Ohjelmistoyrityksen työvaiheiden automatisointi ja optimointi

Meelis Mikko

Opinnäytetyö

Helmikuu 2012

Tietotekniikan koulutusohjelma

Ohjelmistokniikka

Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU

Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

MEELIS MIKKO: Ohjelmistoyrityksen työvaiheiden automatisointi ja optimointi
Opinnäytetyö 33 sivua
Helmikuu 2012

Eye Solutions Oy on ohjelmistoalan yritys, joka kehittää kuvausjärjestelmiä, joissa hyödynnetään eri mobiililaitteita, etenkin älypuhelimia, ja niiden kuvankaappaus ominaisuuksia.

Tämä opinnäytetyö käsittelee yhtiön työtapojen uudelleenajattelua sekä siitä tulevien uusien ideoiden toteutusta. Työtehtäväni kannalta merkittävämmät osa-alueet ovat versiohallinta, versiokäytäntö, koodin automatisoidut käännösmenetelmät aina asennuspaketiksi asti sekä yhtiön tietokanavien pystytys.

Opinnäytetyössä selitetään aina kustakin kohdasta mitä oli vaadittu tai haluttu tulos ja se miten se toteutettiin sekä esille nousseet ongelmat ratkaisuneen. Opinnäytetyössä kuvataan tarkasti ja yksityiskohtaisesti käyttöön otettujen ulkopuolisten ohjelmien asennusta ja niiden muokkausta yhtiön tarpeisiin sopiviksi.

Laajin osa-alue työssä käsittelee prosessia, jossa automatisoidaan lähdekoodin koostamista aina valmiiksi julkaisupaketiksi asti. Tähän käytettiin Hudson nimistä automatisointijärjestelmää, jonka vaiheista, käytöstä ja muokkauksesta kerrotaan hyvin yksityiskohtaisesti. Samalla sivuutetaan siihen liittyviä ongelmia ja kohtia, joita jouduttiin ratkaisemaan, jotta työn tulos olisi mahdollisimman yksinkertainen käyttää.

Työn päämäärä oli että kaikkien yrityksen työntekijöiden tulee pystyä koostamaan yhtiön lähdekoodista valmiita paketteja. Haluttiin myös että että riippuvuudet rakennuksen eri vaiheiden sekä henkilöiden välillä saataisiin poistettua. Myös työntekijöiden käsiksipääsy yhtiön informaatioon, bugikantoihin ja niiden muokkaukseen tulisi tehdä mahdolliseksi sekä helpoksi kaikille työntekijöille.

Avainsanat: Hudson, Mantis, Apache ant, automatisointi, Java, Android, Git

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology
Option of Software Engineering

MEELIS, MIKKO: Work stages automation and optimization in a software company
Bachelor's thesis 33 pages
February 2012

Eye Solutions Ltd. is a software company that develops picture and video capturing solutions by using mobile devices. Company is especially oriented into smart-phones and their capability to record videos and take photos.

This work is about rethinking of work methods, as well as implementation of new work methods. The most important parts of the work are version control, version policy, automatic code compilation and build all the way to the installation package, as well as the information channel setup.

This thesis describes about requirements in each section, how they were implemented as well as what issues rose and how they were solved. This thesis describes exactly the installation process of various needed software and how they were customized for the needs of the company.

The largest section in the thesis is process where compilation of the source code is being automatized up to the final release package. In order to achieve this, an automatized system called Hudson was being used. It is explained thoroughly how it was installed, used and modified. This thesis also contains information about the problems and issues that had to be solved, so the work would end up being as easy to use as possible.

Goal of the project is that every worker in company could build release packages from the source code and dependencies between different phases of the work. Additionally it is desirable that dependencies between people required for the build would be eliminated. Also access to the information, bugs and editing this kind of data should be made as easy as possible and available to everyone in company.

Key words: Hudson, Mantis, Apache ant, automation, Java, Android, Iphone

Sisältö

1 JOHDANTO 6
2 Versiohallintajärjestelmät 7
2.1 Git 7
2.2 Gitin ja Svn:n eroavaisuudet 7
2.3 Gitin tarjoamat mahdollisuudet 9
2.4 Siirtyminen Gitiin 11
2.5 Hudson 12
2.6 Hudsonin asennus ja käyttöönotto 13
2.7 Pakettien rakennusvaiheet Hudsonilla 14
2.8 Versiohallintajärjestelmien käyttö Hudsonilla Eye Solutions:ssa 15
2.9 Apache ant 16
2.10 Rakennuksen ketjutus 17
2.11 Hudsonin liitepaketit 18
2.12 Hudson ja shell skripti 19
2.13 Versointikäytäntyö Hudsonissa 20
2.14 Hudsonin tarjoamat lisäpalvelut 21
2.15 Paketin sekä koodin postitus ja varastointi 22
2.16 Epäonnistuneet rakennusyritykset 23
3 Versiointikäytäntö 23
3.1 Versiokäytäntö Eye Solutionsissa 24
4 Wiki 25
4.1 Wiki yhtiön tarpeisiin 25
4.2 Wikin pystytys 26
5 Mantis Bug Tracker 26
5.1 Asentaminen yhtiön tarpeisiin 28
5.2 Käyttäjäoikeudet, käyttäjäryhmät sekä järjestelmän vakiot Mantis Bug Trackerissa 28
6 Bugien raportointi 29
6.1 Bugien kanssa työskentely 29
6.2 Yhteenveto 31
7 LÄHTEET 32

TERMIT JA LYHENTEET

Build	Ohjelmakooste ja eli käännöstulos lähdekoodista.
Scripti	Käskyjono, joka ajetaan tietyllä logiikalla.
Git	Linuxin toimeasta kehitetty versiohallintajärjestelmä.
SVN	Subversion. Kevyt ja suosittu versiohallintajärjestelmä.
Revisio	Versiohallintajärjestelmän yksi tilanhetki. Voidaan viitata indeksillä.
Javadoc	Automaattisesti generoitu dokumentaatio käännetystä projektista.
Tomcat	Verkkopalvelin ja palvelinsovelma säiliö web-sivuille.
HTTP	Hypertext Transfer Protocol. Hypertekstin siirtoprotokolla.
Apache	HTTP-palvelinohjelma.
Apache ant	Työkalu lähdekoodin buildauksen automatisointiin. XML-pohjainen.
XML	Extensible Markup Language. Standardi tiedon kuvaukseen.
Android	Pääasiassa mobiililaitteille kehitelty javapohjainen käyttöjärjestelmä.
Bugi	Vika tai virhe koodissa joka aiehuttaa toimintavirheitä.
Wiki	Verkkosivusto jota eri käyttäjät voivat muokata haluamansa tavalla.
Mantis	Mantis (bugtracker). Bugihallintajärjestelmä
Windows	Microsoftin kehittämä maailman yleisin käyttöjärjestelmä.
Linux	Linux Torvardsin kehittämä käyttöjärjestelmä Windowsin vaihtoehdoksi.
Mysql	Tietokanta. Tarkoitettu tiedon nopeaan tallennus ja hakupalveluun.

1 JOHDANTO

Eye Solutions on mobiilialustoille jo yli vuoden kameraratkaisuja tehnyt yritys. Yhtiön työvaiheiden standartointi, automatisointi ja tehostaminen olivat jääneet enemmän tai vähemmän taka-alalle, kun koodia oltiin menty kehittämään eri alustoille.

Etenkin versiohallintaan, pakettivarastoon, pakettien rakennusvaiheisiin ja bugien korjaamiseen toivottiin selkeyttä ja parannusta. Minut valittiin hoitamaan tehtävää, jotta pääsisin tutustumaan yhtiön toimintaan, ja myös siksi, että muilla työntekijöillä oli kiireisiä projekteja, eikä kellään ollut aikaa hoitaa kyseisiä tehtäviä.

Sain selvät ja kattavat ohjeet, mitä asioita minun tulisi tutkia ja parantaa. Minulle esiteltiin tuotteet, joita minun olisi tutkittava. Ja jos ne sopisivat yhtiön tarkoituksiin, niin minun tehtävä oli myös asentaa ja ottaa kyseiset tuotteet käyttöön. Pääsin tutustumaan hyvin moneen uuteen tuotteeseen ja alueeseen, sekä sain myös hyvän yleiskuvan työpaikan tuotteista ja toiminnasta.

Ensimmäisenä sekä harjottelussa että tässä työssä, lähdin liikkeelle Git:stä ja vertasin sitä yhtiössä käytössä olevaan Svn:ään. Git on Svn:n tavoin versiohallintajärjestelmä. Sen tarkoitus on pitää yllä lähdekoodiin tehtyjä muokkauksia ja lisäyksiä, sekä tarpeen tullen kumota niitä. Työssä verrataan eroja Git:in ja Svn:n välillä ja kerrotaan miten siirtyminen Git:iin tapahtuisi.

Seuraavaksi työ kertoo Hudsonin asennuksesta ja säädöstä sekä sen soveltamisesta yhtiön eri tuote- ja koodialustoille. Hudson on selaimessa toimiva järjestelmä jonka avulla voidaan luoda automaattisia prosesseja. Niissä voidaan jopa yhdellä painalluksella tuottaa lähdekoodista valmis buildi haluttujen lisätoimintojen kanssa. Luvussa kerrotaan laajasti miten Hudson säädettiin toimintakuntoon ja miten eri mobiilialustojen käännökset saatiin rakentumaan automaattisesti aina alusta loppuun asti.

Luvussa neljä käsitellään yhtiön versiointikäytäntöä. Myös yhtiön versiointikäytäntö oli yksi alueista, jota oli pakko parantaa. Sen epäselvyys ja ristiriitaisuus aiheutti ongelmia automatisoinnin kanssa. Ongelmana oli mm. sovittujen versiokäytäntöjen puuttuminen eri puhelimien koodialustojen välillä. Tähän asti versiot oli nimetty aina ohjelmoijan oman käytännön mukaan. Luvussa kerrotaan miten versiostandardi sovittiin ja mitkä seikat vaikuttivat siihen. Myös Hudsonin tuottamat buildit ja niihin liittyvä versiointikäytäntö selitetään.

