OULUN SEUDUN
AMMATTIKORKEAKOULU

Mikko Korhonen

JATKUVAN INTEGRAATION KÄYTTÖÖNOTTO OHJELMISTO-KEHITYKSESSÄ

JATKUVAN INTEGRAATION KÄYTTÖÖNOTTO OHJELMISTO-KEHITYKSESSÄ

Mikko Korhonen
Opinnäytetyö
Syksy 2013
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, langattomat laitteet

Tekijä: Mikko Korhonen
Opinnäytetyön nimi: Jatkuvan integraation käyttöönotto ohjelmistokehityksessä
Työn ohjaajat: Ensio Sieppi (OAMK) ja Kari Vengasaho (Ouman Oy)
Työn valmistumislukukausi ja -vuosi: Syksy 2013
Sivumäärä: 39

Tämä opinnäytetyö toteutettiin Ouman Oy:n toimeksiannosta. Työn aiheena oli jatkuvan integraation käyttöönotto yrityksessä, ja sen tavoitteena oli edistää yrityksen ohjelmistokehitystä. Tarkoitus oli automatisoida ohjelmistokehityksen osia, jotka vaativat turhaa manuaalista toistamista, kuten ohjelman kääntämisen, sekä ottaa käyttöön ns. integraatiopalvelin. Tämä vähentäisi ohjelmistokehittäjien työtaakkaa ja helpottaisi keskittymistä olennaiseen eli itse ohjelmiston tekemiseen.

Työn tarkoitus oli keskittyä kahden tuotteen, OuflexTool-ohjelmiston ja Ouflex-laitealustan ohjelmiston kehittämiseen. OuflexTool on Windows-käyttöjärjestelmissä toimiva työkalu Ouflex-laitteen sovelluksien tekemiseen. Ouflex-laitealusta on sulautettu ohjelmisto.

Työ voitiin jakaa karkeasti kahteen vaiheeseen, joista ensimmäisessä tutkittiin eri integraatiopalvelimia ja etsittiin yrityksen käyttöön sopiva palvelin. Tätä varten etsittiin useita kymmeniä vaihtoehtoja, joista kolme valikoitui tarkempaan vertailuun. Näitä testattiin todellisen kaltaisessa ympäristössä, ja selvitettiin niiden toiminta nykyisten käytössä olevien työkalujen kanssa. Koska kaikki kolme valittua integraatiopalvelinta täyttivät käytännössä kaikki vaatimukset, oli vertailukriteereinä esimerkiksi käytön helppous sekä yhteensopivuus muiden rajapintojen kanssa. Toinen osa koostui integraatiopalvelimen käyttöönotosta ja määrittämisestä sekä järjestelmän testaamisesta käytännössä.

Työn tuloksena oli toimiva järjestelmä, joka automaattisesti kääntää OuflexTool-ohjelmiston ja testaa sitä automaattisilla käyttöliittymätesteillä. Lisäksi järjestelmä mahdollisti ohjelmiston päivityspakettien teon huomattavasti entistä helpommin. Järjestelmästä jätettiin pois laitealustan ohjelmiston kääntäminen ajanpuutteen vuoksi, ja se onkin ensimmäinen jatkokehitystavoite. Tarkoitus on jatkossa tuoda järjestelmään mukaan muitakin yrityksen tuotteita.

Asiasanat:
jatkuva integraatio, ohjelmistokehitys, Agile, TeamCity

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Wireless Devices

Author: Mikko Korhonen
Title of thesis: Introduction of continuous integration in software development
Supervisors: Ensio Sieppi (OUAS), Kari Vengasaho (Ouman Oy)
Term and year when the thesis was submitted: Autumn 2013
Pages: 39

This thesis work was assigned by Ouman Ltd. The subject of the thesis was the introduction of continuous integration to software development of the company. The main goal of the thesis was to enhance software development by automating repetitive manual tasks and taking extra work load off from developers. Automation included the building of software and testing it with user interface tests created by a tool named Ranorex.

The thesis work was planned to include two different software products, OuflexTool, which is Windows-based software to create applications for Ouflex Device, and the Ouflex device platform, which is embedded software.

