POUŽIŤ DISTRIBUOVANÝ SYSTÉM NA SPRÁVU VERZIÍ V MALOM PROJEKTE?

Zväčša je veľmi dobré, ak história úložiska obsahuje postupnosť všetkých krokov, ktorými sa dostaneme k aktuálnemu stavu. Až pokiaľ tam niekto nevloží pár iso obrazov a v nasledujúcom commite ich nezmaže.
(IRC kanál systému Bazaar)

Matej Kvitkovič

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

Abstrakt. *Táto esej sa zaoberá porovnaním nástrojov na manažment verzií. Existujú dve hlavné architektúry, ktoré tieto systémy implementujú – centralizovaná a distribuovaná architektúra. Pre každú z nich existujú vyspelé nástroje, ktoré majú rôzne vlastnosti a poskytujú rôzne funkcie. Esej porovnáva vlastnosti nástrojov oboch architektúr, uvádza tiež históriu ich používania a aktuálny stav. Hlavnými referenčnými nástrojmi sú Subversion a Git. Následne uvažuje ich použitie v malom softvérovom projekte. Zdôvodňuje voľbu distribuovaného nástroja pre tímový projekte – Digitálne divadlo. Po zvážení potrieb tímu určuje, ako hlavnú a rozhodujúcu výhodu najmä flexibitu vývoja, ktorú distribuované nástroje poskytujú a tým šetria používateľom systému čas a tým aj ďalšie prostriedky.*

Kľúčové slová: *manažment verzií, distribuovaná a centralizovaná architektúra, Git, Subversion*

Úvod

Systém na správu verzií patrí dnes k základným nástrojom softvérového vývoja. Tento systém zjednodušuje vývoj a údržbu programu na ktorom sa podieľa súčasne viacero

účastníkov. Systém na správu verzií, ďalej VCS (angl. Version Control System), poskytuje programátorom možnosť tvoriť kód bez toho, aby ich zasiahli zmeny vykonané inými programátormi. VCS spravujú úložisko (angl. repository) súborov so zdrojovým kódom a ďalších súborov, ktoré sú súčasťou softvérového projektu.

Dôležitou funkciou VCS je udržiavanie histórie zmien v súboroch a poskytovanie možnosti získať ich ľubovoľnú predošlú verziu. Okrem správy verzií jednotlivých súborov VCS spravujú aj logické vzťahy medzi zmenami v súboroch, ktoré odoslal používateľ úložiska v danom čase - verzie. Pri každej verzii vývoja je potrebné evidovať, ktorý používateľ VCS vytvoril novú verziu, prečo ju vytvoril a informácie o podrobnostiach vykonanej zmeny.

Existujú dva základné druhy VCS, ktoré sa delia podľa zvolenej sieťovej architektúry – centralizované a distribuované VCS. Medzi týmito skupinami nástrojov existujú veľké odlišnosti, založené na odlišnom prístupe k správe verzií, ktorý je priamym dôsledkom zvolenej architektúry. Táto esej sa bude zaoberať porovnaním vlastností nástrojov z oboch skupín a ich možným využitím v softvérovom projekte, poznatky aplikujem tiež na tímový projekt.

Pre rôznorodosť jednotlivých VCS a rôznosť prístupov k manažmentu verzií, nie je možné úplne oddeliť všeobecný vplyv vlastností architektúry a vplyv vyplývajúci z vlastností konkrétneho VCS. Pokúsim sa ich oddeliť a zvýrazniť najmä tie kde je vplyv zvolenej architektúry zrejmý alebo kde implementácia určitej funkcie VCS súvisí s touto architektúrou. Taktiež sa medzi jednotlivými VCS líši aj implementácia zvolenej architektúry – to sa týka najmä DVCS, tu sa budem sústrediť najmä na mne známe prostriedky. Tými sú, z jednotlivých skupín – Subversion alebo skrátene SVN, najrozšírenejší centralizovaný systém a Git, ako najrozšírenejší distribuovaný systém. Okrem nich, najmä na strane distribuovaných nástrojov, existuje aj viacero iných, funkcionalitou odlišných nástrojov, avšak keďže som nemal možnosť viacero z nich odskúšať, esej sa bude opierať najmä o možnosti poskytované týmito nástrojmi.

