O TAJNEJ REČI VERZIÍ

Kto hľadá, nájde.

Matej Budzel

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

Abstrakt. *Manažment verzií je v súčasnosti absolútne bežnou praxou ako pri malých, tak aj pri veľkých projektoch. Väčšinu projektov sprevádzajú rôzne komplikácie, ktorým musia čeliť a mnohé z nich môžu ostať ukryté niekde v pozadí, kde potichu vyrábajú problémy. Jedným zo spôsobov ako si pomôcť na ceste k ich odhaleniu, je naučiť sa tajnú reč verzií. Úložiská verzií totiž často zastrešujú obrovské množstvá dát, z ktorých by sa dali správnym prístupom vyčítať mnohé zaujímavé poznatky aplikovateľné aj na projektové riadenie. Spôsob, ako sa k týmto poznatkom dostať, predstavuje spätná analýza repozitára verzií a je vám v tejto eseji poskytujem krátky náhľad do toho, ako takú analýzu uskutočniť, a čo všetko sa dá z nej vyčítať.*

Kľúčové slová: *manažment verzií, spätná analýza, úložisko verzií*

Úvod

Základom väčšiny softvérových, ale aj iných, projektov je tím. Skupina ľudí, ktorá sa spoločne snaží vyrobiť nejaký užitočný produkt. Pri vývoji softvéru pracuje väčšina programátorov na samostatných celkoch, ktoré sa následne spájajú dokopy. Aby sa táto činnosť dala vykonávať jednoduchšie, tak sa zaužívalo používanie centrálneho úložiska dát, ku ktorému majú všetci vývojári prístup a nahrávajú doň svoje výtvory. Avšak postupom času sa objavila potreba pracovať aj s predošlými stavmi systému a tu sa k slovu dostáva manažment verzií.

Podľa [1] možno na toto odvetvie softvérového inžinierstva hľadieť z dvoch smerov. Jednak ako o podporu manažmentu a potom ako podporu vývoja. Najbežnejšie sa v súčasnosti dá stretnúť so systémami na správu verzií, ktoré fungujú práve ako podpora vývoja a manažuje sa pomocou nich najmä zdrojový kód produktov. Keďže sa jedná o pomerne bežnú záležitosť, nebudem sa v tejto eseji ani tak venovať tomu, ako takéto

systémy fungujú, aj keď to naznačím, ale skôr sa pozriem na to ako ich využiť na podporu manažmentu. Jednou z ciest, ako to uskutočniť, je vytváranie spätných analýz úložisiek kam sa verzie ukladajú. Načo by sa takéto analýzy vyrábali? Softvérové systémy môžu byť veľmi komplexné, komplikované a veľké, a teda skrývajú v sebe množstvo rizík. Existuje mnoho manažérskych metód, ako týmto rizikám predísť, ale nie vždy sa dajú odhaliť úplne všetky a niektoré sú skryté hlboko pod povrchom. Jednou z možností, ako si k ich nájdeniu uľahčiť cestu je naučenie sa tajného jazyka verzií, ktorý nám môže napovedať, kde sa tie problémy nachádzajú. A spôsob ako sa tento jazyk naučiť je práve pomocou vytvárania a rozoberania spätných analýz úložísk verzií.

Zdrojový kód náš každodenný

Skôr než sa dostanem k samotnej spätnej analýze považujem za vhodné najprv aspoň v skratke zhrnúť ako manažment zmien prebieha, a to najmä vo forme, s ktorou sa v súčasnosti dá prísť najľahšie do kontaktu.

Ako som už naznačil v úvode, ak sa dnes stretnete s nejakým spôsobom manažmentu konfigurácií bude najpravdepodobnejšie zameraný na správu súborov so zdrojovým kódom. Nie dôvod veľmi sa diviť tomu prečo je to práve tak. Implementácia produktu aj s jej následnými úpravami zvyčajne prebieha celé roky, a za taký dlhý čas sa vyvíja a zdokonaľuje nielen základná línia, ale zvykne sa niekedy pracovať aj na rôznych alternatívach – najmä, ak príde k zmene požiadaviek zákazníka. A to je úrodná pôda pre veľké množstvo vykonaných zmien, a z nich vyplynuvších rôznych verzií, v ktorých je potrebné udržiavať adekvátny prehľad, aby bolo možné sledovať napríklad to, kedy a kde sa opravila nejaká chyba, či kedy a kam sa odklonil smer vývoja.

