Díky podpisu serverové části má klient zajištěno, že se nikdo jiný nevydává za serverovou část a že opravdu komunikuje se správným serverem.

Pro zabezpečení repository se předpokládá nastavení takových práv, aby k datům měl přístup pouze proces serverové části systému. Systém je navržen tak, že v případě potřeby, lze na přání uživatele či zákazníka, implementovat šifrování veškerých dat v repository.

1.6 Funkce systému

Jak již bylo zmíněno v úvodu, jedná se o implementaci systému na správu verzí. Uživatel nebo tým lidí může v takovém systému bezpečně uchovávat jednotlivé verze dokumentů, sledovat jejich změny v čase, porovnávat mezi jednotlivými verzemi a tím zefektivnit produktivitu práce nad jedním nebo více dokumenty.

Systém VCS se skládá ze tří části:

- VCS.Server – jedná se o aplikaci, která je umístěna na počítači, kde bude umístěna Repository
- VCS.Admin – aplikace, která poskytuje administrační rozhraní pro systém (je umístěna na stejném počítači jako VCS.Server
- VCS.Klient – uživatelská (klientská) část systému, slouží pro běžnou práci se systémem

Jedná se o síťovou aplikaci, která předpokládá, že počítač, na kterém běží uživatelská část, má spojení na počítač se serverovou částí. Toto spojení může být realizováno jak přes síť Internet tak i v rámci sítě LAN. Všechny tři části lze mít umístěné i na jednom počítači.

Zde je výčet funkcí, které systém VCS nabízí

- Snadná přenositelnost serveru
- Snadná instalace serverové části i bez instalačního průvodce
- Žádná instalace klientské části
- Možnost připojení libovolného počtu uživatelů
- Správa uživatelů včetně omezení platnosti loginu na určitou dobu
- Umístění repository na lokálním počítači stejně jako na vzdáleném

- Add - Přidání libovolného počtu projektů, složek nebo dokumentů do repository
- Možnost uložení libovolné verze z repository na disk
- Get latest version – navíc možnost přidat informaci ke každému řádku, ze které verze daný řádek pochází, popř. kdo ho vytvořil
- CheckOut
- UndoCheckOut – odblokování souboru s možností jejich smazání nebo ponechání na disku
- Commit
- Přejmenování projektu, složky, dokumentů.
- Smazání dokumentu a jeho obnovení, trvalé smazání projektu, složky, větve
- Prohlížení historie projektu, složky a dokumentů a návrat k libovolné verzi
- Compare – porovnávání mezi souborem na disku a v repository; prohlížení změn mezi libovolnými verzemi dokumentu
- Branch – vytvoření libovolného počtu větví v projektu, včetně zachování kompletní historie
- Statistika – zobrazení počtu změn za určité období apod.
- Možnost vytvořit hierarchickou strukturu repository
- Šifrovaná komunikace mezi klientskou a serverovou části pomocí RSA šifrování
- „Čisté“ pracovní kopie bez interních souborů systému

Systém VCS implementuje všechny základní funkce, které se od takového systému očekávají a navíc dává dohromady implementace podobných systémů pomocí funkcí, které jsou v jedné implementaci, ale v druhé již nikoliv. Z výčtu lze vidět, že již při návrhu byla snaha o co největší flexibilitu systému bez příliš velkého omezení uživatelů.

1.7 Srovnání s jinými známými implementacemi

Na trhu existuje mnoho systémů pro správu verzí. Mezi nejpoužívanější implementace lze zařadit:

- CVS (CVSNT)
- SubVersion
- Microsoft Visual SourceSafe

CVS i SubVersion jsou vydávány pod licencí GNU General Public License a jsou tedy volně k dispozici včetně zdrojových kódů. Systém SourceSafe je produktem společnosti Microsoft a tedy se řídí její licenční politikou.

1.7.1 CVS (Concurrent versions system)

Průkopníkem ve správě verzí byl systém CVS (Concurrent versions system). Jedná se o dosud nejrozšířenější a nejpoužívanější systém pro správu verzi.

Systém CVS používá klient-server architekturu, kde server ukládá verze dokumentů a klienti se připojují k serveru za účelem získání pracovní kopie a jejím pozdějším uložením zpět do repository. Typicky klient a server spolu komunikují přes síť LAN nebo přes Internet, ale také je možné server i klient provozovat lokálně na stejném počítači, pokud je vyžadována správa verzí pouze pro jednotlivce. Serverová část běží pouze na platformě Unix. Klienti ale již mohou běžet nad libovolnou platformou.

Několik klientů může upravovat kopie projektu současně. Systém používá způsob Copy-Modify-Merge. Pokud se klient pokusí uložit změny do repository, server se pokusí provést „merge“. Pokud toto neuspěje, např. proto, že dva klienti se pokusili změnit stejnou řádku v jednom souboru, server neprovede uložení a informuje klienta o vzniku konfliktu. Uživatel již sám musí provést nápravu. Pokud uložení uspěje, číslo verzí se u všech změněných souborů zvýší a CVS server zapíše informaci o popisu verze, času uložení a jména uživatele do logu.

Klienti také mohou porovnávat různé verze souboru, požádat o kompletní historii změn anebo si stáhnout na disk libovolnou verzi z historie k danému datu nebo čísla verze. Klienti také mohou provést „update“, který aktualizuje pracovní kopie na disku.

CVS také může spravovat různé větve (branches) projektu. Např. právě nasazená verze se může přesunout do jedné větve pro opravu chyb a v druhé větvi se může pokračovat na vývoji další verze projektu.

CVS používá „delta compression“ pro efektivní ukládání různých verzí stejných souborů.

Nevýhodou implementace je hlavně omezení uživatele, kdy nelze v repository přejmenovat soubor a také omezená podpora pro soubory v Unicode. Dále také omezená podpora pro soubory s názvem, který je v jiném kódování než

ASCII. Některá omezení odstranila implementace CVSNT, která je na CVS založena. Jedná se především podporu pro více platforem, audit, podpora kódování v Unicode.

1.7.2 SubVersion

Klíčový vývojáři, kteří se podíleli na vývoji CVS přešli na projekt SubVersion, který není zatížen historií jako CVS a byl od počátku vyvíjen jako plnohodnotná náhrada CVS. SubVersion vyšel na začátku roku 2004 a odstraňoval omezení, která implementace CVS obsahovala.
Mezi vylepšení oproti CVS patří:

- Atomický commit. Přerušení během commitu nezpůsobí nekonzistenci nebo porušení dat.
- Možnost přejmenování, kopírování, přesouvání souborů včetně zachování historie
- Podpora pro binární soubory s efektivním ukládáním změn mezi binárními soubory
- Adresáře v repository mají vlastní verze a historii
- Optimalizovaný přístup k repository. Snížena míra nutné komunikace mezi serverem a klientem
- Možnost umístění repository do databáze (BerkeleyDB)

Mezi nejpoužívanější klienty pro SubVersion lze zařadit:

- AnkhSVN – rozšíření pro Microsoft Visual Studio .NET
- Insurrection – webové rozhraní pro Subversion
- TortoiseSVN – klient pro Windows, rozšíření do shellu Windows
- Subclipse – rozšíření do vývojového prostředí Eclipse

1.7.3 Microsoft Visual SourceSafe 6.0

SourceSafe stejně jako předešlé dva produkty je systém pro správu verzí. Umožňuje do repository uložit libovolný typ dokumentu, nicméně předešlé verze byly nestabilní při velkém množství netextových dokumentů. SourceSafe se orientuje především na správu textových dokumentů. SourceSafe je určen především pro platformu Windows, ale existují i verze pro Unix a Macintosh, ale pro tyto systémy neexistuje podpora ze strany společnosti Microsoft.
SourceSafe používá techniku Lock-Modify-Unlock, která znamená to, že pokud chce uživatel soubor upravovat, musí si ho nejprve uzamknout a dokud ho neodemkne, tak žádný jiný uživatel ho nemůže upravovat.

Mezi výhody tohoto produktu patří poměrně nízká cena systému (ve srovnání s komerčními systémy) a velká uživatelská základna. Výhodou je také výborné začlenění do vývojového prostředí Microsoft Visual Studio, jednoduchost a také, že je začleněna jako část MSDN Subscriptions. Je určen především pro menší a středně velké firmy, kde nedochází tak často k jevu, že více uživatelů upravuje ten samý soubor.

Mezi nevýhody patří závislost na prostředí, ve kterém běží. Chybí podpora atomického uložení, kdy v případě havárie počítače, dojde k porušení dat. Mezi

hlavní nevýhody patří Lock-Modify-Unlock implementace, kdy především ve větších týmech dochází ke značnému omezení uživatelů.