There were two major phases in the thesis work, first of which was researching and finding a continuous integration server to use. For this phase, dozens of tools were found and three of those selected for closer review. Those three were tested in a clone of the production environment, and their interfaces to tools already in use were investigated. In practice, all three fulfilled the requirements set for the tool, so the criteria used to select the tool were more subjective, like ease of use and maintenance. The second phase consisted of actual deployment of the integration server and configuring the system to use. This also included testing the function of the system in production environment.

As a result of this thesis work, Ouman Ltd. now has a working continuous integration system to use, which includes an integration server that is configured and functional. The system can build OuflexTool from version control automatically, create a setup program and install the software, then test it automatically. Making of update files for the software is also remarkably easier than before, thanks to automation. Due to time-related issues it was necessary to exclude the Ouflex device platform from the system and leave it for further development of the system.

Keywords:
continuous integration, software, development, Agile, TeamCity

ALKULAUSE

Tämä opinnäytetyö tehtiin keväällä 2013 Ouman Oy:n toimeksiannosta. Ennen opinnäytetyön tekoa olin ollut yrityksessä töissä, ja edellisenä keväänä tein tutkintoon kuuluvan työelämäprojektin Ouman Oy:lle. Tämän takia oli hienoa päästä tekemään myös opinnäytetyö kyseiseen yritykseen.

Haluaisin kiittää erityisesti Kari Vengasahoa, joka toimi opinnäytetyön valvojana yrityksessä. Hänellä oli myös suuri vaikutus opinnäytetyön aiheeseen. Lisäksi haluaisin kiittää Timo Peltolaa, jonka ansiosta olen saanut työskennellä Ouman Oy:llä. Opinnäytetyön kirjallisen osuuden teon ohjaamisesta haluaisin kiittää Ensio Sieppiä, joka toimi innostajana ja ohjaajana työn aikana sekä jo ennen sitä.

Oulussa 21.10.2013

Mikko Korhonen

SISÄLLYS

1 JOHDANTO

Ouman Oy on Kempeleessä sijaitseva yritys, joka valmistaa älykästä ja helppokäyttöistä kiinteistöautomaatiota ja on saavuttanut lämmönsäädön markkinajohtajuuden Suomessa (1, linkki Ouman Oy).

Ouman Oy:n tuotekehitys sisältää paljon ohjelmistokehitystä ja täten myös ohjelmistotestausta. Silloin kun ohjelmistoja kehitetään asiakkaan käytettäväksi, tulee ohjelmiston toiminnan olla varmaa ennen markkinoille viemistä. Ohjelmistojen toistuva manuaalinen testaaminen ei ole tehokasta eikä mielekästä, kun testaus useimmiten liittyy juuri samoihin kohtiin ohjelmistossa. Tämän takia testaus on monesti muodostunut hidastavaksi tekijäksi ohjelmistokehityksessä.

Keväällä 2012 tein Ouman Oy:lle projektin, jonka aiheena oli ohjelmistotestauksen automatisointi. Tässä projektissa etsittiin työkaluja automaattiseen käyttöliittymätestaukseen ja tutkittiin mahdollisuuksia ottaa ne käyttöön Ouman Oy:llä. Kun sopiva työkalu vertailun tuloksena löydettiin, otettiin se myös projektin puitteissa käyttöön ja alettiin tehdä testiympäristöä OuflexTool-ohjelmistolle.

Nyt opinnäytetyönä tehty projekti on jatkoa edellisen kevään projektille. Tarkoituksena on automatisoida ohjelmistotestausta enemmän ja tuoda aikaisemmin tehdyt automaattiset käyttöliittymätestit osaksi kehitystä jatkuvan integraation ja testauksen muodossa. Tarkoituksena on automatisoida kaikki toistoa vaativat toimenpiteet aluksi OuflexTool-työkalun ja Ouflex-laitealustan ohjelmistokehityksessä ja näin helpottaa sekä nopeuttaa kehittäjien työtä.

