Översikt över vanliga metoder & verktyg

Uppgift 1 – Metoder & verktyg för mjukvara(DT042G)

David Hegardt Martin Degerman David Jonsson

April 2, 2016

Contents

Stödjande

GIT

GIT är ett verktyg för versionshantering utvecklat utav skaparen av Linux, Linus Thorvalds. Versionshantering används inom projekt för att kunna spåra ändringar som gjorts i filerna som hör till projektet, oftast används versionshantering för skriven kod i systemutveckling. Versionskontroll är viktigt att använda för att undvika att skriva över kod av misstag, men det är absolut nödvändigt i större projekt där man arbetar med flera personer. Med hjälp av en mjukvara för versionshantering sköts detta av ett program istället för att manuellt spara ner versioner. Med mjukvara för versionshantering kan man sedan gå tillbaka och se tidigare gjorda ändringar i koden.

GIT är ett distribuerat system för versionshantering. Detta innebär att systemet inte bara hanterar versionshantering lokalt, eller på en server utan flera kopior av repositoryt existerar på flera olika maskiner. På detta sätt bygger man ett redundant system, om filerna försvinner från servern eller från en dator så finns det fortfarande kvar flera kopior hos medlemmarna samt på servern där projektet är uppladdat. Med hjälp av GIT kan också flera versioner av projektet finnas lagrat på olika klienter. Fördelen är också att medlemmarna som arbetar i projektet inte behöver vara uppkopplade mot servern när de jobbar mot projektet, det behöver bara göras när man hämtar data från repositoryt (pull) eller när man gjort ändringar som man vill ladda upp (push).
GIT skiljer sig från andra mjukvaror för versionshanteringen, andra system tar hänsyn till vilka filer som har ändrats för att hantera olika versioner av dessa. GIT använder sig

istället av sk snapshots av hela filstrukturen, en snapshot är en bild av hela filsystemet vid ett visst tillfälle. En snapshot skapas när man gör en commit, om ingenting har förändrats så skapas en länk till den tidigare versionen, om något har förändrats så sparas den förändrade information i den nya snapshot kopian.

Github

Github är en hemsida med serverlagring av GIT projekt. Det är fullt möjligt att använda GIT utan att använda Github, men man behöver då själv sätta upp en server. Github bygger på open source principen med att det mesta laddas upp för allmän beskådan, vem som helst kan komma åt och se källkoden för de projekt som är uppladdade på Github. Andra git användare kan också vara med och vidareutveckla projektet, dessa blir då projektmedlemmar som kallas för collaborators. En användare kan begära en pull request på ett projekt, och ägaren av projektet kan sedan gå in och bevilja detta via hemsidan vilket tillåter användaren att ladda ner hela projektet till sin klient. När man använder sig av GIThub så får man en GUI för att överblicka projektet via hemsidan, härifrån kan man se samtliga versioner av projektets filer samt vilka ändringar som gjorts mellan olika versioner. Man kan se vem som gjort vilken ändring och vad som skiljer de olika versionerna åt.

GIT är inte helt enkelt att sätta sig in i till en början, har man inte arbetat med versionshantering tidigare och inte är van vid linux kan det vara svårt, om man jämför med ett verktyg som Subversion är risken för fel större med GIT som är mer komplext med branching och offlinestödet kan göra att det blir problem om man hela tiden arbetar offline och sen tar hem ändringar som gjorts i repositoryt utan att göra det på rätt sätt. GIT är mer anpassat för Linux än för Windows, och är man bekant med att jobba i en terminal är det en fördel. Fördelen är att det är helt open source och lätt att få tag i som privatperson, Github är helt publikt och anpassat för open source utvecklaren.

