GIT som alternativ till CVS/SVN i agila utvecklingsmiljöer

Kristofer Jacobson, Patrick Ivarsson

Abstrakt

En studie om versionshanteringssystemet Git och om möjligheten att använda det som alternativ till CVS/SVN. Studien baseras på ett experiment utfört på ett programvaruprojekt. Projektet är en kombination av två kurser som ges av Lunds Teknologiska Högskola. Frågan som ska besvaras är om Git är enkelt nog och har tillräcklig funktionalitet för att ersätta CVS/SVN som är kursledningens val av system.

1 Introduction

Open Source projekt är något som med åren har blivit väldigt populärt. För många kommersiella program finns idag open source alternativ. Webbläsare, ordbehandlare och programmeringsspråk är alla exempel på detta. Ett exempel på detta är Microsoft Internet Explorer som år 2002 användes av 85,8% av alla internetbesökare [1] men som idag har fått lämna plats åt bland annat Mozilla Firefox och Google Chrome som båda är open source. Firefox och Chrome står nu tillsammans för 72,4% av alla internetbesök och det ökar för varje månad.

För att kunna arbeta effektivt i ett open source projekt behövs en typ av versionshanteringssystem (VCS) som organiserar och för historik över projektfilerna. Vilket VCS är då vanligast inom open source. Microsoft har undersökt frågan i en enkät som över 1000 utvecklare besvarat [2]. Undersökningen visade att år 2011 var Git det mest populära verktyget oavsett vilken plattform som användes. Samma undersökning hade genomförts året innan och visade att Gits popularitet hade ökat kraftigt. Vad detta beror på är vad vi vill ta reda på i den här djupstudien.

När man läser till Civilingenjör i Datateknik på Lunds Tekniska Högskola går man en kurs som heter Programvaruutveckling i Grupp (PVG) där ett mjukvaruprojekt utvecklas av en grupp med 8-10 utvecklare under en sju veckors period. Som VCS har de kursansvariga valt att använda systemet Apache Subversion (SVN). Gruppen leds av två coacher som hjälper gruppen att komma framåt och utvecklas. I vår roll som coacher väljer vi att istället använda oss av Git som VCS och se hur bra det fungerar. Våra tidigare erfarenheter av VCS är med SVN så vår studie kommer gå ut på att först lära oss Git och sen försöka lära ut det till gruppen. Kan Git göra samma jobb som SVN lika bra eller kanske bättre?

Studien är strukturerad med en kort förklaring av hur vår situation ser ut först följt av en beskrivning av hur gruppens kurs samspelar med coachernas kurs. Därefter följer en kort beskrivning av vad ett VCS bidrar med samt en översiktlig beskrivning av de system som behandlas i studien. Efter det går vi igenom hur studien har genomförts för att sen avslutas

med en resultat- och slutsatsdel. För att lättare följa med och förstå texten i studien finns det i Sektion 6 en terminologitabell där begrepp förklaras kortfattat.

2 BAKGRUND

Här följer en beskrivning över hur de två kurserna samspelar samt lite mer ingående hur de olika VCS fungerar och skillnader mellan dem.

2.1 Vår situation

Projektet som vi ska utföra vår studie på är ett delat projekt mellan två kurser. Den ena kursen går ut på att utveckla ett mjukvaruprojekt över en sju veckor lång period. Den andra kursen går ut på att gå in i en ledarroll och coacha en av grupperna under deras projekt. Projektet ska simulera ett verkligt utvecklingsprojekt med en kund och Extreme Programming (XP) [3] som utvecklingsmetod. Gruppen kommer att använda sig utav utvecklingsverktyget Eclipse och programmeringen sker i Java. Gruppen har innan projektets början fått lära sig XP och SVN genom föreläsningar och laborationer. De har fått lära sig praktikerna som används inom XP [3]. För att testa koden under laborationerna kommer JUnit att användas som verktyg.

2.2 Versionshantering

I mjukvaruprojekt behöver utvecklarna dela filer och dokument med varandra. I ett mindre projekt med ett fåtal utvecklare går det att komma undan med att skicka filer via email och att tillsammans sitta och sammanfoga dokument. Detta kan med tiden bli mycket tidskrävande och komplicerat med många versioner av samma dokument i omlopp. Det är i ett sådant läge som ett VCS kan användas och förenkla arbetet. Ett VCS tillåter att ett godtyckligt antal utvecklare jobbar med samma projekt och delar alla projektfiler på en plats, även kallat ett repositorie (hädanefter refererat till som repo). När en fil ändras sparas informationen om vad som har ändrats samt att filen får ett nytt versionsnummer. På detta vis håller systemet reda på vilken som är den senaste versionen av en fil samt möjliggör för utvecklarna att återställa en tidigare version av filen.

