MANAŽMENT VERZIÍ ALEBO JEDNA VERZIA NIKDY NESTAČÍ

Prvá verzia
základ vývoja.

Juraj Spusta

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
spustaj[zavináč]gmail[.]com

Abstrakt. *Životný cyklus vývoja softvérového projektu prebieha vždy v niekoľkých etapách alebo cykloch. Pri každom kroku vývoja je preto potrebné manažovať exitujúce zdrojové súbory a dokumenty, ktoré sa pri procese tvorby systému používajú. Je potrebné uchovávať tieto súbory tak, aby bolo možné riešiť prípadné kolízie, vrátiť sa k predošlým verziám projektu, alebo monitorovať priebeh celého procesu. Na uľahčenie tejto práce sa v posledných dvoch dekádach používajú systémy pre manažment verzií a zmien v projektoch. V tejto eseji sa zaoberám popisom týchto systémov, spôsobom ich použitia a výhodami, ktoré prinášajú vývojárom pri práci a komunikácií v rámci tímu. Zároveň však poukazujem aj na problémové situácie, ktoré môžu nastať pri ich používaní.*

Kľúčové slová: *CVS, SVN, version management system*

Úvod

Či už sa pozeráme na problematiku vývoja softvérov alebo softvérových systémov z pohľadu študentských zadaní na školách alebo veľkých firemných projektov, je zrejmé, že manažment používaných súborov, či už zdrojových kódov alebo projektových dokumentácií, je dôležitým prvkom pri dosiahnutí úspechu. Aj keď už asi dvadsať rokov sa vyvíjajú a zároveň používajú podporné prostriedky, ktoré by som chcel predstaviť v tejto eseji, veľa vývojárov sa s nimi stretne až v prvej pracovnej pozícií a časť z nich ich vôbec nepoužíva. Týmto podporným prostriedkom mám na mysli version management system (*vms*). V ďalších častiach tejto eseje sa budem venovať definovaniu týchto

prostriedkov z pohľadu funkcionality a použiteľnosti, rozoberiem jednotlivé výhody a nevýhody a porovnám centralizovaný prístup s distribuovaným prístupom. Na záver načrtnem možné výhody pri použití v rámci výučby na niektorých univerzitách.

Popis a funkcionalita

Jedným z najznámejších a zároveň najstarších *vms* je Concurrent Versions System (CVS). V poslednom desaťročí vzniklo veľa ďalších druhov podobných systémov, ale pri nasledujúcom popise a charakteristikách budem vychádzať z CVS.

Základným architektonickým prvkom CVS je prístup klient – server, inak nazývaný centralizovaný prístup. Pripojenie na server je možné pomocou LAN ale aj pomocou Internetu, z čoho vyplýva, že práca na projekte alebo zadaní je možná z hociktorého miesta, ktoré toto pripojenie poskytuje a používateľ nie je viazaný len na jedno miesto výkonu prace.

Server slúži ako záloha dát - takzvaný depozitár, kde sú uložené všetky súbory, ktoré používateľ potrebuje pri práci. Zriadenie servera je jednoduché a prístup doň môže byť umožnený viacerým používateľom súčasne, čím sú tieto systémy priamo predurčené pre prácu v malých ale aj veľkých tímoch.

Čo sa týka pohľadu klientskej časti, bolo už spomenuté, že miesto pripojenia a tým pádom aj práca so súbormi je možná kdekoľvek. Pred samotným pripojením je nutné, aby používateľ vytvoril lokálny depozitár na svojom počítači. V tomto adresári sa potom budú zobrazovať súbory, s ktorými bude pracovať.