Navyše pre manažment verzií zdrojového kódu existujú dobre spracované a účinné systémy. Základom je centrálne úložisko, repozitár, kde sa všetky verzie ukladajú a majú tak všetci členovia vývojového tímu prístup nielen vždy k najaktuálnejšiemu stavu produktu, na ktorom pracujú, ale aj k jeho starším verziám. Ďalej okrem ukladania verzií ponúkajú tieto systémy aj nástroje na koordináciu práce viacerých ľudí. Pomocou nich sa dajú nájsť riadky, v ktorých prebehli zmeny, či dokonca vedia určiť, ktoré riadky boli zmazané, alebo ktoré sú úplne nové. Vďaka tomu sa dajú zmeny v tých istých súboroch spôsobených viacerými ľuďmi zlúčiť aj bez toho, aby vznikol konflikt, pretože nie vždy sa robia úpravy na presne tých istých miestach.

Odkladanie rôznych verzií a používanie na to určených nástrojov sa teda môže javiť ako užitočné a zmysluplné, ale na základe čoho okrem zmenených riadkov v zdrojovom kóde možno úložisko verzií spätne analyzovať? Odpoveď na túto otázku je: metadáta. Tie možno tiež radiť medzi esenciálne zložky systémov na správu verzií a pomocou nich možno určiť aj to, kto a kedy zmenu vykonal. A z takýchto informácií už možno usudzovať aj zaujímavé závery, najmä ak si uvedomíme, že v úložiskách býva uchovaných zvyčajne veľké množstvo dát.

Nespraviť znova tú istú chybu

Pokusov o spätnú analýzu prebehlo viacero, ale ja som sa zameral na dva konkrétne. Prvý prebehol v univerzitnom prostredí a skúmali sa repozitáre viacerých malých študentských

tímov v rámci predmetu podobnému tímovému projektu na našej fakulte. Na opačnú stranu, pri druhej analýze sa použilo úložisko verzií projektu, ktorý má otvorený zdrojový kód a je vyvíjaný veľkým tímom dobrovoľných programátorov z celého sveta. Aj keď sú analyzované projekty diametrálne odlišné, dajú sa v dosiahnutých výsledkoch vybadať mnohé spoločné znaky, čo je spôsobené najmä tým, že oba použili na správu verzií systémy, aké som spomínal v predošlej časti, teda také, ktoré pracujú so zdrojovým kódom. A preto prv než bližšie opíšem špecifické zistenia oboch projektov zhrniem čo (približne) rovnaké sa dá z nich vyčítať.

V oboch prípadoch prebehlo ohodnocovanie pomocou rôznych metrík, ktoré boli založené na sledovaní operácií vykonaných v systéme v závislosti od času. Jednalo sa najmä o sledovanie kvantitatívnych zmien v dĺžke vytoreného kódu. Pri takomto prístupe sa dá na vývoj celého projektu hľadieť z viacerých uhlov:

- Práca tímu ako celku
- Prispievanie konkrétneho jednotlivca
- Súbory, s ktorými sa pracovalo

Ak sa na obsah úložiska pozrieme z pohľadu celého tímu, tak ako najzjavnejšia metrika sa javí nárast veľkosti projektu čo do počtu riadkov. Taktiež sa dá vyčítať aj to, po akej krivke tento rast prebiehal, a môže to byť zaujímavá informácia, pretože konštantne rastúca krivka je nedosiahnuteľný ideál a reálne výsledky majú všemožné tvary a prehyby, za ktorými vždy v pozadí stojí nejaká konkrétna príčina. Ďalej je možné sledovať na istej úrovni aj komunikáciu medzi členmi tímu. V tomto smere sa dajú dedukcie vytvárať podľa pomeru úspešných spojení zmien, vykonaných na tom istom súbore viacerými ľuďmi, v porovnaní s nezlučiteľnými kolíziami. Napríklad v tímoch, kde sa každý hrá na svojom piesočku a neinformuje ostatných o tom, čo robí alebo čo sa chystá urobiť, môže byť počet kolízií výrazne vyšší.