En ytterligare funktion i VCS är möjligheten att slå samman två versioner av samma fil, även kallat merge. Om flera utvecklare arbetar i samma version av en fil samtidigt och försöker spara den i det gemensamma repot kommer en sammanslagning att behöva göras. Vid enklare ändringar i filen kan ett VCS bidra med att automatiskt slå samman filerna. Däremot om ändringarna är komplicerade eller på samma ställe i filen markerar systemet konflikten och ber utvecklaren att lösa konflikten.

I projekt kan det ibland finnas behov av att utifrån en punkt utveckla systemet parallellt men åt olika håll. Med ett VCS kan man då skapa en förgrening av programmet, även kallad branch. Ett exempel på detta är om utvecklargruppen förbereder för en release av mjukvaran. Då behövs en stabil version av mjukvaran där funktionaliteten är helt färdigställd. För att inte hela projektet ska stanna upp för detta kan man välja att göra en branch när projektet är i ett stabilt läge. Releasearbetarna kan då

arbeta på den nya branchen utan risk för att ny, halvfärdig kod följer med i releasen. De andra utvecklarna kan fortsätta att arbeta som tidigare och vid ett senare tillfälle kan man välja att återigen slå ihop branchen med huvudprojektet.

2.3 CVS/SVN

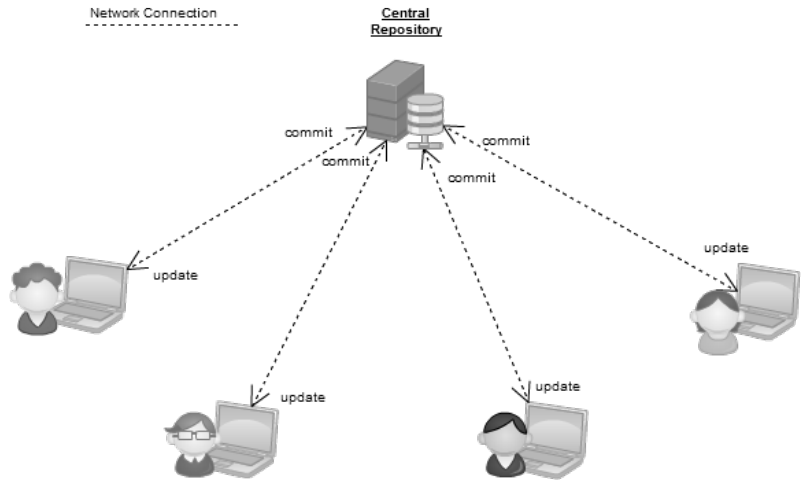

Fig. 1: Visar upplägget i ett centraliserat versionshanteringssytem. Alla användare kopplar upp sig mot ett gemensamt repo

Datavetenskapsinstitutionen har valt att lära ut ett VCS kallat Concurrent Versions System (CVS) till PVG-projektet. CVS lanserades år 1990 och är ett centraliserat VCS (CVCS). Att det är centraliserat innebär att det har ett centralt repo som alla utvecklare kopplar upp sig mot (se Fig. 1). I repot finns alla filer samt information om alla ändringar som gjorts. När en utvecklare checkar ut projektet skapas en lokal kopia av det på utvecklarens dator. Därefter görs alla ändringar lokalt och när koden är klar laddar utvecklaren upp den i repot och löser eventuella merge-konflikter. Skulle ny kod bli uppladdat på repot går det att göra en uppdatering och på så sätt få de senaste ändringarna i sitt lokala repo. Allt detta kan göras via ett inbyggt användargränssnitt(GUI) i Eclipse.

CVS utvecklas inte längre men det finns många CVS-kloner som har samma funktionalitet som CVS. Klonerna är främst utvecklade för att lösa buggar och lägga till funktioner som saknades i den sista släppta versionen av CVS. Subversion (SVN) är en av dessa kloner och är systemet som används under PVG-projekten. Funktionerna som finns i CVS finns även i SVN och fungerar analogt. Även för SVN finns det GUI att installera till Eclipse för att grafiskt hantera projektfilerna.

2.4 Git

För att hantera källkoden till Linuxkärnan valde Linus Torvalds att utveckla sitt eget system [4]. Tidigare hade ett system kallat BitKeeper används men när projektets gratislicens gick ut behövdes ett nytt gratis VCS. Enligt Torvalds kunde inget av de dåvarande gratisalternativen leverera den funktionalitet som behövdes så han valde att istället utveckla sitt eget. Det beslutet resulterade i Git.