Prvou a hlavnou úlohou CVS je vedenie histórie všetkých uložených súborov, to znamená každej zmeny, ktorá bola vykonaná na jednotlivých súboroch a bola potvrdená na serveri. Táto funkcia zabezpečuje, že je kedykoľvek možný návrat k predchádzajúcej verzii. Často sa totiž stáva, že nastane podstatná zmena v súbore so zdrojovým kódom, ale po nejakom čase zistíme, že zmena nebola potrebná alebo dané riešenie bolo chybné. V takom prípade, pokiaľ si zálohujeme všetky verzie súboru osobitne, by sme museli nájsť daný súbor, kde ešte nebola daná zmena vykonaná. V prípade, že ale prepisujeme stále rovnaký súbor by mohla táto chyba viesť k strate nášho úsilia a vynaloženého času. Ak ale používame CVS a pri každej významnej zmene uložíme verziu na server, je jednoduché kedykoľvek nájsť a vrátiť sa k staršej verzii. Väčší poriadok v rámci verzií môžeme dosiahnuť písaním logov ku každej zmene, ktorú potvrdíme na server. Zároveň sa tak dá sledovať aj priebeh práce na projekte.

Ďalšou úlohou CVS je pomoc pri vývoji projektov v tímoch, respektíve v skupinách s počtom členov dva a viac. Pri práci v tíme treba totiž riešiť zložitejšie problémy ako pri jednotlivcoch. Systém CVS je zameraný primárne na implementáciu projektu. Gro funkcionality sa zameriava na problémy úpravy a konzistencie jednotlivých súborov v prípade, keď k nemu pristupujú viacerí používatelia. Tento problém by sa dal vyriešiť uzamknutím tohto súboru tak, aby ho mohla upravovať len jedna osoba v danom čase. Daný prístup je však často nežiadúci a preto je potrebné zabezpečiť možnosť práce viacerých používateľov súčasne (Concurrent). To sa dá dosiahnuť tak, že každý z vývojárov bude začínať s rovnakou verziou. Následne každý vypracuje zmeny, za ktoré bol zodpovedný. Na záver je potrebné dané zmeny postupne zlúčiť (merge). Je potrebné, aby sa vždy aktuálna verzia spojila so zmenami od jedného používateľa, čím vznikne nová

verzia. Tento krok podporuje vzájomnú komunikáciu členov tímu riešiacich rovnakú úlohu a prispieva ku konzistencii a lepšiemu pochopeniu daných zmien, ale aj samotného celku.

V nasledujúcom zozname uvedieme príklad niektorých príkazov, ktoré ponúka manažment verzii. Príkazy sú z programu, ktorý aktívne používam – TortoiseSVN.

- Add - Pridanie súboru na server, slúži vlastne na vytvorenie prvej verzie zo strany používateľa (prvú verziu je možné vytvoriť aj priamym zápisom na server)
- Delete - Odstránenie súboru zo servera
- Commit - Zapísanie aktuálnej verzie na server, pri commit-e sa väčšinou uvádza aj popis zmien vykonaných v danom súbore (log)
- Update (to revision) - Do lokálneho depozitára sa nahrá aktuálna verzia súboru, alebo verzia, ktorú si zvolíme
- Merge - zlúči novú verziu v lokálnom depozitári (vetvu - branch) s verziou umiestnenou na serveri (trunk)
- Get (Release) lock - Získanie exkluzívneho prístupu na prácu so súborom (premenovanie, zapisovanie, mazanie...), respektíve povolenie prístupu ostatným používateľom
- Diff - porovná aktuálnu verziu v lokálnom depozitári s predošlou verziou

Vzhľadom k popisu CVS by sa zdalo, že tento systém je stopercentne výhodný. Nie je to však úplné pravda, preto zhrniem jeho výhody a doplníme niektoré nevýhody tohto systému.