Viides ja kuudes luku käsittelevät puolestaan Wiki:a ja Mantis:ia. Luvussa kerrotaan miten nämä asennettiin ja miten ne muokattiin yhtiön omiin tarpeisiin sopiviksi, siten että sekä asiakkaat että työntekijät voivat käyttää samaa palvelinta. Koska eri sidosryhmille haluttiin eri oikeuksia ja näkyyvyysalueita Mantis järjestelmässä, piti järjestelmää muokata tarpeisiin sopivaksi.

2 VERSIOHALLINTAJÄRJESTELMÄT

Versiohallintajärjestelmän tarkoitus on pitää yllä tietoa muutoksista ja lisäyksistä, joita tulee siellä oleviin tiedostoihin ja projekteihin. Tämäntyyppiset järjestelmät ovat suunnattu ohjelmointiin ja niiden lähdekooditiedostoihin. Versiohallintajärjestelmän tarkoitus sananmukaisesti on hallita eri versioita siellä olevista ja sinne tulevista uusista tiedostoista ja niihin tehdyistä muutoksista. Versiohallinnan tarkoitus on ylläpitää lisäyksiä ja muutoksia, joita on tehty eri revisioissa sekä pystyä palautumaan aikaisempiin revisioihin tarvittaessa. Järjestelmässä näkyy myös tieto siitä, kuka on tehnyt, mitä on tehnyt ja milloin on tehnyt. (Version Control for Designers)

Etenkin isoissa ohjelmistoprojekteissa versiohallintajärjestelmän tarkoitus on merkittävä, sillä projektiin tekee muutoksia useampi eri henkilö. Tällöin on tärkeä että muutokset ja lisäykset tulevat nopeasti ja helposti kaikille käyttäjille. Myös virhetilanteissa järjestelmästä on mahdollista saada selville, mikä ja kenen muutos on aiheuttanut projektissa ilmenevän uuden virheen. Lisäksi voidaan poistaa myös kyseinen virheellinen koodipäivitys ja jatkaa ilman sitä. (Version Control for Designers)

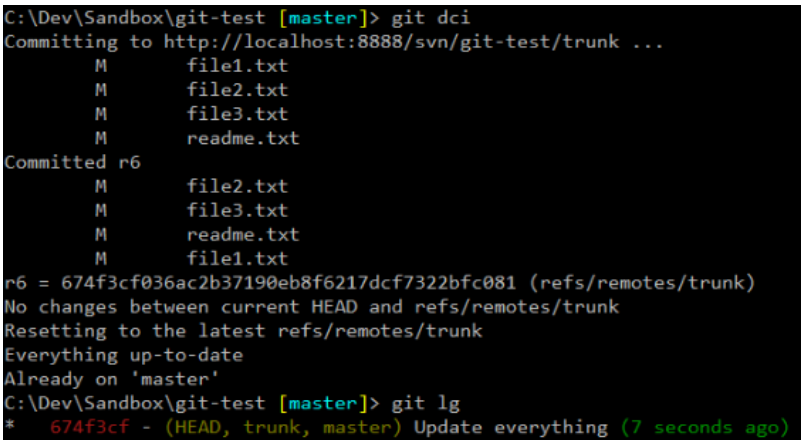
2.1 Git

Git (kuva 1) on versiohallintajärjestelmä, joka toimii yleisempien käyttöjärjestelmien lisäksi myös monessa harvemmin käytetyssä käyttöjärjestelmässä. Git-versiohallintajärjestelmän kehittämisen taustalla on ollut Linux Torvards ja hänen vaatimukset sekä hänen tyytymättömyys SVN-versiohallitnajärjestelmän toimintaan etenkin Linux-alustalla. Tovardsin mielestä SVN oli huono ja hidas versiohallintajärjestelmä, eikä SVN-järjestelmän kehittäjät olleet halukkaita tekemään muutoksia joita Tovards halusi. (Git – software – Wikipedia)

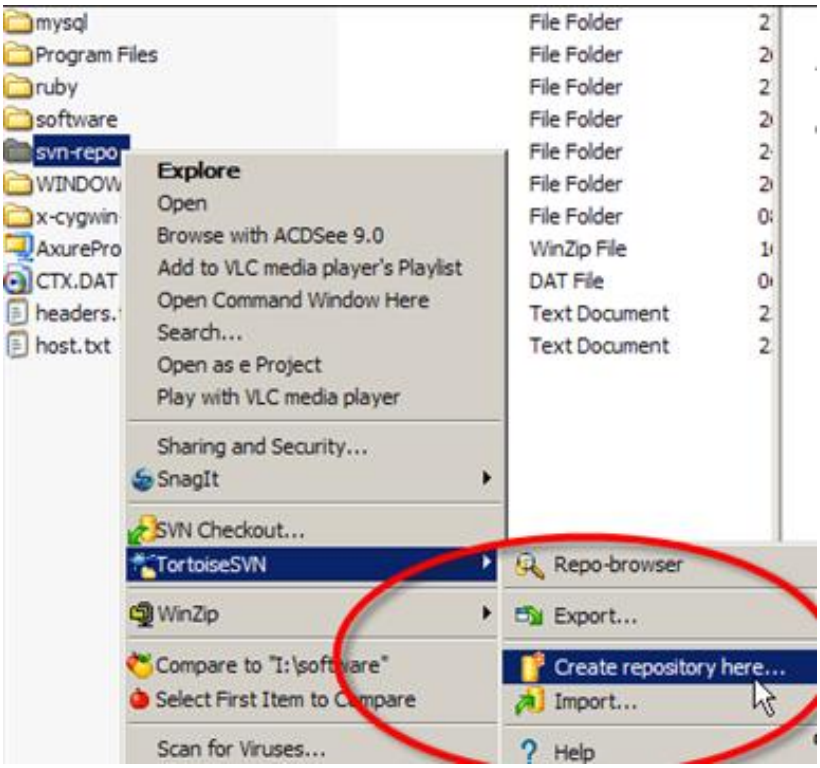
2.2 Gitin ja Svn:n eroavaisuudet

Eye Solutions Oy:ssä on käytössä toistaiseksi SVN-versiohallintajärjestelmä. Kyseinen järjestelmä on huomattavasti Git:iä yksinkertaisempi, mutta silti täyttää versiohallinnan perusvaatimukset (kuva 2). SVN ja Git ovat perustoiminnoiltaan lähes samanlaiset,

vaikka käskyt eroavat komentonimiltään toisistaan. Tiedon lisäys järjestelmään, järjestelmästä uusimman päivityksen hakeminen yms. perustoiminnot löytyvät molemmista. SVN:n ja Git:in isommat erot tulevatkin esiin vasta, kun siirrytään monimutkaisempiin, raskaimpiin ja vaativampiin tehtäviin. Näitä eroja ovat mm. järjestelmän nopeus, käyttötehokkuus sekä mahdollisuus tarkemmin säädettäviin komentokäskyihin. Näissä kaikissa Git:iä pidetään parempana, tai ainakin Gitin kehittäjät itse väittävät näin asioiden olevan. Komentoparametrien kohdalla tämä voidaan todeta, sillä SVN:n puolella ei kovin monimutkaisiin yhdistelmäkomentoihin ole mahdollisuutta, toisin kun GIT:issä. (gitutorial , git- the fast version control system , Apache subversion, Git – Svn crash course)



Kuva 1. Git:in käyttöä Linuxissa.



Kuva 2. Svn:n käyttö on helppoa Windows koneilla. Komennot on ingegroitu valikkoihin suoraan.

Kun tutkitaan tarkemmin miten järjestelmät toimivat niin huomataan, että Git:in ja SVN:n välillä on paljon eroa. Näkyvimpiä eroja ovat esim. .svn ja .git päätteiset hakemistot projektipoluissa, jotka sisältävät itse revisiotiedot. SVN:n tapauksessa jokainen hakemisto sisältää aina yhden .svn loppuisen hakemiston. GIT:in tapauksessa taas vain projektin kantahakemistossa on yksi .git loppuinen hakemisto, jossa tieto on. (git- the fast version control system, Apache subversion, Git – Svn crash course)

Eye Solutions halusi lisätietoa GIT:istä ja sen toimintoista, sekä miten hyvin SVN:ässä käytetyt toiminnot ja palvelut saataisiin tehtyä GIT:in puolella. Lisäksi haluttiin tutkia olisiko mahdollisesti hyötyä GIT:in tarjoamista uusista toiminnoista. Lisäksi työssä tutkitiin myös siirtymistä ja mahdollista revisiotietokannan kääntämistä SVN:stä GIT:iin. (Git – Svn crash course)

2.3 Gitin tarjoamat mahdollisuudet

Koska molemmista versiohallintajärjestelmistä löytyvät samat perusominaisuudet ja toiminnot, keskitytään enemmän asioihin, jotka eroavat näissä kahdessa järjestelmässä. Git:in on näistä kahdesta se monipuolisempi, tarkoittaa se että seuraavissa kappaleissa

keskitytään enimmäkseen Git:in tarjoamiin palveluihin, joita Svn:stä ei löydy. (Git – Svn crash course)

Versiohallintajärjestelmän tehtävänä on varmistaa, että siihen lisättyjä muutoksia, muokkauksia tai lisäyksiä pystytään helposti hallitsemaan, selamaaan ja tarvittaessa peruuttamaan. Tämä tarkoittaa sitä, että nämä molemmat järjestelmät tarjoavat kattavat työkalut eri revisioiden välisiin toimenpiteisiin. Molemmissa järjestelmissä pitää ensiksi kertoa järjestelmälle, mitä sinne laitetaan, ja mikä suljetaan pois hakemistorakenteista. Useimmiten esim. ohjelmistoprojekteissa binääritiedostot yms. jätetään pois, koska niiden laittaminen järjestelmään ei hyödynnä mitään ja käyttää paljon kovalevytilaa. Sen sijaan ohjelmistoprojektin lähdekoodit, kuten myös erilaiset resurssitiedostot ja kuvat jne. tulevat sinne. (Git – Svn crash course , git- User's Manual)

Perustoimintoja ovat esimerkiksi tiedostojen lisäys järjestelmään, tiedostojen poisto järjestelmästä, tiedoston tai tiedostojen palautus tiettyyn vanhempaan versioon ja tiedostojen automaattinen yhdistäminen. Tämä yhdistäminen tässä tilanteessa tarkoittaa tapausta, jossa kahdelta tai useammalta eri käyttäjältä on tullut muutoksia samaan tiedostoon. Näitä muutoksia ei järjestelmä voi automaattisesti yhdistää. Ristiriidat on selvitettävä manuaalisesti, valiten mitkä kohdat tulevat mukaan yhdistettyyn versioon ja mitkä ei. Riittää kun tämä tehdään vain kerran ja kenen tahansa osapuolen toimesta.