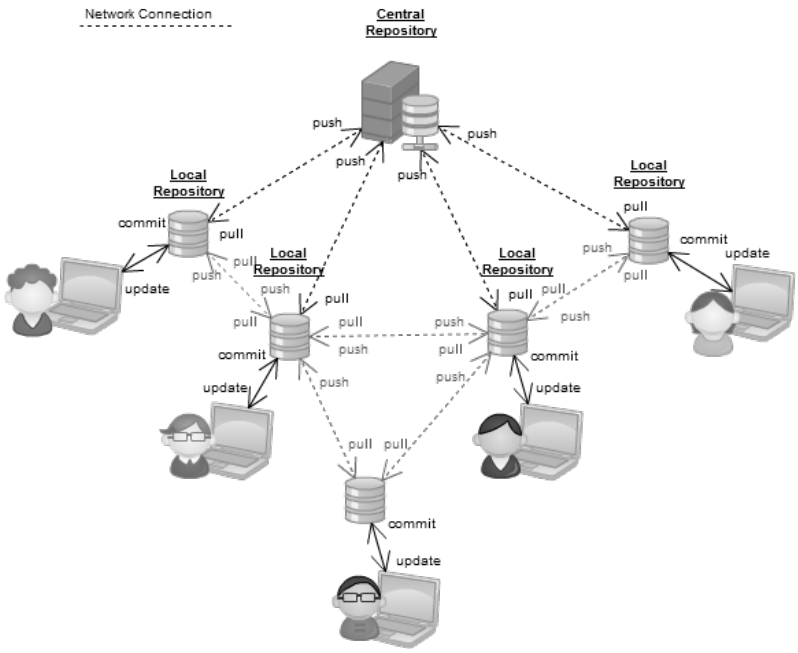

Fig. 2: Visar upplägget i ett decentraliserat versionshanteringssytem. Alla användare kan koppla upp sig mot ett centralt repo men kan också dela filer mellan varandra

Till skillnad från CVS och SVN är Git ett decentraliserat VCS(DVCS). Detta innebär att det inte finns ett gemensamt repo som alla utvecklare använder sig av. Istället delar utvecklarna filerna mellan sig (se Fig. 2). Alla utvecklare har ett eget lokalt repo med alla filer och all filhistorik och när man behöver den senaste versionen av koden hämtar man den från de andra utvecklarna. När en utvecklare själv har gjort ändringar kan han trycka ut den nya koden till de andra utvecklarna. Git kan även användas som ett centraliserat VCS genom att alla hämtar från och trycker upp uppdateringar på en server. En av fördelarna med ett decentraliserat system är att två personer kan arbeta med en del i projektet och dela experimentell kod mellan sig utan att det påverkar resten av teamet.

Arbetsflödet i Git påminner mycket om det som används i CVS/SVN. Man arbetar med det lokala repot på samma vis som man arbetar med ett repo i CVS/SVN. Varje gång man gjort ändringar i koden gör man en commit till det lokala repot. När koden är redo att skickas ut till de andra utvecklarna trycker man ut koden till dem, vilket kallas att göra en push. Genom att commita till sitt eget repo får man även lokalt en historik som är enkel att navigera i då man alltid kan återställa filerna till en tidigare commit.

Git kan användas både genom terminaler och genom grafiska GUI. Det finns både plugins och fristående program för att hantera sitt repo. Med installationen av Git medföljer även två simpla GUI som kan användas för att skapa och hantera branchar. Till Eclipse finns pluginet EGit vilket medför att man kan hantera sitt repo inifrån Eclipse.

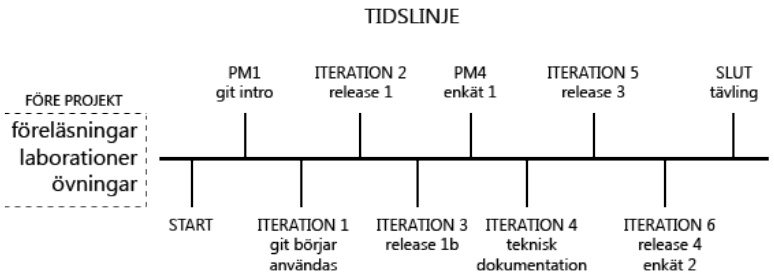

Fig. 3: Visar tidsupplägget i projektet med viktiga händelser (PM = Planeringsmöte)

3 UTFÖRANDE

3.1 Upplägg

Vi lade upp arbete så att vi inte skulle lägga för mycket fokus på att använda Git som VCS då det är mycket ny information. Vårt team som vi skulle utbilda har inte varit i någon kontakt med Git innan men har inte heller några djuprotade arbetssätt med något annat VCS. Vi har delat upp upplägget för gruppen i tre faser. Första fasen är rent teoretisk och vi försöker hålla den på så grundläggande nivå som möjligt. Vi måste ständigt hålla i åtanke att för de flesta i gruppen som vi introducerar Git till har aldrig arbetat med något projekt av denna storlek innan. Detta innebär att de har fullt upp med mycket annat och ger vi dem för mycket teori så kommer den ändå inte befästas. Gruppen har redan en god insikt till vikten av att kunna arbeta parallellt och vill lära sig mer om det.