1.7.4 Srovnání

Úkolem této práce bylo vytvořit systém, který by nebyl pouhou kopií těchto systémů. Do systému byly implementovány funkce, které nejsou zahrnuty v žádné z těchto nejrozšířenějších implementacích a také zahrnuty funkce, které v některé implementaci jsou a v jiné chybí. Nejblíže má systém k produktu SourceSafe společnosti Microsoft, který má podobné předpoklady jako zadání tohoto systému (jednoduchost, přehlednost, zaměření na menší a středně velké týmy), ale v mnohém ho rozšiřuje. Stejně jako klientská část systému SourceSafe, i klientská část tohoto systému poskytuje přímý pohled na repository.

Funkce systému VCS	CVS	SubVersion	Microsoft SourceSafe
Rename		x	x
Delete	x	x	x
Commit	x	x	x
Checkout	x	x	x
Get	x	x	x
History	x	x	x
Update	x	x	x
Branch	x	x	x
Get with Info		x	
Hieararchie			x
Statistika		x	
Práva na projekt			
Xcopy instalace			

Tabulka 1.2: Srovnání s jinými produkty

V tabulce 1.2 jsou zahrnuty pouze vybrané funkce ze systému VCS, který byl předmětem této bakalářské práce. Uvedené konkurenční systémy mají další funkce, které zde nejsou uvedeny a které tento systém VCS neimplementuje. Úkolem tohoto systému nebylo udělat kopii existujícího řešení, ale navrhnout systém, který by se odlišoval, a který by implementoval funkce, které v žádném konkurenčním prostředí nejsou.
V tabulce 1.2 nejsou zahrnuty výhody, které vycházejí z návrhu a jsou popsány v kapitole 1.5 – Implementace. Nespornou výhodou tohoto systému je nový návrh nezatížený žádnou historií.

2. Uživatelská příručka

2.1 Instalace serverové části

2.1.1 Systémové požadavky

Pro správnou funkčnost systému je třeba na počítači, na kterém bude umístěna serverová část, mít nainstalované následující součásti

- Operační systém Microsoft Windows 98/2000/2003/XP
- .NET Framework 1.1.

V případě, že bude serverová část umístěná na jiném počítači než klientská

- Připojení do vnitřní sítě (popř. sítě Internet), kde se nachází klientská část

2.1.2 Instalace bez instalačního průvodce

1. Zkopírovat obsah instalačního archivu do libovolného adresáře na disku
2. Na disku vytvořit adresář, který bude sloužit jako repository (úložiště)
3. V adresáři z bodu 1 upravit soubor „vcs_settings.xml“

Klíč	**Hodnota**
vcs_root	Umístění repository – adresář z bodu 2
privateKeyLocation	Umístění privátního klíče – defaultní klíč se nachází v adresáři z bodu 1
vcsAdminLocation	Cesta k administračnímu prostředí

Tabulka 2.1: Hodnoty konfiguračního souboru

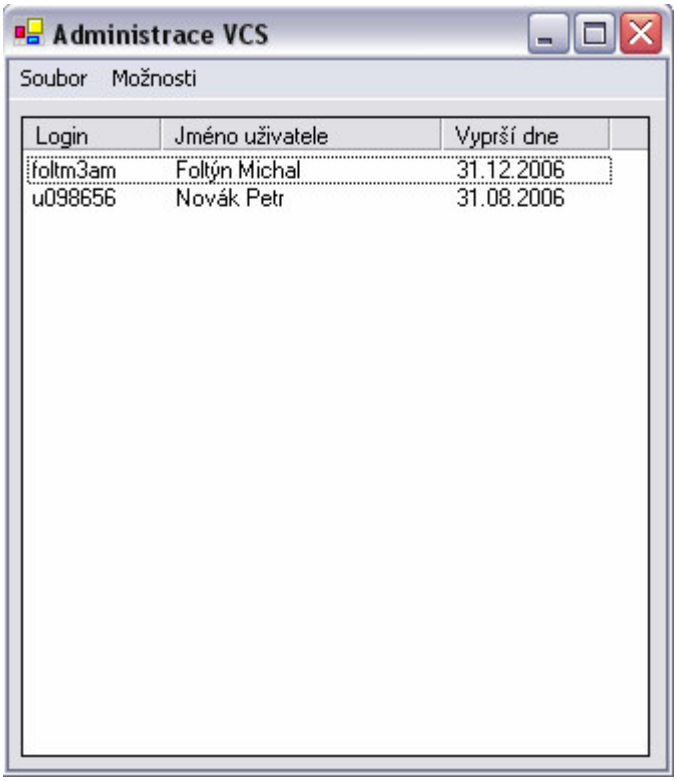
Tyto hodnoty je důležité správně nastavit, jinak systém nebude fungovat správně. Vlastní privátní klíč lze vygenerovat v administračním prostředí. S privátním klíčem se vygeneruje i veřejný klíč, který je nutné dát všem klientským částem.

4. Upravit soubor „app.config“, který se nálezá v adresáři z bodu 1; zde v případě potřeby změnit číslo portu, přes který se budou klienti připojovat k serverové části
5. Pomocí administračního prostředí přidat uživatele, kteří budou moci k systému přistupovat.

2.2 Administrační prostředí (VCS.Admin)

2.2.1 Popis

Administrační prostředí systému slouží pro správu uživatelů, kteří mohou k repository přistupovat. Pouze uživatelé zde uvedení mohou se systémem pracovat. Administrační prostředí, jak již sám název napovídá, by měl sloužit pouze administrátorům, a tedy by k němu neměli mít přístup neoprávnění uživatelé.



Obr. 2.1: Administrační prostředí

2.2.2 Přidání uživatele

Uživatele přidáte následujícím způsobem:

1. V hlavním menu kliknete na Možnosti -> Přidat uživatele
2. Otevře se dialogové okno (obr. 2.2)
3. V dialogovém okně vyplníte všechny potřebné informace, včetně data vypršení, do kdy je účet platný a kliknete na tlačítko Uložit
4. Pokud vše proběhlo v pořádku, v hlavním okně administračního rozhraní uvidíte přidaného uživatele v seznamu uživatelů

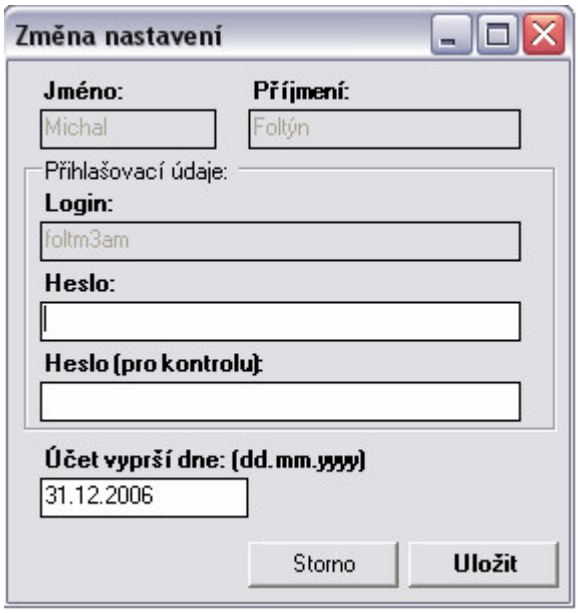
Obr. 2.2: Přidání uživatele

2.2.3 Odebrání uživatele

1. V seznamu uživatelů v hlavním okně vyberete uživatele, kterého chcete odebrat
2. V hlavním menu kliknete na „Možnosti -> Odebrat uživatele“
3. Po potvrzení daný uživatel zmizí ze seznamu uživatelů a tento uživatel se již nemůže přihlásit do repository

2.2.4 Změna nastavení uživatele

1. V seznamu uživatelů v hlavním okně vyberete uživatele, u kterého chcete změnit nastavení
2. V hlavním menu kliknete na Možnosti -> Změnit nastavení
3. Otevře se stejné dialogové okno jako při přidávání uživatele (obr. 2.3)
4. Nyní můžete změnit heslo a také datum vypršení účtu
5. Pro potvrzení klikněte na tlačítko Uložit
6. V případě, že jste změnili datum vypršení, změna se projeví v seznamu v hlavním okně.



Obr. 2.3: Změna nastavení uživatele

2.2.5 Generování vlastních klíčů pro šifrování

Klíče slouží pro šifrování komunikace mezi serverovou částí a klientskou, aby nedošlo k vyzrazení přihlašovacích údajů a vlastních dat. Součástí instalace je i výchozí privátní a veřejný klíč. Tyto je doporučeno nepoužívat a vygenerovat si vlastní pomocí následujícího postupu.