Pri konkrétnych jednotlivcoch si pri spätnej analýze úložiska môžeme v prvom rade všimnúť ich prínos v počte pridaných riadkov, a podobne ako pri tíme sa dá určiť aj to, ako sa úroveň aktivity menila v čase. Okrem toho sa dá sledovať aj na koľkých rôznych súboroch jednotlivec pracoval či ako často si vyberal z úložiska aktuálne najnovšiu verziu. Z manažérskeho pohľadu je dobré poznať okrem samotného prínosu v počte naťukaných znakov aj to, akým spôsobom sa jedinec integroval ako člen tímu. V takom prípade môže opäť dosť napovedať počet úspešných spojení jednotlivcom vykonaných zmien so zmenami od ostatných členov v porovnaní s kolíziami, ktoré sa mu podarilo vytvoriť. Podľa charakteru sa dá odhadnúť aj pracovný charakter jednotlivca – napríklad od aktívneho testovača, ktorý aj opravuje chyby sa očakáva, že vykoná množstvo menších zmien vo viacerých súboroch, zatiaľ čo v prípade programátora, ktorý má na starosti novú funkcionalitu, je predpokladaným výsledkom jeho práce skôr súvislý sled nových riadkov.

Predošlé aspekty boli postavené zo strany tvorcov, teraz sa pozrime na vec zo strany tvoreného. Zas a znova sa ako najzjavnejšia metrika núka sledovanie zmien v počte riadkov kódu pre každý súbor, čím sa môže dať vyjadriť jeho komplexnosť. Avšak zaujímavé a napovedajúce môžu byť aj porovnania toho, ako sa súbory od seba líšia veľkosťou, taktiež ako veľkosť súboru súvisí s počtom jeho verzií, aké sú rozdiely v dĺžke období, počas ktorých sa na súbore pracovalo alebo aké sú rozdiely v počte

programátorov, ktorí v súbore vykonali zmeny. V poslednom prípade sa môže pridať aj počet kolízií či úspešných spojení v rámci jedného súboru, z čoho sa dá odhadnúť problematickosť súboru, ak v ňom často vykonávajú zmeny mnohí ľudia a ich snahu sprevádzajú početné kolízie.

Z posledných troch odsekov je zrejmé, že analyzovať úložisko verzií možno naozaj z viacerých stránok, ale čo sa dá takýmto spôsobom získať? Ako sa z toho poučiť, či ako to využiť na monitorovanie projektového riadenia? Ak sa analyzuje úložisko verzií projektu, ktorý už bol ukončený a poznáme okrem zaznamenaného priebehu aj počiatočnú špecifikáciu a konečný výsledok, je možné určiť viaceré problematické miesta, z nich sa dá vziať ponaučenie, a potom sa im možno v budúcnosti vyvarovať. Je možné napríklad nájsť nedostatky v internej komunikácii v tíme, ak sú známe postupy, ktorými sa uskutočňovala, a tiež aj fakty o tom, ako sa prejavila vo výsledku v počte kolízií medzi programátormi.

Použiteľnosť možno vidieť aj v prípade ak sa jedná o veľký projekt, na ktorom pracuje mnoho ľudí. Môže sa stať, že sa niektorý jednotlivec nenápadne skrýva za výsledky davu, ale výsledky jeho činnosti zaznamenané v úložisku ho môžu odhaliť a pri ďalšom projekte si ho môžu manažéri lepšie ustrážiť. Obdobne ak je známa pozícia, na ktorej mal jednotlivec vystupovať, je možné z úložiska podľa charakteru zmien, ktoré vykonal, ohodnotiť správnosť jeho aktivít, a z toho vytvárať dôsledky do budúcna. Ďalej analýza verzií súboru môže odhaliť napríklad návrhové problémy, pretože ak sa vyvíjali súbory, ktoré predstavujú hrdlo príliš úzke na to, koľko programátorov cez neho prechádza, asi niečo nebolo úplne v poriadku.