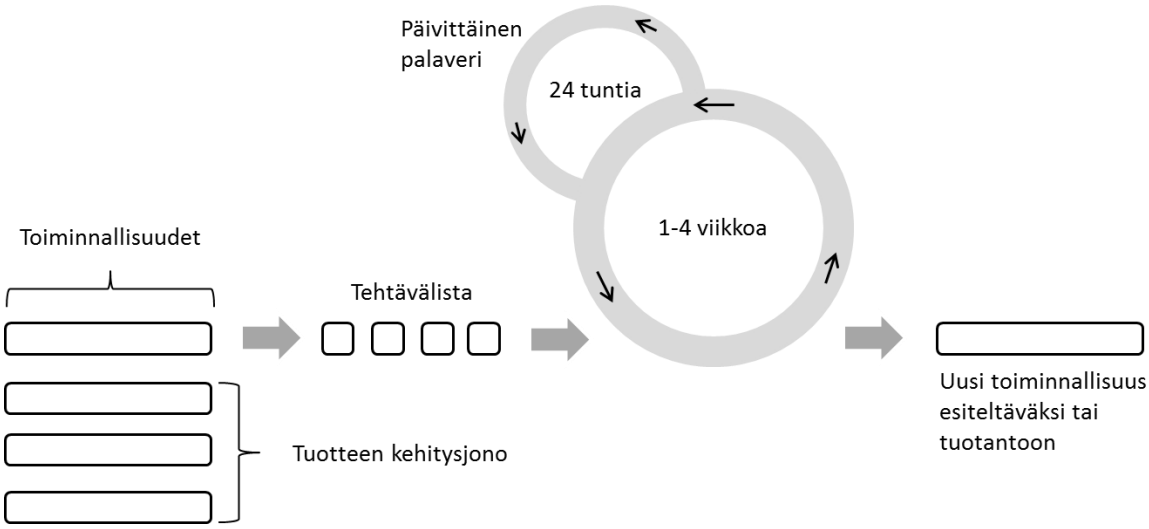
2 JATKUVA INTEGRAATIO OHJELMISTOKEHITYKSESSÄ

Kun tietokoneohjelmistoa kehittää useampi kuin yksi ihminen kerrallaan, pitää jokaisen tekemät muutokset yhdistää jossain vaiheessa. Tämä vaihe on nimeltään integraatio. Perinteisesti se on kuulunut projektin loppuvaiheeseen, mutta jatkuvassa integraatiossa sitä pyritään tekemään usein. (2.)

Jatkuva integraatio on nykyisin olennainen osa ohjelmistokehitystä, ja Ouman Oy:llä ohjelmia jo tehtiin osittain jatkuvan integraation perusteiden mukaisesti. Suuri osa työstä oli kuitenkin manuaalista ja aikaa vievää, joten integrointia ei voitu tehdä niin usein.

2.1 Agile-menetelmät

Jatkuva integraatio on osa ns. ketteriä Agile-menetelmiä. Agilen tarkoitus on mahdollistaa kehittäjäryhmän tehokas toiminta, vaikka vaatimukset ohjelmalle muuttuvat joskus hyvinkin nopeasti. Menetelmien ytimessä on nopea kehitystahti, jossa tehdään ohjelmaa useissa lyhemmissä aikaväleissä. Joka aikavälin lopussa pitäisi olla toimiva ohjelma jossa on pieniä lisäyksiä ja muutoksia. (3, s. 9–11.) Kuvassa 1 on esitetty Agile-projektin toimintamalli.



KUVA 1. Agile-projektin toimintamalli (mukaillen 4)

Tarkemmin sanottuna jatkuva integraatio kuuluu yleisimmin Extreme Programming -menetelmään, jossa kehittäjätiimi käsittelee vaatimuksia, suunnittelua ja testausta jatkuvasti, verrattuna vesiputousmalliin, jossa kaikki ovat omia vaiheitaan projektin elinkaarella (3, s. 16–19). Näitä toistetaan nopeissa sykleissä, jolloin joustavuus lisääntyy ja tiimi saa nopeammin palautetta työnsä tuloksista.