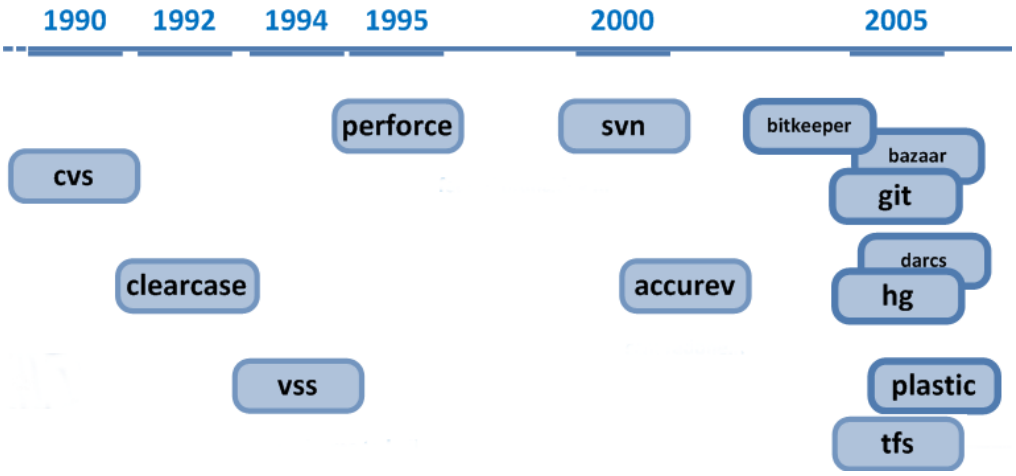
V tíme softvérového projektu sme využili distribuovaný nástroj Git. V ďalších kapitolách okrem výhod a nevýhod oboch architektúr uvediem aj dôvody, ktoré nás viedli k tomuto rozhodnutiu.

História VCS

Prvé systémy na manažment verzií vznikli v roku 1982. Tieto archaické systémy dokázali operovať len na jednotlivých súboroch, nepoznali ešte pojem úložiska. CVS (Concurrent Version System), ktorý vznikol v roku 1990 je najstarším, v širšej miere využívaným systémom. Je to centralizovaný VCS, avšak s mnohými nedostatkami a chybami. Tento systém sa majoritne používal desať rokov, kedy ho nahradil jeho nástupca – Subversion. Niektoré väčšie firmy ho pre vysokú náročnosť presunu svojich veľkých projektov využívajú dodnes. Subversion odstránil mnohé nedostatky CVS, ako bola neatomickosť operácií, nemožnosť jednoduchej rekonštrukcie histórie úložiska alebo chýbajúca podpora detekcie presunu a premenovania súborov.

V čase vzniku SVN začal vývoj prvých distribuovaných systémov. Za účelom kolaboračného vývoja jadra systému Linux vznikol v roku 2002 distribuovaný VCS (ďalej DVCS), Bitkeeper. Tento komerčný systém však zadarmo sprístupňoval iba časť

dostupných funkcií a navyše bol na vtedajšie požiadavky pomalý a preto bolo viacero pokusov o tvorbu rozšírení, v reakcii na čo v roku 2005 prestali autori poskytovať Bitkeeper zadarmo úplne. Vzápätí na to vytvoril programátor Linuxu, Linus Torvalds vlastný DVCS – Git, ktorý sa veľmi rýchlo stal najpoužívanejším DVCS. Okrem týchto nástrojov v tom čase vznikli aj ďalšie DVCS. Oproti systému Git sa až tak výrazne nepresadili, avšak dodnes sú používané. Za zmienku stoja systémy Darcs, Bazaar a Mercurial. Vzniklo taktiež niekoľko centralizovaných nástrojov, ako napríklad TFS (Team Foundation Server) od spoločnosti Microsoft [1].



Obr. 1. Časová os vzniku VCS.

Dvoma najpoužívanejšími systémami na kontrolu verzií sa stali teda Git a SVN a sú nimi dodnes. V posledných rokoch sa výraznejšie nepresadil žiaden nový VCS, vývoj sa sústredí najmä na zdokonaľovanie tých existujúcich. Postupný sa ukázalo, že výhody, ktoré priniesli DVCS sú pri zvyšovaní efektívnosti práce dôležité – nasvedčujú tomu dva faktory.

Jedným z nich je neustále sa zvyšujúci podiel distribuovaných systémov a to výrazným spôsobom, najmä v oblasti open-source projektov, keďže v komerčných firmách sú väčšie obavy o kontrolu nad vloženými súbormi, ktorú distribuované systémy neposkytujú.

