Verzovací systémy
Version Control Systems

Marek Novosád

Bakalářská práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Marek Novosád**
Osobní číslo:	**A15795**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Verzovací systémy**
Téma anglicky:	**Version Control Systems**

Zásady pro vypracování:

1. **Vypracujte literární rešerši.**
2. **Popište základní principy verzování.**
3. **Porovnejte nejpoužívanější systémy.**
4. **Vytvořte vlastní webovou vizualizaci vzniku, slučování a evidence různých verzí.**
5. **Použijte open-source technologie.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **COLLINS-SUSSMAN, Ben, Brian W FITZPATRICK a C PILATO. Version control with Subversion. 1st ed. Sebastopol, CA: O'Reilly Media, 2004, xvii, 299 p. ISBN 0596004486.**
2. **CHACON, Scott. Pro Git. Praha: CZ.NIC, 2009, 263 s. CZ.NIC. ISBN 978-80-904248-1-4.**
3. **FOGEL, Karl a Moshe BAR. Open source development with CVS. 2nd ed. Scottsdale, AZ: Coriolis Group Books, 2001, xxi, 345 p. ISBN 078-8581001738.**
4. **NAGEL, William A. Subversion version control: using the Subversion version control system in development projects. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005, xxii, 343 p. ISBN 0131855182.**
5. **O'SULLIVAN, Bryan. Mercurial: The Definitive Guide. 1st ed. United States of America: O'Reilly Media, 2009. ISBN 978-0596800673.**

Vedoucí bakalářské práce: **doc. Ing. Martin Sysel, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **24. února 2017**

Termín odevzdání bakalářské práce: **24. května 2017**

Ve Zlíně dne 24. února 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis diplomanta

ABSTRAKT

Tato bakalářská práce se zabývá verzovacími systémy. Postupně jsou v ní vysvětleny základní operace verzování a větvení. Dále popisuje a porovnává nejpoužívanější verzovací systémy. Jsou zmíněny i zásuvné moduly ve funkci verzovacích systémů. Na konci teoretické části jsou popsány technologie využité při tvorbě praktické části.

V praktické části je vytvořena vizualizace vzniku, slučování a evidence verzí.

Klíčová slova: Verzovací systém, Větvení, Vizualizace, jQuery

ABSTRACT

This bachelor's thesis engages version control systems. Gradually, it explains basic versioning and branching operations. It also describes and compares most commonly used versioning systems. The work also mentions the plug-in modules in the versioning function. At the end of the theoretical part are described the technologies used in creating the practical part.

Practical part contains visualization of creation, merging and registration version.

Keywords: Versioning system, Branching, Visualization, jQuery

Poděkování:

Rád bych poděkoval vedoucímu mé práce doc. Ing. Martinu Syslovi, Ph.D. za cenné názory, připomínky a zapůjčení materiálů k vypracování této bakalářské práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato práce se v teoretické části věnuje verzovacím systémům. Ty jsou rozděleny na Centralizované a Distribuované verzovací systémy. Dále jsou v teoretické části vysvětleny základní operace pro práci s verzovacími systémy a princip větvení. V další části jsou pak popsány a porovnány nejpoužívanější verzovací systémy kterými jsou Git, Mercurial, Subversion, Fossil a Bazaar. V práci je zmínka i o zásuvných modulech ve funci verzovacího systému a popsán VersionPress jako jeden z nich. Na konci teoretické části jsou popsány technologie jQuery a BootsTrap, které byly použity pro vytvoření praktické práce. V části o jQuery jsou zmíněny jedny z hlavních výhod této knihovny, popsány selektory knihovny jQuery a popis co to jQuery vlastně je. U BootsTrapu je zmíněna jeho hlavní výhoda, která je využita v projektu Vizualizace Verzování.

V praktické části jsou popsány technologie použité k vytvoření vizualizace. Samotná vizualizace obsahuje ukázku vzniku, slučování a evidence různých verzí.

I. TEORETICKÁ ČÁST

1 HISTORIE

První verzovací systém, který dal základ i všem ostatním verzovacím systémům byl Source Code Control System (SCCS). Vznikl v roce 1972 v bellových laboratořích a jeho tvůrcem byl Marc J. Rochkind.[11] V roce 1982 ho nahradil pokročilejší Revision Control System (RCS), který vyvinul Walter F. Tichy z Purdueovy univerzity.[12] Dalším verzovacím systémem byl Concurrent Version System (CVS) jehož první verzi vyvinul Dick Grune ve formě shellových skriptů. V roce 1989 Brian Berliner navrhl a přepsal systém CVS. CVS byl systém, který již pracoval s rozsáhlými projekty. Udržoval pořádek v souborech tím, že všechny projekty byly umístěny v jednom adresáři (repozitáři). Byl často využívaný i díky možnosti práce v síti nebo multiuživatelskému režimu. Pro systém CVS existují i nadstavby pro použití v grafickém režimu, jednou z nich je například WinCVS.[13] Na začátku 21. století vznikl Subversion (SVN), jehož vývoj řídila firma CollabNet jelikož se rozhodla nahradit dříve používaný systém CVS. Vývoj Subversionu započal v květnu roku 2000 detailní analýzou a trval 14 měsíců do 31. srpna 2001 kdy se stal použitelným a dostal na starost správu zdrojových kódu samotného Subversionu.[8] V dalších letech vznikají verzovací systémy s jinou architekturou. V následujících letech se začína pracovat na decentralizovaných verzovacích systémech. Jedním z prvních je GNU Arch, který byl později nahrazen systémem Bazaar.[14] Bazaar je dodnes stále využíván tisíci projekty mezi které patří například Unubtu, MySql, InkScape a jiné.[15] V roce 2002 se objevuje verzovací systém Darcs, který nabízí volnější způsob práce a jednodušší uživatelské rozhraní. Nevyžaduje centrální server a pracuje i v režimu offline.[16] V roce 2005 již přichází dnes nejvyužívanější verzovací systémy Git, Mercurial a dříve zmíněný Bazaar.[9][10][15] O rok později ještě Fossil.[17]

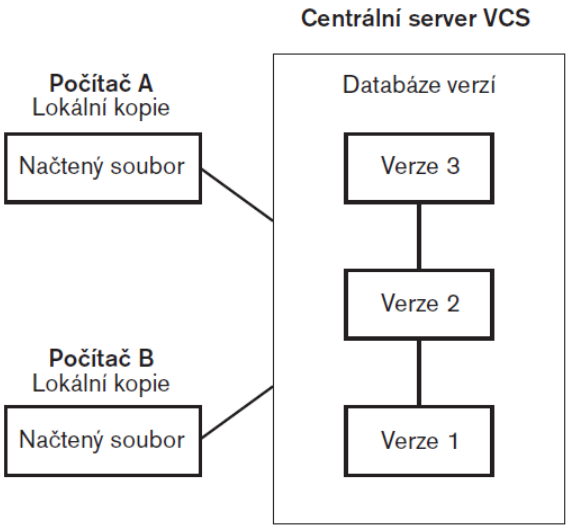
2 VERZOVACÍ SYSTÉMY

Verzovací systémy jsou systémy, které zaznamenávají změny v souboru nebo ve více souborech v průběhu času. Existuje mnoho důvodů ,proč verzovací systémy využívat. Mezi ty hlavní důvody patří: [5]

- Budou za vás sledovat historii a vývoj vašeho projektu. Takže nebudete muset dál sledovat kdo, kdy, proč a jak něco v projektu změnil. Verzovací systém to udělá za vás.
- Řeší také problémy týmové spolupráce, kdy se například bude snažit více lidí současně provést potencionálně nekompatibilní změny. Systém tyto konflikty pomáhá identifikovat a řešit.
- Snadná oprava chyb. Později po provedení změn v souboru či souborech zjistíte, že změna byla chybná a pomocí verzovacího systému se vratíte do doby před zavedením této chyby.