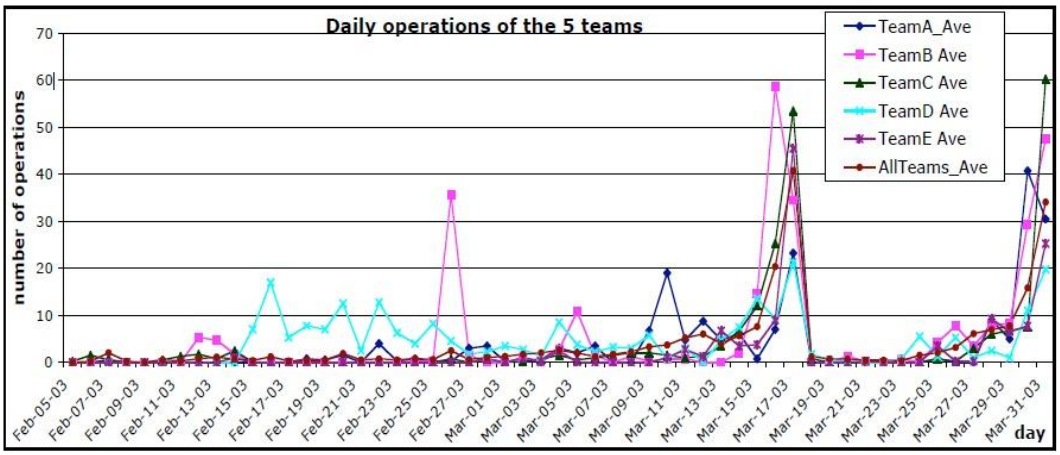
Ale teraz prejdime k tomu, čo konkrétne zistili spomínané dve prípadové štúdie a najmä na špeciality, ktorými sa odlišujú od opísaného zovšeobecnenia.

Laboratórne myši

Spätná analýza úložisiek verzií, ktoré vznikli počas semestrálneho tímového projektu na Albertskej univerzite v Edmontone v Kanade [6], bola v podstate od samého počiatku zamýšľaná a aj realizovaná ako vedecký pokus. Na sledovanie a ohodnocovanie zmien sa použilo špeciálne upravené prostredie Eclipse pre programovanie v jazyku Java, s ktorým pracovalo dvadsaťtri štvorčlenných tímov a časová organizácia predmetu bola rozdelená na tri fázy, z ktorej každá končila odovzdaním vypracovaných materiálov. Cieľom pokusu bolo otestovanie a ohodnotenie vytvoreného prostredia v nasadení.

Motiváciou tvorcov pre vyhotovenie prostredia bolo umožniť pedagógovi, ktorý predmet vedie, jednotne a prehľadne sledovať priebeh aktivít jednotlivých tímov, vďaka čomu by mohol odhadnúť prípadné riziká a poskytnúť tak študentom adekvátnu spätnú väzbu, aby dosiahli lepšie výsledky. Základným problémom pri výučbe totiž je, že študenti sú väčšinou pomerne leniví a najradšej robia všetko na poslednú chvíľu. A presne toto sa aj potvrdilo aj v tejto štúdií, tak ako môžete vidieť na Obrázku 1. Krivky aktivity tímu výrazne stúpajú okolo dátumov keď sa blížil termín odovzdania, zatiaľ čo po iné dni sa celková činnosť pohybovala kdesi okolo nuly. Okrem toho, ak sa spojí študentská záľuba v odkladaní zadaní s tendenciou zahmlievať svoju nečinnosť, môže sa stať, že termín odovzdania jednoducho nestihnú a majú problém na krku. Takejto situácií by sa dalo predísť práve sledovaním tímovej aktivity a ak by pedagóg spozoroval, že na

úložisku tímu je podozrivé ticho, mohol by promptne zareagovať a členom tímu pripomenúť, aby už začali pracovať.