1. V hlavním menu kliknete na „Možnosti -> Vygenerovat klíče…“
2. Postupně se otevřou dvě dialogové okna pro zadání cesty, kam klíče uložit
3. Privátní klíč umístěte k serverové části, veřejný předejte všem klientům, kteří se budou chtít k serverové části připojit

2.3 Spuštění serverové části

Serverovou část spustíte souborem „vcs.server.exe“, který je umístěn v adresáři, kam jste systém nainstalovali, popř. umístili ručně. Od chvíle spuštění serverové části, se mohou klienti připojovat a pracovat se systémem. Klienti se mohou připojovat na adresu daného počítače a na port, který jste uvedli při instalaci systému.

2.4 Instalace klientské části

2.4.1 Systémové požadavky

Pro správnou funkčnost klientské části je třeba mít na počítači nainstalované následující součásti

- Operační systém Microsoft Windows 98/2000/2003/XP
- .NET Framework 1.1.
- Na stejném nebo jiném počítači mít správně nastavenou a spuštěnou serverovou část a možnost se na daný počítač připojit

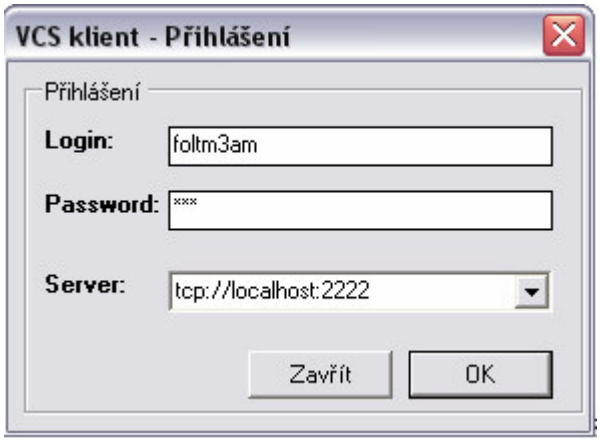
2.4.2 Spuštění klientské části

Instalace klientské části se nemusí provádět. Klientskou část lze spustit bez jakékoliv instalace, pouze spuštěním souboru „vcs.exe“. Klientská část je tedy přenosná a nezávislá na prostředí, ve kterém běží a lze ji přenášet z počítače na počítač.

2.5 Popis klientské části

2.5.1 Přihlášení k serverové části

Po spuštění klientské části se zobrazí přihlašovací dialog (obr. 2.4).



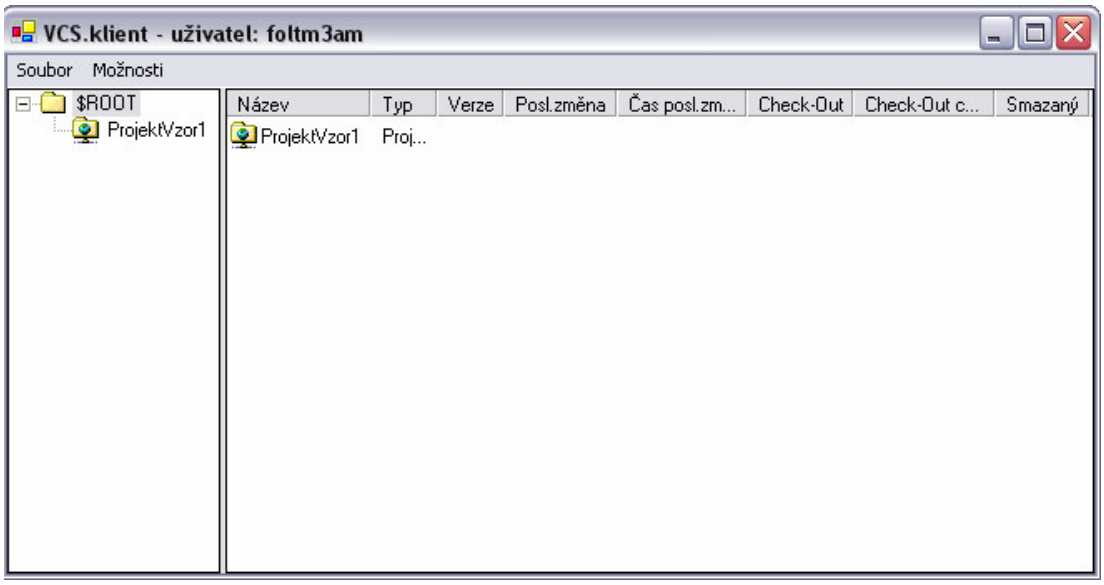
Obr. 2.4: Přihlašovací dialog

Do příslušných poliček vyplňte vaše přihlašovací jméno a heslo.
Adresu serveru zadávejte ve tvaru: „tcp://server.vcs.cz:XXXX“.

V případě úspěšného přihlášení se vám zobrazí hlavní okno klientské části s pohledem na repository (obr. 2.5).

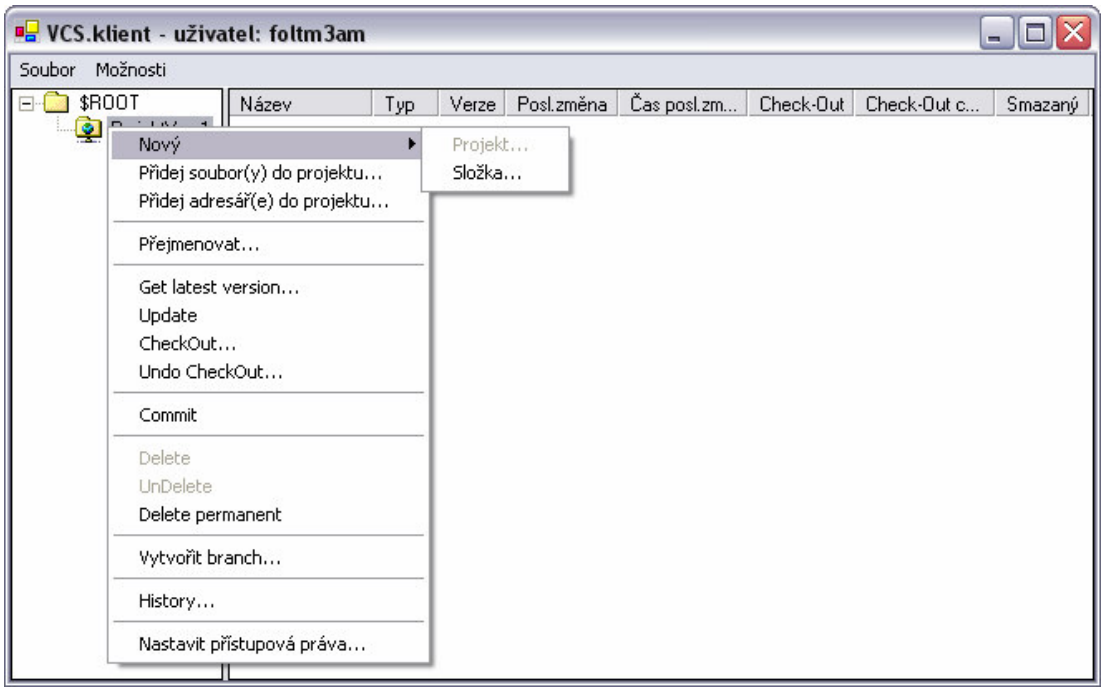
2.5.2 Hlavní okno klientské části

Hlavní okno zobrazuje strukturu a obsah repository na serveru, na který jste se přihlásili. Tedy v hlavním okně můžete vidět aktuální stav repository, projekty a dokumenty uloženém vámi nebo jiným uživatelem.



Obr. 2.5: Hlavní okno klientské části

V levé části můžete vidět stromovou strukturu repository a v pravé části okna obsah aktuálně označené složky.
Pravá část dále obsahuje informace o dokumentech jako jeho název, číslo poslední verze, kdo a kdy provedl poslední změnu na dokumentu, zda-li je soubor uzamčen a na jakém místě se nachází.
Funkce, které můžete používat najdete buď v hlavním menu a nebo v seznamu po kliknutí pravého tlačítka myši nad složkou či dokumentem (obr. 2.6).