Då vi också var nya till Git bara någon månad tidigare, så förstår vi deras situation och har lättare för att skriva ner en kort guide på de mest nödvändigaste kommandona för att kunna arbeta med vårt projekt [Appendix A]. Denna guide delade vi ut på första planeringsmötet så alla kunde komma igång med att klona ner och arbeta med koden på första långlabben. Vi tilldelade även en i gruppen att läsa på mer teori samt att testa lite själv, på så vis har de en VCS expert inom gruppen då det kan kännas lättare att prata med varandra.

Fas två uppstod vid första labben och för det flesta var därmed deras mer teoretiska lärande färdigt. Här efter fick de uteslutande lära sig av vad de såg och gjorde. Det vill säga att de såg vilka fördelar Git har och i vilka situationer man bör använda sig av olika funktioner. I fas två var gruppen fortfarande väldigt outbildad vilket ledde till att vi coacher fick gå runt och hjälpa dem vid de mer komplicerade kommandon och händelser. Till att börja med så behövde de inte lära sig dessa alls utan bara kunna det som gavs till dem i fas ett. Allt efter som så fick de själva skriva in kommandona och då vi stod bakom och instruerade och endast kontrollerade så att allt gjordes rätt. Vissa i gruppen som inte ville lära sig så mycket om Git stannade i denna fas.

Efter några veckor av labbande anpassade sig fler och fler i gruppen efter arbetssättet och de tillkallade oss inte då de redan visste vilka kommandon som skulle skrivas. De var då som fas tre inleddes med mycket mer självgående användande utav Git. De visste själva vad som skulle göras och kom själva med förslag när de ville brancha för att till exempel arbeta i tätt samarbete med en annan grupp, t.ex med argument för att uppnå ett

bättre resultat snabbare. För sådana strukturella beslut som om det skull gå snabbare att två grupper branchar ut och arbetar tillsammans godkännde vi det först. Detta gjorde vi inte för att de tekniskt skulle ha svårt med det utan att vi ville utvärdera om det skulle finns någon vinning i det.

Det var för det mesta väldigt säkert att låta grupperna själva experimentera så fort de hade lärt sig grunderna eftersom Git är väldigt robust och man kan nästan alltid återskapa ändringarna. Att återskapa ändringar var speciellt endkelt för oss i Git då vi hade minst fem stycken lokala repon med komplett historik samt en server. Då de följer den utdelade guiden och ofta commitar upp till sina egna lokala repon så kan man återställa så gott som allt.

3.2 Planeringsmöten

I slutet av varje långlabb fick studenterna skriva ett par meningar om den specifika XP practice som de hade fokuserat på. Vi lade även till att de ska skriva om saker och ting som gick bra och mindre bra med gruppen och vårt arbete, där vi belyste att om det var några problem med Git så skulle de gärna ta upp dem då vi coacher lättare skulle kunna ta tag och åtgärda dem. Dessa punkter togs sedan upp på våra schemalagda planeringsmöten för diskussion och reflektion. Vi kom med förslag till lösningar på hur vi skulle åtgärda dessa och helst så vi attnågon i gruppen hade något sätt. Ett exempel på en sådan diskussion var om vi behövde skriva en ytterligare guide för att sätta in gruppen i lite svårare funktioner eller om de kunde lära sig dem som beskrivet ovan i 3.1

I slutet av planeringsmötena delade vi ut spikes till studenterna och om det hade varit något som flera uppfattade som svårt eller problematiskt inom Git så spred vi kunskapen genom att någon fick göra en grundligare undersökning varför det blev så och hur vi skulle kunna göra i fortsättningen. För det absolut mesta så visste vi coacher redan svaret men vi tryckte hårt på att teamet skulle besitta all kunskap själva och inte vara beroende utav oss coacher. Det kunde även förekomma flera tillfällen då gruppen tyckte att det var lättare eller smidigare att fråga varandra istället för oss coacher vilket stödjer att det skulle vara bättre med att gruppen vet allt som de borde.

Det var viktigt att vi alla i gruppen också skulle kunna ha tillgång till koden hemma för att kunna göra eventuella spikes så som till exempel kod granskning eller undersöka förslag till hur det är bäst att vidareutveckla en viss gren av programmet. För att detta skulle fungera så skrev vi ihop en kort guide på den gemensamma trac-hemsidan. Vi lade också upp ett shellscript som konfigurerar Git till deras inställningar utan att studenterna själva behöver skriva några kommandon.