Myös järjestelmästä muiden käyttäjien tekemien muutosten päivittäminen sekä omien tehtyjen muutosten lisäys järjestelmään ovat jokapäiväisiä perustoimintoja versiohallintajärjestelmissä. Näissä päivitys- ja lisäystoiminnoissa on liitteenä tieto, joka kertoo mikä on muuttunut kyseisissä tiedostoissa tai tiedostossa. Koko tiedoston sisältöä ei versiohallintajärjestelmässä lähetetä, ellei tiedosto puutu kokonaan.

Peruskäytössä nämä ominaisuudet ovat usein riittäviä eikä näin ollen tule välttämättä tarvetta monimutkaisemmille ominaisuuksille. Git kuitenkin tarjoaa lukuisia muita ominaisuuksia niistä kiinnostuneille peruskäyttäjille, kuten myös tehokäyttäjille.

Merkittävämpiä lisäominaisuuksia peruskäyttäjälle Git:issä ovat komennot “git stash” eli väliaikaissäiliö, sekä “git branch” eli haara (tosin svn:ssä on myös nykyisin

jonkinlainen tuki haaroille), kuten myös koukut “hooks” sekä termi ”biscet”, jolle ei löydy suomennosta. (Git – Svn crash course , git- User's Manual)

Stash:in eli säilön tarkoituksena on toimia väliaikaisena varastona, johon käyttäjä voi laittaa viimeisemmät tehdyt muutokset, joita ei vielä ole lähetetty versiohallintajärjestelmään päivityksenä. Tämä on esimerkiksi kätevä tilanteessa, jossa käyttäjä joutuu vaihtamaan kesken aloitetun työn alueen A parissa uuteen ja kiireisempään aiheeseen B. Koska A jäi kesken, niin sen muutokset voi haitata B:n toimintaa, mutta myös A:n palauttaminen alkutilanteeseen tarkoittaisi, että A:ssa tehdyt muutokset jouduttaisin tekemään taas uusiksi myöhemmin. Säilön tarkoituksena on säilöä näitä keskeneräisiä muutoksia, siten että vaikka A:n muutokset palautetaan lähtötilanteeseen; saadaan A:ssa jo tehdyt muutokset myöhemmin uudelleen esiin ja niistä voidaan jatkaa. (git- User's Manual)

Koukut ovat puolestaan Gitin:n toimintoihin sidottuja skripti logiikka kutsuja, joilla voidaan sitoa Git:in ulkopuolisia toimintoja. Näitä kutsutaan, kun tietty Git:in toiminto tai tilanne tapahtuu tai arvo tule voimaan. Tämä tarkoittaa, että Git on mahdollista liittää kiinni muihin järjestelmiin, missä se palvelee vain yhtä osaa isommasta automatisoidusta kokonaisuudesta. (git- User's Manual)

Haarat taas ovat nimensä mukaan tietystä kantaversiosta haarautuneita versioita, jotka joskus ovat olleet samanlaisia, mutta joista esim. on päätetty lähteä kehittämään kahta tai useampaa erilaista projektia. Git tarjoaa kattavat työkalut, joiden avulla voidaan haaroja yhdistää osittain tai kokonaan, tarkistaa eroja tai siirtää kantahaaraa eteenpäin yhdistämällä väliin jääneet muutokset. Vaikka perusasiat haaroista ovat helppoja, niin mitä syvemmälle mennään huomataan, että asiat mutkistuvat ja vaativat pitkäaikaisen syventymisen. Gitissä taitava käyttäjä voi saada haluamansa tuloksen hetkessä, kun taas sitä taitamaton käyttäjä joutuu ensin syventymään hyvin pitkäänkin ongelmaan. (git- User's Manual)

2.4 Siirtyminen Gitiin

Siirtyminen yhdestä versiohallintajärjestelmästä toiseen on sinänsä yksinkertaista, jos aloitetaan tyhjältä pöydältä eli vanhan versiojärjestelmän sisältöä ei tuoda mukana, vaan

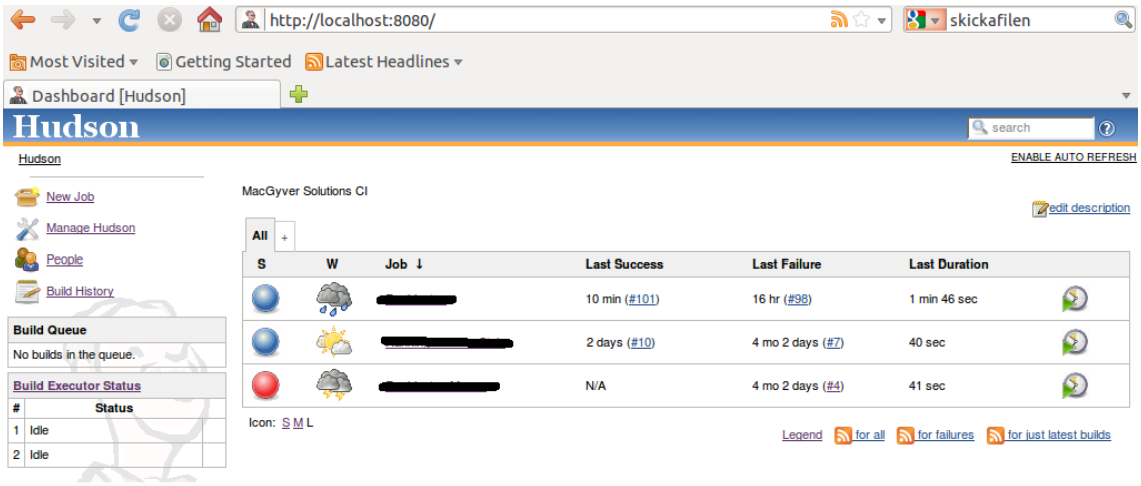
koodi- ja resurssitiedostot lisätään uuteen järjestelmään ja aloitetaan alusta. Tällä tavalla tietoja ei häviä, koska aina tarvittaessa voidaan katsoa vanhan järjestelmän puolelta revisioita jotka on liitetty ennen siirtymistä uudeen versiohallintajärjestelmään. Tämä tosin hankaloittaa työtä, jos joudutaan usein tarkistamaan asioita vanhasta versiohallintajärjestelmästä.

Koska Git ja Svn ovat laajassa käytössä olevia järjestelmiä on kyseisten järjestelmien tietokannan kääntäminen mahdollista ainakin Svn:stä Git:iin. Tämä ei kuitenkaan ole kovin helppoa vaan asiaan piti perehtyä jonkin aikaa. Ikävä kyllä, kumpikaan järjestelmien kehittäjistä ei halunnut tukea kyseistä operaatiota tai edes jakaa tietoa siitä. Lopulta asiasta löytyi tietoa ja peretyttyä asiaan huomasin, että tähän tarvittiin vain muutama rivi Linux-skriptia, sekä noin tunti käännösaikaa. (Migrate your subversion repository to a git repository)

Koska yhtiölläni oli kiire muiden projektien kanssa ja Git:in opettelu ja siirtyminen olisi vienyt jonkin verran aikaa kaikilta työntekijöiltä, päätettiin se jättää myöhemmäksi. Tällöin olisi paremmin aikaa opastaa henkilökuntaa uuden järjestelmän ominaisuuksiin, jotta niitä tulisi käytettyä ja siten siirtymisestä olisi näkyvää hyötyä.

2.5 Hudson

Hudson (kuva 3) on ilmainen automaattinen buildausjärjestelmä, joka keskittyy jatkuvaan integraatioon. Se on Extreme Programming:in (XP) kehittämä järjestelmä, joka helpottaa ja automatisoi yhtiön buildaustoimintaa sekä vähentää riippuvuutta eri elinten välillä. Hudson tarjoaa järjestelmän, johon voidaan liittää säädettävissä olevia buildaus projekteja, jolla jokaisella yleensä saadaan jokin kokonaisuus buildattua lisätoimintoineen. Parhaimmillaan yhdellä hiiren painalluksella voidaan suorittaa viimeisimmän koodirevision haku versiohallinnasta, käännös, testaus ja onnistuessa postitus halutuille henkilöille. Lisäksi jos valittaessa saadaan vielä käännetty paketti varastoon. Tämä kaikki on säädettävissä ja yksi projekti voi koostua niin monesta eri vaiheesta kuin tarvitaan. Lopputuloksena voisi esimerkiksi olla käännöspaketti, javadoc tai vaikka vain ilmoitus sähköpostiin siitä, että projekti kääntyi tai ei kääntynyt, tai tämä kaikki yhdessä. (Hudson (software) – Wikipedia, Meet Hudson – hudson- Hudson wiki)



Kuva 3. Hudsonin päänäkymä. Hudsoniin on lisätty kolme projektia.