Obr. 2.6: Vyskakovací okno při kliknutí pravým tlačítkem na projekt

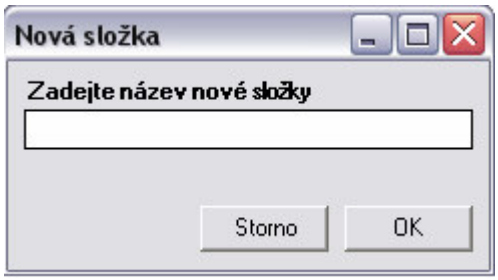
Aplikaci ukončíte kliknutím v menu na Soubor -> Konec.

2.5.3 Přidání projektu do repository

V repository si můžete vytvořit jednoduchou strukturu, kdy na jedné úrovní budete mít všechny projekty anebo hierarchickou strukturu, kdy si můžete jednotlivé projekty rozdělit do nějaké logické struktury. Tato hierarchie se vytváří pomocí složek.

Vytvoření složky

1. Kliknete pravým tlačítkem na objekt v repository, kde chcete novou složku vytvořit
2. Ze seznamu funkcí vyberete a kliknete na „Nový -> Složka…“
3. Zobrazí se dialog pro zadání jména nové složky



Obr. 2.7: Vytvoření složky

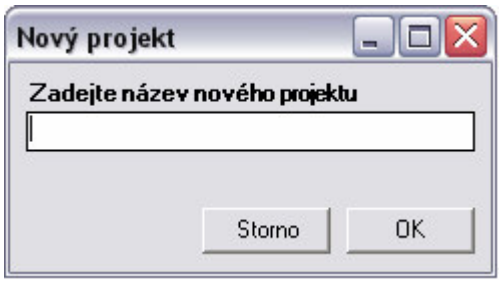
4. Zadejte název a klikněte na tlačítko OK
5. V případě úspěšného přidání se složka zobrazí v hlavním okně v repository

Takhle můžete pomocí složek vytvořit libovolnou strukturu repository a např. rozlišit zaměření projektů apod.

Dokumenty lze v systému umístit pouze do projektů. Složky mimo projekty slouží pouze pro vytvoření hierarchie. Vytváření projektů probíhá podobným způsobem jako u složek.

Vytvoření projektu

1. Kliknete pravým tlačítkem na objekt v repository, kde chcete nový projekt umístit
2. Ze seznamu funkcí vyberete a kliknete na „Nový -> Projekt…“
3. Zobrazí se stejný dialog jako při přidávání projektu pro zadání jména nového projektu (obr. 2.9)



Obr. 2.8: Vytvoření projektu

4. Zadejte název a klikněte na tlačítko OK
5. V případě úspěšného přidání se projekt zobrazí v hlavním okně v repository

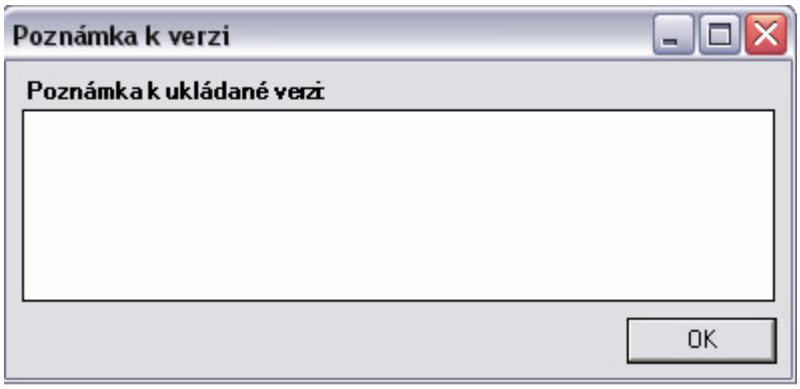
Nyní můžete do repository přidávat dokumenty a tyto dokumenty spravovat.

2.5.4 Přidání souborů a adresářů do projektu v repository

Přidání souborů a adresářů lze provádět pouze v projektu nebo jeho podsložce. Systém neumožňuje přidávat dokumenty mimo projekt.

Přidání souborů probíhá následujícím způsobem:

1. Vyberte a klikněte pravým tlačítkem myší na projekt nebo libovolnou složku v projektu, do které chcete soubor(y) přidat
2. V seznamu vyberte „Přidej soubor(y) do projektu…”
3. Otevře se dialogové okno pro zadání souborů z disku, které chcete do projektu umístit
4. Po úspěšném potvrzení se otevře dialogové okno pro zadání poznámky k nové verzi (obr.2.9)
5. Po vložení poznámky se provede uložení souborů do projektu a v hlavním okně můžete vidět nově přidané soubory



Obr. 2.9: Vložení poznámky k verzi

Poznámka k verzi umožňuje uživateli vložit k dané verzi nějakou informaci, která charakterizuje či jinak popisuje danou verzi.

Přidání adresářů z disku a jeho obsahu do projektu

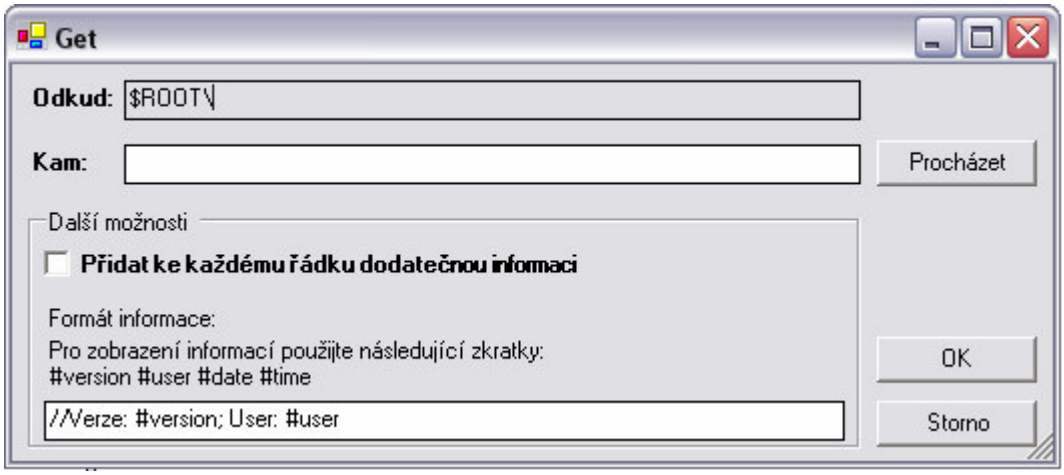
1. Vyberte a klikněte pravým tlačítkem myší na projekt nebo libovolnou složku v projektu, do které chcete adresář(e) a jeho veškerý obsah přidat
2. V seznamu vyberte „Přidej adresáře(y) do projektu…"
3. Otevře se dialogové okno pro zadání adresáře z disku, který chcete do projektu umístit
4. Po úspěšném potvrzení se otevře dialogové okno pro zadání poznámky k nové verzi (obr. 2.9)
5. Po vložení poznámky se provede uložení adresáře a jeho obsahu do projektu a v hlavním okně můžete vidět nově přidaný adresář a jeho obsah

2.5.5 Přejmenování objektů v repository

1. Vyberte v hlavním okně objekt, který chcete přejmenovat
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Přejmenovat"
3. Zobrazí se dialogové okno pro zadání nového názvu, podobné jako při vytváření složky nebo projektu
4. Po úspěšném potvrzení názvu se daný objekt v repository přejmenuje a v hlavním okně můžete vidět změnu

2.5.6 Získání poslední verze z repository (Get latest version)

1. Vyberte v hlavním okně objekt, jehož aktuální verzi chcete uložit na disk
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Get latest version"
3. Otevře se dialogové okno pro podrobnější nastavení (obr. 2.10)
4. Tlačítkem „Procházet" vyberte cestu, kam chcete daný objekt uložit
5. V případě, že chcete ke každému řádku v souboru přidat dodatečnou informaci, zaškrtněte příslušné pole
6. Po kliknutí na tlačítko OK se na disk uloží poslední verze vybraného objektu



Obr. 2.10: Get latest version

Ke každému řádku v souboru si můžete nechat vygenerovat informace o tom, z jaké verze daný řádek pochází, kdo a kdy ho do souboru vložil. V příslušném poli můžete zadat vlastní formátování informace. Povolené proměnné jsou v tabulce 2.2.

Proměnná	Popis
#version	Číslo verze, ze které daný řádek pochází
#user	Jméno uživatele, který řádek do souboru vložil
#date	Datum vložení řádku
#time	Čas vložení řádku

Tabulka 2.2: Formátování výstupu

2.5.7 Získání poslední verze k úpravě (CheckOut)

Pokud chcete dokument z repository upravovat, musíte na něm provést operaci „CheckOut“. Tato operace stejně jako „Get“ stáhne na disk poslední verzi z repository a navíc soubory označí jako vyhrazené pro vaši potřebu. Zpětné uložení změn do repository se provádí operací „Commit“.