Extreme Programming -menetelmän työtahti on yleensä viikko, eli joka viikon lopussa on uusi versio ohjelmistosta julkaistavaksi. Tätä tuotosta voidaan esitellä asiakkaalle ja saada palautetta tähän asti tehdystä työstä. Viikon työn jälkeen ei vielä ole liian myöhäistä muuttaa suuntaa, jos tulos ei ole asiakasta miellyttävä. Tässä tapauksessa asiakasta edustaa tiimissä mukana oleva jäsen, joka toimii ohjaajana asiakastarpeille. (3, s. 18–20.) Ouman Oy:llä julkaisutahti on rauhallisempi, mutta OuflexTool-työkalun kehitystiimi toimii hyvin itsenäisesti ja on mukana suunnittelussa ja määrittelyssä sekä kehityksen suuntaa arvioidaan ja ohjataan usein.

2.2 Jatkuva integraatio

Tarve jatkuvalle integraatiolle on tullut suurista ohjelmistoprojekteista, joissa perinteisesti integraatiovaihe on koko projektin lopussa. Tämä voi olla jopa useamman vuoden ohjelmoinnin päätös, jossa yritetään saada nopeasti kaikkien tekemät osat toimimaan keskenään oikein. Usein tässä vaiheessa on jo kiire saada ohjelmisto valmiiksi, mutta integrointiin menevää aikaa ei pysty lainkaan ennustamaan. (2.) Tämän takia on hyvä, että ohjelmaa olisi integroitu ennen projektin loppuvaihetta, koska tuo loppuvaiheen integrointi voi kestää jopa kuukausia ja altistaa kehittäjät stressille ja hätäisille päätöksille (3, s. 183).

Jatkuva integraatio parantaa tilannetta, koska siinä tarkoitus on aina pitää toimiva versio ohjelmasta versionhallinnassa kaikkien saatavilla. Tähän toimivaan versioon jokainen kehittäjä sitten integroi omat muutoksesta hyvin usein, jopa useita kertoja päivässä. Jatkuvan integraation seurauksena kaikkien kehittäjien uusimmat muutokset ovat aina mukana versionhallinnassa olevassa ajantasaisessa koodissa, lukuun ottamatta viimeisen muutaman tunnin työtä. Tämän seurauksena kehittäjän integroidessa jälleen uutta muutosta tuohon koodiin huomataan ristiriidat hyvin pian. Nopeasti havaitut ongelmat on helpompi korja-

ta. (3, s. 183.) Työn alkaessa Ouman Oy:llä integrointia suoritettiin useammin projektin aikana, mutta ei kuitenkaan joka muutoksen jälkeen tai edes päivittäin, ja tässä oli ensimmäinen kehitettävä alue.

Jatkuvan integraation periaatteisiin kuuluu, että jos jotain voi automatisoida, se pitäisi automatisoida. Lisäksi kaikki mitä tarvitaan ohjelman kääntämiseen, tulee olla versionhallinnassa saatavilla. (2.) Myös kaikki mahdolliset oheistoiminnot, kuten asennuspaketin teko, on hyvä automatisoida ja pystyä tekemään millä vain koneella, jolla saa tiedostot versionhallinnasta. Tämän avulla käännös ja muut toiminnot seuraavat aina samaa kaavaa, eikä kehittäjien välisiä eroja pääse syntymään.