Yhtiömme johto oli kiinnostunut järjestelmästä, koska aikaisemmin uusien versioiden tekeminen vaati aina sen, että joku projektin ohjelmoijia oli läsnä ja käänsi sekä lähetti paketin sitä tarvitseville. Tämä oli tosi iso ongelma, kun esimerkiksi projektin ainoa asiasta tietävä ohjelmoija oli kesälomalla.

Myös itse kääntäminen ja käännettyjen pakettien varastointi olivat ohjelmoijan vastuulla. Tästä riippuvuudesta haluttiin eroon. Lisäksi otin selvää myös autommaattisesta testauksesta, joka on yksi Hudsonin tarjoamista mahdollisuuksista, mutta sitä yhtiössämme ei otettu käyttöön.

2.6 Hudsonin asennus ja käyttöönotto

Hudsonin voi ladata internetistä ilmaiseksi .war -pakettina. Sen asennus on helppoa, jos Tomcat tai Apache servletti-säiliö on jo asennettu. Jos näin ei ole, on ensin asennettava jompi kumpi viimeksi mainituista tai muu vastaavanlainen. Hudson asennus aloitettiin puhtaalta Linux-koneelta, johon asennettiin kaikki komponentit aina kun tuli tarve siihen. Jotta Hudson saatiin toimimaan jouduttiin asentamaan java-ympäristö, Tomcat, sähköpostipalvelin Hudsonin automaattisia postinlähetyksiä varten. Myös jouduttiin muokkaamaan Hudsonin porttia- sekä oikeusasetuksia. Vielä oli muokattava Linuxin alkukäynnistys skriptejä, jotta Hudsoni olisi käynnistynyt automaattisesti vaikka palvelinkone olisi käynnistetty uudelleen tai kaatunut. Sähköpostipalvelimen asentamiseen meni odotettua enemmän aikaa, koska palvelimen piti tukea normaalin

postin lähetyksen lisäksi myös liitetiedostojen lähetystä. Tämä oli kuitenkin saatava toimimaan, jotta Hudsonilla olisi voinut toimittaa käännettyjä paketteja suoraan sähköpostiin. (Installing Hudson – hudson- Hudson wiki)

Kun itse Hudson on asennettu ja käynnistetty Tomcatissa, avataan se selaimesta käsin antaen osoitteeksi palvelinkoneen ip-osoite. Ennen seuraava vaihetta piti turvallisuuden vuoksi asettaa portit kuntoon, koska ei haluttu Hudsonin näkyvän ulkoverkkoon. Hudson teki ensikertakäynnistyksen yhteydessä pienen automaattisen itsesäädön ja antoi käyttäjän tehdä ylläpitäjätilin. Sen jälkeen päästiin tutkimaan ja muokkamaan itse Hudsonin tarjoamia palveluja ja asetuksia. Vaikka Hudsonissa oli paljon säädettävää ja muokattavaa, niin järjestelmä on tietotekniikan taidot omaavalle henkilölle hyvin helppokäyttöinen ja on myös intuitiivinen.

Asetuksista pystyy määrittelemään koneen, jolla Hudsonin tietokanta on, sekä halutessa linkittämään lisää koneita Hudsonin pääpalvelimen taakse nopeuttamaan prosesseja. Lisäksi asetuksista pystyy säätämään ulkoasua, projekteja, käyttäjiä sekä liitännäisiä, joista kerrotaan lisää myöhemmin.

2.7 Pakettien rakennusvaiheet Hudsonilla

Ennen kun projekteja kannattaa lisätä Hudsoniin, on suotavaa määrittää päälohkot projekteille. Tämä siksi, jotta projektit pysyisivät paremmin järjestyksessä. Tähän päälohkojen luontiin ja niiden sisällön järjestämiseen on eri vaihtoehtoja, ja siten projektejä pystyy ryhmittelemään monella eri tavalla.

Hudsonissa yhden projektin rakennusvaiheet koostuvat rastitettavista arvovalinnoista, jotka yleensä sijoittuvat ennen rakennusketjun alkamista tai sen jälkeiseen toimintaan. Näitä ovat esimerkiksi se, että säilötäänkö projektin .apk paketti vai ei ja tyhjennetäänkö projektin kotihakemisto ennen, kuin haetaan versiohallintajärjestelmästä uusimmat muutokset. Jos jokin ruutu rastitetaan, pystyy yleensä valitun toiminnon yksityiskohtia säätämään tarkemmin.

Toisen osan rakennusmäärittelyssä muodostavat itse vapaat skriptikentät, jossa voidaan käyttää taustalla pyörivän käyttöjärjestelmän kaikkia mahdollisia komentorivikäskyjä hyväksi. Myös osittain valmiiksi täytettyjä käskyjä löytyy Hudsonista ja niissä päätoiminto on jo ennalta määrätty ja käyttäjän ei tarvitse esimerkiksi, kun kertoa vain hakemistopolku. Nämäkin toiminnot on mahdollista määrittää alusta asti komentoriviskriptillä, eikä näin ollen mitään puolivalmista rakennusvaihetta ole pakko käyttää jos toiminto halutaan määritellä itse tarkemmin. Kaikki on siis mahdollista tehdä itse toimintoaskel kerralla, ja projektiin voi lisätä niin monta rakennusaskelta kun halutaan. Lisäksi projekteja on mahdollista linkittää keskenään, mutta tästä lisää myöhemmin.

2.8 Versiohallintajärjestelmien käyttö Hudsonilla Eye Solutions:ssa

Pakettivarastot tai versiohallintajärjestelmät, kuten on asiaa kutsuttu tässä asiakirjassa, ovat tärkeä osa Hudsonia, sillä uusien buildiversioiden rakennus alkaa aina viimeisimmän revision noutamisella versiohallintajärjestelmästä. Sen integrointi buildiin on siis tärkeä osa koko automaatiojärjestelmän toimintaa.

Normaalisti haluttu versio on viimeinen revisio projektista ja se haetaan aina automaattisesti versiohallintajärjestelmästä. Niin tehdään kaikissa Eye Solutionissa projekteissa ennen kuin sitä aletaan buildamaan Hudsonilla. Hudson antaa säätää tarkkaan mitä hakemistoja ja tiedostoja haetaan versiohallintajärjestelmästä. Ensimmäinen tapa on käyttää poissulkumenetelmää jossa valitaan mukaan kansiot jotka halutaan hakea, poislukien tietyt kansiot tai tiedostot. Toinen vaihtoehto on määrittää halutut hakemistot ja tiedostot yksi kerralla. Yleensä ensimmäinen vaihtoehto on helpompi ylläpitää, sillä kun projektiin lisätään tiedostoja, ei niitä tarvitse lisätä Hudsonin hakuskriptiin. Lisäksi harvoin ylimääräisen tiedon hakemisella on muuta haittaa, kun itse hakemiseen mennyt ylimääräinen aika sekä turhaan käytetty tila Hudson-palvelimella.

Normaalisti kun haetaan versiohallintajärjestelmästä tietoa, niin joudutaan syöttämään tunnus sekä salasana. Onneksi tämä oli Hudsonissa mahdollista tehdä automaattiseksi järjestelmän omilla valmiilla toiminnoilla. Toinen tapa olisi toteuttaa uusimman revision haku liittämällä projektin ketjun alkuun shell skripti, jossa haetaan ”svn

update” – komennolla viimeisin revisio. Tätä valitettavasti ei ole mahdollista saada toimimaan ilman, että käyttäjä joutuisi syöttämään manuaalisesti salasanan joka kerta kun Hudsonin projektin kyseinen vaihe tulee vastaan. Tämä olisi kova takaisku silmälläpitäen automatisointia. Edellämainittu automaattinen tapa kuitenkin aiheuttaa joskus hankaluuksia, sillä Hudsonin valmis toiminto revision hakemiselle ei ole kovin joustava.

Hudsoni pitää myös tietoa järjestelmässä siitä, mistä revisioista jokin käännös tai projekti on aikanaan tehty. Tämä on varsinkin kätevä, kun toiminto on automaattinen ja sisältää revision numeron lisäksi myös revisiossa ohjelmoijien tekemät kommentit versiohallintajärjestelmään. Sen avulla buildaaja saa kätevästi yhteenvedon siitä, mitä on muuttunut projektin koodissa sitten viime revision. Tämä voidaan siten myös liittää jaettavaan pakettiin mukaan versiologiksi.