Obrázok 1 - priemerný denný počet operácií študentských tímov [6]

Ďalšie problémy tímu ako celku môžu spočívať v komunikácií a v rozdelení náročnosti úloh. Samotní študenti si totiž nemusia uvedomovať, že niečo v tomto smere robia nesprávne, a teda o tom ani neinformujú pedagóga. Ten však analýzou úložiska môže odhaliť či existuje množstvo nezlučiteľných kolízií značiacich možné nedostatky vo vzájomnom informovaní sa o postupe alebo aj to, či sú niektorí členovia výrazne aktívnejší ako ostatní. Podľa toho sa dá so študentmi konzultovať ich ďalší postup.

Nepríjemné skryté problémy sa môžu vyskytnúť aj na úrovni jednotlivcov. Jednak je pre pedagóga pomerne náročné odhaliť, či sa niektorý študent len nevezie s tímom a jeho aktivita je minimálna, pokým ho náhodou ostatní členovia takpovediac „neudajú“ alebo či niekto nie je až príliš aktívny, a nezasahuje príliš často, a chybne, do výsledkov práce svojich kolegov. Vďaka analýze úložiska sa takéto anomálie dajú odhaliť a taktiež sa môžu dať včas riešiť.

Užitočné informácie sa dajú získať aj pri pohľade na jednotlivé súbory, pretože sa podľa nich dajú odhaliť nedostatky v návrhu. Ak totiž existuje príliš konfliktný súbor, na ktorom pracuje príliš mnoho ľudí naraz, môže pedagóg navrhnúť, aby sa zvážila taká zmena, ktorou by sa kolíziám dalo predísť – napríklad rozdelenie na viac súborov s menším vzájomným previazaním a podobne.

Osobne si myslím, že v tomto prípade sa jedná o naozaj veľmi užitočné využite možnosti spätne analyzovať úložisko verzií, pretože sa takýmto spôsobom dá predísť mnohým zbytočným komplikáciám. Aj celkové riadenie tímového predmetu sa pre pedagóga stáva jednoduchším, pretože nemusí občas tápať v tme, ale má sa o čo oprieť. Samozrejme treba podotknúť, že názory a dôsledky vydedukované na základe získaných dát nemusia byť vždy úplne presné. Môžu byť dokonca až mylné. Celá práca na projekte takéhoto typu totiž predstavuje pomerne málo človeko-dní, operácií teda nie je uskutočnených veľmi veľa a študenti často ani nemajú predošlé skúsenosti s tým, ako správne používať systém na správu verzií. Takže informácie môžu byť skreslené

a neodzrkadľujúce skutočný stav problému. Ďalšou prekážkou je aj nárazové fungovanie študentov, ktoré nedáva veľa priestoru na poskytnutie rád a najmä na vypočutie a aplikovanie. Lenže nič nie je zbytočné a ak sa preskúmajú paralely medzi výsledným produktom a poznatkami z úložiska, dá sa lepšie pochopiť zmýšľanie študentov, a na základe toho lepšie viesť predmet na ďalší rok. Je to akási forma nenápadnej, utajenej spätnej väzby od študentov, ktorú poskytnú všetci a povinne.