Druhým je fakt, že aj do pôvodne výsostne centralizovaných nástrojov ako SVN alebo TFS postupne pribúdajú funkcie pôvodne vlastné iba DVCS. Najjasnejším príkladom je podľa môjho názoru to, že napríklad v SVN je už dnes možné vykonávať niektoré operácie na lokálnom počítači bez sieťového pripojenia k centrálnemu úložisku.

Ďalším trendom vo vývoji VCS je ich čoraz väčšia integrácia s ďalšími podpornými prostriedkami, či už sa jedná o systémy na manažment úloh, systémy na manažment obsahu webu (Content management system). Táto integrovateľnosť môže byť jedným z dôvodov voľby konkrétneho VCS.

Porovnanie dôležitých vlastností VCS

Mnohé výhody a nevýhody prostriedkov na manažment verzií sú úzko späté so spôsobom ich využitia. Pri výbere konkrétneho prostriedku je potrebné vytýčiť požiadavky na VCS, uviesť ich prioritu a konfrontovať ich s vlastnosťami architektúr, ale aj jednotlivých nástrojov.

Uloženie histórie zmien

Jednou zo základných vlastností, ktorú by dnes mali poskytovať už všetky moderné nástroje, je efektívna stratégia fyzického uloženia histórie súborov v úložisku. Poznáme dve základné stratégie - Snapshot a Changeset. Prvá z nich ukladá v úložisku všetky zmenené súbory, čo predstavuje pri väčšom počte zmien veľké množstvo dát. Pri stratégii Changeset sa v úložisku nachádza iba rozdiel, delta, medzi jednotlivými verziami. Nevýhodou tohto prístupu je nutnosť prepočtu vždy pri získavaní konkrétnej verzie súboru – avšak pri možnostiach dnešných výpočtových prostriedkov by sa tento problém mohol výraznejšie prejaviť, iba pri dlhotrvajúcich projektoch s tisíckami zmien. A aj na tento existuje v moderných nástrojoch jednoduchý liek – kombinácia oboch prístupov, kde sa vždy po určitom počte zmien daného súboru tento uloží celý.

Problémom, načrtnutým v myšlienke eseje, s ktorým sa lepšie dokážu vysporiadať centralizované úložiská sú binárne súbory. Tieto nie je možné efektívne uložiť Changeset stratégiou a preto v DVCS, kde sa vytvára veľké množstvo commitov (odoslaní novej verzie súborov) tvoria veľký overhead (dáta potrebné na uloženie histórie).

Implementácia stratégie uloženia zmien je dôležitá vlastnosť, ale vplyv architektúry VCS na ňu nie je nijak výrazný, skôr sa jedná o obmedzenia kladené zvolenou architektúrou, s ktorými sa tvorcovia konkrétnej aplikácie musia pri je vývoji popasovať. Pre nás, ako používateľa VCS, sú dôležité najmä vlastnosti konkrétneho VCS a vlastnosti vyplývajúce z implementovanej stratégie – veľkosť overheadu a rýchlosť odozvy VCS.

Spolupráca a vetvenie

Dôležitou vlastnosťou VCS je mechanizmus tvorby viacerých paralelných vetiev vývoja, t.j. *rozvetvenie* (angl. branching).

Centralizované nástroje dovoľujú vývoj kolaboráciou viacerých užívateľov pomocou jednej vetvy – aj keď s obmedzeniami, v menších projektoch sa často používa. Vetvy sa zväčša vytvárajú len pre špecifické potreby – uloženie stabilnej verzie projektu (tzv. release branches) alebo rozdelenie aplikácie na viacero funkčných verzií. V lepšom prípade sa pri väčších aplikáciách vyvíjajú jednotlivé komponenty programu v oddelených vetvách. SVN, napríklad, neeviduje zmeny medzi vetvami vývoja a teda použitie postupov ako je vytvorenie vlastnej vetva pre každého používateľa úložiska je neefektívne. Výhodou centralizovaných nástrojov je možnosť uzamykať súbory, aby nedošlo k ich vzájomnému prepísaniu pri spájaní vetiev, čo pri DVCS nie je možné.

V distribuovaných nástrojoch sa vetvy používajú oveľa intenzívnejšie, vetvenie na najzákladnejšej úrovni prebieha pri každom získaní a zmene súborov úložiska.