Výhody:

- Uchovávanie histórie na serveri, možnosť prístupu k ľubovoľnej verzii
- Zobrazenie rozdielov medzi dvoma rôznymi verziami
- Návrat ľubovoľnej časti projektu k vybranej verzii
- Podpora súčasného vývoja viacerými používateľmi a riešenie konfliktných situácií pri uplatňovaný zmien
- Podpora delenia projektu na viacej častí (vetiev) – paralelný vývoj, spájanie jednotlivých vetiev navzájom a s hlavnou vetvou
- Nezávislosť na operačnom systéme, vývojovom modeli a programovacom jazyku
- Takmer neobmedzený prístup k súborom projektu (pripojenie do Internetu)

Nevýhody:

- Náročná revízia binárnych súborov – *vms* sú primárne určené pre textové súbory
- Riziko nekonzistentnosti projektu, alebo straty dát pri nesprávnom používaní príkazu *merge* a nedostatočnej komunikácií v rámci tímu

Centralizovaný vs. distribuovaný prístup

Doteraz spomínaný CVS je typickým predstaviteľom centralizovaného prístupu k manažmentu verzii. Medzičasom vznikla aj vylepšená verzia tejto aplikácie s názvom

Subversion (SVN). Ja používam pri práci TortoiseSVN. Všetky však patria do rovnakej kategórie.

V poslednej dekáde sa začal rozvíjať aj nový druh *vms* s distribuovaným prístupom. Jednými z najznámejších systémov tohto druhu sú Git a Mercurial. Hlavným rozdielom je architektúra peer-to-peer, kde používatelia majú každý vo svojom depozitári históriu celého projektu a zmeny komunikujú priamo medzi sebou namiesto so serverom.

Systém vetvenia, spájania vetiev a následného publikovania zmien je ďalším z výrazných rozdielov. Pri použití SVN vytvoríme novú vetvu, na ktorej môže pracovať viac ľudí súčasne. Každý osobitne vytvára vlastnú verziu, ktorú bude neskôr chcieť spojiť do jedného celku. Ako príklad uvediem nasledujúcu situáciu: Sú dvaja vývojári, pričom každý začal s rovnakou verziou 1. Prvý z nich vytvorí novú verziu a spojí (merge) ju s hlavnou vetvou, pričom tak vznikne verzia 2. Ak chce teraz druhý vývojár publikovať svoje zmeny musí najprv svoju prácu spojiť s verziou 2 a až potom môže danú verziu publikovať na serveri ako číslo 3. Tu nastáva riziko, že pri druhom spájaní dôjde ku kolízii alebo môže dôjsť k tomu, že výsledná verzia bude nepoužiteľná, čím dôjde o svoju prácu. Preto pri používaní SVN je dôležité, aby používatelia pracujúci s rovnakými súbormi medzi sebou komunikovali zmeny, ktoré vykonávajú.

Pri použití systémov Git alebo Mercurial má každý vývojár lokálnu kópiu celého projektového adresára. To znamená, že aj keď pracuje viac vývojárov na rovnakej časti projektu, každý si uchováva svoje zmeny len lokálne a ak potrebuje zmeny od iného člena tímu, tak sa s ním spojí a spraví si ich kópiu. Zatiaľ čo teda v prípade SVN zmeny, ktoré spravíme na serveri, sú automaticky publikované pre všetkých, v prípade distribuovaných systémov sú zmeny len lokálne a až neskôr sa môžeme rozhodnúť publikovať ich verejne.

V prípade distribuovaných systémov je celkový pohľad na vetvenie v projekte odlišný, lebo každá operácia *commit* môže byť chápaná ako vytvorenie novej menšej vetvi v projekte. Preto pri používaní týchto systémov dochádza častejšie k vetveniu a následne aj spájaniu daných vetiev. História projektu je tým pádom viac stromová ako v prípade SVN, kde je viac-menej lineárna.

Pri výbere jednej či druhej alternatívy prístupu môžu rozhodovať fakty, ktoré sú uvedené v tabuľke 1.