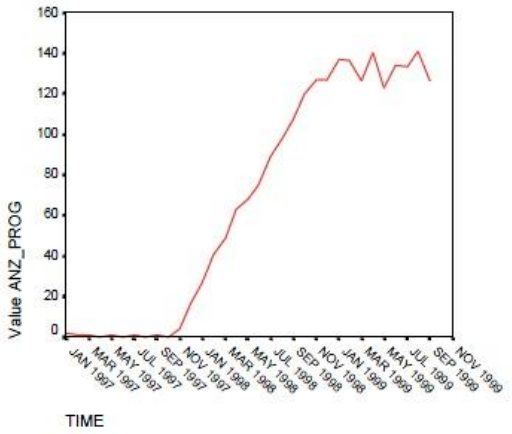
Dátoví archeológovia

Zatiaľ čo pri predošlej štúdií sa skúmané úložisko vytváralo cielene na testovacej vzorke, v tomto druhom prípade sa použili dáta, ktoré vznikli počas vývoja reálneho projektu, pri ktorom nie je dôležitá ani tak cesta vývojárov ako skôr celkový výsledok. Konkrétne sa pristupovalo k repozitáru projektu GNOME, ktorý sa slúži ako prostriedok na zobrazenie virtuálnej pracovnej plochy. Hlavnou motiváciou, prečo bol zvolený práve tento projekt bol fakt, že sa vyvíja s otvoreným zdrojovým kódom, a práve takýto typ softvéru chceli páni Koch a Schneider preskúmať [5].

Význačným znakom tohto projektu a aj mnohých iných, ktoré sú vyvíjané podobným štýlom je, že na ňom pracovalo a pracuje výrazne viac ľudí ako pri klasickom komerčne vyvíjanom projekte, ale každý z nich venuje práci na ňom podstatne menej času. Vďaka tomu sa počas takéhoto projektu do úložiska verzií zvykne zaniesť enormné množstvo informácií, ktoré analytikom uľahčuje život, pretože môžu dospieť k presnejším výsledkom. Pri tejto konkrétnej analýze sa použili rôzne techniky dolovania dát, a keďže sa skúmali dáta viac-menej historického charakteru, možno vravieť o akomsi archeologickom hľadaní stratených a tajomných artefaktov ukrytých kdesi hlboko pod povrchom.

Takže bola objavená softvérová Trója?

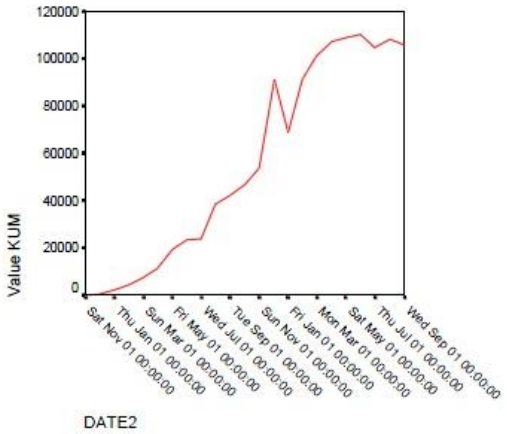
Z globálneho hľadiska sa opäť sledovali zmeny v náraste objemu napísaného kódu v závislosti od času, avšak zaujímavejšie poznatky sa dajú získať pri zohľadnení spomínaného špecifika veľkej variability pracovnej sily. Počas celého sledovaného obdobia sa do projektu zapojilo 301 programátorov a na Obrázku 2 môžete vidieť, ako sa priebežne menil počet aktívnych zúčastnených. Zaujímavým na tejto krivke nie je ani tak strmý nárast, ktorý značí popularizáciu projektu, ale skôr náznak ustálenia v poslednom období. Ten je vlastne v rozpore s celou myšlienkou vyvíjania softvéru s otvoreným kódom, pretože pri ňom sa očakáva priebežne rastúci počet členov, čím sa docieľuje zrýchľovanie kumulovania vykonanej práce, a teda projekt môže rýchlejšie napredovať. Preto stojí za to zamyslieť sa nad príčinami takéhoto stavu. Ak vylúčime možnosti, že sa z projektu stala sektárska záležitosť, ku ktorej sú pripúšťaní len vyvolení, alebo že o projekt ustal záujem, čo je nepravdepodobné pri pohľade na náhly prechod zo strmého rastu do ustálenej polohy, tak môžeme dospieť k záveru, že projekt je jednoducho zo softvérovo-architektonického pohľadu zle navrhnutý. Môže na ňom totiž v jednom časovom okamihu pracovať len limitovaný počet ľudí, a v takom prípade by sa malo zvážiť prepracovanie architektúry systému.