2.1 Centralizované verzovací systémy

Centralizované verzovací systémy (CVCS – Centralized Version Control Systém)– Tyto systémy např. CVS, Subversion nebo Perforce obsahují serverovou část, která ukladá všechny verze souborů. Jednotlivý uživatelé si mohou z centrálního repozitáře potom stahovat konkrétní soubory. Tento systém řeší problém spolupráce v týmu, kdy má každý pracovník přístup ke všem verzím (viz obrázek 1.). [2]



Obrázek 1 Diagram centralizované správy verzí.[2]

Mezi výhody Centralizovaných verzovacích systému patři: [2]

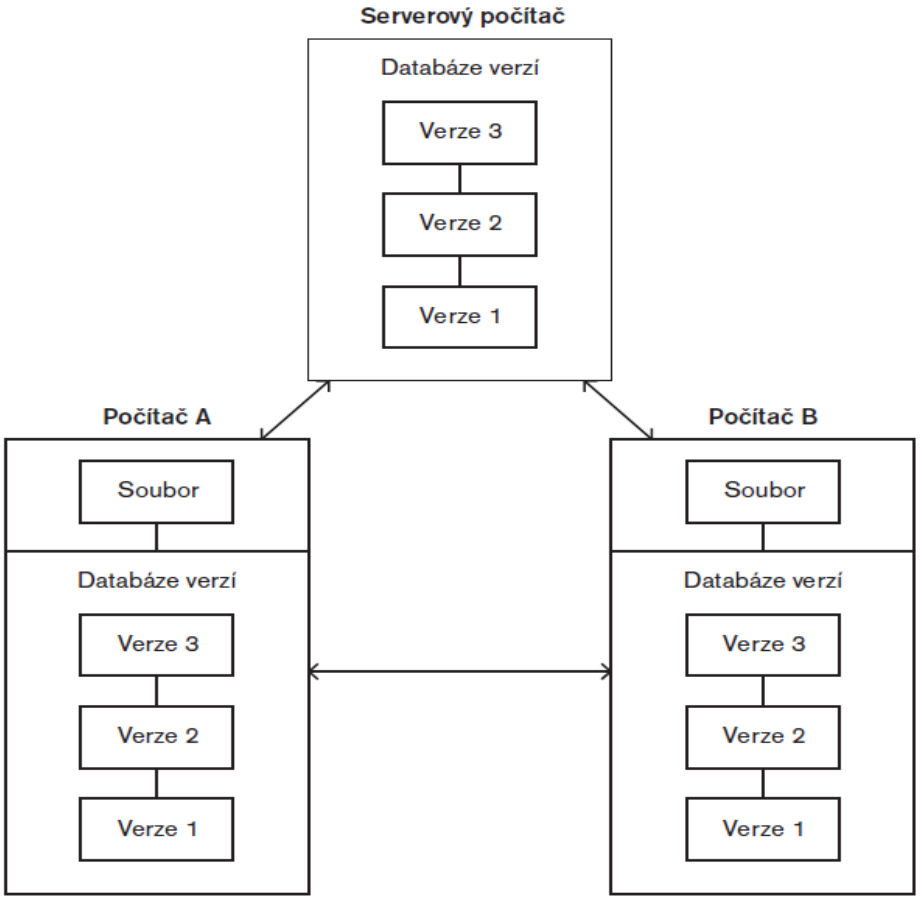
- Podstatně jednodušší správa CVCS než práce s lokálními databázemi na jednotlivých klientech.
- Administrátoři mají přesnou kontrolu nad prací a právy jednotlivých uživatelů.
- Každý uživatel přibližne ví, co dělají ostatní účastnící projektu.

Tato koncepce má však 2 velké nevýhody: [2]

- Při výpadku centrálního serveru je riziko kolapsu celého projektu. Po dobu výpadku nebude možné stahovat ani ukládat jednotlivé verze.
- Pokud by došlo k poruše disku, na kterém je uložena centrální databáze bez jakýchkoliv záloh, dojde ke ztrátě všech dat.

2.2 Distribuované verzovací systémy

Distribuované verzovací systémy (DVCS – Distributed Control Version Systém)– Mezi tyto systémy patří např. Git, Mercurial, Darcs nebo Bazaar. U systému DVCS nestahují uživatelé pouze aktuální verzi souboru, ale uchovávají kompletní kopie (viz obrázek 2.). [2]



Obrázek 2 Diagram distribuované správy verzí.[2]

Hlavní vyhodou těchto systémů je, že pokud dojde ke kolapsu serveru, nebo smazání dat, lze je snadno obnovit zkopírováním repozitáře od libovolného uživatele. Každý uživatel má totiž plnohodnotnou zálohu všech dat. Naopak nevýhodou tohoto systému je potřeba synchronizace repozitáře tak, aby byly vždy soubory aktuální.[2]

3 ZÁKLADNÍ PRINCIPY VERZOVÁNÍ

Při práci s verzovacími systémy mame k dispozici 18 zakládních operací pro centralizované verzovací systémy. Pro distribuované verzovací systémy jsou k dispozici 3 operace navíc: Clone, Push a Pull.

3.1 Create

Vytvoření nového prázdného repozitáře. Repozitář je místo, kde se budou ukládat všchny soubory. Repozitář obsahuje celou historii verzí.[7][4]

Při vytvoření repozitáře VCS očekává určitou idntifikaci, jako kde má být repozitář vytvořen a jaký ma mít název.[7]

3.2 Checkout

Vytvoření nové pracovní kopie. Tato operace se používá pokud je potřeba vytvořit pracovní kopii již existujícího repozitáře. Pracovní kopie se využívá v případě, že není možné upravovat soubor přímo, ale je nutno jej upravit bokem a poté tuto kopii sloučit s originálem. [7][4]

3.3 Commit

Aplikuje změny z pracovní kopie do hlavního repozitáře. Obvykle se také poskytuje zpráva nebo komentář, vysvětlující provedené změny. Tato zpráva zůstává součástí historie repozitáře. [7][4]

3.4 Update

Aktualizuje pracovní kopii se změnami v hlavním repozitáři. Update je zrcadlová operace k operaci Commit, obě operace přesouvají změny mezi pracovní kopii a repozitářem. [7][4]

3.5 Add

Přidání souboru nebo adresáře. Operace se používá pokud je nutno přidat existující soubor nebo adresář, který ještě není pod správou verzí, do hlavního repozitáře. Položka není přidána do repozitáře okamžitě, ale až po provení operace commit. [7][4]

3.6 Edit

Upravení souboru. Při použití operace Checkout se vytváří pracovní kopie, která je dále upravována a očekává se, že změny budou součástí repozitáře. U většiny nástroju správy verzí není nutno zahrnovat operaci Edit přímo do VCS. Stačí upravit soubor pomocí libovolného editačního programu a VCS upravený soubor později zapíše do repozitáře. [7]

3.7 Delete

Smazání souboru nebo adresáře. Při použití operace bude smazán soubor nebo adresář z repozitáře. Při pokusu o odstranění upraveného souboru z pracovní kopie může VCS hlásit, že změny provedené v pracovní kopii nejsou uloženy v repozitáři. Soubory smazané z repozitáře nejsou ve skutečnosti smazány úplně. Je pouze vytvořena nová verze souboru, která neobsahuje předešlou verzi, ale repozitář stále obsahuje i starší verze se smazaným souborem. [7][4]

3.8 Rename

Přejmenování souboru nebo adresáře. Při přejmenovávání souborů nebo adresářů v pracovní kopii dochází k přejmenování okamžitě. Pro přejmenování souboru nebo adresáře v repozitáři je nutné čekat na provedení změn. V praxi se nedoporučuje najednou přejmenovávat a měnit obsah souboru z důvodu možných chyb ze strany VCS. [7][10]

3.9 Move

Přesunutí souboru nebo adresáře. Při přesunutí souborů nebo adresářů v pracovní kopii dochází ke změnám okamžitě. Pro přesunutí souboru nebo adresáře v repozitáři je nutné čekat na provedení změn. [7][4]