Subversion version control

Subversion är ett vertyg för versionshantering inom systemutveckling. Verktyget togs fram som en vidareutveckling till en annan mjukvara för versionshantering, CVS. CVS(Concurrent Version System) hade brister som man ville lösa genom att ta fram ett nytt verktyg. Subversion precis som GIT är open source och används ofta inom större utvecklingsprojekt. Stora företag som håller på med systemutveckling använder ofta Subversion som verktyg för versionshantering i sina projekt. Det som skiljer Subversion mellan andra verktyg t.ex GIT är att i Subversion arbetar man alltid mot en server, lösningen är centraliserad och utvecklarna arbetar gemensamt i filerna som ligger där, vilket inte ger utvecklarna möjlighet att ladda ner projektet offline. För att säkerthetställa att alla alltid arbetar live mot samma server och kan på så sätt se alla ändringar direkt så har man valt denna centraliserade lösning.

Detta är såklart en nackdel då man vill arbeta offline, och det finns tillägsmöjligheter att kunna kopiera hela projektet lokalt, men det är inte standard på samma sätt som med GIT. Den stora fördelen är att det är enklare att lära sig Subversion och risken för

fel är mindre än med GIT som är betydligt mer komplext.

Testverktyg

Google test

Google test är ett ramverk för enhetstestning i C++. Enhetstestning är väldigt enkelt, och går i stort sett ut på att berätta vad en funktion förväntas göra, och se om den också gör det.

När man väl kommit igenom installationen och fått ordning på de grundläggande inställningarna, är det ungefär lika svårt att köra sina tester som att kompilera och köra sitt program. Resultatet av testerna är nästan omöjligt att missförstå, och visas direkt i terminalen.

Det enda som egentligen riskerar ställa till det för en nybörjare är just de första inställningarna. Den som vill använda ramverket hänvisas till dokumentationen för sitt buildsystem, och endast den som har tillräcklig tur kommer snubbla över en forumtråd eller annan obskyr informationskälla med information om möjligheten att använda en linkerflagga för att bygga testprogrammet.

Hur själva testklasserna och testerna utformas är däremot mycket väldokumenterat. Med början i ett primerdokument om grunderna kan även en fullständig nybörjare snabbt komma in i arbetet, och också gå vidare till mer avancerade guider.

Gcov

GNU/gcc tillhandahåller verktyget gcov för code coverage-testning. Även här är den grundläggande användningen väldigt enkel, och går i stort sett ut på att man anropar gcov med källkoden som ska testas, för att sedan köra resulterande fil. Resultatet presenteras i defaultläget som en sammanställning av antal kodrader, antal lästa, och lästa i procen, men flera flaggor kan användas för att få fram mer utförlig information. Code coverage blir som mest värdefullt när det kombineras med enhetstestning, och förmågan att se inte bara hur testad kod klarar sig, utan också hur stor del av koden som testerna faktiskt går igenom kan bidra till både säkrare och mer optimerad kod.

Nackdelen med både enhetstestning och code coverage är förstås att programmering är en för komplex syssla för att så enkla kvantitativa mätmetoder ska kunna ge entydiga resultat, mer än i de allra enklaste fallen. Ingenting säger att de tester som *inte* skrivits inte heller hade behövts. Och inga garantier finns för att de tester som skrivits är meningsfulla, hur stor del av koden som än omfattas. Risken finns till och med, och då särskilt för nya programmerare, att man lockas att jaga procent, istället för att ta fram så relevanta tester som möjligt.

Utvecklingsmetoder

Vattenfallsmetoden

Vattenfallsmetoden är inte specifikt utformat för mjukvaruutveckling, utan har sitt ursprung i tillverkningsindustrin, och bygger på ett sekventiellt tillvägagångssätt där man successivt betar av ett antal utvecklingssteg. Stegen kan summeras som: kravanalys, design, kodning, testning, och slutligen installation och underhåll av produkten. Utvecklingscykeln är linjär, vilket innebär att när man väl är klar med ett steg så återkommer i regel inte till det, alltså måste alla delar i programmet vara implementerade innan man går vidare till nästa steg.

Krav: Utvecklingen av programvara som sker sekventiellt kan tendera att glida från kundens krav vid större, längre projekt, där man har många som arbetar på samma projekt där viktig kunskap kan försvinna mitt i projektet om någon slutar. Det ställer stora krav på dokumentation. Dessutom är möjlighet att ändra kraven mycket begränsade, och mycket kostsamma om så sker.

Testning: I vattenfallsmetoden testar och buggfixar du produkten utvecklingscycelns slutfas.