Jednou z hlavných podmienok flexibilnej pracovnej skupiny malého projektu – a teda aj nášho tímového projektu, je možnosť samostatne pracovať na pridelenej časti systému

bez potreby centrálneho správneho bodu – ktorým by som v opačnom prípade, ako manažér vývoja musel byť ja.

Integrácia zmien

Dôležitou súčasťou použitia VCS je proces integrácie zmenených súborov a vetiev, ktorý sa nazýva *spojenie* (angl. merging). Pri riadení tohto procesu je potrebné riešiť významný problém - riešenie konfliktov, ktoré vzniknú upravením tých istých súborov a ich častí viacerými programátormi. Pri riešení konfliktov dochádza pri použití VCS k najväčšiemu množstvu chýb. Dôležitá je správna voľba algoritmu na spájanie súborov, dobrá podpora riešenia konfliktov v konkrétnom nástroji a najmä ich správne používanie.

Spôsob práce v centralizovaných nástrojoch a odosielanie zmien po dlhších časových úsekoch často spôsobuje to, že konflikty nie je možné vyriešiť automatizovane. Proces spájania po dlhšom časovom úseku tak často zahŕňa spoluprácu všetkých zainteresovaných prispievateľov. Môže to byť proces trvajúci aj niekoľko hodín. Existuje možnosť metodickej integrácie zmien v krátkych úsekoch do spoločnej vetvy, avšak toto zas kvôli existencii centrálneho uzla vyžaduje výraznú mieru komunikácie medzi prispievateľmi pracujúcimi na jednej vetve.

V distribuovanom nástroji sa využíva možnosť odovzdania každej menšej zmeny okamžite po jej vykonaní. To dáva prispievateľovi lepší prehľad o zmenách, ktoré vykonal. Po zavedení jednotnej metodiky odovzdávania súborov dokáže prispievateľ tieto zmeny odosielať samostatne alebo s minimálnou asistenciou ostatných prispievateľov a manažérov podporných prostriedkov a vývoja. Vykonanú zmenu navyše nie je nutné okamžite odoslať do centrálneho úložiska, ale môže ju získať napríklad iba kolega, ktorý spolupracuje na vývoji danej časti funkcionality. Nástroj Git toto rieši oddelením akcie odovzdania zmeny (vytvorenie novej verzie vetvy) a odoslania zmeny. Ďalšou výhodou je napríklad možnosť získať súbory aj z lokálneho úložiska – takto si môžem overiť a získať funkcionalitu potrebnú pre môj ďalší postup bez toho aby som musel otravovať kolegu. V Gite sa to vykonáva akciou získavania (angl. *pull*), ktorá je opačná k akcii odoslania. Rozdelenie štandardnej akcie, *commitu*, známej z centralizovaných VCS, na tieto menšie akcie, má pozitívny vplyv na integritu hlavnej vývojovej vetvy – nachádza sa tam aktuálnejšia verzia aplikácie. To umožňuje všetkým programátorom pracovať so stabilnejším kódom a znižuje to čas potrebný na hľadanie chýb [2].

Samozrejme tento proces vyžaduje naozaj kvalitné komentovanie vykonaných zmien. Toto môže v tíme so slabšou spoluprácou predstavovať aj jeden z hlavných problémov distribuovaného prístupu – skúmanie konkrétnych zmien v množstve nič nehovoriacich odovzdaní môže predstavovať výrazné plytvanie času. U distribuovaných nástrojov existujú automatické nástroje automatizované usporiadanie množstva malých odovzdaní (*rebase* príkazy) [3], ktoré tento problém dokážu čiastočne riešiť. V tímovom projekte sa budeme sústrediť najmä na presnú špecifikáciu toho, kedy odovzdávať zmeny a ako by malo vyzerať dobré odovzdanie.

Kontrola verzus flexibilita

Centralizované nástroje poskytujú možnosť väčšej kontroly pri správe, odovzdávaní alebo spájaní vetiev. Okrem presnejšieho určenia oprávnení jednotlivých prispievateľov je

možné využiť tiež automatizované skripty na kontrolu správnosti a kompatibility vykonaných zmien, toto pri DVCS z dôvodu neexistencie centrálneho úložiska nie je tak jednoduchým spôsobom možné.