3.10 Status

Seznam úprav, které byly provedeny v pracovní kopii. Operace Status zobrazuje změny, které budou provedeny v repozitáři pokud použijeme na soubor operaci Commit. [7][4]

3.11 Diff

Zobrazí podrobnosti o úpravách, které byly provedeny na pracovní kopii. Operace status obsahuje seznam změn, ale žádne jejich podrobnosti. Pokud chce uživatel znát přesně jaké změny byly provedeny u souborů, je třeba použít operaci Diff. [7][4]

3.12 Revert

Vrátí zpět změny, které byly provedeny v pracovní kopii. Kompletní vrácení pracovní kopie do doby jejího vytvoření pomocí operace Checkout. [7][4]

3.13 Log

Zobrazí změny provedené v repozitáři. Repozitář udržuje každou verzi, která kdy existovala. Operace Log všechny tyto verze a jejich změny zobrazuje spolu s dalšímy daty jako jsou: [7][4]

- Kdo provedl změnu
- Kdy byla změna provedena
- Zprávu nebo komentář o změně od uživatele

Nástroje pro správu verzí umožňují vypsat jen určité informace, které žádáme. Jako například jaké změny provedl určitý uživatel, nebo vypsání všech provedených změn za určité období. [7]

3.14 Tag

Spojuje smysluplné jméno s konkrétní verzí v repozitáři. Nástroje pro správu verzí poskytují způsob, jak označit konkrétní okamžik v historii repozitáře se srozumitelným názvem pro uživatele. [7][10]

Například if(-43==e) na if(==ERR_FILE_NOT_FOUND errorCode). [7]

3.15 Branch

Vytvoří další vývojovou línii. Operace Branch se použije, pokud je žádoucí aby se projekt vyvíjel ve více směrech. [7][10]

3.16 Merge

Použití změn z jedné větve do druhé. Obvykle používaná operace pokud bylo rozvětvení použito například pro opravení chyb v projektu. Díky této operaci není třeba chyby opravovat ručně v jiné větvi a stači je spojit. [7][4]

Při využívání operace Merge však hrozí riziko, že vznikne takzvaný Konflikt. Konflikt vznikne pri pokusu o spojení dvou větví, ve kterých jsou provedeny odlišné změny na stejném míste. V některých případech nedokáže verzovací systém sám konflikt vyřešit a je nutný zásah uživatele. Uživatel pak musí provést spojení ručně, přepsáním konfliktních řádků, nebo využít software pro toto určený. Jedním z nich je například KDiff3, pomocí kterého je možné určit jak se zachovat ke konfliktním řádkům .

3.17 Resolve

Řešení konfliktů vzniklích použitím operace Merge. V některých situacích vyžaduje operace Merge lidský zásah. Merge automaticky spojí vše co může být provedeno bezpečně. Vše ostatní je vyhodnoceno jako konflikt. Operaci Resolve použije uživatel, aby informoval VCS jak s konfliktem zacházet. [7][10]

3.18 Lock

Operace Lock zabraňuje ostatním uživatelům upravovat soubor. Používá se pro získání výhradního práva pro úpravu souboru. Hlavní využití u binarních souborů, které nemohou být automaticky sloučeny pomocí operace Merge. [7][10]

3.19 Clone

Operace Clone vytvoří novou instanci repozitáře, která je jeho kopií. Jedná se o podstatný rozdíl mezi centralizovanými a decentralizovanými verzovacími systémy. Po vytvoření repozitáře pomocí operace Create se využívá operace Clone pro vytvoření instance pro každého uživatele (lokálního repozitáře). [7][10]

3.20 Push

Překopírování změn z lokálního repozitáře do vzdáleného repozitáře. Operace slouží k synchronizaci mezi jednotlivými repozitáři. Obvykle je vzdáleným repozitářem myšlen repozitář, ze kterého byl soubor vytvořen pomocí operace Clone. Operaci Push může

omezit VCS na posílání pouze některých lokálních změn, jelikož vzdalený repozitáře může obsahovat věci, které nejsou v lokálním repozitáři obsaženy. [7][10]

3.21 Pull

Překopírování změn ze vzdáleného repozitáře do lokálního repozitáře. Stejně jako operace Push tak i Pull slouží k synchornizaci mezi repozitáři. Operace se využije v případě, že je nutné aktualizovat soubory v lokálním repozitáři. Aktualizace souboru se provádí vetšinou z repozitáře, ze kterého byl soubor vytvořen. [7][10]

Pro úplnou synchronizaci dvou instancí se doporučuje první použít operaci Pull pro stažení dat ze vzdáleného repozitáře a následně operaci Push pro aktualizaci změn na vzdáleném repozitáři. [7]

4 VĚTVENÍ

Větvení je jednou z hlavních výhod verzovacích systému zejména pro vývojáře. Verzování neznamená jen možnost vrátit se zpět na některou z predchozích verzí, ale podporuje i rozdělení na paralelní větve. To znamená, že se verzovaný soubor nebo projekt může v určitý moment rozdělit na více větví. Každá z těchto větví se pak dále vyvíjí jako samostatný projekt a neovlivňuje ostatní větve.[2]

Velkou roli hraje větvení také ve chvíli, kdy se zpětně objeví v softwaru chyba a je nutné ji neprodleně opravit. Například z důvodu, že se jedná o verzi, která je již vydána a používána. Není totiž možné provést opravu chyby ve stávajícím stavu projektu, na kterém pokračuje vývoj a dále se na něm pracuje. Tato verze není stabilní a otestovaná a nebylo by možne ji s opravou hned vydat. Chybu je nutné opravit na poslední stabilní verzi a tuto verzi vydat, tedy z poslední vydané verze vytvořit novou větev a v ní provést opravu chyby. Na této větvi pak stačí přetestovat opravenou funkcionalitu popřípadě související části, které by mohla oprava ovlivnit a ne celý projekt. Nicméně se chyba musí opravit i v hlavní vývojové linii, aby se neobjevila v příští verzi. Také je možné provést merge větve s opravou do hlavní vývojové linie, ale hrozí vznik konfliktu pokud již byly provedeny nějaké změny v této části kódu.[2]

Při vývoji softwaru se také větvení využívá pro ukladání rozdělané práce vývojáři, díky čemuž se neovlivňuje hlavní linie vývoje. Po dokončení práce pak stačí provést merge větve s novým vývojem do větve, ze které byla tato větev vytvořena.

Větvení také může sloužit k oddělení zákaznických větví. Kde jádro softwaru je vždy stejné a jsou provedeny vždy pouze úpravy do větve pro konkrétního zákazníka.

5 NEJPOUŽÍVANĚJŠÍ VERZOVACÍ SYSTÉMY

V následující části budou popsány nejpoužívanější verzovací systémy. Zmíněny budou zastůpci od centralizovaných i distribuovaných verzovacích systémů, jejich výhody i nevýhody.

5.1 Subversion (SVN)

Obrázek 3 Logo verzovacího systému subversion [8]

Patří mezi centralizované verzovací systémy. Subversion byl vytvořen jako alternativa k CVS (Concurrent Version System), která měla opravovat některé jeho chyby při zachování kompatibility. [1]

Subversion pracuje napříč sítí, může tedy být využíván z více počítačů. Uživatelům dovoluje upravovat a spravovat stejné soubory dat ze svých pozic. Provedené změny jsou tedy zaznamenány rychleji a verzovány. Tedy i chráněny před případnými chybami možností se vratit k předchozí verzi. [1]

Vlastnosti subversionu:

- Verzování adresářů:

Nejen, že subversion sleduje historii jednotlivých souborů, ale také implementuje virtuální verzovaný souborový systém, který sleduje změny v celém adresářovém stromě v průběhu času. Všechny soubory a adresáře jsou opatřeny verzí.