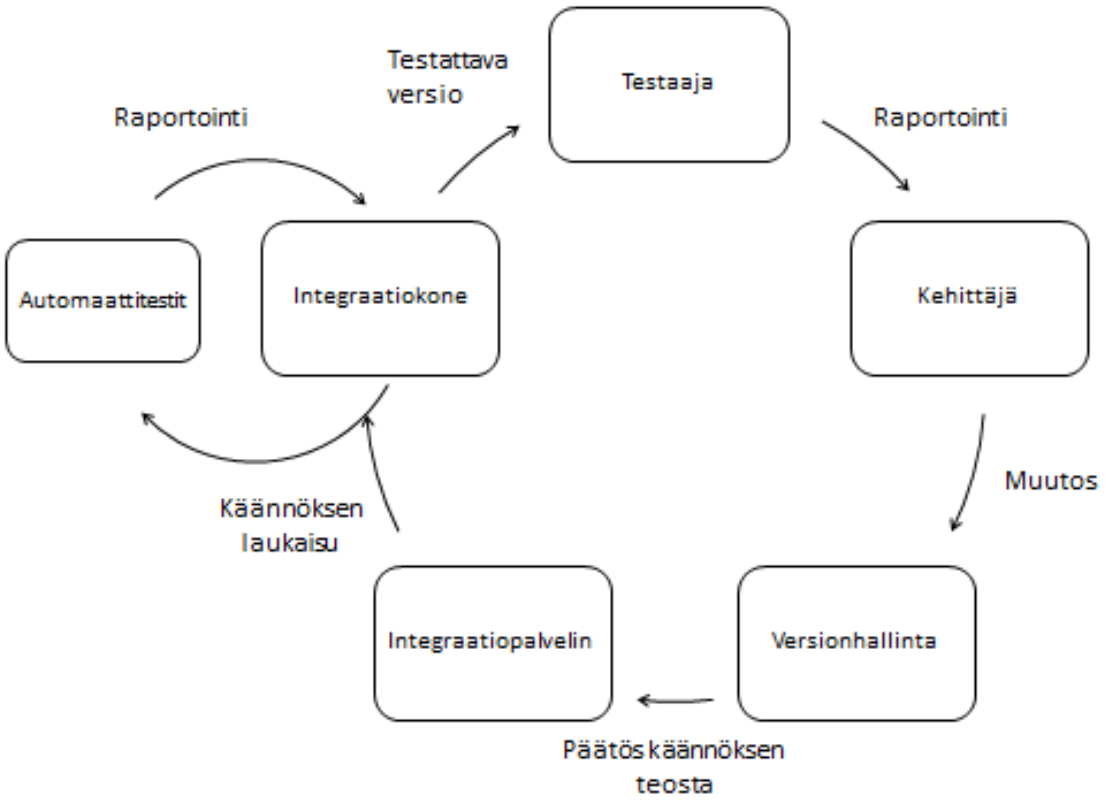
2.3 Jatkuvan integraation työkalut

Käännöksen automatisointi on jatkuvan integraation hyvä alku. Käännöksen tekoon on hyvä käyttää yhtä siihen tarkoitukseen pyhitettyä tietokonetta, jota kutsutaan myös integraatiokoneeksi. Näin ympäristö on aina sama, eikä yksittäisen kehittäjän käyttämä kehitysympäristö vaikuta kääntämiseen. (2.) Tätä helpottavat myös integraatiopalvelimet tai -työkalut, jotka automaattisesti kääntävät ohjelman.

Tässä työssä tutustutaan kolmeen integraatiotyökaluun ja vertaillaan niiden soveltuvuutta. Integraatiotyökalujen tarkoitus on hakea koodi versionhallinnasta, ja tehdä käännös sekä muita toimenpiteitä, mukaan lukien testaaminen, automaattisesti. Tällöin kehittäjän tehtäväksi integroinnissa jää koodin vieminen versionhallintaan ja mahdollisten ristiriitojen selvittäminen. Lisäksi työkalut auttavat tulosten selkeässä esittämisessä ja saatavuudessa huomattavasti.

Jatkuvan integraation periaatteen mukaan käännöksen tulee olla itsensä testaava, eli käännös tulkitaan epäonnistuneeksi, jos testeistä joku epäonnistuu (2). Tässä työssä testeinä ovat Ranorex-työkalulla tehdyt käyttöliittymätestit, joiden ajaminen vie pidemmän aikaa kuin perinteisten yksikkötestien. Käyttöliittymätestit antavat paremman kuvan siitä, miten ohjelma toimii käyttäjän näkökulmasta ja miltä ohjelma käyttäjästä näyttää. Koska testien ajamiseen menee pidemmän aikaa, tulee käännösprosessista kaksiosainen. Ensimmäisessä

osassa pelkästään käännetään testattava ohjelma, ja se tulkitaan onnistuneeksi, jos käännösvirheitä ei tule. Toisessa osassa ohjelmistosta tehdään myös asennuspaketti, se asennetaan testikoneeseen ja ajetaan käyttöliittymätestit. Tämän vaiheen onnistuminen riippuu testien onnistumisesta. Kuvassa 2 on esitetty integraatiopalvelimen periaate.