Under de senare veckorna då gruppen arbetade mer självständigt med Git så skapade vi brancher även för de spikes som kräver något ur repot men även ska uppdatera repot till en ny version när spikesen var klara. Kursledningen hade beslutat att det inte fick skrivas någon kod som trycks upp i repot mellan långlabbarna men gruppen fick till exempel uppdatera JavaDoc eller teknisk dokumentation. Det var viktigt för oss att dessa spikes inte blev för många och att de inte jobbade direkt på vår huvudbranch (master) utan att vi innan långlabben istället kunde

merga ihop brancherna. Dessa merges skulle alltid gå automatiskt då det inte skulle vara flera brancher med samma ändringar, även om detta inte alltid var fallet. Till exempel så kunde vi en vecka ha en spike med JavaDoc uppdateringar och en spike med att uppdatera manualen. Dessa två blev således brancher för att kunna arbeta hemifrån utan att störa andra i gruppen som skulle ladda ner koden då de skulle fått en halvt färdig skriven JavaDoc eller manual. Direkt då långlabben startade tog vi en utav brancherna och mergade ihop den med master. Detta blev i Git endast en fast-forward och inga merge konflikter då det bara var en person som hade gjort ändringar. Sedan gick vi till den andra spiken och gjorde det samma. Git löste denna merge-konflikten automatiskt då den tekniska dokumentationen inte hade gjort några ändringar i Java-filer och JavaDocen inte skulle ha gjort några ändringar i den tekniska dokumentationen.

Ledningen för kursen hade även godkänt att större refaktoriseringar utav hela programmets arkitekt kunde vara bra att göra hemma då det för övrigt var produktionsstopp och annars skulle bli väldigt stora och svåra merge konflikter. Då vi hade dessa spikes var det viktigt att den spikegruppen var den enda som arbetade med koden och att alla i gruppen hade en god förståelse på vilka ändringar som skulle göras. Alla i gruppen skulle även veta hur systemet skulle komma att se ut efteråt. Det var viktigt att lägga detta arbete på en egen branch då arbetet kunde göras stegvis med flera commits under tiden. Att göra refaktoriseringar stegvis var inte bara bra för att andra lättare skulle kunna integrera sin kod utan att hela gruppen skulle lättare kunna gå tillbaka om något inte blir bra. En annan mycket viktig aspekt på varför refaktoriseringen inte fick ske på huvudbranchen var om den inte skulle lyckas bli klar så skulle det inte vara några problem och alla kunde bara arbeta på som vanligt. Skulle refaktoriseringen sedan bli klar någon timme in under labben så kan man fortfarande merga ihop den då. Det medför dock att man måste vara noggrann med den funktionalitet som har tillkommit under tiden.

3.3 Git-konfiguration

Git är mycket fritt och kan användas på många olika sätt där ingen är bunden att ställa sig i en hierarki. Vi valde dock att arbeta med Git på ett lättförståeligt sätt där alla hämtade(pull) det senaste från en server. På detta vis löste vi det krångliga som kan uppstå med att hämta från flera repon. Alternativet skulle vara att när en grupp har blivit klar med en story så hämtar alla ner hans version och mergar ihop med sitt eget.Det skulle dock bli lite rörigt i vår situation då vi enligt XP anda gjorde parbyten och vilket snabbt slutar med att man inte ha någon aning om vilken dator och inloggning man sitter på. Ett annat vanligt sätt att arbeta på är att ha en så kallad Gatekeeper [10] som alla hämtar ifrån. När ett par av utvecklare är färdiga med en story så säger man till Gatekeepern som då hämtar från deras, kontrollerar ändringarna och kanske testkör programmet. När han är nöjd säger han till alla att det finns en ny version hos honom som alla ska hämta. Detta sätt är för visso mer säkert och röd kod sprider sig inte när någon har gjort fel. Dock så tyckte vi coacher att detta inte känns så agilt och den snabbheten som är så viktig går lite förlorad. Det

skulle även betyda att teamet skulle vara beroende av denna enda "bättre vetande" personen som skulle granska alla arbeten, vilket vi inte heller tyckte går enligt XP. Vi ville med vår grupp uppnå en väldigt självgående sammansättning där vi coacher inte skulle ha någon avgörande roll, och vi vill även att gruppen skulle fungera vi eventuella bortfall. Vi valde därför det upplägget som vi gjorde och delade istället ut en task på att något annat utvecklingspar granskar en story innan den anses helt färdig.

3.4 Enkätundersökning

För att mer konkret kunna analysera vår utveckling och ge oss feedback om hur bra materialet och vår undervisning är så har vi lämnat ut en enkät [Appendix B]. I enkäten adresserade vi de olika faserna och se hur framgångsrika de har varit. Vi har ställt enkla frågor för att se hur lätt det var att ta till sig teorin i fas ett och att komma igång med ett naturligt arbetsflöde i fas två. Vi undersökte huruvida de känner sig bekväma med Git och får djupare förståelse för det genom terminalen i steg tre, samt huruvida de skulle kunna tänka sig att arbeta med det i framtiden för en eventuellt fas fyra. Vi gav ut enkäten två gånger under kursens gång för att kunna analysera några förändringar. Den första gången efter cirka halva tiden då de flesta i gruppen var i mitten eller slutet av fas två och den andra gången vi undersöker var på sista mötet då allt arbete med Git var avklarat.