- Atomické operace:

Zajišťují, že buď se do uložiště provedou všechny změny, nebo žádná z nich. Tohle zabraňuje problémům, které by mohli nastat, kdyby byla provedena pouze část z požadovaných změn. [1]

- Verzované metadata:

Každý soubor a adresář má sadu vlastností (klíčů a jejich hodnot). Všechny tyto vlastnosti jsou také opatřeny verzí v průběhu času, stejně jako obsah souborů. [1]

- Konzistentní zpracování dat:

Subversion zjišťuje rozdíly v souborech za použití binárního rozdílového algoritmu, který pracuje stejně pro textové i binární soubory. Oby typy souborů jsou uloženy stejně komprimované v uložišti, a rozdíly jsou přenášeny v obou směrech v rámci sítě. [1]

- Efektivní větvení a značkování:

Náklady na větvení a značkování nemusí být vždy úměrné velikosti projektu. Subversion proto vytváří větve a značky tím, že zkopíruje projekt, pomocí mechanismu podobnému k tvrdému odkazu. Čímž se tyto operace velmi zjednoduší a provadějí se velmi rychle. [1]

Výhody subversionu:

- Systém založený na CVS
- Využívá Atomické operace
- Široký výběr plug-inů pro vývojářské prostředí
- Nevyužívá modelu peer to peer
- Levné operace větvení

Nevýhody subversionu:

- Nedostatečné příkazy pro správu uložiště
- Pomalejší srovnávací rychlost

5.2 GIT

Obrázek 4 Logo verzovacího systému Git [9]

Byl vyvinut v roce 2005 zejména Linusem Torvaldsem (tvůrcem Linuxu), aby nahradil BitKeeper, který přestal být zdarma. Měl sloužit hlavně pro vývoj linuxového jádra. Hlavními požadovanými vlastnostmi systému Git byly: [2]

- Rychlost
- Jednoduchý desing
- Silná podpora nelineárního vývoje (tisíce paralelních větví)
- Plná distribuovatelnost
- Schopnost efektivně spravovat velké projekty, jako je linuxové jádro (rychlost a objem dat)

Hlavním rozdílem systému Git proti všem ostatním verzovacím systémům je způsob, jakým Git zpracovává data. Většina verzovacích systémů ukládá informace jako seznamy změn jednotlivých souborů. Tedy uložené informace jsou pro ně sadou souborů a seznamů změn v čase. Git chápe uložené informace jako sadu snímků vlastního systému souborů. Pokud v systému zapíšete (uložíte) stav projektu, Git „vyfotí“ jak vypadají všechny soubory v daném okamžiku, a uloží reference na tento snímek. Soubory, ve kterých nebyly provedeny změny nejsou znovu ukládány, ale uloží se pouze odkaz na předchozí soubor, který je identický se souborem v nové verzi. [2]

Jelikož je Git distribuovaným verzovacím systémem tak většina jeho operací vyžaduje pouze lokální soubory k činnosti. Například pro procházení historie projektu nemusí Git vyhledávat informace na serveru, ale načte je z lokálního uložiště uživatele. Díky využívání lokalního uložiště je jen velmi málo operací, které Git nemůže provadět offline nebo bez připojení k VPN. Umožňuje tedy i práci offline a při následujícím připojení se k síti proběhne synchronizace dat. Jako ochranu před ztrátou nebo poškozením dat provádí systém před každým uložením dat kontrolní součet, který je následně používán pro identifikaci dané operace. Není tedy možné změnit obsah jakéhokoli souboru nebo adresáře, aniž by to Git nevěděl. Mechanismus, který zajišťuje ochranu dat pomocí

kontrolního součtu se nazývá SHA-1. Jedná se o řetězec 40 hexadecimálních znaků (0-9; a-f) vypočítaný na základě obsahu souboru nebo adresářové struktury systému. [2]

Pro spravované soubory využívá Git tři základní stavy: [2]

- Zapsáno (committed) – data jsou bezpečně uložena v lokální databázi.
- Změněno (modified) – v souboru byly provedeny změny, ale prozatím ještě nebyl zapsán do databáze.
- Připraveno k zapsání (staged) – změněný soubor v jeho aktuální verzi je určen k zapsání v další revizi.

Výhody Gitu:

- Velké zvýšení rychlosti
- Levné operace větvení
- Lokální databáze – možnost práce offline

Nevýhody Gitu:

- Omezená podpora ve Windows oproti Linuxu
- Nutné zaučení pro ty, kteří používali SVN
- Pro jednotlivé vývojáře není optimální

5.2.1 Git Extension

Je sada nástrojů zaměřená pro intuitivní práci s Gitem k dispozici pro operační systémy Windows, Linux a Macintosh OS X. K dispozici je také Visual Studio plug-in pro přímé použití Gitu z vývojářského prostředí.

Funkce:

- Integrace Gitu do průzkumníku Windows
- Podpora plug-inu pro Visual Studio 2005 a vyšší
- Bohaté uživatelské rozhraní
- Instalační balíček zahrnuje Git, Git Extensions a nástroj pro slučování Kdiff3 (podpora 32bit i 64bit systému).

5.3 Mercurial

Obrázek 5 Logo verzovacího systému Mercurial [10]

Mercurial má vyjímečnou sadu vlastností, díky kterým je dobrou volbou mezi verzovacími systémy a patří mezi distribuované systémy. Je jeden z nejjednodušších verzovacích systémů na ovládání a naučení se s ním. Jeho příkazy a vlastnosti jsou jednotné a konzistentní, takže můžete sledovat jen několik obecných pravidel namísto řady vyjímek. S Mercurial můžete začít pracovat na nějakém menším projektu za pár chvil, protože základní operace jako vytvoření, změny větve, přenos změn (ať už lokálně nebo přes síť), historie a stav jsou velmi rychlé. Užitečnost Mercurialu není omezena pouze na malé projekty. Je používán i u projektů, které mají stovky nebo tisíce uživatelů obsahující desítky tisíc souborů a stovky megabajtů dat. [5]

Výhody Mercurialu:

- Snadnější ovládání než u Gitu
- Lepší dokumentace
- Distribuovaný model

Nevýhody Mercurialu:

- Nepodporuje sloučení rodičů
- Založeno spíše na rozšířeni než skriptovatelnosti

5.4 Fossil

Obrázek 6 Logo verzovacího systému Fossil[17]

Fossil je jednoduchý distribuovaný verzovací systém pro správu souborů s vysokou spolehlivostí a následujícími vlastnostmi. Obsahuje integrované sledování chyb, wiki a technické poznámky. Integrované webové rozhraní, které je intuitivní s bohatou škálou informačních stránek včetně příkladů. Nepoužívá žádné vlastní protokoly nebo TCP porty. Pro síťovou komunikaci využívá HTTP, HTTPS nebo SSH, takže funguje napříč firewallů a proxy serverů. Je robustní a spolehlivý, obsah ukládá do SQLite databáze.[17]

5.5 Bazaar

Obrázek 7 Logo verzovacího systému Bazaar[15]

Autorem verzovacího systému Bazaar (zkráceně Bzr) je Martin Pool a jeho vývoj je sponzorován firmou Canonical. [15]

Mezi jeho hlavní výhody patří:[15]

- Jednoduchost a snadné použití: Propracované cross-platform GUI, příkazová řádka s online pomocí pro každý příkaz.
- Práce offline: Jako pro každý distribuovaný verzovací systém je velkou výhodou práce bez připojení ke vzdálenému serveru
- WorkFlow: I když je Bazaar distribuovaným systémem tak při přechodu z projektu pod centralizovanou správou umožňuje takto pokračovat. Přechod pak může být zaváděn postupně.

6 VERZOVACÍ SYSTÉM JAKO PLUGIN

Význam verzovacích systémů je ve výpočetní technice tak velký, že se objevují i jako zásuvné moduly do jiných programů. Zásuvný modul neboli plugin rozšiřuje funkci softwaru, do kterého je nainstalován a nedokáže fungovat samostatně.