1. Vyberte v hlavním okně objekt, který chcete uložit na disk a upravovat
2. Klikněte pravým tlačítkem myši na tento objekt a z vyskakovacího okna vyberte „CheckOut“
3. Zobrazí se dialog pro zadání cesty, kam se má pracovní kopie z repository uložit.
4. Po potvrzení cesty se soubory (složky) uloží na disk a v repository se označí jako „CheckedOut“
5. Od této chvíle na souboru můžete pracovat pouze vy a dokud soubor neuvolníte nikdo jiný nemůže soubor upravovat.
6. V hlavním okně můžete vidět, že soubory, na kterých jste provedl(a) „CheckOut“ se označily jako uzamčené vaší osobou.

Uvolnění souboru provedete buď uložením změn pomocí operace „Commit“ anebo operací „UndoCheckout“, která pouze soubor uvolní, ale již neuloží změny.

2.5.8 Uvolnění souborů (UndoCheckOut)

Tuto operaci lze provádět pouze nad soubory, které jsou uzamčené vaší osobou.

1. Vyberte v hlavním okně objekt, který chcete uvolnit
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „UndoCheckOut“
3. Od této chvíle je dokument k dispozici všem uživatelům
4. V hlavním okně můžete vidět, že soubory, na kterých jste provedl(a) „UndoCheckOut“ se označily jako volné.

2.5.9 Aktualizace pracovních kopií (Update)

Operace „Update” aktualizuje všechny soubory, které máte jako CheckedOut. To znamená všechny soubory, které máte vyhrazené a uzamčené výhradně pro sebe. Díky této funkci máte zajištěno, že na disku máte vždy poslední verzi dokumentů.

1. Vyberte v hlavním okně objekt, který chcete aktualizovat
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Update“
3. Po kliknutí na „Update“ se provede stažení aktuální verze na disk

2.5.10 Uložení změn do repository (Commit)

Pokud chcete uložit změny, které jste na dokumentech provedl(a), použijte funkci „Commit“. Funkce „Commit“ uloží všechny soubory, které máte označeny jako „CheckedOut“.

1. Vyberte v hlavním okně objekt, který chcete uložit
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Commit“
3. Zobrazí se dialogové okno pro zadání poznámky k verzi
4. Po zadání poznámky se uloží soubory do repository a v hlavním okně můžete vidět, že se u změněných souborů zvýšila verze

2.5.11 Smazání souboru z aktuální verze (Delete)

Pokud chcete smazat soubor z repository, použijte funkci „Delete“. Funkce „Delete“ ovšem fyzicky nesmaže soubor z repository, pouze ho vyřadí z aktuální verze. To znamená, že pokud se bude chtít vrátit k předešlé verzi, tak v ní bude zahrnut. Pokud chcete soubor úplně vymazat z repository (tz. i z historie), tak použijte funkci „Delete permanent“.

1. Vyberte v hlavním okně objekt, který chcete smazat
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Delete“
3. Zobrazí se dialogové okno pro zadání poznámky k verzi

4. Po zadání poznámky se objekt smaže z aktuální verze a v hlavním okně můžete vidět, že daný objekt z repository zmizel. (Pokud v hlavním menu kliknete na „Možnosti -> Zobrazit i smazané soubory“, tak v okně můžete vidět i soubory, které byly smazány)

2.5.12 Obnovení smazaného souboru (Undelete)

Pokud chcete smazaný soubor obnovit, použijte funkci „Undelete“. Tato funkce uloží daný dokument do nové verze.

1. Vyberte v hlavním okně objekt, který chcete obnovit
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Undelete“
3. Zobrazí se dialogové okno pro zadání poznámky k verzi
4. Po zadání poznámky se objekt obnoví z aktuální verze a v hlavním okně můžete vidět, že daný objekt z repository stal aktivní.

2.5.13 Trvalé smazání objektu z repository

Tato operace provedete úplné smazání objektu z repository. Rozdíl oproti operaci „Delete“ je, že „Delete Permanent“ vymaže i historii tohoto souboru. To znamená, že pokud se budete vracet k libovolné verzi, tento objekt již v ní nebude zahrnut.

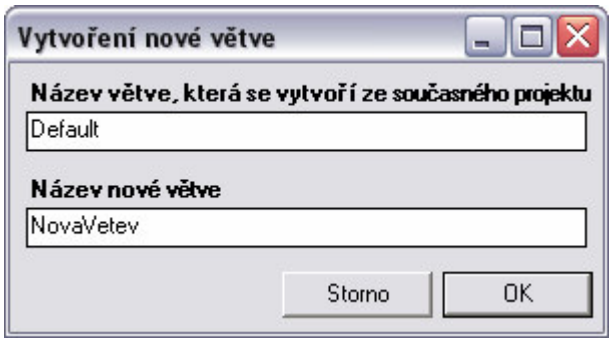
1. Vyberte v hlavním okně objekt, který chcete trvale smazat
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Delete permanent“
3. Zobrazí se potvrzení, zda-li opravdu chcete daný objekt smazat
4. Po potvrzení se objekt a jeho historie vymažou z repository a v hlavním okně můžete vidět, že se již objekt v repository nevyskytuje.

2.5.14 Vytvoření větve projektu (Branch)

Větev slouží k rozdělení projektu na dvě části. Používá se např. v případě, kdy byla ukončena část vývoje a je třeba zároveň pracovat na dalším vývoji a zároveň opravovat chyby z předchozího vývoje.
Vytváření větví lze rozdělit na dvě části:

Vytvoření větve z projektu

1. Vyberte v hlavním okně projekt, ve kterém chcete vytvořit větev
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Vytvořit branch“
3. Zobrazí se dialog pro zadání názvu první a druhé větve



Obr. 2.11: Vytvoření větve z projektu

4. Po zadání názvů dojde k rozdělení projektu na dvě větve. V hlavním okně můžete vidět, že došlo k vytvoření dvou větví

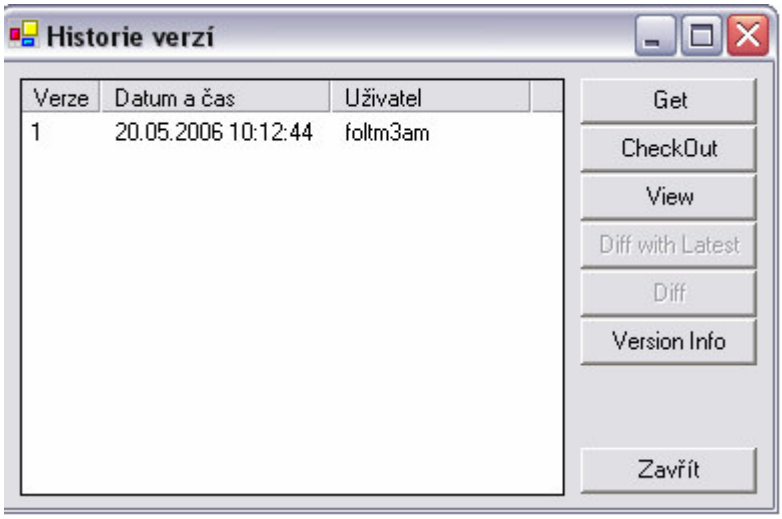
Vytvoření větve z již existující větve

1. Vyberte v hlavním okně větev, ze které chcete vytvořit další větev
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „Vytvořit branch“
3. Zobrazí se stejný dialog jako v předchozím případě pro zadání názvu nové větve
4. Po zadání názvů dojde k rozdělení vytvoření nové větve. V hlavním okně můžete vidět, že v projektu vznikla další větev.

2.5.15 Zobrazení historie

Historii si můžete nechat zobrazit pro libovolný projekt, jeho podsložku nebo dokument. V historii můžete vidět kdo a kdy prováděl změny a na jakých souborech byly změny provedeny.