2.9 Apache ant

Apache ant on XML:ään pohjautuva työkalu Java:lle. Se mahdollistaa monivaiheisen rakennusskriptitiedoston kirjoittamisen XML muodossa (kuva 4). Siellä skriptin eri vaiheet suoritetaan määritellyssä järjestyksessä. Apache ant on monipuolinen ja kaikki javapohjaiset projektit kattava kääntäjä, olkoon sitten haluttuna tuloksena portletit, servletit tai vaikkapa Android:in .apk-paketit. Jälkimmäinen tosin tarvii Android kirjastot asennettuna, jos halutaan tehdä Android buildeja. (Apache ant – User Manual)

```
<project name='Flex' default='mxml'>

    <target name='mxml'>
        <property name='compiler' location='C:\flex\bin\mxmlc.exe' />
        <exec executable='${compiler}' failonerror='true'>
            <arg line='-output C:\baadshah\IDE\flex\rachel.swf' />
            <arg line='C:\baadshah\IDE\flex\rachel.mxml' />
        </exec>
        <echo>mxml file compiled successfully</echo>
        <echo>swf file generated successfully</echo>
    </target>

</project>
```

Kuva 4. build.xml tidosto jonka avulla Apache ant rakentaa projekteja

Apache ant XML skriptissä voidaan määritellä globaaleja vakioita, kuten esim. tiedostopolkuja tai muita vastaavia arvoja, sekä myös paikallisia arvoja. Ketjut suoritetaan läpi aina niin sanotussa riippuvuusjärjestyksessä. Jos jokin osa on riippuvainen jostakin muusta osasta niin se suoritetaan ennen sitä. Näin ollen koko ketju aina aloitetaan siitä ensimmäisestä osasta, joka ei ole riippuvainen mistään muusta osasta. Koko ketjua kutsuttaessa kirjoitetaan komentoriville aina viimeisemmän osan nimi, jolloin kaikki muutkin vaiheet tulevat suoritetuksi. (Apache ant – User Manual)

Apachen itse suosittelema tapa on pitää eri loogiset vaiheet omana osana. Suositeltava ja myös käytetyin järjestys voisi siis olla: 1) Käännetään .java tiedostot .class tiedostoiksi. 2) Kopioidaan .class tiedostot rakennuskansioon 3) Luodaan lopullinen paketti 4) Poistetaan ylimääräiset tiedostot. Lisäksi monimutkaisemmissa projekteissa väliin voi tulla lukuisia lisävaiheita. Ensimmäiseen vaiheisiin voidaan laittaa useampi eri logiikkavaihtoehto, jolloin lopputulos voi olla erilainen vaikka jotkut osat olisivatkin samoja eri projekteissa. (Apache ant – User Manual)

Apache ant ja sen palvelut olivat yksi merkittävin osa Hudsonin projekteissa, sillä kaikki javakoodit käännettiin Apache antin avulla. Androidin tapauksessa Eclipsen tuottamasta androidManifest.xml -tiedostosta sai automaattisesti rakennettua Apache ant:in build.xml -tiedoston. Mutta esimerkiksi servlettien ja portlettien tapauksessa build.xml –tiedosto jouduttiin rakentamaan manuaalisesti.

Myös Javadoc tuotettiin Apache antin avulla joko omalla erillisellä build.xml:llä tai liittämällä se yhdeksi käännöksen vaiheeksi. Ylläpidettävämmäksi osoittautui määritellä Javadocin teko erikseen Hudsonissa, jolloin sen sai pois päältä sieltä käsin, ilman että jouduttiin muokkamaan build.xml tiedostoa.

2.10 Rakennuksen ketjutus

Myös yksi merkittävistä Hudsonin palveluista on projektien ketjutus. Tämä mahdollistaa moniosaisen projektin rakennemallin, jossa yhden projektin ei tarvitse edes olla kokonaisuus vaan osa. Rakennettaessa useita projekteja Hudsonille tulee vastaan tilanteita, joissa esimerkiksi eri projekteissa on yhteneviä osia. Jos nämä osat vielä muuttuvat päivittäin niin muutokset jouduttaisiin tekemään kaikkiin projekteihin.

Tässä tapauksessa järkevämpi tapa olisi siis tehdä yhtenevä osa yhdeksi osaksi eri ketjuja, jossa eri projektit kutsuisivat samaa yhtenevää osaa, mutta muuten olisivat eri projekteja. Jos siis yhtenevä osa muuttuu tarvitsee muutokset tehdä vain yhteen projektiin.

Vaikka tämä onkin idealtaan hyvä ja järkevä tapa niin valitettavasti Hudsonin kankeus ja jäykkä osien välinen runko syövät tämän ominaisuuden käytettävyyttä. Tämä etenkin siksi, koska projektit ovat rippuvaisia eri osien sisäisistä muuttujista ja niiden arvoista. Ja näitä arvoja ei voi siirtää osien välillä järkevällä tavalla. Tämän ongelman kiertäminen vaatisi paljon tallennus- sekä latausskriptejä, ja siten ylläpidettävyys sekä selkeys huonontuisivat. Tästä syystä kyseistä ominaisuutta ei voitu hyödyntää juurikaan Eye Solutionin Hudson projekteissa.

2.11 Hudsonin liitepaketit

Hudsonin liitepaketit jaetaan kolmeen eri luokkaan. Ensimmäisen muodostavat niin sanotut ydinpaketit, jotka tulevat Hudsonin mukana ja ovat kiinteä osa sitä. Toisen ja kolmannen osan muodostavat ulkopuolisten tuottamat liitepaketit. Toisen ja kolmannen luokan välillä jako perustuu siihen, että ovatko ne laatutestattuja vai ei. Liitännäisiä on tarjolla satoja ja monessa tapauksessa samaan asiaan voi olla tarjolla monta erilaista liitännäistä. Liitännäisten laatu vaihtelee paljon ja joistakin liitännäisistä on vaikea löytää edes dokumentaatiota tai tietoa, puhumattakaan käyttöesimerkeistä. Liitännäisiä Hudsonista voi asentaa tai poistaa ylläpitäjä. (Plugins – hudson- Hudson wiki)

Liitännäiset ovatkin tärkeä osa Hudsonia, etenkin jos järjestelmästä halutaan muokata juuri omaan tarpeseeseen sopiva. Hudson ei itse tarjoa välttämättä kaikkea käyttäjän haluamia ominaisuuksia ja palveluja, joten monesti ollaan liitännäisten tarjonnan varassa. Jos liitännäistäkään ei ongelmaan löydy ratkaisua, niin silloin usein joudutaan asia hoitamaan shelli skriptillä, josta enemmän ensi kappaleessa.

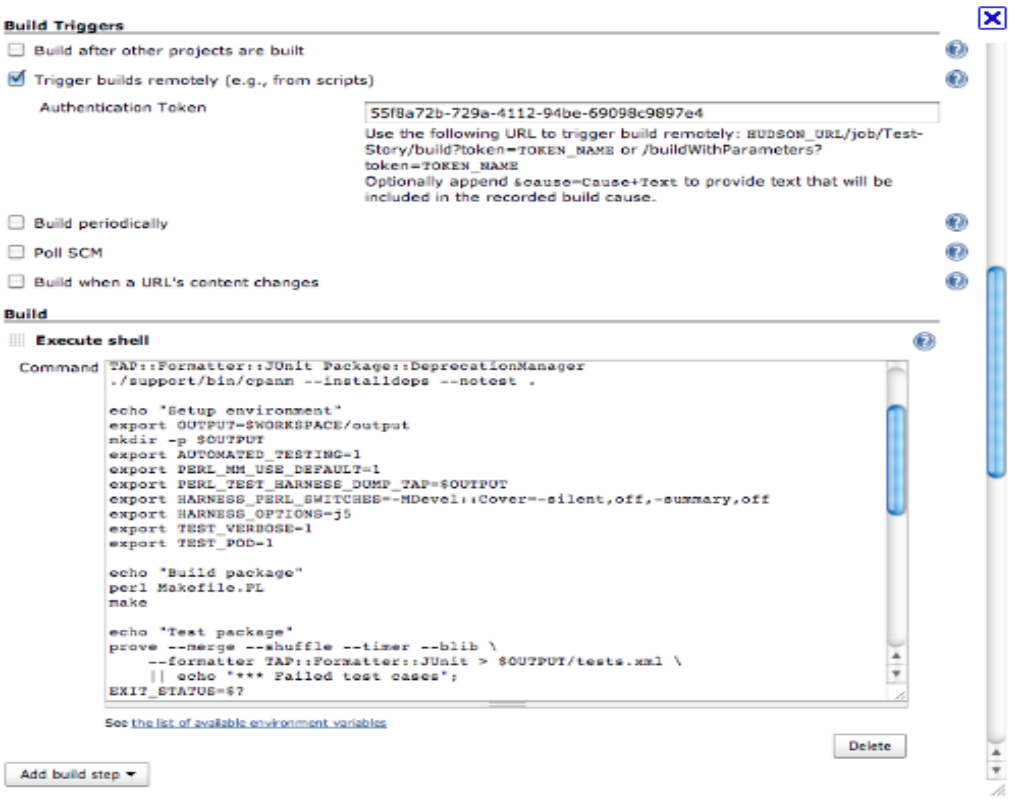
Liitännäisistä onneksi löytyy hyviäkin ja läheskään kaikki eivät ole huonosti dokumentoidut. Jotkut liitännäisistä ovat lähes pakollisia jos käyttäjä haluaa tehdä vähänkin monimutkaisempia projekteja. Toiset taas helpottavat tai automatisoivat

asioita suuresti. Jotkut liitännäisistä puolestaan keskittyvät parantamaan Hudsonin käyttöliittymää tai ulkoasua.(Kuva 5).

Kuva 5. Yksi monista Hudson liitännäisistä; Chuck Norris- littäntäpaketti. Kaikki littännäiset eivät aina välttämättä tuo uusia toimintoja lisää.

2.12 Hudson ja shell skripti

Shelli skripti (kuva 6) viittaa eri alustojen komentorivikieleen, johon voidaan kirjoittaa koodinomaista logiikkaa. Hudson mahdollistaa shelli skriptin ajon kesken työvaiheiden, jolloin kaikki toiminnot jota ei ole mahdollista toteuttaa Hudsonin valmiilla palveluilla tai liitännäisillä, voidaan toteuttaa shelli skriptillä. Shelli skriptin käyttäminen ei ole kovin helppoa jos ei tunne taustajärjestelmän käskyjä, eikä haluttujen tulosten saaminen ole kovin nopeata jos halutaan suorittaa vähänkin monimutkaisempia toimintoja. (Shells – hudson- Hudson wiki)



Kuva 6. Hudsonin rakennusvaiheen määrittelyä, jossa Linuxin shell skripita

Hudsonin ja sen liitännäisten jätettyä ongelmia usein ratkaisutta, tai puolitiehen, oli turvauduttava shelli skriptiin. Se oli ainoa takuuvarma työkalu, jolla jokin asia voitiin toteuttaa juuri haluamalla tavalla, joskin kovan työn kautta. Käytännössä työn loppuvaiheessa huomattiin, että shelli skripti osoittautui aina kätevämmäksi, kun kankeat valmiit toiminnot ja sen käyttö korvasi melkein kaikki valmiit Hudsonin palvelut. Lopulta jopa versiohallintajärjestelmän käyttö, varmuuskopiointi tai sähköpostinkin lähettäminen oli paljon tarkemmin säädeltävissä Linuxin komentojen avulla, kun valmiiden syötekenttien avulla Hudsonissa. Tästä syystä työn loppuvaiheessa lähes kaikki toiminnot oli toteutettu kokonaan shelli skriptillä.