6.1 VersionPress

VersionPress je zásuvným modulem pro systém WordPress, který zajišťuje správu webového obsahu. Hlavním úkolem VersionPressu je jakoukoliv změnu na webu fungujícím pod WordPressem uložit. Změny ukádá formou commitů do systému Git, ktery běží pod VersionPressem. Ukládání neprobíhá pouze na úrovni souborů, ale i databáze. [18]

Funkce VersionPressu:

- Sledování změn: Všechny provedené změny jsou zapsány a zobrazeny v přehledné tabulce obsahující datum,čas a název provedené změny.[19][18]
- Možnost vrácení změn: Pokud je potřeba vrátit některou z dříve provedených změn stačí jedno kliknutí v tabulce přehledu. [19][18]
- Zálohování: Inkrementální zálohování, které ukládá jen provedené změny a historii změn zajistí, že velikost zálohovaných souborů bude co nejmenší. [19][18]
- Práce s větvením: VersionPress také podporuje práci s větvemi. Je tedy možné začít vyvíjet novou verzi stávajícího webu pomocí vytvoření nové větve a v ní provádět všechny potřebné úpravy. I když v průběhu času přibudou na stránce nové články, komentáře apod. stačí provést operaci Pull z aktuální verze webu a spojit ji s novou pomocí operace Merge. [19][18]

7 POPIS POUŽITÝCH TECHNOLOGIÍ

K vizualizaci vzniku, sloučení a evidence verzí byly použity technologie jQuery, HTML(HyperText Markup Language), CSS(Cascading Style Sheets) a BootsTrap.

7.1 jQuery

Obrázek 8 Logo knihovny jQuery [21]

jQuery je javascriptová knihovna pro poskytnutí víceúčelové abstraktní vrstvy pro webové skriptování. Je tedy ideálním pomocníkem v každé situaci, ve které je potřeba skriptovat. Klade velký důraz na rychlost, jednoduchost a čitelnost. jQuery knihovna je dostupná zdarma ke stažení a je multiplatformní.[20][21]

Proč si zvolit knihovnu jQuery pro práci? Využíva znalosti jazyka kaskádových stylů a pracuje s jeho selektory. Je rozšiřitelná, existuje velké množství zasuvných modulů a umožňuje tvorbu nových podle vlastních potřeb. Jako abstraktní vrstva sjednocuje běžné úkoly díky čemuž zjednodušuje a zmenšuje zdrojový kód. Snaží se eliminovat rozdíly mezi různými webovými prohlížeči a tím umožňuje používat stejné funkce na různých platformách. [20][21]

Základní funkce knihovny jQuery pomáhájí s následjícími úlohami: [20]

- Vybrat objekty DOM(Document Object Model): Díky knihovnám javascriptu nemusí vývojáři psát velké množství kódu, aby prošli strom modelu DOM a našli konkrétní část dokumentu HTML a pracovali s ní. Knihovna jQuery obsahuje velké množství selektorů, kterou mají vývojáři k dispozici. Selektory představují jednoduchý způsob jak získat určitou část dokumentu, bez zkoumaní či manipulování s modelem DOM.
  Ukázka: $('div.content').find('p') [20]
- Upravovat vzhled webové stránky: Jazyk Kaskádových stylů (CSS) je sice dobrým způsobem pro definování a změny vzhledu webové stránky, ale jeho velkým nedostatkem je, že všechny webové prohlížeče nepodporují standardy

stejně. Díky knihovně jQuery je možné tento nedostatek eliminovat a spoléhat se na stejné standardy napříč všemi webovými prohlížeči. Nespornou výhodou je také možnost měnit třídy a jednotlivé vlastnosti na různých částech dokumentu.

Ukázka: $('li > p:first').removeClass('active') [20]

- Měnit obsah modelu DOM: Přehledné rozhraní API (Application Programming Interface) umožňuje například měnit text, seřazovat seznamy, vkládat nebo zaměňovat obrázky nebo i přepisovat a rozšiřovat celou strukturu dokumentu HTML.
  Ukázka: $('#container').append('více') [20]
- Reagovat na akce: Knihovna také nabízí způsob jak snadno reagovat na události na webu jako například klepnutí na odkaz bez „znečišťování“ kódu obsluhující funkcí. Knihovna také odstraňuje rozdíly pro obsluhu událostí mezi různými webovými prohlížeči pomocí rozhraní API.
  Ukázka:
  ```
  $('button.show-details').click(function(){
      $('div.details').show();
  });[20]
  ```
- Animace: Knihovna nabizí množství efektů a také nástroje pro tvorbu nových vizuálních efektů. Pokud je požadováno nějaké interaktivní chování, je nutné poskytnout také vizuální odezvu například posouvání nebo prosvítání.
  Ukázka: $('div.details').slideDown();[20]
- Načítaní dat ze serveru bez nutnosti obnovení webové stránky: tuto funkci zajišťuje technologie Ajax (Asynchronous JavaScript and XML) tedy asynchronní javascript a xml. Postupem času se ale tato technologie rozrostla a v dnešní době reprezentuje větší skupinu technologií pro komunikaci mezi klientem a serverem. Tak jako u všech jiných funkcí i zde knihovna jQuery eliminuje složitost související se specifickými odchylami jednotlivých prohlížečů.
  Ukázka: $('div.details').load('more.html #content'); [20]
- Zjednodušování: Velkou mírou zjednodušuje programování v jazyce JavaScript. Knihovna nepřidává jen nové funkce, ale rozšiřuje základní konstrukci jazyka JavaScript. Například procházení a manipulaci s poli.
  Ukázka:
  ```
  $.each(obj, function(key, value){
      Total += value;   }); [20]
  ```

Selektory jsou hlavním důvodem, proč je knihovna jQuery tak oblíbená. Selektor je textový výraz, díky kterému můžeme vyhledávát a upravovat elementy v dokumentu.

7.1.1 Selektory knihovny jQuery:

- ("*") – vybere všechny elementy dokumentu.[22]
- (":animated") – vybere všechny elementy u kterých je aktivní animace v době spuštění selektoru[22]
- ("[attribute|='value']") – vybere elementy, které mají zadaný atribut s hodnotou buď rovnou nebo začínající tímto řetězcem, následovaným pomlčkou (-). [22]
- ("[attribute*='value']") – vybere elementy, které mají zadaný atribut s hodnotou obsahující danému řetězci. [22]
- ("[attribute~='value']") – vybere elementy, které mají zadaný atribut s hodnotou obsahující daný řetězec oddělený mezerami. [22]
- ("[attribute$='value']") – vybere elementy, které mají zadaný atribut s hodnotou končící přesně daným řetězcem. Při srovnání se rozlišují velké a malé písmena. [22]
- ("[attribute='value']") – vybere elementy, které mají zadaný atribut s hodnotou přesně rovnou zadanému řetězci. [22]
- ("[attribute!='value']") – vybere elementy, které buď nemají zadaný atribut, nebo mají zadaný atribut, ale neshoduje se se zadaným řetězcem. [22]
- ("[attribute^='value']") – vybere elementy, které mají zadaný atribut začínající přesně zadaným řetězcem. [22]
- (":button") – vybere všechny elementy button. [22]
- (":checkbox") – vybere všechny elementy, které jsou zaškrtnuty nebo vybrány. [22]
- ("parent > child") – vybere všechny přímé potomky zadaného rodiče. [22]
- (".class") – vybere všechny elementy se zadanou třídou. [22]
- (":contains(text)") – vybere všechny elementy obsahující zadaný text. [22]
- ("ancestor descendant") – vybere všechny elemety, které jsou potomky zadaného předka. [22]
- (":disabled") – vybere všechny elementy, které jsou ve stavu disabled. [22]
- ("element") – vybere všechny elementy se zadaným názvem tagu. [22]
- (":empty") – vybere všechny elementy, které nemají potomky. [22]
- (":enabled") – vybere všechny elementy, které jsou ve stavu enabled. [22]