4 RESULTAT

Här följer de resultat som vi har som vi samlat in under studien. De innehåller dels resultaten från den enkät som delas ut vid två tillfällen till gruppen. Vi tar även upp våra egna observationer över hur inlärningen har utvecklats med tiden.

4.1 Git i agila utvecklingsprojekt

Av de observationer som vi har kunnat göra iform utav coacher har vi kommit framtill att vi ser tydliga framsteg både i förståelsen för hur ett projekt av större storlek fungerar och vilka nya problem som uppstår. Framför allt så har vi observerat att studenterna ser stor nytta av ett versionshatneringssystem och den gemensamma uppfattningen är att Git fungerar väldigt bra. Vi själva tycker att problemen kring verktyget är få och förhållande små. Då vi pratar med andra grupper, som läser samma kurs, så framgår det tydligt att vi har mycket mindre problem. Detta tror vi dels för att Git har mycket bättre merge-vertyg och att vi inte använder oss utav något eclipse plugin. Eftersom vi använder git i terminalen får vi även en effekt av att man tänker till en extra gång innan man gör något. Vi tror att det är bra och tror inte att det leder till att man sprider sin kod mer sällan utan att utvecklarna tänker till innan och inte sprider felaktig kod lika ofta.

Vi ser att git växer mer och mer i de framtidsrapport vi har studerat och en intressant tanke för kursen skulle vara att ersätta CVS/SVN med Git. Gruppen utvecklar i Eclipse som IDE, och den senaste versionen av programmet kommer med en mycket uppskattad version av EGit/JGit. Vi

coacher har testat detta och ser fördelarna med att använda ett grafiskt plugin för att underlätta för utvecklarna. Den förberedande laborationen, se figur 3 , skulle då inrikta sig på att lära ut Egit. Problemet på denna lösning är att Instutitionen förnärvarande endast levererar en Eclipse version som är två versioner bakom.

4.2 Utveckling

4.3 Enkätundersökning

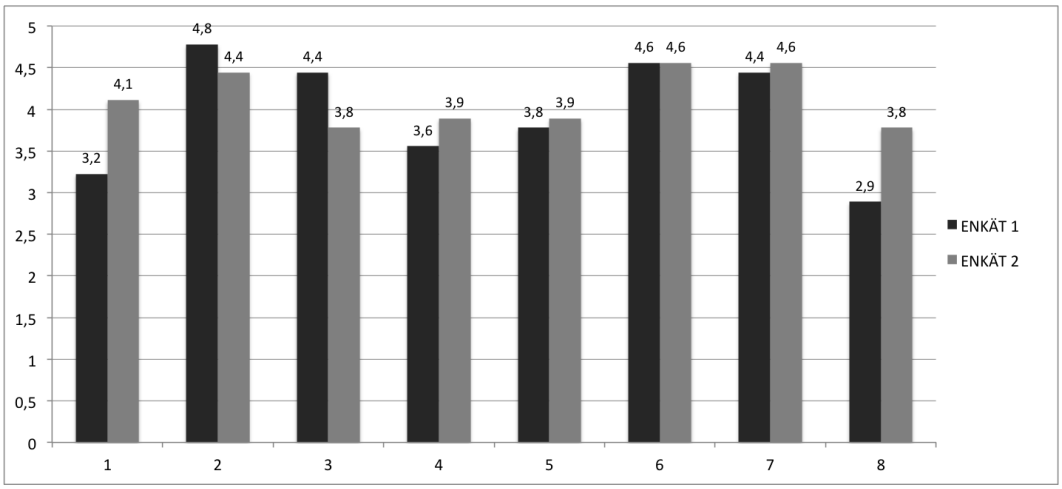

Fig. 4: Visar resultaten för den enkät som delats ut vid två tillfällen under kursen (se Appendix B). Första enkäten(mörkgrå) delades ut tre veckor in i projektet och den andra(ljusgrå)sex veckor in

Här följer de frågor som fanns med i enkäten som delades ut samt en kort diskussion om resultaten.

4.3.1 Snabbguiden gjorde det enkelt att förstå grunderna i Git

Resultatet visar att de tyckte guiden var ok. På första enkäten är detta det näst lägsta resultatet vilket sedn ändrades mycket till den andra enkäten. Vi tror detta mest beror på att de känner sig säkrare med Git och att de inte kommer ihåg den första förvirringen lika väl. Vi är nöjda med resultatet då det var en av de punkter som skulle vara svårast.