Koko opinnäytetyöstä noin kolmannes meni Hudsonin muokkaamiseen ja siihen projektien pystyttämiseen, josta taas puolestaan shelli skriptin opettelu ja kirjoitus veivät isoimman osan ajasta. Etenkin yksi testiprojekti, jossa käytettiin Androidin sijaan iPhone alustaa, vei paljon aikaa. Tämä jouduttiin tekemään lähes kokonaan raskailla shelli skripti logiikoilla.

2.13 Versointikäytäntyö Hudsonissa

Koska Hudson ei tarjonnut yhtiön tarpeisiin sopivaa versiointipalveluja, piti se toteuttaa käyttämällä kolmansien osapuolien liitepakettia, jolla saatiin kysyttyä tietoarvoja järjestelmästä ja siten saatiin käyttöön tarvittavat tiedot. Kyseessä oli Hudsonin versiointi -liitäntäpaketti. Lisäksi avuksi tarvittiin shelli skriptiä itse versionumeron laskemiseen, sekä lopullisen arvon syöttämiseen Hudsonin tietokantaan.(Hudson version plugin)

Versiointiäytännöstä käytiin keskustelua ja neuvottelua sekä ohjelmoijien että myös johdon kanssa. Koska joitakin projekteja rakennettiin paljon, ei haluttu että pääversionumerointi kasvaisi mielettömän nopeasti. Silti haluttiin erotella eri Hudson-buildid toisistaan, jotta myöhemmin voitaisiin tarkistaa lähdekoodi, josta jokin buildi oli tehty. Tämän vuoksi pidettiin parhaimpana vaihtoehtona laittaa Hudsonissa rakennetuille projekteille aina juokseva lisänumero normaalin versionumeron lisäksi. Tämä tarkoitti sitä, että varsinainen koodissa säädetty versionumero ei ikinä kasvanut Hudsonissa. Sen sijaan versionumeroon lisättiin piste sekä juokseva lisänumero, joka alkoi ykkösestä. Kun varsinainen versionumero muuttui nollattiin aina Hudonin liukuva numero myös. Hudsonin liukuva versionumero kasvoi vain onnistuneissa buildeissa.

Käytännössä tämä toteutettiin sillä tavalla, että shell skriptillä otettiin androidManifest.xml tiedostosta ylös projektin versionumero, lisättiin siihen liukuva versionumero Linuxin ”awk” -komennon avulla. Sitten korvattiin alkuperäisen androidManifest.xml tiedoston versionumero uudella Hudsonin liukuvan versionumeron sisältäneellä versionumerolla.

2.14 Hudsonin tarjoamat lisäpalvelut

Tähän saakka käsiteltyjen ominaisuuksien lisäksi tarjoaa Hudson vielä huomattavan määrän muita palveluita ja työkaluja. Nettikäyttöliittymän kautta on säädettävissä kaikki Hudsonin asetukset eikä Hudsonin lähdekoodiin tarvitse siis koskea ollenkaan. Samoin Hudsonin liitännäisetkin asentuvat selaimessa.

Hudsonissa voi säätää monenlaisia asetuksia alkaen projektin ryhmistä päätyen etäkoneisiin, joita voi lisätä äärettömän määrän jakamaan buildauksesta aiheutuvaa työkuormaa. Myös globaaleja muuttujia, käyttäjäoikeuksia ja tiedostopolkuja pystyy

säätämään. Asetuksia ei ole paljon, mutta kaikki asiat, jolla on merkitystä löytyvät joko Hudsonin peruspaketista tai ainakin liitännäisistä. Jotkut liitännäispaketit saattavatkin keskittyä pelkästään asetuksiin ja tuoda näin ollen lisää säätövaraa Hudsonin toimintaan.

Itse palveluista tärkeimpiä ovat Hudsonin Maven rajapinta, jatkuva integraatio, Javadocin tuotto, buildien varastointi sekä revisiotietojen säilytys. Mavenia eikä jatkuvaa integraatiota tässä työssä käsitellä, sillä ne eivät olleet tarpeellisia alueita yhtiön työnmäärittelyssä. Javadocin automaattinen generointi sekä esille asettelu taas olivat kätevä lisä, etenkin kun sen toiminnon lisäämiseen ei tarvinnut tehdä juurikaan ylimääräistä työtä. Sen avulla kuitenkin saa näkyville tiedot koko käännetyn lähdekoodin luokista ja funktioista sekä revisioiden kommentit.

2.15 Paketin sekä koodin postitus ja varastointi

Myös sähköpostipalvelut ovat mainitsemisen arvoinen lisäosa Hudsonissa. Hudsonissa oletuksena sähköpostipalvelut keskittyvät buildin epäonnistumisesta tai onnistumisesta ilmoittaviin huomautuksiin. On kuitenkni myös mahdollista saada lähetettyä käännöksen lopputuloksia eteenpäin sähköpostissa. Tämä oli etenkin kätevä yhtiön älypuhelinkehityksen kannalta, sillä buildin lopputukoksena syntyvät .apk paketit ovat suoraan asennettavissa kännykkään sähköpostista.

Käännöstulosten sekä revisiotietojen säilytys jokaisesta tehdystä buildista on tärkeä ominaisuus Hudsonissa. Sen avulla voidaan säilöä eri käännettyjä versioita niin kauan kun on tarvetta siihen. Sitä voidaan esimerkiksi hyödyntää, kun yritetään paikantaa jonkun ohjelmistoon tulleen bugin paikannusta tiettyyn revisioon. Se myös automaattisesti varastoi kaikille asiakkaille lähettämät buildit siten, joten vanhat buildit löytyvät aina helposti varastosta.

2.16 Epäonnistuneet rakennusyritykset

Sähköpostihälytysten lisäksi epäonnistuneista buildeista on mahdollista saada esiin logitietoa. Koko rakennusprosessista jää näkyviin konsoliin lokikirja, josta näkee mihin osaan Hudsonilla käännetty projekti kaatui. Mikä tahansa virhetilanne joko shell skriptissä tai integroiduissa Hudsonin palveluissa aiheuttaa aina epäonnistuneen rakennusyrityksen. Jos yksikin osa epäonnistuu, niin rakennusvaihe keskeytyy. Epäonnistuneiden buildien tapauksessa on mahdollista myös liittää automaattisia lisätoimintoja sähköpostihuomautuksen lisäksi, jotka laukaistaan jos buildi ei onnistu.

3 VERSIOINTIKÄYTÄNTÖ

Versiokäytäntö on erittäin tärkeä osa ohjelmistoyrityksessä etenkin jos yhtiöllä on lukuisia tuotteita sekä tiheä julkaisuväli uusille versioille. Versiokäytäntöjä on monia ja siihen ei ole olemassa tiettyä oikeata standardia. Sen sijaan versiointistandardeja on useita ja joskus niitä sekoitetaan vielä keskenään. Versio voi olla yhtä hyvin koodinimi, numero, pisteytetty numerosarja tai numerosarja, jossa on myös mukana kirjaimia. Versionumeron pituudella tai pisteiden määrällä ei ole rajaa vaan useimmiten käytetään parhaiden tarkoitukseen sopivaa tapaa. Eniten käytetty tapa on kaksi tai kolmeosainen numeroyhdistelmä (joskus myös sisältää kirjaimia), jossa jokainen numero on eritetty keskenään toisistaan pisteellä. Tällä menetelmällä ensimmäinen numero kertoo ohjelman valmistumistilanteesta, jossa nolla viittaa yleensä, että tuote ei ole vielä julkaisuasteinen. Toinen numero usein viittaa tärkeisiin päivityksiin, joka tuo ison muutoksen tai lisäyksen tuotteeseen. Kolmas numero usein on liukuvampi ja sitä kasvatetaan aina, kun tuotteesta julkaistaan uusi versio pienin korjauksin. Tämä on siis käytäntö jos käytetään kolminumeroista koodia. Kahden numeron järjestelmä usein viittaa siihen, että ohjelmasta ei julkaista kovin usein uusia versioita vaan muutokset ovat isompia kokonaisuuksia, jotka sisältävät paljon pieniä sekä isompiakin muutoksia. Myös kirjaimien käyttö- kuten a (alpha), b (beta) tai rc (release canditate) ovat joskus lisäliitteinä. Samoin versioissa voidaan viitata kantoihin, kuten esimerkiksi 3.1.x – kannalla tarkoitetaan kaikkia versioita jotka alkavat numeroilla 3.1, riippumatta mitä sen jälkeen tulee. (Software versioning, wikipedia)

Versioinnilla on lukuisia merkityksiä. Paitsi, että se erottaa tuotteen eri versiot toisistaan, tarjoaa se myös visuaalisuutta asiakkaalle siitä, että tuotetta kehitellään, kuten myös kaikille osapuolille siitä missä vaiheessa tuotekehitystä ollaan. Vaikka jälkimmäinen ei aina noudata standardeja, on yleinen käytäntö se, että kun aletaan lähestyä 1 alkavaa johtavaa numeroa, alkaa tuote olla valmiina julkaisuun. Merkittävin on kuitenkin versioinnin tuoma mahdollisuus viitata tiettyyn tilaan tuotteesta. Näin varmistetaan että kommunikaatio osapuolien välillä on helpompaa, kun tiedetään että puhutaan juuri tietystä samasta tuoteversiosta. (Software versioning, wikipedia)

3.1 Versiokäytäntö Eye Solutionsissa

Versiokäytäntö oli yhtiössä yksi osa-alueista, jossa toivottiin parannusta. Vanha versiokäytäntö vaihteli eri tuotteissa, sekä merkitsevien numeroiden määrän suhteen, kuten myös sen suhteen, että päivitettiinkö niitä ollenkaan. Vaihtelu oli tuotekohtaista ja käytäntö vaihteli etenkin eri henkilöiden välillä. Tämä aiheutti ongelmia etenkin jos versionumeroa ei kasvatettu ollenkaan. Lisäksi, välttämättä ei tiedetty, mikä buildi jollakin laitteella tai asiakkaalla oli käytössä. Haluttiin, että käyttöön otettaisiin selvä ja toimiva yhtiön sisäinen versiointistandardi.