1. Vyberte v hlavním okně objekt, pro který chcete zobrazit historii
2. Klikněte pravým tlačítkem myši na tento objekt a ze vyskakovacího okna vyberte „History…“
3. Otevře se následující dialogové okno (obr. 2.12)



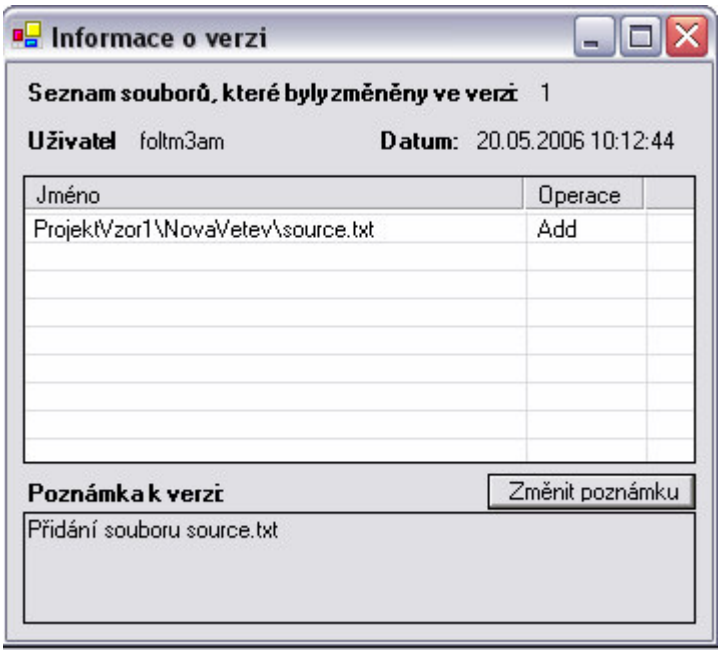
Obr. 2.12: Historie

V tabulce lze vidět čísla verzí a jména uživatelů, kteří verzi uložili. Na pravé straně lze vidět tlačítka, které nad vybranou verzí provádí operace.

Text tlačítka	Funkce
Get	Provede stažení verze z historie na disk
CheckOut	Provede stažení verze z historie na disk a soubory označí jako CheckedOut
View	Zobrazí obsah dokumentu v dané verzi
Diff with Latest	Dokument v dané verzi porovná s poslední verzí
Diff	Porovná dvě označené verze mezi sebou
VersionInfo	Zobrazí detailní informace o verzi (seznam změněných souborů, poznámku)

Tabulka 2.3: Seznam funkcí v historii

Označením verze ze seznamu a kliknutím na tlačítko VersionInfo se zobrazí podrobnější informace o verzi (obr. 2.13).



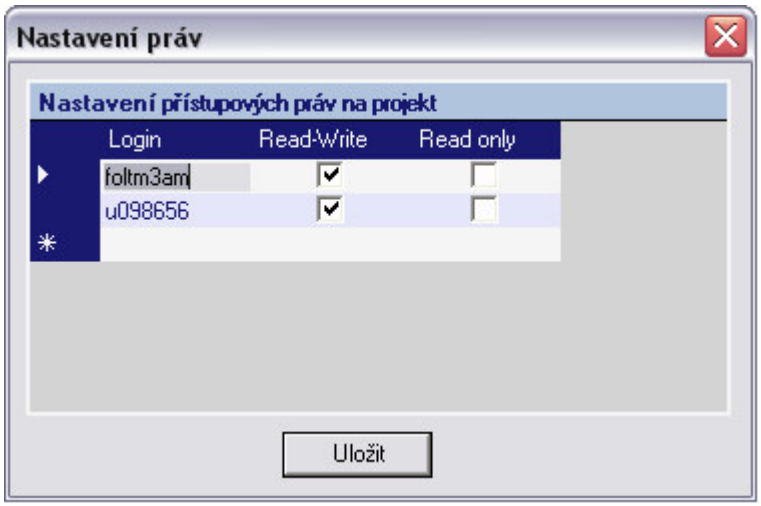
Obr. 2.13: Podrobné informace o verzi

V dialogovém okně lze vidět soubory, které byly v dané verzi zahrnuty a operace, které se provedly (např. změna, přidání, odebrání).
V dolní části lze vidět poznámku, kterou lze kliknutím na tlačítko „Změnit poznámku“ upravovat. Po kliknutí na toto tlačítko se zobrazí dialog na změnu poznámky.

2.5.16 Změna nastavení práv na projektu

Tato funkce je užitečná v případě, že chcete některým uživatelům povolit ukládání do repository a některým povolit pouze čtení, popř. úplně zamezit přístup k projektu.

1. V hlavním okně vyberte projekt, pro který chcete změnit oprávnění
2. Klikněte na něj pravým tlačítkem myši a z vyskakovacího okna vyberte „Nastavit přístupová práva…“
3. Otevře se vám dialogové okno se seznamem uživatelů (obr. 2.14)



Obr. 2.14: Nastavení práv

4. V daném okně nastavte práva, přičemž pokud nezaškrtnete ani jedno pole, znamená to, že daný uživatel nemá k projektu přístup.
5. Pro uložení klikněte na tlačítko „Uložit“

2.5.17 Přihlášení jako jiný uživatel

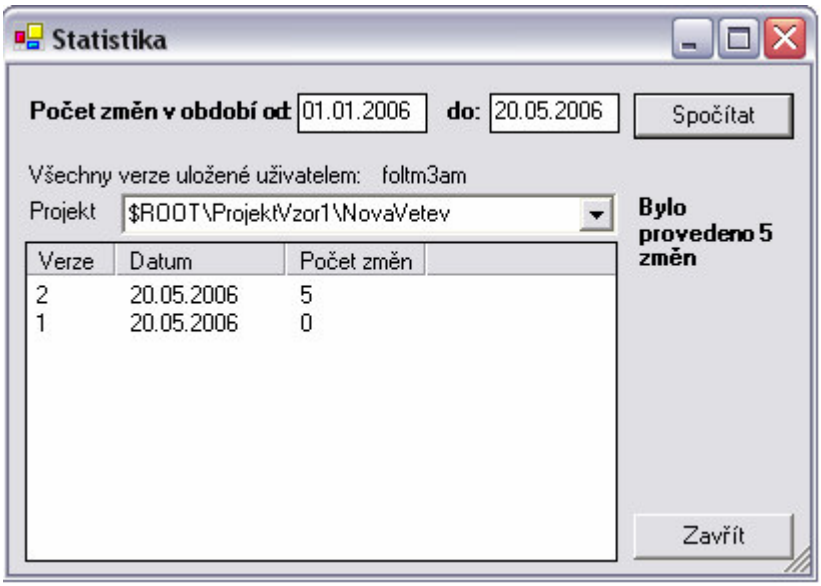
Pokud se chcete přihlásit jako jiný uživatel nebo se připojit na jinou repository, postupujte následovně:

1. V hlavním okně aplikace klikněte v menu na „Soubor -> Přihlásit“ se jako jiný uživatel…
2. Zobrazí se stejný dialog jako při spuštění programu
3. Zadejte přihlašovací údaje a adresu počítače, kde běží serverová část
4. Pokud přihlášení proběhlo úspěšně, zobrazí se potvrzují dialog.

2.5.18 Statistika

V aplikaci lze rovněž sledovat statistiky o četnosti změn pro daného uživatele.

1. V hlavním okně aplikace klikněte v menu na „Možnosti -> Statistika…“
2. Zobrazí se dialogové okno (obr. 2.15)



Obr. 2.15: Statistika

V okně lze vidět seznam všech projektů v repository. Dále pak seznam verzí a počet změn v této verzi. V okně lze také zadat časový interval a po kliknutí na tlačítko „Spočítat“ se zobrazí počet změn, které uživatel za daný časový úsek provedl.

2.5.19 Zobrazení smazaných souborů

Pokud chcete v hlavním okně vidět i soubory, které se byly někdy v historii smazány a v poslední verzi se již nevyskytují, postupujte následovně:

1. V menu klikněte na „Možnosti -> Zobrazit” i smazané soubory
2. V hlavním okně nyní uvidíte i soubory, které byly smazány. Tyto soubory jsou označeny písmenem x v sloupci Smazané.

3. Programová dokumentace

Tato kapitola obsahuje popisy funkcí, které serverová část poskytuje klientským částem. Součástí je popis funkce, deklarace, seznam a význam parametrů a další poznámky.

3.1 Atributy

3.1.1 RepositoryNodeName

Název prvního uzlu, který pojmenovává Root repository

Deklarace:
public static string RepositoryNodeName;

3.1.2 RepositoryNodeName

Jméno aktuálně přihlášeného uživatele do repository

Deklarace:
public string RepositoryUserName;

3.1.3 PreparedFiles

Seznam souborů, které jsou připraveny ke stažení klientem

Deklarace:
public DSRepositoryFile PreparedFiles

3.1.4 FailedFiles

Seznam souborů, u kterých nastala chyba a nelze na nich provádět operaci.

Deklarace:
public DSRepositoryFile FailedFiles

3.2. Obsah složky v repository

3.2.1 RepositoryDirContent

Vrací obsah adresáře v repository pro přihlášeného uživatele

Deklarace:
public DSDirectory RepositoryDirContent(string repositoryPath)

Parametry:
repositoryPath - Relativní cesta do adresáře v repository

Poznámka:
Funkce je přetížena.