Obrázok 2 - priemerný počet aktívnych programátorov [5]

Pohľad na participujúcich programátorov ako na skupinu priniesol aj ďalšie zaujímavé zistenie. Ak sa vezme do úvahy počet zmien v riadkoch kódu, ktoré jeden programátor vykonal a zohľadní sa aj dĺžka obdobia počas, ktorého sa na projekte zúčastňoval, tak sa dá vybadať, že aj keď na systéme pracuje množstvo ľudí z viacerých končín sveta, stále existuje úzka skupinka ľudí, ktorí sú akýmsi centrálnym pohonom, ktorý celý projekt posúva vpred.

Posledným zaujímavým aspektom, z globálneho pohľadu, na ktorý by som rád poukázal, je sledovanie celkového vývoja projektu. Respektíve jeho čiastkových zložiek, ak sa jedná o veľký projekt ako je GNOME. Graf na Obrázku 3 znázorňuje počet riadkov kódu časti jadra celého projektu a je zjavné, že sa dospelo priebežne do pomerne ustáleného stavu. Čiže aj takýmto spôsobom sa dá vyhodnocovať sledovanie dosahovania požadovaných výsledkov a približovanie sa k cieľu.



Obrázok 3 - celkový počet riadkov kódu časti jadra projektu GNOME [5]

Pri ostatných náhľadoch na projekt, teda na súbory alebo na konkrétnych jednotlivcov sa dajú väčšinou získať obdobné informácie, ako som spomínal vyššie v spoločnom zhrnutí. Nájde sa ešte pár špecifík. Napríklad zaujímavé výsledky sa dajú získať aj zo sledovania aktivity jednotlivcov, ale z pohľadu riadenia sa pri takomto type projektu nedajú v tomto smere odvádzať veľmi silné dôsledky. Každý programátor pracuje na projekte dobrovoľne, a teda nie je nijak viazaný dodaním požadovaných výstupov a ani ich nemožno od neho vymáhať.

Takže aká by bola odpoveď na otázku, ktorá je o pár riadkov vyššie?

Žiadnu softvérovú Tróju sa v tomto smere nepodarilo nájsť, ale bola by chyba túto analýzu zatracovať. Jednak pomohla metódami softvérového inžinierstva ozrejmiť fungovanie vývoja softvéru s otvoreným zdrojovým kódom, ale poskytuje aj priestor na poučenia sa. Na poučenie sa z cudzích chýb. Pretože znova sa dostávame k tomu, že ak máme prístup k úložisku verzií a aj k výslednému projektu, čo v tomto prípade máme, a bez problémov, tak hĺbkovým preskúmaním ich vzájomného vzťahu môžeme odhaliť rôzne nenápadné chyby, ktoré mohli spôsobiť mnoho komplikácií. A budeme ich tak potom vedieť identifikovať v budúcnosti skôr než by začali spôsobovať problémy.

Osobne si myslím si, že nemá veľký zmysel robiť spätnú analýza úložiska verzií pri takýchto projektoch na pravidelnej báze. Na monitorovanie projektu totiž väčšinou postačuje sledovanie dodržiavania plánu a spĺňania zadaných úloh, ale ak je podozrenie, že vývoj projektu sa niekde zadrháva, tak môže byť analýza repozitára užitočná. Napríklad pri odhaľovaní rôznych nenápadných škodcov. Dá sa totiž očakávať, že sa objavia, a to najmä pri takýchto veľkých projektoch, do ktorých sa môže zapojiť prakticky ktokoľvek, a ktoré väčšinou neponúkajú motiváciu v podobe finančného či iného materiálneho obohatenia.