Tab. 1. Porovnanie preferencii centralizovaných a distribuovaných systémov

Centralizovane Systémy	**Distribuovane Systémy**
Úzko spojený tím, možná osobná komunikácia	Menej zviazaný tím, členovia často cestujú (napr. za zákazníkmi)
Pevná štruktúra tímu, obmedzenie prístupu k súborom podľa postavenia v hierarchii	Nedá sa priamo obmedziť prístup k súborom, do úvahy prichádza len oddelenie súborov do rôznych zložiek (nie je žiadaný prístup)
Menšia zodpovednosť z pohľadu používateľov, vývojári publikujú zmeny aj bez bližšej komunikácie so svojimi spolupracovníkmi	Členovia tímu viacej komunikujú navzájom zmeny, ktoré chcú publikovať
Vetvenie v projekte a spájanie daných vetiev je menej časté, preto pri daných operáciách je väčšie riziko, že spojenie bude neúspešné alebo dôjde ku kolízií	Vetvenie a spájanie je častým javom, preto sú tieto prostriedky lepšie vybavené pre narábanie s týmito operáciami
Pomerne jednoduché hľadanie chyb v lineárnej	Manuálne hľadanie chýb je zložitejšie kvôli

histórii verzii	stromovej štruktúre histórie, podpora pre vytváranie jednoduchých skriptov na automatizovane vyhľadávanie chýb (aj pre veľký počet verzií)
V prípade ukladania binárnych súborov na server sa šetrí miesto na diskoch (existuje len jedna kopia histórie)	Manažovanie binárnych súborov je náročné na diskový priestor jednotlivých členov tímu

Vyber systému pre manažment verzií je otázka s prekvapivo malým počtom priamych odpovedí. Základné problémy, ktoré treba zvážiť sú, s akým druhom dát pracuje daný tím a aký typ interakcie medzi členmi tímu požadujeme. Ak používate väčšie množstvo často editovaných binárnych dát, distribuovaný systém zrejme nebude vyhovovať vaším požiadavkám. Ak sú vaše prednosti agilnosť, invencia a práca na diaľku, distribuované systémy sú pre vás oveľa vyhovujúcejšie a centralizované systémy by váš tím len spomalili [3].

Výhody nasadenia v školskom prostredí

Všetky doteraz spomenuté systémy sa úspešne používajú v praxi pri vývoji softvérových projektov. Účelnosť a funkcionalita týchto systémov je na vysokej úrovni a preto významne pomáhajú pri implementácií projektov. Vyvstáva však otázka, či by sa tieto systémy nedali použiť aj v inom smere ako priamo v praxi.

Väčšina univerzít má svoje vlastné informačné systémy na komunikáciu so študentmi, či už ohľadom všeobecných informácií alebo zadaní na jednotlivé predmety. Niektoré predmety majú aj systémy, ktoré slúžia nielen na komunikáciu ale aj na zadávanie úloh, ich odovzdávanie a následné zobrazovanie ich hodnotenia. V prípade absencie systému sa používa iba jednoduchá elektronická komunikácia v podobe emailu.

Navzdory týmto možnostiam existuje viacej prípadov, keď sa celkom úspešne podarilo nahradiť vyššie uvedené prostriedky systémom pre manažment verzii v prípade predmetov zameraných na programovanie a softvérové inžinierstvo. V týchto prípadoch bolo motiváciou nielen výučbový charakter, oboznámenie študentov s týmito prostriedkami, ale aj uľahčenie celého procesu pri vypracovávaní zadania.

Z pohľadu oboznámenia sa s týmito prostriedkami je veľmi žiadané, aby prvá skúsenosť s danými systémami prebehla už na univerzite, kde sa môžu študenti v priebehu niekoľkých kurzov alebo semestrov podrobne oboznámiť s funkcionalitou daných prostriedkov, čo im pomôže pri nástupe do praxe (zamestnania). Tieto prostriedky totižto nielen pomáhajú pri vývoji projektu, ale taktiež pomáhajú jednotlivcom lepšie porozumieť komunikácií a práci v tíme.