V malom nezávislom tíme s dobrou komunikáciu však nie je kontrola natoľko dôležitá, najmä ak je spojená s dobrou komunikáciou. Navyše pri použití distribuovaného systému, nie je až natoľko potrebná. Výnimkou je riadenie integrácie zmien, ktoré je najcitlivejším bodom VCS. V distribuovanom systéme je za predpokladu, že sa jedná o malý tím, je jedným z možných jednoduchých riešení využitie jedného úložiska ako kvázi centrálneho úložiska, kde budú prispievatelia odovzdávať všetky zmeny. Následne je možné centrálne riadenie spájania jedným alebo viacerými prispievateľmi – ostatní prispievatelia môžu o spojenie požiadať, ale až určení prispievatelia ho môžu vykonať.

Oprava chýb

Dôležitou vlastnosťou VCS je využitie histórie zmien na opravu chýb, kedy je potrebné zistiť v ktorej verzii táto chyba vznikla a aplikovať jej opravu na všetky verzie a prípadne aj ďalšie vetvy v ktorých sa vyskytuje.

Najjednoduchším spôsobom opravy chýb, ktorý sa využíva najmä v centralizovaných nástrojoch, kde na opravu chýb medzi vetvami a verziami vetiev neexistujú pokročilejšie nástroje, je ručná identifikácia a oprava chyby vo všetkých relevantných vetvách. Ručná kontrola je v centralizovaných nástrojoch jednoduchšia, pretože história zmien je oveľa lineárnejšia, ako pri DVCS.

V distribuovaných nástrojoch sú prítomné pokročilé nástroje na čiastočnú automatizáciu procesu opravy chýb. Každý konkrétny nástroj implementuje vlastný spôsob – používané metódy v nástroji Git sú napríklad *bisect* (identifikácia všetkých verzií vetvy s chybou) a *cherry picking* alebo *daggy fix* (oprava chýb) [3].

Je potrebné poznamenať, že potenciál pokročilých nástrojov DVCS v tímovom projekte naplno nevyužijeme, avšak existuje možnosť reálneho využívania nášho produktu a najmä jeho ďalšej úpravy pre iné podmienky (vytvárame systém obohatenej reality, ktorý nemusí byť vhodný do všetkých reálnych podmienok) a v tom prípade by použitie takýchto nástrojov už bolo aktuálnejšie.

Zálohovanie

Z dôvodu existencie lokálnych úložísk na počítačoch všetkých prispievateľov je u DVCS veľkou výhodou eliminácia centrálneho bodu zlyhania (angl. single point of failure), ktorý u CVCS môže pri niektorých druhoch softvérových aj hardvérových chýb spôsobiť problémy, často nie je potrebné ani zálohovanie úložísk, keďže zmeny sú väčšinou odosielané po malých častiach [4].

Využitie DVCS odbremeňuje prispievateľov od potreby zálohovania zmien, čo z pohľadu nášho tímu, najmä z časového hľadiska veľkou výhodou – nemusíme sa spoliehať na iné mechanizmy zálohovania.

Požiadavky na prispievateľov

Veľmi dôležitou vlastnosťou VCS sú jeho požiadavky na prispievateľov a to z viacerých hľadísk.

Zo stránky hardvérového vybavenia, v centralizovanom prístupe je potrebné zabezpečiť neustálu dostupnosť centrálneho úložiska, čo za určitých podmienok môže predstavovať výrazný problém. Pre SVN aj Git existujú internetové úložiská, avšak ich využitie pre citlivejšie dáta je otázne, navyše majú obmedzenia.

Zo stránky softvéru, keďže vývoj v tímovom projekte prebieha najmä na našich lokálnych počítačoch, je potrebné aby použitie všetkých funkcií VCS bolo bezplatné.

Zo stránky siete, použitie DVCS umožňuje prácu bez sieťového pripojenia, na druhej strane, ale u DVCS dochádza pri prvom pripojení vždy k replikácii celého úložiska, čo môže predstavovať väčšie množstvo prenesených dát a času.

S absenciou závislosti na pripojení na centrálny server sa zvyšuje plynulosť práce, keďže operácie odovzdania zmien, zobrazenia histórie alebo vrátenia predošlej verzie je možné vykonať lokálne [3].