KUVA 2. Integraatiopalvelimen periaate

Kun jatkuvan integraation seurauksena jokaisen muutoksen jälkeen tehdään ohjelman käännös ja asennuspaketti, on testaajille aina saatavilla uusin mahdollinen versio. Tätä versiota voivat tarkastella myös muut projektissa mukana olevat henkilöt, esimerkiksi tutkiessaan uusia ominaisuuksia. Tämä tuo läpinäkyvyyttä kehitysprosessiin, kun uusimmat tuotokset ovat aina saatavilla. Integraatiotyökalujen myötä on myös mahdollista liittää tehdyt muutokset tiet-

tyyn versioon, jolloin työn etenemistä voidaan seurata, ja tiedetään mitä muutoksia missäkin versiossa on tullut.

Integraatiotyökalun eli palvelimen käyttö ei ole pakollista, mutta se helpottaa monessa tilanteessa jatkuvan integraation käyttöönottoa, kun kaikille on helpompi nähdä, mitä integraatiokoneella tapahtuu, ja monet tehtävät on helppo automatisoida. Tärkeintä on kuitenkin se, että kehittäjät ovat mukana ja toimivat jatkuvan integraation mukaan, koska ketään ei voi pakottaa integroimaan tiheään ja pitämään huolta siitä, että käännös onnistuu aina. (3, s. 186.) Lisäksi integraatiopalvelin tarvitsee ylläpitäjän, joka huolehtii siitä, että palvelin toimii jatkuvasti, jotta kehittäjät voivat luottaa siihen.

3 KEHITYSYMPÄRISTÖ

3.1 Ohjelmistokehityksessä käytössä olevat työkalut

3.1.1 Versionhallinta Subversion

Ouman Oy:llä on käytössä ohjelmistokehityksessä versionhallintaohjelmisto Subversion. Subversion perustuu avoimeen lähdekoodiin ja on ilmaiseksi käytettävissä. Subversion (lyhennetään SVN) pitää kirjaa ohjelmiston koodin muutoksista ja mahdollistaa monipuolisen versioinnin. Subversion luo palvelimelle hakemistorakenteen joka sisältää trunk-, tags- ja branches-hakemistot.

Trunk-hakemisto sisältää aina viimeisimmän koodin kaikkine uusine muutoksineen, ja sitä käytetään integraatiojärjestelmässä testattavana pohjana. Tags-hakemistoon tehdään jokaisesta versiosta oma kansio, joka sisältää kaikki sen version koodit ja tiedostot. Näin myös jälkikäteen on haettavissa suoraan tietty versio ja sen sisältämät muutokset. Branches-hakemisto sisältää ohjelman haaroja, joita voidaan käyttää esimerkiksi jonkin ominaisuuden testaamisen eriyttämiseen päähaarasta.

Versionhallinnan käyttö koostuu itse versionhallinnassa olevista tiedostoista sekä käyttäjän sieltä itselleen hakemasta työkopiosta. Työkopio on käyttäjän paikallisella koneella, ja siinä olevia tiedostoja voidaan muokata kuten tavallisia tiedostoja. Kun paikallisessa työkopiossa tehdyt muutokset haluaa vielä versionhallintaan muiden saataville, täytyy tehdä ns. kommitointi eli ”commit”. Tämä on Subversionin komento, joka vie muokatut tiedostot versionhallintaan. Mikäli juuri tuo muokattu tiedosto on muuttunut työkopion hakemisen ja kommitoinnin välillä, eli joku muu on sitä välissä päivittänyt, täytyy kehittäjän ensin integroida molempien muutokset yhteen. Mitä harvemmin tätä tekee, sitä enemmän muutoksia versionhallintaan on voitu laittaa.

Mikäli jotkin tiedostot ovat toisistaan riippuvaisia, ei se vaikuta kommitointia tehdessä mihinkään. Kuitenkin tiedosto, joka vaikuttaa juuri uusimpaan muutokseen, on voinut muuttua, eikä kehittäjän tekemä muutos toimi kuten pitäisi. Lisäksi jos muutoksia tehdessä luo uusia tiedostoja tai muokkaa useampia tiedos-

toja, tulee kaikki ne viedä versionhallintaan, jotta ohjelmisto toimii oikein. Tämä ei kuitenkaan aina ole niin yksinkertaista, kun tiedostoja on paljon, ja vaikka ne sijaitsevat samassa kansiossa, ei kaikkia ole automaattisesti versioitu.