3.2.2 RepositoryDirFilesList

Vrací seznam souborů v daném adresáři v repository (nerekurzivní)

Deklarace:
public string[] RepositoryDirFilesList(
string relativePath,
bool validOnly)

Parametry:
relativePath - Cesta k adresáři v repository
validOnly - Zda-li vrátit pouze platné dokumenty

3.2.3 RepositoryDirDirsList

Vrací seznam všech složek v daném adresáři v repository

Deklarace:
public string[] RepositoryDirDirsList(
string relativePath,
bool validOnly)

Parametry:
relativePath - Cesta k adresáři v repository
validOnly - Zda-li vrátit pouze platné složky

3.3 Nová verze

3.3.1 StartNewVersion

Připraví repository na vložení nové verze projektu.

Deklarace:
public void StartNewVersion(
string relativePath,
string versionNote,
bool clearFileLists)

Parametry:
relativePath - Relativní cesta k projektu(může být i k lib. složce v projektu
versionNote - Poznámka k verzi
clearFileLists - Zda-li se mají vymazat seznamy úspěšných a neúspěšných souborů

Poznámka:
Tato metoda musí být zavolána před každou operací, která vytváří novou verzi v repository.

3.3.2 CloseVersion

Uzavře verzi a uloží ji do repository

Deklarace:
public void CloseVersion()

Poznámka:
Tato metoda musí být vždy volána až po zavolání StartNewVersion

3.4 Add, Get, CheckOut

3.4.1 AddFileToRepository

Přidá soubor do repository

Deklarace:
public void AddFileToRepository(
string repositoryPath,
string fileName,
Diff.DiffList_TextFile sLF)

Parametry:
repositoryPath - Relativní cesta do repository, kde má být soubor umísťen
fileName - Název přidávaného souboru
sLF - Přidávaný soubor

3.4.2 PrepareRepFilesForCopy

Sestaví seznam souboru z repository, které lze stáhnout na disk k určité verzi.

Deklarace:
public void PrepareRepFilesForCopy(
string repositoryPath,
string[] fileNames,
bool checkOut,
int version,
bool clearLists)

Parametry:
repositoryPath - Relativní cesta do adresáře v repository, kde se soubory nacházejí
fileNames - Seznam souborů
checkOut - Zda-li se provádí CheckOut
version - Číslo verze
clearLists - Zda-li se mají nejprve seznamy souborů vymazat

Poznámka:
Tento seznam je umístěn v PreparedFiles. Ve FailedFiles je seznam souborů, které nelze stáhnout. Funkce je přetížena.

3.4.3 PrepareRepDirectoryForCopy

Sestaví seznam souborů z daného adresáře, které lze získat z repository k určité verzi.

Deklarace:
public void PrepareRepDirectoryForCopy(
string repositoryPath,
string dirName,
bool checkOut,
int version,
bool clearLists)

Parametry:
repositoryPath - Relativní cesta v repository
dirName - Název adresáře
checkOut - Zda-li se má provést CheckOut tohoto souboru
version - Číslo verze
clearLists - Zda-li se mají nejprve seznamy souborů vymazat

Poznámka:
Funkce je přetížena.

3.4.4 GetRepositoryFileStream

Vrací stream na soubor v repository

Deklarace:
public StreamReader GetRepositoryFileStream(
string repositoryPath,
string fileName)

Parametry:
repositoryPath - Relativní cesta do repository
fileName - Název soubor

3.4.5 GetRepositoryFile

Načte z repository poslední verzi souboru

Deklarace:
public Diff.DiffList_TextFile GetRepositoryFile(
string relativePath,
string fileName)

Parametry:
relativePath - Relativní cesta do Repository>
fileName - Jméno souboru

Poznámka:
Funkce je přetížena.

3.4.6 GetRepositoryFileWithInfo

Načte z repository soubor. Přidává dodatečnou informaci ke každému řádku v souboru.

Deklarace:
public Diff.DiffList_TextFile GetRepositoryFileWithInfo(
string relativePath,
string fileName,
string noteFormat)

Parametry:
relativePath - Relativní cesta k souboru
fileName - Jméno souboru
noteFormat - Formát dodatečné informace

3.4.7 CheckoutFiles

Funkce označí všechny soubory v seznamu jako CheckOut.

Deklarace:
public void CheckoutFiles(
DSRepositoryFile filesToCheckOut,
string relativePathStart,
string outputPath)

Parametry:
filesToCheckOut - Seznam souborů, na kterých se má provést CheckOut
relativePathStart - Relativní cesta do repository
outputPath - Do jakého adresáře na disku se soubory uložily

Poznámka:
Funkce používá seznam PreparedFiles. Do seznamu FailedFiles ukládá soubory, které nelze uzamknout.

3.4.8 RepositoryFileUndoCheckout

Provede UndoCheckOut na zadaný soubor v repository

Deklarace:
public void RepositoryFileUndoCheckout(
string repositoryPath,
string fileName)

Parametry:
repositoryPath - Relativní cesta do repository
fileName - Název souboru, na kterém se má provést UndoCheckOut

3.4.9 UndoCheckOut

Provede UndoCheckOut na seznam souborů a adresářů.

Deklarace:
public void UndoCheckOut(
string repositoryPath,
string[] files,
string[] dirs,
bool clearLists)

Parametry:
repositoryPath - Relativní cesta do repository
files - Seznam souborů
dirs - Seznam složek
clearLists - Zda-li vymazat seznam úspěšných a neúspěšných souborů

Poznámka:
Funkce plní seznam PreparedFiles soubory, na kterých se provedla operace. Funkce je přetížena.

3.5 Update

3.5.1 PrepareFilesForUpdate

Sestaví seznam souborů, které lze přihlášeným uživatelem aktualizovat.

Deklarace:
public void PrepareFilesForUpdate(
string repositoryPath,
string[] files,
string[] dirs,
bool clearLists)

Parametry:
repositoryPath - Relativní cesta do repository
files - Seznam souborů
dirs - Seznam adresářů
clearLists - Zda-li se mají PreparedFiles a FailedFiles vymazat

Poznámka:
Funkce sestaví do seznamu PreparedFiles seznam souborů, které jsou checked-out právě přihlášeným uživatelem a tedy připraveny pro Update

3.6 Commit

3.6.1 PrepareListForCommit

Připraví seznam všech souborů, na kterých lze provést Commit.

Deklarace:
public void PrepareListForCommit(
string repositoryPath,
string[] fileNames,
string[] dirNames,
bool clearLists)

Parametry:
repositoryPath - Relativní cesta do repository
fileNames - Seznam souborů
dirNames - Seznam adresářů
clearLists - Zda-li se mají vymazat seznamy souborů ke stažení a nepovolených souborů

Poznámka:
Zahrnuje i soubory, které jsou v libovolných podadresářích daných adresářů.

3.6.2 CommitFile

Provede Commit zadaného souboru

Deklarace:
public void CommitFile(
string repositoryPath,
string fileName,
Diff.DiffList_TextFile discFile)

Parametry:
repositoryPath - Relativní cesta do složky v repository, kde se soubor nachází
fileName - Název souboru
discFile - Soubor na klientském počítači

Poznámka:
Funkce vyžaduje otevřenou verzi, tedy je nutné před voláním této metody zavolat metodu StartNewVersion

3.7 Rename

3.7.1 RenameFile

Metoda přejmenuje soubor v repository

Deklarace:
public void RenameFile(
string repositoryPath,
string currentFileName,
string newFileName)

Parametry:
repositoryPath - Relativní cesta do repository
currentFileName - Název souboru
newFileName - Nový název souboru

Poznámka:
Metoda vyžaduje otevřenou verzi, to znamená před voláním této metody musí být zavolána metoda StartNewVersion

3.7.2 CanRename

Ověří, zda-li se může provést přejmenování položky v repository

Deklarace:
public string CanRename(
string repositoryPath,
string currentName,
string newName,
RepositoryItemType type)

Parametry:
repositoryPath - Relativní cesta do adresáře v repository
currentName - Současný název položky
newName - Nový název položky
type - Typ položky

Poznámka:
Funkce ověřuje, zda-li soubor není nějakým uživatelem blokován a zda-li neexistuje položka se stejným názvem.