Z pohľadu riadenia jednotlivých zadaní, tieto systémy uľahčujú prácu vedúcim projektov semestrálnych alebo ročníkových prac, ale aj vedúcim cvičení s jednoduchými zadaniami. V prípade cvičení sa dajú systémy pre manažment verzií použiť nasledovne:

- Vedúci vytvorí server a pre každého študenta adresár (depozitár)
- Do depozitárov môže umiestniť zadanie ale aj prvú verziu súboru, ktorý by študenti normálne odovzdali emailom alebo do iného systému; prvá verzia môže byt prázdny súbor alebo súbor s predpísanou štruktúrou

- Študent si zriadi lokálny depozitár či už v škole alebo aj doma, takže môže pracovať na danom zadaní kdekoľvek a kedykoľvek
- Pri vypracovávaní zadania študent aktualizuje pôvodnú verziu a môže pridávať poznámky (log) ku priebehu vývoja
- Vedúci má vypracované zadania na jednom mieste a zároveň má aj ďalšie informácie, ktoré mu môžu pomôcť pri hodnotení: z priebehu histórie môže vidieť kedy študent začal pracovať, ako často a akým spôsobom zadanie riešil
- Pri samotnom hodnotení môže vedúci prideliť známku, ale na základe predchádzajúcich informácií môže poskytnúť študentom aj širšie hodnotenie a názor k postupu vypracovania
- Pri zverejnení známok stačí na server umiestniť súbor so známkami, takže všetci študenti majú prístupné výsledky v rovnakom čase

V prípade skupinových zadaní (tímových projektov) je využitie týchto systémov ešte efektívnejšie a poskytuje hlbšiu analýzu postupov pri riešení projektu. Pomáhajú študentom nielen pri komunikácii, ale aj pri plánovaní a rozdeľovaní jednotlivých úloh.

Záver

Či už teda použijeme *vms* pri riešení zadaní, alebo ich využívame pri práci v tímoch na reálnych projektoch, prinášajú nám tieto systémy evidentné výhody. Odbremeňujú vývojárov od namáhavej správy súborov a zabraňujú prípadnej strate dát a nekonzistencii celého projektu. Z pohľadu tímov majú *vms* taktiež významný socializačný efekt, lebo pomáhajú pri spolupráci, synchronizácii a komunikácii vo vývojovom tíme. Je však potrebné naučiť sa s týmito prostriedkami správne narábať a zároveň vybrať vhodný typ, ktorý bude vyhovovať naším potrebám. Preto by malo byť motivujúce pre vysoké školy a univerzity, ktoré sa zaoberajú výučbou softvérového inžinierstva, aby sa ich študenti už v skorom štádiu štúdia oboznámili so systémami pre manažment verzií.

Použitá literatúra

1. Beck, J.: Using the CVS version management system in a software engineering course. Journal of Computing Sciences in Colleges archive Volume 20 , Issue 6 (June 2005), 57-65
2. Hartness, K.T.N.: *Eclipse and CVS for group projects*. Journal of Computing Sciences in Colleges archive Volume 21 , Issue 4 (April 2006) 217-222.
3. O'Sullivan, B.: Making sense of revision-control systems. Communications of the ACM Volume 52, Issue 9, 1 September 2009, Pages 57-62
4. Reid, K.L., Wilson, G.V.: Learning by doing: introducing version control as a way to manage student assignments. *Technical Symposium on Computer Science Education archive*, Proceedings of the 36th SIGCSE technical symposium on Computer science education table of contents St. Louis, Missouri, USA(2005) 272-276.

Annotation

Version management or one version is not enough

Development lifecycle of the software project always runs in several phases or iterations. In every step of development it is necessary to manage source files and documents, which are used in this process of development. It is vital to store these files in a way that it is possible to solve collisions, return to the previous version or manage whole process of development. To ease this work, version management systems were used in past two decades. In this essay I present a description of these systems, way of their usage and advantages they bring to developers in their work and communication in team. I also refer to problem situations that could arise while using these systems.