4.3.2 Det är lätt att få igång ett bra workflow (liknande update, update, update, commit för SVN)

Väldigt högt resultat. Det var det högsta vi fick på första enkäten vilket kan verka en aning märkligt då frågan inte är helt skild från den första. Resultatet gick ner något till den andra enkäten, vilket kan bero på att de börjar använda svårare funktioner.

4.3.3 Att använda Terminalen gör att man förstår bättre hur Git fungerar

Gruppen har även här varit nöjda. Denna fråga har den största sänkningen mellan enkät ett och två. Vi tror att denna sänkningen även här beror på att de använder Git mer avancerat då vissa av dessa funktioner presenterar information man måste ta till sig via terminalen. Vi anser att det är på denna punkt ett grafikt gränssnitt har en stor fördel över terminalen.

4.3.4 Git hanterar mergkonflikter på ett bra sätt

Resultatet speglar ganska bra stämningen i gruppen. Mergekonflikter är alltid tråkigt och vid vissa fall blir folk irriterade på att det borde kunna lösas automatiskt. Samtidigt så får gruppen höra av andra studenter att alternativen inte verkar vara bättre, snarare motsatsen.

4.3.5 Git fungerar bra tillsammans med Eclipse

Förvånandsvärt högt resultat här. Vi valde att inte använda en integrerat pluggin då vi övervägde andra aspekter så som robusthet och förståelse. Vi hade dock förväntat oss ett lägre resultat här efter som man i eclipse själv måste updatera projektet varje gång. Vi tror att det relativt höga resultatet även här kommer för att de har hört av andra studenter att alternativa plugin till SVN har krånglat mycket.

4.3.6 När vi stöter på problem med Git kan ni som coacher ofta lösa det

Folk har bra tilltro till oss som coacher som har bestått genom kursen. Detta visar om något att man inte behöver lång upplärningstid för att lära sig Git då vi coacher var nya till det bara någon månad innan.

4.3.7 Mitt helhetsintryck av Git är att det är ett bra versionshanteringssystem

Ett högt resultat på helhetsintrycket gör att vi är nöjda med valet utav git och rekommenderar det för liknande situationer och framför allt för kursen ifråga.

4.3.8 Jag kommer fortsätta använda Git som versionshanteringssystem

Angående om de skulle använda Git i forstättningen är det lite olika. Ingen har varit med och sett hur man initierar och skapar starten. Vissa har även blivit lite rädda då vi använde oss utav ett eget skrivet shell-script som de ser som mycket svårt. Ökningen till den andra enkät undersökningen tror vi mycket beror på att de känner att de kan hantera svårare funktioner och mer bekväma med de funktioner de redan använder. När de känner att de kan lära sig mer saker inom Git så minskar rädslan för det okända.

4.4 Gruppens kommentarer

I enkäten som gruppen fick fylla fanns även rutor för att fylla i vad de tycker har varit bra och mindre bra med att använda Git. Här följer en summering av deras kommentarer:

4.4.1 Vad som fungerade bra

De flesta i gruppen fann det relativt enkelt att förstå grunderna i Git. De tyckte att det kändes stabilt då inga krasher uppkom. Vad som också gillades var att det är svårt att göra destruktiva fel i Git, dvs Git märker om man använder kommandon fel eller i fel ordning och visar varningar. Detta har motverkat små fel som kan ge stora konsekvenser. När det gällde att hantera merge-konflikter i koden fick systemet bra kritik och det var många som kommenterade att det löste många mergar utan att användaren aktivt behövde göra något. Gruppen kände allmänt att de hade fått tillräckligt med stöd från coacherna.

4.4.2 Vad som kunde fungerat bättre

Det mest delade svaret på vad som kunde bli bättre var hanteringen av filer som inte ska synkroniseras med repot. Det uppkom en del problem med binära filer som hamnade i repot och ställde till med många små problem. De tyckte även att automatiska merge-konflikter kunde fungerat bättre i textdokument t.ex. JavaDoc där det vid vissa tillfällen uppkom, vad gruppen tyckte var, onödiga merge-konflikter. Till sist var det ett antal i gruppen som önskade att det hade funnits ett grafiskt användargränssnitt i form av ett plugin till Eclipse.

4.4.3 Coachernas reflektioner

Efter att ha läst igenom och sammanställt gruppens individuella kommentarer kommer vi till slutsatsen att vi är nöjda med studien. Det var övervägande positiv kritik från gruppen och det mesta av de problem som beskrevs kan på ett eller annat sätt åtgärdas.

Önskemålet om ett grafiskt användargränssnitt kan enkelt uppfyllas med pluginet EGit till Eclipse. Anledningen till att detta inte användes var att det inte kunde installeras på ett enkelt sätt på skolans datorer. I den senaste versionen av Eclipse är detta plugin dessutom inbyggt och integrerat med användargränssnittet.