Asiasta käytiin neuvottelua eri ohjelmoijien sekä johdon kanssa ja lopulta päätettiin, että otetaan käyttöön neljänumeroinen versiokäytäntö. Käytännössä kyse oli kolmenumeroisesta versiokäytännöstä, johon lisättiin jo aikaisemmin mainittu Hudsonin lisäämä liukuluku. Tämä siksi, että jokainen Hudsonin buildi olisi ainutlaatuinen numeron suhteen ja siten helposti viitattavissa. Kolme ensimmäistä numeroa kuvaisivat versiotilaa yleisempien versiostandardin tapaan. Ensimmäinen numero esittäisi tuotteen yleistä kehitystilaa, joka kasvaisi vain jos saavutettaisiin joku uusi virstanpylväs kehityksessä. Toinen viittaisi uuteen merkittävään muutokseen tuotteessa sekä kolmas kuvaisi pienempiä muutoksia sekä korjauksia.

Koska yrityksen tuotteet jakautuvat useaan eri itsenäiseen osaan on yrityksessä jokaiselle tuotteelle oma versionumero. Vaikka tuotteita tarjotaankin paketissa, vaihtelee tilattujen osien määrä sen mukaan, mitä osia asiakas tilaa. Siten moduulikohtainen versionumerointi toimii parhaiten.

4 WIKI

Wiki on verkostoitunut tietokanta, jolla voidaan esittää ja asiakirjoja sekä linkittää niiden sisältöä toisiinsa. Wiki-palveluja on useita erilaisia riippuen käyttötarkoituksesta. Wiki:n tarkoitus yleensä on esittää ja tarjota palveluja kasvavalle ja muuttuvalle tieto- sekä dokumentaatiokannalle, jossa eri käyttäjät eri oikeuksineen voivat lisätä tai muokata Wikin sisältöä. Wikit useimmiten toimivat omilla yksinkertaisilla syntakseilla, jolla voidaan luoda peruselementtejä kuten: otsikoita, alaotsikoita sekä tauluja ja muita visuaalisia objekteja. Myös yksinketaiset html syntaksit toimivat useimmissa wikeissä. Visuaalisen käyttöliittymäkerroksen lisäksi wikit tarvitsevat toimiakseen tietokantaan. Wikit ovat kevyet rakenteeltan ja palveluiltaan, mutta saattavat sisältää paljon tietosisältöä ja siksi tietokanta saattaa kasvaa isoksi. (Wiki – Wikipedia)

4.1 Wiki yhtiön tarpeisiin

Yksi lopputyöni osa-alueista oli wikin pystytys palvelemaan yhtiön dokumentointi-tarpeita. Koska vuoden jälkeen yhtiöltä silti puuttui informaatiotietokanta, piti kaikki tieto saada kirjattua johonkin yhteiseen järjestelmään. Tästä syystä yhtiölle luotaisiin oma sisäinen wiki- sivusto, johon kaikki tieto voitaisiin laittaa korvamaan hajanaiset paperilla lojuvat merkinnät ja muistiinpanot. Tämä oli etenkin siksi tärkeä, koska aikaisemmin ei yhtiössä ollut mitään tapaa järjestää tietoa sähköpostin ja hajanaisten paperilappujen lisäksi.. Wikin tarkoitus oli siis yleisesti toimia dokumenttivarastona, johon kaikki työntekijät sekä johto pääsivät käsiksi. Siihen tulisi kaikki alkaen työntekijöiden yhteystiedoista päätyen seuraavan version vaatimuslistoihin ja bugikantoihin. Myös käyttöohjeita sekä muuta olennaista tietoa tulisi sinne helpottamaan työtä ja vähentämään toistuvia kysymyksiä työntekijöiden välillä.

Tutkimuksen jälkeen päätin ottaa käyttöön Mediawikin. Itse wiki palveluntaroajan suhteen ei yhtiöllä ollut toivetta, joten päätin käyttää yhtä eniten levinneistä wiki palveluista. Tämä päätös siksi, koska Mediawikistä löytyi hyvin dokumentaatioita sekä käyttö- että asennusohjeita ja se oli helposti asennettavissa Linuxin LAMP (linux, apache,mysql sekä php, perl tai python) palvelimelle. Turvallisuussyistä wiki suljettiin sisäverkon ulkopuolelta.

4.2 Wikin pystytys

Wiki pystytettiin tuoreelle komentorivipohjaiselle Linux koneelle. Ennen, kun voitiin asentaa Mediawikin .jar –paketti piti asentaa sen esivaatimukset. Tämä tuotti jonkin verran ongelmia, sillä en aikaisemmin ollut pystyttänyt Linuxiin nettisovellusta, puhumattakaan palvelimista ja tietokannoista. Tähän löytyy kuitenkin lukuisia oppaita netistä. Palvelinkoneseen asennettiin Ubuntu versio, joka on yksi eniten levinneimmistä Linux-versioista, joten esivaatimusten asentamisesta löytyi helposti ohjeita alusta loppuun. Eniten ongelmia tuotti vaadittavien pakettien ja niiden eri versioista aiheutuvat ristiriidat. Koska ohjeet oli tehty tietyille versioille apachesta ja mysql:stä ei vanhat ohjeet toiminut uusien versioiden kanssa ja siksi piti olla tarkkana että käytti juuri oikeiden versioiden ohjetta. (Manual:Installing MediaWiki)

Pienen esiasennuksen jälkeen saatiin Mediawiki pystyyn (kuva 7) Apache palvelimelle ja sidottua se omaan mysql tietokantaan. Itse Mediawikin muokkaaminen ja käyttäjäoikeuksien muuttaminen oli yksinkertaista. Koska kaikilla ei ollut kokemusta wikin käytöstä tai muokkauksesta, piti siitä vielä laatia ohjeet itse uuteen wikiin.

Kuva 7. Mediawiki asennuksen jälkeen.

5 MANTIS BUG TRACKER

Mantis Bug Tracker (kuva 8), on suosittu ilmainen bugien kanssa työskentelyyn suuniteltu järjestelmä, joka on kirjoitettu PHP:llä. Toimiakseen se vaatii verkkopalvelimen sekä jonkin tietokannannoista, tässä tapauksessa mysql:n. Se toimii

Windows, Linux ja Mac Os käyttöjärjestelmissä sekä kaikilla selaimilla. (Mantis bug tracker).

Kuva 8. Mantis Bug Trackerin yleisnäkymä

Se tarjoaa lukuisia palveluita bugien kanssa työskentelyyn ja hienon muokattavissa olevan kapseloinnin eri käyttäjäryhmien välille. Bugien raportoinnille löytyy valmiit kaavakkeet, joita täyttämällä saadaan niiden korjauksen kannalta olennaiset asiat ylös. Bugeille löytyy omat vaihetilat, joiden välillä eri bugeja on helppo siirtää täyttäen muutaman lisäkentän, jossa kerrotaan siirron syyt. Bugeja on myöskin mahdollista järjestää eri ryhmien alle. Yleisesti ottaen ryhmittely on Mantis Bug Trackerin vahvimpia puolia sillä järjestelmä pyrkii pitämään bugilistat siistinä, että myös informatiivisina. Näin ollen voidaan helposti esittää lukuisia eri tilastoja, joilla avulla voidaan seurata miten yhtiössä bugien hoito toimii.

Toinen Mantis Bug Trackerin vahva puoli on käyttäjien ja niille avautuvien näkymien sekä oikeuksien kapselointi. Tämä mahdollistaa sen, että yksi bugijärjestelmä kattaa kaikki tarpeet ja vaatimukset, mitä yhtiöllä on eri asiakkaiden kanssa. Kaikki oikeudet on säädettävissä ja halutut asiat voidaan pitää tai poistaa niille kuuluvilta tai kuulumattomilta ryhmiltä. Esimerkiksi näin voidaan välttää ettei asiakkaat pääse käsiksi

niille kuulumattomiin tietoihin. Toisaalta näin voidaan sulkea myös pois turhaa informaatiota niiltä ryhmiltä, jolle kyseinen tieto ei ole tarpeellista.

5.1 Asentaminen yhtiön tarpeisiin

Mantisin asentaminen oli jokseenkin samanlaista kuin Mediawikin asentaminen. Tuotteilla oli samat esivaatimukset. Kuten siis Mediawikikin, pohjautuu myös Mantiksen .war paketti LAMP- palvelimeen. Koska Linuxiin oli jo asennettu valmiiksi Apache palvelin, mysql, php sekä python niin toista kertaa ei tarvinnut näitä enää asentaa. Tosin mysql:ään jouduttiin tekeemään oma tietokanta Mantis:lle, kuten myös asettamaan Mantikselle oma portti Apacheen. Näin ollen samalla palvelimella pyöri sekä Mantis että Mediawiki.

Mantis Bug Tracker jäi asennuksen jälkeen hetkeksi sivuun, mutta suunnitelmien kehittyessä yhtiössä tarvittiin paikka, johon asiakkaat voisivat raportoida bugeja. Koska Mediawiki oli vain yhtiön sisäisessä verkossa ei sitä voitu käyttää bugien raportointiin asiakkaiden puolelta. Toisaalta se ei ollut luontainen palvelu kyseiseen tarkoitukseen muutenkaan. Tehtäväni olikin laatia valmiiksi kattavat ohjeet Mantis Bug Trackerin käytöstä. Ohjeissa piti käydä ilmi paitsi mihin kaikkeen ohjelmalla pystyy myös se miten bugien raportointi sekä seuraaminen toimivat. Laadin asiasta Powerpoint-esityksen, jossa ainakin bugien raportointi sekä niille annetut asetukset olivat selitetty.