Ergonómia

U výberu VCS netreba zabúdať na prácu s jeho rozhraním. Uvádzam rôzne výhody a nevýhody architektúr, avšak jedným z dôležitých bodov, najmä s ohľadom na tímový projekt, je aj zohľadnenie predchádzajúcich skúseností a preferencií členov tímu. Vo väčšej firme by výber jedného alebo viacerých prostriedkov zohľadňoval najmä existujúce firemné postupy a používané prostriedky. Keďže v našom projekte sme začali vo vývoji úplne od začiatku. toto malo menšiu váhu. Preto sme mohli výraznejšie zhodnotiť preferencie jednotlivých členov. V našom tímovom projekte nemala väčšina členov tímu, okrem mňa, žiadne skúsenosti s VCS. Keďže všetci kolegovia používajú primárne OS Windows a sú zvyknutí na jeho ovládacie prvky, musel zvolený prostriedok poskytovať intuitívne grafické rozhranie, čo výraznejšie nahrávalo SVN s jeho kvalitnými implementáciami rozhraní integrovanými do prostredia Windows Explorer, ako napríklad TortoiseSVN. Nakoniec sme túto myšlienku zavrhli, keďže aj pre distribuované prostriedky existujú veľmi dobré rozhrania, menovite napríklad rozhranie v aplikácii GitExtensions.

Hybridné VCS

Špeciálnu zmienku si zaslúžia systémy, ktoré kombinujú distribuované a centralizované nástroje [5]. Použitie takéhoto systému umožní súbežné odosielanie zmien napríklad do úložiska Git a do úložiska SVN.

Tieto nástroje sa využívajú najmä kvôli používateľom so skúsenosťami s SVN a podľa mojej mienky, výhody, ktoré poskytuje SVN, nie sú dôvodom štúdia oboch nástrojov a duplicitného odosielania zmien.

Jednou možnosťou, ktorú možno zvážime je využitie niektorého nástrojov v neskoršej fáze tímového projektu. Jednou z požiadaviek na náš systém je totiž aj umiestnenie zdrojových súborov v úložisku SVN, po ukončení projektu.

Záver

Analyzoval som vlastnosti centralizovaných a distribuovaných prostriedkov na manažment verzií a vývoj ich použitia v čase. Napriek tomu, že som popísal najmä

množstvo výhod DVCS, nie je možné uviesť jednoznačný záver, platný pre každý malý projekt. Ale ak zvážime štandardný projekt, bez závislosti na technológiách a preferencií jednotlivých ľudí, distribuovaný systém sa ukazuje ako výhodnejší. Pre konkretizáciu uvediem hlavný dôvod výberu systému Git, pre náš tímový projekt.

Hlavným dôvodom použitia DVCS je flexibilita takýchto systémov, ktorá vyplýva takmer zo všetkých uvedených vlastností. Zníženie nárokov na komunikáciu, jednoduchšie riešenie konfliktov, pokročilé možnosti opravy chýb, nenáročnosť na používateľa úložiska po zadefinovaní správnych postupov – toto všetko sú faktory, ktoré prispievajú k tomu, aby používatelia venovali viac času práci na projekte a menej riešeniu problémov.

Použitá literatúra

1. 2010. The version control timeline.
   http://codicesoftware.blogspot.com/2010/11/version-control-timeline.html
2. DE ALWIS, B.: Why are software projects moving from centralized to decentralized version control systems? In: CHASE '09 Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering , 2009, s. 36-39
3. O'SULLIVAN, B.: Making Sense of Revision-control Systems In: Queue - File Systems, vol. 7, 2009, no.7, pp. K 42-51.
4. Clatworthy, Ian. 2007. Distributed Version Control Systems - Why and How. ianclatworthy.files.wordpress.com/2007/10/dvcs-why-and-how3.pdf
5. Naganov, Mikhail. 2008. Hybrid Version Control
   http://codemate.wordpress.com/2008/05/29/hybrid-version-control/

Annotation

Should a distributed control system be used in a small project?

This essay covers comparison of version control systems. There are two main architectures these systems implement – centralis ed and distributed architecture. For each of them, there are many mature tools with different attributes and features. Essay compares features of tools of both architectures, covers history of their development and usage and their current state. The two tools used for reference are Subversion and Git. Essay considers the usage of those tools in a software project. It arguments the use of a distributed tool for Team project - Digital Theatre. After considering the needs of our team, it marks the flexibility od development process which is offered by distributed tools as the most significant feature. It saves time and with that all the other resources required.