- (":eq(index)") – vybere element v indexu n v seskupené množině. [22]
- (":even") – vybere i prvky s nulovým indexem. [22]
- (":file") – vybere všechny elementy typu file. [22]
- (":first-child") – vybere všechny elementy, které jsou prvními potomky rodičů. [22]
- (":first-of-type") – vybere všechny elementy, které jsou první mezi sourozenci stejného názvu elementu. [22]
- (":first") – vybere první odpovídající element z DOMu. [22]
- (":focus") – vybere element pokud je aktuálně „zaměřen“. [22]
- (":gt(index)") – vybere všechny elementy v indexu, který je větší než index uvnitř seskupené řady. [22]
- ("[attribute]") – vybere elementy, podle zadaného atributu bez odhledu na jejich hodnotu. [22]
- (":has(selector)") – vybere elementy, které obsahují alespoň jeden element, který odpovídá zadanému selektoru. [22]
- (":header") – vybere všechny elementy, které jsou jako záhlaví (h1, h2 atd.). [22]
- (":hidden") – vybere všechny skryté elementy. [22]
- ("#id") – vybere element s daným ID atributu. [22]
- (":image") – vybere všechny elementy typu image. [22]
- (":input") – vybere všechny vstupni elementy: input, textarea, button, select. [22]
- (":lang(language)") – vybere všechny elementy zadaného jazyka. [22]
- (":last-child") – vybere všechny elementy, které jsou posledním potomkem rodiče. [22]
- (":last-of-type") – vybere všechny elementy, které jsou poslední mezi sourozenci stejného názvu elementu. [22]
- (":last") – vybere poslední odpovídající element. [22]
- (":lt(index)") - vybere všechny elementy v indexu, který je menší než index uvnitř seskupené řady. [22]
- ("[attributeFilter1][attributeFilter2][attributeFilterN]") – vybere elementy, které se ovpovídají všem zadaným filtrům atributů. [22]

- ("selector1, selector2, selectorN") – vybere všechny kombinované výsledky zadaných selektorů. [22]
- ("prev + next") – vybere všechny další elementy odpovídající hodnotě "next", které jsou předcházeny sourozenci "prev".[22]
- ("prev ~ siblings") – vybere všechny elementy sourozence, které následují po elementu "prev", mají stejného rodiče a odpovídají filtrování na "siblings".[22]
- (":not(selector)") – vybere všechny elementy, které neodpovídají zadanému selektoru. [22]
- (":odd") – vybere liché elementy s nulovým indexem. [22]
- (":only-child") – vybere všechny elementy, které jsou jediným potomkem jejich rodiče. [22]
- (":only-of-type") – vybere všechny elementy, které mají sourozence se stejným názvem elementu. [22]
- (":parent") – vybere všechny elementy, které mají alespoň jednoho potomka. [22]
- (":password") – vybere všechny elementy typu "password".[22]
- (":radio") – vybere všechny elementy typu "radio".[22]
- (":reset") – vybere všechny elementy typu "reset".[22]
- (":root") – vybere element, který je kořenem dokumentu. [22]
- (":selected") – vybere všechny vybrané elementy. [22]
- (":submit") – vybere všechny elementy typu "submit".[22]
- (":target") – vybere cílový element označený identifikátorem fragmentu URI dokumentu. [22]
- (":text") – vybere všechny elementy typu "input".[22]
- (":visible") – vybere všechny viditelné elementy. [22]

7.2 BootsTrap

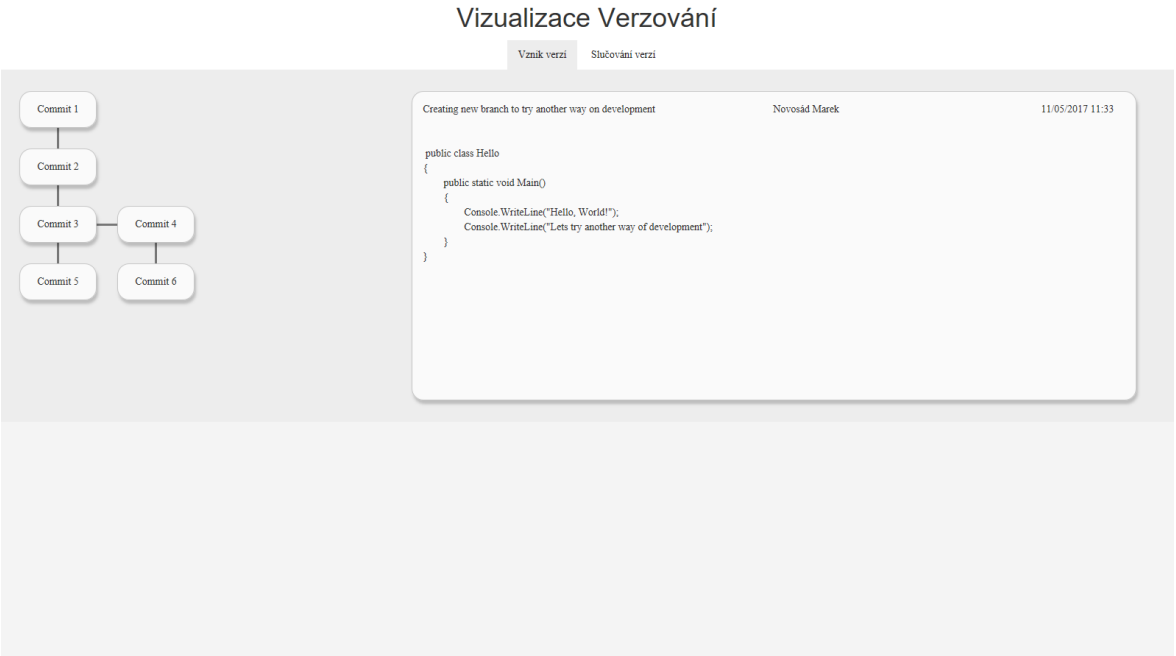
Obrázek 9 Logo frameworku BootsTrap [23]

BootsTrap je framework, který obsahuje kolekci nástrojů pro tvorbu a úpravu vzhledu webových stránek a aplikací. Nespornou výhodou tohoto frameworku je, že do projektu ve kterém je použit přidává responsivitu. Responzivita je způsob stylování zaručující, že bude zobrazení webové stránky optimalizované pro všechny druhy a velikosti nejrůznějších zařízení (tablety, mobilní telefony, notebooky apod.). [23] [24]

II. PRAKTICKÁ ČÁST

8 VIZUALIZACE VERZOVÁNÍ

Hlavní myšlenkou této praktické části bakalářské práce je znázornit a vysvětlit procesy verzování. Konkrétně jsou vizualizovány případy větvení a slučovaní včetně ukázkových dat. Projekt je vytvořen pomocí technologií HTML, CSS, jQuery a BootsTrap.



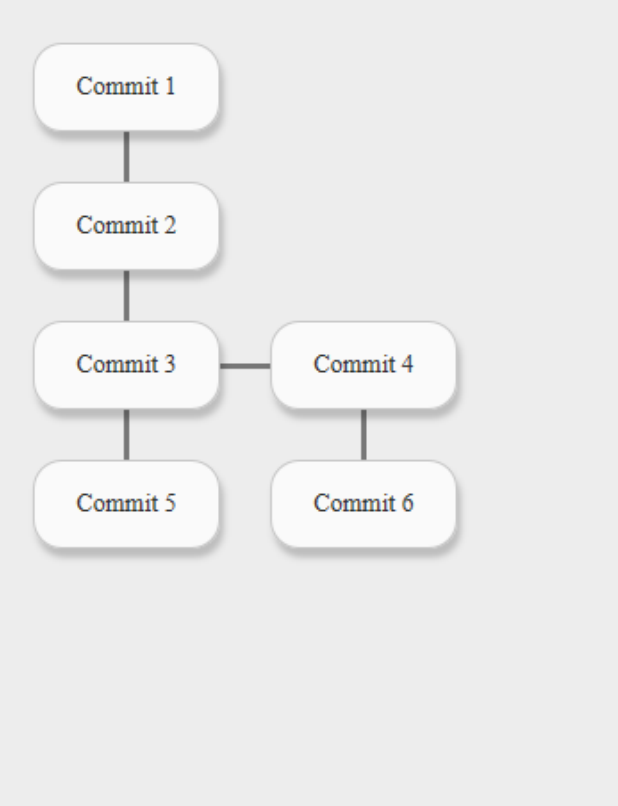
Obrázek 10 zobrazení vizualizace verzování

Na obrázku 10. je vidět základní rozložení stránky zobrazující vizualizaci verzování. Na levé straně je vytvořen strom z jednotlivých commitů. Na pravé straně pak data které sebou commit nese. V prvním řádku se zobrazuje název commitu pro snadnou identifikaci následuje jméno autora commitu a jako poslední je zobrazen čas kdy se změna provedla. Pod těmito informacemi je místo pro zobrazení dat a změn, které se udály v dokumentu.