Problemet med att filer som inte skulle ligga på repot hamnade där ändå kan vi som coacher ta på oss. I Git definierar man en fil med olika mappar, filtyper och filnamn som ska exkluderas från repot. När vi skrev ihop den filen misstolkade vi hur mappar skulle definieras. Efter att problemet uppdagats och felet åtgärdats fungerade allt bra.

5 SLUTSATS - GIT SOM STANDARD I XP PROJEKT

Vi har under denna kurs och studie kommit fram till att Git fungerar utmärkt som versionshanteringssystem i agila utvecklingsmiljöer. Vi har också fastslagit att det är möjligt att introducera Git, om även på ett mycket grundläggande plan, till en grupp studenter tillika utvecklare som aldrig arbetat med det innan. Verktyget har överlag fått god till mycket god kritik av gruppen. Vi visar härmed att Git skulle vara ett bättre alternativ än nuvarande SVN för kursen med bättre framtidsutsikter vilket är viktigt för blivande Civilingenjörer.

6 TERMINOLOGI

Uttryck	*Betydelse*
Versionshanteringssystem (VCS)	System som hanterar historik för källkod och dokument.
Repositorie (Repo)	Server där projektets dokument, filer och historik lagras.
Merge	När man slår ihop två versioner av samma dokument eller fil.
Branch	En kopia av projektet som utvecklas för sig själv, parallellt med projektet.
Centraliserat VCS (CVCS)	VCS där alla utvecklare kopplar upp sig mot ett gemensamt repositorie.
Decentralisert VCS (DVCS)	VCS där alla utvecklare kan välja att koppla upp sig mot ett gemensamt repositorie eller att dela projektfiler mellan sig.
Concurrent Versions System (CVS)	Är ett centraliserat VCS som idag inte utvecklas längre.
Subversion (SVN)	System som bygger på CVS men som fortfarande utvecklas aktivt.
Git	Är ett decentraliserat VCS.
Java	Är ett programmeringsspråk som används i studien.
Commit	Termen i CVS/SVN för när en utvecklare har ändrat i en fil och lägger upp ändringen i VCS.
Push	Term för när en utvecklare som använder Git trycker ut sitt repositorie till andra utvecklare.
Update	Används i CVS/SVN när en utevecklare hämtar hem de senaste versionerna av projektet.
Pull	Samma som Update för CVS/SVN men för Git.
eXtreme Programming (XP)	Utvecklingsmetod framtagen av Kent Beck.
Story	När en funktion ska utvecklas i XP skrivs denna som en story som beskriver vad som ska göras.
Task	När en story delas upp i mindre problem blir dessa tasks.
Spike	Egentid för att experimentera fram lösningar eller ideer.
Refaktorisering	När kod förenklas och förbättras genom att minska komplexitet och göra saker mer tydliga.
Gate Keeper	Vid användande av Git kan alla nya ändringar skickas till en person som granskar och därefter godkänner eller avslår ändringarna.
JavaDoc	Dokumentation som beskriver en Java-klass och alla dess publika metoder.
JUnit	Program som kan användas för att skriva enhetstester för Java-program.
Trac	En Wiki hemsida som gruppen använder för att strukturera sitt arbete.

REFERENSER

[1] Browser Statistics. W3Schools.com. http://www.w3schools.com/browsers/browsers_stats.asp.(February 2012)

[2] Open Source Developer Preferences Survey. Microsoft Developer Network (MSDN). http://blogs.msdn.com/b/codeplex/archive/2011/07/11/survey-results-open-source-developer-preferences-june-2011.aspx (June 2011)

[3] Beck, K., *Extreme Programming Explained: Embrace Change*. Reading, Massachusetts: Addison-Wesley, 2000.

[4] McMillan, R. *After controversy, Torvalds begins work on "git"*. http://www.infoworld.com/t/platforms/after-controversy-torvalds-begins-work-git-721. (April 2005)

[5] Bryan O'Sullivan. *Making sense of revision-control systems*. Commun. ACM 52, 9. (September 2009)

[6] Yip, Chen & Morris. *Pastwatch: a Distributed Version Control System*. MIT Computer Science and AI Laboratory. (May 2006)

[7] A. Mockus.*Amassing and indexing a large sample of version control systems: Towards the census of public source code history*. Avaya Labs. Res. (May 2009)

[8] Hedin, Bendix & Magnusson. *Teaching extreme programming to large groups of students*. Elsevier. (2005)

[9] Asklund, Bendix & Ekman. *Software Configuration Management Praticies for eXtreme Programming Teams*. Department of Computer Science, Lund Institute of Technology. (August 2004)

[10] Brian Berliner and Nayan B. Ruparelia. 2010. Early days of CVS. SIGSOFT Softw. Eng. Notes 35, 5 (October 2010), 5-6. DOI=10.1145/1838687.1838689 http://doi.acm.org/10.1145/1838687.1838689