3.7.3 RenameDir

Přejmenuje složku, projekt, větev v repository

Deklarace:
public void RenameDir(
string repositoryPath,
string dirName,
string newDirName)

Parametry:
repositoryPath - Relativní cesta ke složce v repository
dirName - Název složky
newDirName - Nový název složky

Poznámka:
Funkce provede přejmenování složky a aktualizuje informace o cestách v projektech

3.8 Delete

3.8.1 Delete

Vymaže soubory a složky ze současné verze projektu

Deklarace:
public void Delete(
string repositoryPath,
string[] files,
string[] dirs,
bool clearLists)

Parametry:
repositoryPath - Relativní cesta do adresáře v repository
files - Seznam souborů, které se mají smazat
dirs - Seznam složek, které se mají smazat
clearLists - Zda-li se mají vyčistit seznamy úspěšných a neúspěšných souborů

Poznámka:
Metoda vyžaduje otevřenou verzi, je tedy nutné před voláním této funkce zavolat metodu StartNewVersion.
Metoda vymaže pouze soubory, ke kterým má přihlášený uživatel právo.

3.8.2 UnDelete

Provede UnDelete na seznam souborů a složek v repository

Deklarace:
public void UnDelete(
string repositoryPath,
string[] files,
string[] dirs,
bool clearLists)

Parametry:
repositoryPath - Relativní cesta do repository
files - Seznam souborů, na kterých se provede UnDelete
dirs - Seznam složek, na kterých se provede UnDelete
clearLists - Zda-li se mají vymazat seznamy úspěšných a neúspěšných souborů

Poznámka:
Funkce obnoví soubory, které byly v některé z předešlých verzí smazány. Metoda vyžaduje otevřenou verzi, je tedy nutné před voláním této funkce zavolat metodu StartNewVersion.
Metoda obnoví pouze soubory, ke kterým má přihlášený uživatel právo.

3.8.3 DeletePermanent

Smaže trvale zadanou složku z repository

Deklarace:
public void DeletePermanent(string relativePath)

Parametry:
relativePath - Relativní cesta ke složce, která se má smazat

Poznámka:
Tato metoda vymaže složku úplně z repository včetně její historie.
Po provedení této funkce, v repository nezůstane žádná informace o existenci složky.

3.9 Historie

3.9.1 GetHistoryForFile

Vrací historii pro daný soubor

Deklarace:
public BDO.DSProjectHistory GetHistoryForFile(
string repositoryPath,
string fileName)

Parametry:
repositoryPath - Relativní cesta k souboru v repository
fileName - Název souboru

3.9.2 GetHistoryForDir

Vrací historii pro daný adresář

Deklarace:
public BDO.DSProjectHistory GetHistoryForDir(
string repositoryPath,
string dirName)

Parametry:
repositoryPath - Relativní cesta k adresáři v repository
dirName - Název adresáře

3.9.3 GetHistoryForProject

Získá historii projektu ze zadané relativní cesty (relativní cesta může být k lib. složce v projektu)

Deklarace:
public DSProjectHistory GetHistoryForProject(string relativePath)

Parametry:
relativePath - Relativní cesta do repository

3.9.4 RepositoryVersionInfo

Vrací informace o konkrétní verzi projektu

Deklarace:
public DSProjectHistory RepositoryVersionInfo(
string relativePath,
int version)

Parametry:
relativePath - Relativní cesta do repository
version - Číslo verze, u které chceme získat informace

3.10 Projekt

3.10.1 CreateNewProject

Vytvoří nový projekt v zadané cestě

Deklarace:
public bool CreateNewProject(
string repositoryPath,
string projectName)

Parametry:
repositoryPath - Relativní cesta do repository, kde se má nový projekt vytvořit
projectName - Název projektu

Poznámka:
Funkce je přetížena.

3.10.2 GetAllProjects

Vrací seznam cest ke všem projektům v repository, ke kterým má uživatel přístup

Deklarace:
public string[] GetAllProjects()

Poznámka:
Funkce vrací pouze ty projekty, ke kterým má přihlášený uživatel právo alespoň pro čtení.

3.10.3 ProjectExists

Ověřuje, zda-li na zadané cestě již neexistuje projekt se stejným názvem.

Deklarace:
public bool ProjectExists(
string relativePath,
string projectName)

Parametry:
relativePath - Relativní cesta do složky
projectName - Název projektu

Poznámka:
Vrací pouze informaci o existenci projektu, ale může existovat složka se stejným názvem

3.10.4 IsProjectAvalaible

Zjišťuje rekurzivně, zda-li v adresáři neexistuje platný soubor, který by byl označen jako CheckOut

Deklarace:
public bool IsProjectAvalaible(string relativePath)

Parametry:
relativePath - Relativní cesta k projektu

3.10.5 SaveProjectHistory

Uloží historii projektu do Repository

Deklarace:
public void SaveProjectHistory (
string repositoryPath,
string projectName,
DSProjectHistory dsProject)

Parametry:
repositoryPath - Relativní cesta do repository
projectName - Název projektu
dsProject - Historie projektu

3.10.6 GetRightsForProject

Vrací nastavení práv pro daný projekt

Deklarace:
public DSProjectHistory GetRightsForProject(
string repositoryPath,
string projectName)

Parametry:
repositoryPath - Relativní cesta do repository
projectName - Název projektu

Poznámka:
Funkce je přetížena.

3.10.7 SaveRightsForProject

Uloží nastavení práv do projektu

Deklarace:
public void SaveRightsForProject(
string repositoryPath,
string projectName,
DSProjectHistory dsRights)

Parametry:
repositoryPath - Relativní cesta do repository
projectName - Název projektu
dsRights - Nastavení práv

3.10.8 ChangeProjectVersionNote

Funkce změní poznámku k již uložené verzi

Deklarace:
public void ChangeProjectVersionNote(
string relativePath,
string projectName,
int version,
string newNote)

Parametry:
relativePath - Relativní cesta k projektu v repository
projectName - Název projektu
version - Číslo verze
newNote - Nová poznámka

Poznámka:
Funkce je přetížena.

3.10.9 HasUserAccessToProject

Zjišťuje, zda-li uživatel má přístup k projektu

Deklarace:
public bool HasUserAccessToProject(
string repositoryPath,
string projectName)

Parametry:
repositoryPath - Relativní cesta do repository
projectName - Jméno projektu

Poznámka:
Funkce je přetížena.

3.11 Branch

3.11.1 MoveProjectToBranch

Přesune projekt do větve

Deklarace:
public void MoveProjectToBranch(
string relativeName,
string dirName)

Parametry:
relativeName - Relativní cesta do repository
dirName - Název adresáře

Poznámka:
Funkce, pouze vytvoří z projektu větev. Pro vytvoření další větve je třeba zavolat metody CreateBranch.

3.11.2 CreateBranch

Vytvoří novou větev z již existující větve

Deklarace:
public void CreateBranch(
string relativeName,
string currentBranchName,
string newBranchName)

Parametry:
relativeName - Relativní cesta do repository
currentBranchName - Název současné větve
newBranchName - Nový název větve

4. Závěr

Úkolem bylo vytvořit systém pro správu verzí, který by byl použitelný i v konkurenci podobných a hodně rozšířených systémů, čehož se podle mého názoru podařilo dosáhnout. Věřím, že systém najde uplatnění jak u méně zkušených uživatelů, kteří se dosud obávali složitosti ovládání, tak u uživatelů pokročilých, které odrazuje komplikovaná instalace či náročná správa existujících systémů.

Při tvorbě tohoto systému jsem implementoval funkce, které v ostatních produktech chybí nebo jsou implementovány v jednom a v druhém již nikoliv. Přesto stále existuje relativně velké množství funkcí, které v tak krátkém čase nebylo možné implementovat. Snažil jsem se demonstrovat, že lze navrhnout a vytvořit systém, který se v základních funkcích přibližuje již používaným systémům a navíc ukazuje, že stále existují funkce, jimiž lze tyto systémy rozšiřovat. Při navrhování VCS jsem zohledňoval předpokládaný budoucí vývoj skrze transparentní kód a důkladnou dokumentaci. Nepodařilo se mi vytvořit takový systém pro správu verzí, který by mohl zcela nahradit některý z existujících systémů, avšak domnívám se, že při dalším vývoji lze takového cíle dosáhnout využitím této práce jako základu, který jsem v rámci bakalářského projektu navrhl a implementoval.

5. Literatura

[1] Collins-Sussman, Fitzpatrick, Pilato: Version Control with Subversion, http://svnbook.red-bean.com/
[2] CVS - Concurrent Versions System: http://www.nongnu.org/cvs/
[3] Küng, Onken, Large: TortoiseSVN - http://tortoisesvn.berlios.de/?q=doc_release
[4] The Code Project – http://www.codeproject.com
[5] Wikipedia – http://www.wikipedia.com