Data obsažená v commitech jsou pouze ukázkové z tohoto důvodu byly zvoleny jednoduché texty, na kterých je poměrně jednoduché pochopit princip fungovaní verzovacích systémů.

9 POPIS VZNIKU VERZÍ

Pro vizualizaci vzniku verzí jsem zvolil jednoduchou ukázku o pár commitech na kterých si postupně vysvětlíme fungování větvení. Změny, které proběhly v po sobě jdoucích commitech budou vždy stručně popsány a zvýrazněny.



Obrázek 11 strom vzniku verzí

Commit 1: Creating new project and pushing to server

První commit je zároveň i vytvořením dokumentu a první verze na serveru, který je pod správou verzovacího systému. Jelikož se tímto commitem projekt zakládá bude to pro něj tedy i hlavní větev. V našem případě obsahuje první commit tyto data:

```
+ public class Hello
+ {
+         public static void Main()
+         {
+                 Console.WriteLine("Hello, World!");
+         }
```

```
+ }
```

Commit 2: Adding new text line in console

Druhý commit v sobě nese již změnu nebo úpravu vzniklého projektu. Na obrázku 11. je vidět, že se jedná o úpravu nebo změnu v hlavní větvi. Data commitu 2 jsou následující:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
+       Console.WriteLine("This is new added code for commit");
    }
}
```

Commit 3: Adding another line in console

Třetí commit následuje druhý commit v hlavní větvi a přidává další výpis do konzole. Data commitu 3:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
        Console.WriteLine("This is new added code for commit");
+       Console.WriteLine("This new line represents another commit");
    }
}
```

Commit 4: Creating new branch to try another way on development

Při čtvrtém commitu již dochází k rozvětvení projektu na hlavní a jednu vedlejší větev. Změny provedené ve vedlejší větvi nemají žádný vliv na hlavní větev. V commitu dochází ke smazaní provedených změn v commitu 2 a 3, které můžou být nahrazeny novým zpusobem řešení funkce nebo jen opravou dané funkce.Data commitu 4:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
-       Console.WriteLine("This is new added code for commit");
-       Console.WriteLine("This new line represents another commit");
+       Console.WriteLine("Lets try another way of development");
    }
}
```

Commit 5: Here can continue development on master branch

Pátý commit je opět v hlavní větvi a nijak neovlivňuje větev vedlejší. Může na ní nadále pokračovat vývoj. Data commitu 5:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
        Console.WriteLine("This is new added code for commit");
        Console.WriteLine("This new line represents another commit");
+       Console.WriteLine("In this branch we can continue with working on project
+       or fixing some bugs");
```

```
        }
}
```

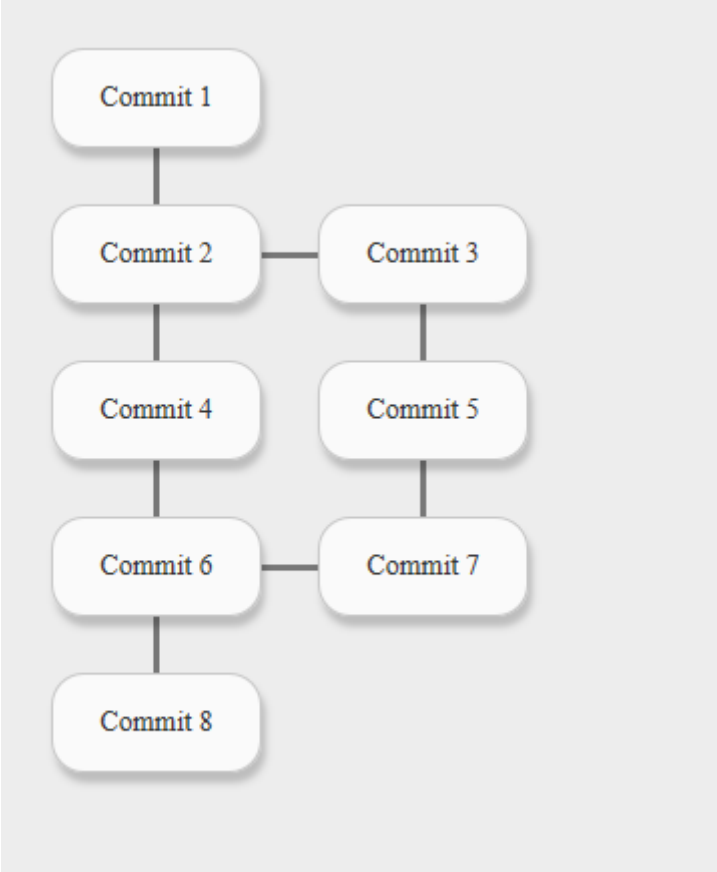
Commit 6: New development without influence of master branch

Šestý commit představuje druhou úpravu, která nadále rozšiřuje vedlejší větev. Data commitu 6 jsou následující:

```
public class Hello
{
        public static void Main()
        {
                Console.WriteLine("Hello, World!");
                Console.WriteLine("Lets try another way of development");
+               Console.WriteLine("New way of development can continue in this
                branch");
        }
}
```

10 POPIS SLOUČENÍ VERZÍ

Pro vizualizaci sloučení verzí jsem zvolil ukázku, která obsahuje zároveň i rozvětvení projektu. Na ukázce bude stručně vysvětleno fungovaní této operace včetně možných potíží při slučování verzí.



Obrázek 12 strom vzniku a sloučení verzí.

Commit 1: Creating new project and pushing to server

První commit je zároveň i vytvořením dokumentu a první verze na serveru, který je pod správou verzovacího systému. Jelikož se tímto commitem projekt zakládá bude to pro něj tedy i hlavní větev. V našem případě obsahuje první commit tyto data:

```
+ public class Hello
+ {
+         public static void Main()
+         {
+                 Console.WriteLine("Hello, World!");
```

```
+        }
+ }
```

Commit 2: Adding new text line in console

Druhý commit v sobě nese již změnu nebo úpravu vzniklého projektu. Na obrázku 11. je vidět, že se jedná o úpravu nebo změnu v hlavní větvi. Data commitu 2 jsou následující:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
+       Console.WriteLine("This is new added code for commit");
    }
}
```

Commit 3: Created new branch for adding new feature

Ve třetím commitu již dochází k rozvětvení projektu na hlavní a jednu vedlejší větev. V commitu dochází k přidání dalších dat oproti hlavní větvi. Data commitu 3:

```
public class Hello
{
        public static void Main()
        {
                Console.WriteLine("Hello, World!");
                Console.WriteLine("This is new added code for commit");
+               Console.WriteLine("Developer adding new feature to project in own
                branch");
        }
}
```

Commit 4: Adding or changing code in master branch

Čtvrtý commit navazuje na commit 2 a rozšiřuje tak hlavní větev o nová data. Data commitu 4:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
        Console.WriteLine("This is new added code for commit");
+       Console.WriteLine("Developer continue in his work on project");
    }
}
```