Muutamia Mantiksen säädettäviä yleisasetuksia ja niiden tarpeellisuuta jouduttiin miettimään yhtiön näkökulmasta. Tämä esimerkiksi tuli esiin kun luotiin käyttöjärjestelmäprofiileja. Koska Mantis Bug trackeria ei virallisesti oltu suunniteltu mobiilikehitykseen, ei esimerkiksi käyttöjärjestelmän syöttö bugikaavekkeeseen ollut järkevää. Tämä siksi, että Android puhelimia oli yhtiössä käytössä kymmeniä eri malleja ja samoissa malleisakkin saattoi vielä olla omat käyttöjärjestelmäversiot. Tästä syystä ei lienenyt järkevää ottaa mukaan käyttöjärjestelmää bugiraportteihin.

5.2 Käyttäjäoikeudet, käyttäjäryhmät sekä järjestelmän vakiot Mantis Bug Trackerissa

Tehtäväni oli saada Mantis Bug Tracker toimimaan siten, että sitä olisi mahdollista käyttää niin eri asiakkaiden kun myös sisäisten työntekijöiden toimesta. Tästä syystä jouduttiin muokkaamaan paljon alkuperäisiä vakioasetuksia, jotta saatiin eroteltua eri käyttäjäryhmät toisistaan ja jakaa oikeuksia heille sen mukaan miten he niitä tarvitsevat. Etenkin asiakastiedot oli pidettävä erillään toisista asiakkaista, ja siksi asiakkaille ei sallittu muita oikeuksia kuin vain nähdä tiedot omista raportoiduista bugeista ja niiden tilasta. Myöskin työntekijöiden sekä johdon omat raportoimat bugit haluttiin pitää pois näkyviltä asiakkaiden näkymästä. Näin ollen saatiin aikaan järjestelmä, joka tukee useamman asiakkaan bugiraportointia, mutta lisäksi toimii myös yhtiön sisäisenä bugijärjestelmänä.

6 BUGIEN RAPORTOINTI

Bugien raportointi oli myöskin tärkeä osa alue yhtiön toiminnassa, sillä huomattiin lukuisien bugien ja eri ohjelmistoversioiden aiheuttavan sekaannusta tässä. Samaa bugia saatettiin korjata monta kertaa tai pahimmillaan jättää korjaamatta. Bugien raportoinnille ei yhtiössä ollut mitään käytössä olevaa järjestelmää, joten sille toivottiin pikaista ratkaisua. Yksi nopea ehdotus oli bugien raportointi Wikiin, kuten aikaisemmin mainittiin, ja niin myös tehtiin. Käytännön kokemuksen kautta wiki osoittautuikin riittäväksi neljän ohjelmoijan tarpeisiin. Sinne pystyi helposti raportoimaan bugeja kuka tahansa nopeasti, eivätkä listat ikinä olleet liian pitkiä tai sekavia. Myös kuka tahansa pääsi yhtiön sisältä helposti katsomaan, mitä on jo raportoitu ja mitä ei tai missä vaiheessa jokin bugin tila on. Myöhemmin, kun asiakassuhteita ja palveluita alettiin miettimään tarkemmin, huomattiin, että oli laadittava myös järjestelmä, joka palvelee asiakkaita.

6.1 Bugien kanssa työskentely

Tämän hetkinen tilanne Eye Solutionsissa on se, että Mediawiki on yhä käytössä yhtiön sisäissessä bugiraportoinnissa. Kyseessä on siis ihan vain dokumenttisivu, johon voidaan kirjoittaa bugeja ja niitä voidaan yliviivata, kun ne on korjattu tai hoidettu. Tämä on kätevä sekä nopea tapa raportoida jokapäiväisiä bugeja eikä kattavia Mantis Bug Trackerin bugiraportointipalveluja tarvita. Useimmat bugit korjataan nopeasti

muutaman päivän sisällä, joten ne eivät ikinä ehdi vanhentua tai unohtua. Useimmiten mainitaan asia myös huoneen yli suullisella viestillä ohjelmoijalle, joka on vastuussa kyseisestä bugista. Tämä onkin toimiva tapa pienessä yrityksessä sisäiseen käyttöön.

Mantis Bug Trackerissa bugit pitää raportoida ensin järjestelmään jonkun toimesta. Valmiisiin kaavakkeisiin joudutaan syöttämään tiedot missä, miten ja milloin bugi saatiin aikaan sekä tietoa onko bugi toistuva / toistettavissa tai mikä sen prioriteetti on. Tämä on kätevä asiakassuhteissa, koska muuten voisi jäädä tärkeätä informaatiota saamatta, koska asiakas ei välttämättä osaa itse kertoa ohjelmoijille tarpeellisia yksityiskohtia asiasta. Tämän virallisen bugijärjestelmän etuja on myös se, että kun tieto on kirjallisena ja kattava ei haittaa vaikka bugin korjaus siirtyisi, sillä tieto sekä yksityiskohdat pysyvät tallessa eivtäkä siten pääse unohtumaan. Myöskin Mantis Bug Trackerin tuomat tilastopalvelut ovat hieno visuaalinen lisä bugipalveluissa sekä mahdollinen mainoskeino asiakkaita varten.

6.2 Yhteenveto

Yhtiössä oli paljon haastavia tehtäviä ja tutkimiskohteita, joista jokaista tuli koitettua ja testattua. Monet niistä otettiin käyttöön ja ne ovat edelleenkin käytössä jokapäiväisesti yhtiössämme. Näistä esimerkkejä ovat yhtiön wiki sekä Hudson serveri. Myös Mantis Bug trackerille löytyy luultavasti enemmän käyttöä tulevaisuudessa.

Git on jäänyt toistaiseksi sivuun, koska se ei olisi tuonut merkittäviä parannuksia tai uudistuksia Eye Solutionsille. Siitäkin on tehty tutkimus ja asiakirjat, joiden avulla siirtyminen Git:iin on helpompaa jos joskus niin halutaan.

Työ oli siis haastava, mutta erittäin opettava. Samalla tutkiessani ja testatessani työni päätehtäviä opin myös paljon yhtiön tavoista ja tuotteista. Etenkin Androidi alusta tuli jo osittain tutuksi, vaikka se ei ollut työni varsinainen tutkimus- tai kehitysaihe.

Yhtiössä työtapojen ja standardien kehitys sekä automatisointi on tärkeää. Etenkin tähän pitäisi kiinnittää huomiota alkuvaiheessa. Muuten ne voivat jäädä helposti huomiotta ja opitaan sekä totutaan tekemään asioita vähemmän tehokkailla tavoilla. Tällöin työntekijöiltä menee hukkaan paljon tehokasta työaikaa ja virhealttius kasvaa.

7 LÄHTEET

git- the fast version control system [online] [viitattu 21.8.2011] Saatavissa:
http://git-scm.com/

git- User's Manual [online] [viitattu 21.8.2011] Saatavissa:
http://schacon.github.com/git/user-manual.html

Version Control for Designers[online] [viitattu 21.8.2011] Saatavissa:
http://hoth.entp.com/output/git_for_designers.html

gitutorial - [online] [viitattu 21.8.2011] Saatavissa:
http://schacon.github.com/git/gittutorial.html

Apache subversion - [online] [viitattu 21.8.2011] Saatavissa:
http://subversion.apache.org/docs/community-guide/

Git – Svn crash course [online] [viitattu 21.8.2011] Saatavissa:
http://git.or.cz/course/svn.html

Migrate your subversion repository to a git repository [online] [viitattu 21.8.2011] Saatavissa:
http://www.jonmaddox.com/2008/03/05/cleanly-migrate-your-subversion-repository-to-a-git-repository/

Git – software (Wikipedia) [online] [viitattu 21.8.2011] Saatavissa:
http://en.wikipedia.org/wiki/Git_(software)

Meet Hudson – hudson- Hudson wiki [online] [viitattu 21.10.2011] Saatavissa:
http://wiki.hudson-ci.org/display/HUDSON/Meet+Hudson

Hudson (software) - Wikipedia [online] [viitattu 21.10.2011] Saatavissa:
http://en.wikipedia.org/wiki/Hudson_(software)

Installin Hudson – hudson- Hudson wiki [online] [viitattu 21.10.2011] Saatavissa:
http://wiki.hudson-ci.org/display/HUDSON/Installing+Hudson

Apache ant – User Manual [online] [viitattu 25.10.2011] Saatavissa:
http://ant.apache.org/manual/index.html

Plugins – hudson- Hudson wiki [online] [viitattu 25.10.2011] Saatavissa:
http://wiki.hudson-ci.org/display/HUDSON/Plugins

Shells – hudson- Hudson wiki [online] [viitattu 25.10.2011] Saatavissa:
http://wiki.hudson-ci.org/display/HUDSON/Shells

Hudson version plugin [online] [viitattu 27.11.2011] Saatavissa:
http://wiki.hudson-ci.org/display/HUDSON/Version+Number+Plugin

Software versioning, wikipedia [online] [viitattu 28.11.2011] Saatavissa: http://en.wikipedia.org/wiki/Software_versioning

Wiki – Wikipedia [online] [viitattu 28.11.2011] Saatavissa: http://fi.wikipedia.org/wiki/Wiki

Manual:Installing MediaWiki [online] [viitattu 28.11.2011] Saatavissa: http://www.mediawiki.org/wiki/Manual:Installing_MediaWiki

Mantis bug tracker [online] [viitattu 28.11.2011] Saatavissa: http://www.mantisbt.org/