Spiral-modell

Spiral-modellen är en metod som lånar element från olika utvecklingsmetoder som vattenfall, iterativa metoder och RAD (rapid application development). Focus ligger på att minimera risken för projekt, och är särskilt lämpad för större projekt. Varje iteration är uppdelad i 4 stadier, först identifiera mål, sedan analysera risker, sedan utveckling, och sist planering av nästa iteration.

Krav: Utvecklingcyklen sker iterativt, vilken medför att man vid varje iteration klargör vilka mål man har och i slutet av varje iteration utvärderar resultatet.

Test: I spiral-modellen är testningen en riskbedömning, det gäller att hitta en balans då man inte vill släppa en bristfällig produkt, samtidigt som man inte kan ha allt för omfattande testning då detta gynnar konkurrenter.

Scrum

Som ett svar på traditionella, sekventiella och "kantiga" utvecklingsmetoder kom de agila utvecklingsmetoderna. Scrum är den vanligaste av de agila metoderna. Här vill man istället för att ha en linjär process så har man korta iterationer (även kallade sprinter) där man implementerar en liten del av programmet i varje iteration. En iteration består av ett planerings-, utvecklings-, test- och ett utvärderingsstadie.

Krav: Kraven kan relativt enkelt uppdateras vid en iteration om det faller inom ramen för ekonomi och tid, vilket bidrar till att man har stor chans att träffa de mål som kunden ställer. Detta ställer också högre krav på delaktighet i projektet från kundens sida.

Testning: Vid varje inkrementering testas nyatillkomna funktioner i programmet. Eftersom fokus läggs på korta iterationer kan inte jättemycket tid läggas på testning vid varje iteration, vilket kan leda till bristfällig kod.

Extrem programmering

Extrem är ytterliggare en agil metod, som lägger fokus på korta utvecklingscykler och frekventa releaser. Under ett projekts cykel lägger man stor vikt vid att hela tiden granska befintlig kod, och därför främjar man utövandet av par-programmering, alltså att två personer arbetar vid samma arbetsstation. Det kan tilläggas att extrem programmering inte behöver motsäga andra agila metoder som t.ex. Scrum utan de har helt enkelt focus på olika delar i utvecklingsprocessen, och snarare kan kompletera varandra.

Krav: Vid extrem programmering sätter man inte upp fasta mål från början, utan låter dem successivt växa fram under projektets gång. Kraven formuleras också som acceptanstest, som körs och publiceras under projektets gång.

Testning: Som nämndes ovan, är acceptanstest en viktig del av extrem programmering, utöver detta uppmuntras inom extrem programmering omfattande testning, av små beståndsdelar av källkod.

Referenser

Ben Collins-Sussman, C. Michael Pilato, Brian W. Fitzpatrick. 2011. “Version Control with Subversion.” http://svnbook.red-bean.com/en/1.7/svn.intro.whatis.html.

Chacon, Scott, and Ben Straub. 2016. “Pro GIT - 2nd Edition.” https://git-scm.com/book/en/v2.

Donahue, Billy. 2016. “Googletest - Primer.” https://github.com/google/googletest/commits/master/googletest/docs/Primer.md.

unknown. 2016a. “Spiral Model - Wikipedia, the Free Encyclopedia.” https://en.wikipedia.org/wiki/Spiral_model.

———. 2016b. “Spiral Model - Wikipedia, the Free Encyclopedia.” https://en.wikipedia.org/wiki/Extreme_programming.

———. 2016c. “Gcov - a Test Coverage Program.” https://gcc.gnu.org/onlinedocs/gcc/Gcov.html#Gcov.

———. 2016d. “Scrum (Software Development) - Wikipedia, the Free Encyclopedia.” https://en.wikipedia.org/wiki/Scrum_%28software_development%29.

———. 2016e. “Software Development Process - Wikipedia, the Free Encyclopedia.” https://en.wikipedia.org/wiki/Software_development_process.

———. 2016f. “Waterfall Modell - Wikipedia, the Free Encyclopedia.” https://en.wikipedia.org/wiki/Waterfall_model.