Záver

Okrem textov, z ktorých som vychádzal primárne, som prišiel do kontaktu aj s ďalšími tromi, pomocou ktorých by sa dala táto esej rozšíriť o ďalšiu obsahovú úroveň. Opis analýzy repozitárov som zámerne, z dôvodu rozsahu článku, zúžil len na sledovanie verzií počas implementačnej fázy pomocou systémov pracujúcich so zdrojovým kódom. Avšak zaujímavé výsledky by sa mohli dosiahnuť aj preskúmaním vlastností rôznych verzií vytvorených už počas rannejších fáz vývoja – počas analýzy a návrhu. Tejto problematike sa venuje [2]. V takomto prípade však narážame na nedostatky klasických systémov, ktoré nedokážu pracovať so zložitou sémantikou, a tak, aby bolo možné zaznamenávať stavy počas týchto fáz, keď sa tvoria najmä rôzne diagramy, bol navrhnutý spôsob ako takéto objekty efektívne uchovávať v priestore verzií [4]. V momente, keď už boli vytvorené metódy a prostriedky, ako uchovávať verzie počas analýzy a návrhu, tak sa mohli stanoviť aj metriky, ako hodnotiť spätnú analýzu takýchto repozitárov a bola vytvorená aj jednoduchá prípadová štúdia [3]. Výsledky podobne ako pri štúdiách spomínaných v tomto článku vyšli opäť z pohľadu užitočnosti spätnej analýzy pomerne pozitívne.

Softvérové systémy sú často veľmi komplexné, pracuje na nich množstvo ľudí a ukrýva sa v nich množstvo skrytých alebo nejasných nástrah, ktoré rovnako nenápadne pracujú na komplikácií života zúčastnených ľudí. Preto by sa možnosť analyzovania

úložiska vytvorených verzií produktu nemala zanedbávať, a malo by sa k nej občas pristúpiť. Už len preto aby sa skontrolovalo či je všetko tak ako má.

Za veľmi vhodné považujem takýto prístup najmä pri študentských tímových projektoch, kde je pravdepodobnosť pochybenia skutočne vysoká, ale mohol by nájsť, alebo už aj našiel, úrodnú pôdu aj vo veľkých komerčných firmách ako podpora riadenia. Veď nakoniec ovládanie každej reči, aj tej tajnej, ktorou k nám prehovárajú verzie, je užitočné.

Použitá literatúra

1. Conradi, R., Wesfechtel, B.: Version models for software configuration management. In: *ACM Computing Surveys*, Vol. 30, No. 2, (1998) 232 – 282.
2. Kaur, P., Singh, H.: Version Management and Composition of Software Components in Different Phases of Software Development Life Cycle. In: *ACM SIGSOFT Software Engineering Notes*, Vol.34, No. 4, (2009), 1-9.
3. Kaur, P., Singh, H.: Metrics Suite for Component Versioning Control Mechanism in Component-based Systems. In: *Journal of Software Engineering*, Vol.4, No. 3, (2010), 231-243.
4. Kelter, U., Ohst, D.: A Fine-grained Version and Configuration Model in Analysis and Design. In: *ICSM '02: Proceedings of the International Conference on Software Maintenance*, IEEE Computer Society, Washington D.C., (2002), 521 - 527.
5. Koch, S., Schneider, G.: Results from Software Engineering Research into Open Source Development Projects Using Public Data. In: *Informationswirtschaft*, H.R. Hansen und W.H. Janko (Hrsg.), Wirtschaftsuniversitaet, Wien, No. 22, (2000).
6. Liu, Y., Stroulia, E.: Reverse Engineering the Process of Small Novice Software Teams. In: *Proceedings of the 10th Working Conference on Reverse Engineering*, 2003.

Annotation

About secret language of versions.

Version management is nowadays a common practice for small as well as for large projects. Almost all projects have to face various complications while many of them may remain hidden somewhere in the background and produce problems quietly. One way to help with their discovery is to learn the secret language of versions. Version repositories often contain enormous amounts of data, which can offer a lot of interesting information applicable to project management, if the correct approach is used. The Reverse engineering of version repository is a way to gain that knowledge and in this essay, I will offer you a short insight into how such an analysis is conducted and what can be read from it.