Commit 5: Adding new feature to own branch

V pátem commitu dochází k rozšíření vedlejší větve o další změny nebo úpravy. Data commitu 5 jsou následující:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
        Console.WriteLine("This is new added code for commit");
        Console.WriteLine("Developer adding new feature to project in own branch");
+       Console.WriteLine("Here is implemented new feature by developer in his branch");
    }
```

```
}
```

Commit 7: Adding new feature to own branch 2

Sedmý commit je posledním commitem do vedlejší větve. Tedy poslední udělaná úprava před sloučením s hlavním větví. Data commitu 7:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
        Console.WriteLine("This is new added code for commit");
        Console.WriteLine("Developer adding new feature to project in own branch");
        Console.WriteLine("Here is implemented new feature by developer in his branch");
+       Console.WriteLine("Finishing work on new feature");
    }
}
```

Commit 6: Merging branches

Šestý commit provádí samotné sloučení obou větví. V tomto případě je nutné si dát pozor na možnost vzniku takzvaného konfliktu. Ten vzniká pokud se na dvou větvích stejného projektu provádí změny na stejném místě. Data commitu 6:

```
public class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World!");
        Console.WriteLine("This is new added code for commit");
```

```
            Console.WriteLine("Developer continue in his work on project");
+           Console.WriteLine("Developer adding new feature to project in own
            branch");
+           Console.WriteLine("Here is implemented new feature by developer in his
            branch");
+           Console.WriteLine("Finishing work on new feature");
        }
}
```

V našem případě by tedy ke konfliktu došlo, protože byl kód upravován v obou větvích na stejném místě. Ve většině případů při vzniku konfliktu je nutný zásah uživatele, protože verzovací systém nedokáže určit co se má stát s daty které se „překrývají“. V tomto případě byl konflikt vyřešen jednoduchým posunutím všech změn z vedlejší větve pod změny v hlavní větvi.

Commit 8: Adding comments to code

Osmý commit znázorňuje pokračovaní v práci na hlavní větvi která již obsahuje sloučená data z vedlejší větve. Data commitu 8:

```
public class Hello
{
    public static void Main()
    {
+           //Vizualizace Verzování Novosád Marek
            Console.WriteLine("Hello, World!");
            Console.WriteLine("This is new added code for commit");
            Console.WriteLine("Developer continue in his work on project");
            Console.WriteLine("Developer adding new feature to project in own
            branch");
            Console.WriteLine("Here is implemented new feature by developer in his
            branch");
```

```
                Console.WriteLine("Finishing work on new feature");
        }
}
```

ZÁVĚR

Cílem této práce bylo popsat aktuální stav verzovacích systémů, popsat a vysvětlit základní principy verzování, porovnat a popsat nejpoužívanější systémy a vytvořit vizualizaci vzniku a slučování verzí.

V teoretické části je popsána historie verzovacích systémů od doby vzniku prvního verzovacího systému, až po současný stav. Dále je vysvětleno základní rozdělení verzovacích systémů včetně jejich výhod a nevýhod. V popisu základních principů verzování jsou popsány a vysvětleny základní operace pro práci s verzovacím systémem. V práci jsou také zmíněny nejpoužívanější verzovací systémy, každý je popsán a zmíněny jsou výhody a nevýhody konkrétních systémů. Speciální zmínka v práci patří větvení, jelikož se jedná o jednu z nejdůležitějších funkcí. Dále jsou popsány technologie jQuery a BootsTrap ,které byly využity při tvorbě praktické části.

Praktická část obsahuje vizualizaci slučování a vytváření verzí. Jsou zde popsány konkrétní data použité pro znázornění principů verzování. Každá operace je zvýrazněna a jsou použita základní data pro snadné pochopení fungování verzovacích systémů.

SEZNAM POUŽITÉ LITERATURY

[1] COLLINS-SUSSMAN, Ben, Brian W FITZPATRICK a C PILATO. Version control with Subversion. 1st ed. Sebastopol, CA: O'Reilly Media, 2004, xvii, 299 p. ISBN 0596004486.

[2] CHACON, Scott. Pro Git. Praha: CZ.NIC, 2009, 263 s. CZ.NIC. ISBN 978-80-904248-1-4.

[3] FOGEL, Karl a Moshe BAR. Open source development with CVS. 2nd ed. Scottsdale, AZ: Coriolis Group Books, 2001, xxi, 345 p. ISBN 078-8581001738.

[4] NAGEL, William A. Subversion version control: using the Subversion version control system in development projects. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005, xxii, 343 p. ISBN 0131855182.

[5] O'SULLIVAN, Bryan. Mercurial: The Definitive Guide. 1st ed. United States of America: O'Reilly Media, 2009. ISBN 978-0596800673.

[6] Distribuované verzovací systémy. ABC linuxu [online]. [cit. 2016-04-12]. Dostupné z: http://www.abclinuxu.cz/clanky/distribuovane-verzovaci-systemy-uvod-1

[7] SINK, Eric. Version control by example. 1st ed. Champaign, Ill.: Pyrenean Gold Press, 2011. ISBN 9780983507901.

[8] Subversion. Apache Subversion [online]. [cit. 2016-05-16]. Dostupné z: https://subversion.apache.org/

[9] Git [online]. [cit. 2016-05-16]. Dostupné z: https://git-scm.com/

[10] Mercurial [online]. [cit. 2016-05-16]. Dostupné z: https://www.mercurial-scm.org/

[11] Source Code Control System [online]. [cit. 2017-04-02]. Dostupné z: http://sccs.sourceforge.net/

[12] Revision Control System [online]. [cit. 2017-04-02]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=897

[13] Concurrent Version System [online]. [cit. 2017-04-02]. Dostupné z: http://www.nongnu.org/cvs/

[14] GNU Arch [online]. [cit. 2017-04-02]. Dostupné z: https://www.gnu.org/software/gnu-arch/

[15] Bazaar [online]. [cit. 2017-04-02]. Dostupné z: http://bazaar.canonical.com/en/

[16] Darcs [online]. [cit. 2017-04-02]. Dostupné z: http://darcs.net/FrontPage

[17] Fossil [online]. [cit. 2017-04-02]. Dostupné z: https://www.fossil-scm.org/index.html/doc/trunk/www/index.wiki

[18] Version Press [online]. [cit. 2017-05-08]. Dostupné z: https://docs.versionpress.net/en

[19] Jan Bien [online]. [cit. 2017-05-08]. Dostupné z: https://www.janbien.cz/borek-bernard/

[20] CHAFFER, Jonathan a Karl SWEDBERG. Mistrovství v jQuery: [kompletní průvodce vývojáře]. Brno: Computer Press, 2013. Mistrovství. ISBN 978-80-251-4103-8.

[21] JQuery [online]. [cit. 2017-05-21]. Dostupné z: http://jquery.com/

[22] JQuery: Selectors [online]. [cit. 2017-05-21]. Dostupné z: http://api.jquery.com/category/selectors/

[23] BootsTrap [online]. [cit. 2017-05-21]. Dostupné z: https://www.ambitas.sk/blog/svet-ambitas/bootstrap-framework-a-jeho-variabilita

[24] BootsTrap [online]. [cit. 2017-05-21]. Dostupné z: http://getbootstrap.com/getting-started/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SCCS Source Code Control System.

RCS Revision Control System.

CVS Concurrent Version System.

SVN Subversion.

CVCS Centralized Version Control System.

DVCS Distributed Version Control System.

VCS Version Control System.

VPN Virtual Private Network.

SHA-1 Secure Hash Standard.

TCP Transmission Control Protocol.

HTTP HyperText Transfer Protocol.

HTTPS HyperText Transfer Protocol Secure.

SSH Secure Shell.

GUI Graphical User Interface.

HTML HyperText Markup Language.

CSS Cascading Style Sheets.

DOM Document Object Model.

API Application Programming Interface

XML eXtensible Markup Language

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

P I CD obsahující bakalářskou práci a projekt Vizualizace Verzování