Integraatioyökaluja vertailtaessa tuli ottaa huomioon, että työkalu osaa hakea koodit ja tiedostot Subversion-versionhallinnasta täysin ilman käyttäjän puuttumista tapahtumiin.

3.1.2 Virheraportointityökalu Jira

Ouman Oy:llä on käytössä ohjelmistokehityksessään ohjelmointivirheiden ja muiden kirjausten kuten parannusehdotusten seurantaan Atlassian-nimisen yrityksen tekemä Jira-järjestelmä.

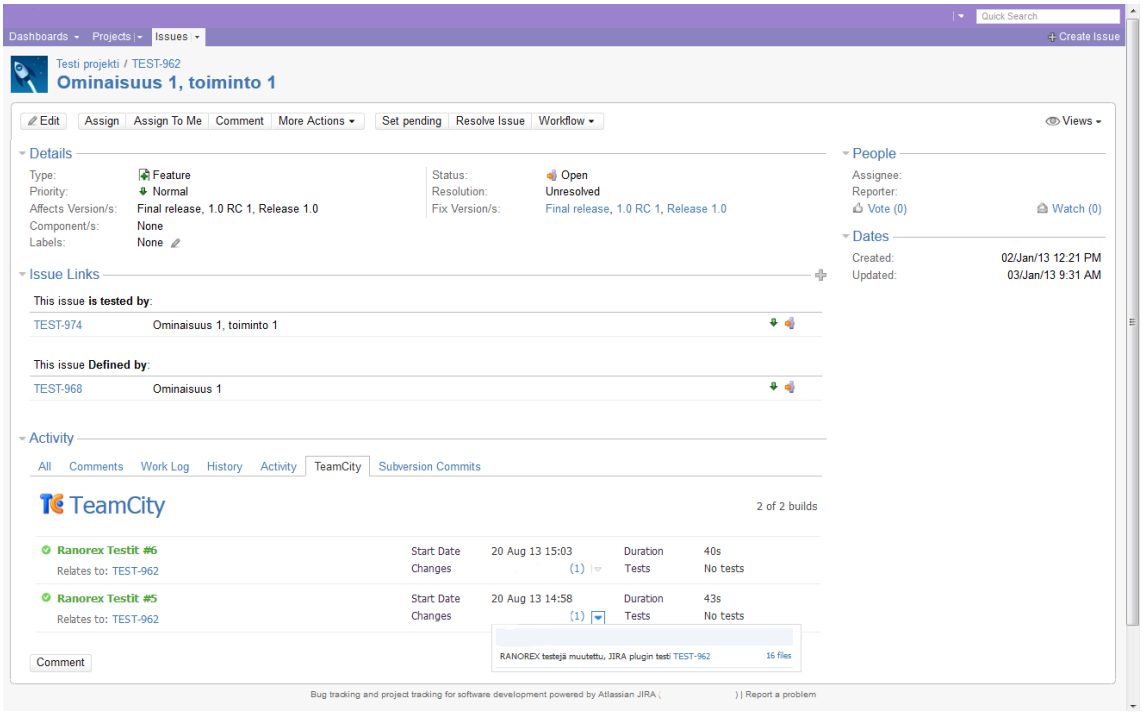
Järjestelmään kerätään mm. eri projektien, eli kehitettävien ohjelmistojen, kaikki olemassa olevat virheet, parannusehdotukset ja tulevat ominaisuudet. Nämä voidaan järjestää järjestelmässä versioiden mukaan, jolloin saadaan selkeästi jaettua tehtävää työtä sekä jäljitettyä, milloin mikäkin ongelma on korjattu. Testaajat voivat raportoida virheistä suoraan Jiraan, jolloin kehittäjä saa siitä tiedon ja kaikki sinne kirjattu on myös pysyvästi tallessa. Kirjauksen korjattuaan kehittäjä merkitsee sen korjatuksi seuraavaan versioon, ja kun versio julkaistaan, testaaja näkee, mitä korjattuja asioita uusi versio sisältää.

Jira itsessään toimii yhteen versionhallinnan kanssa siten, että kun versionhallintaan tehdään kommitointi, joka korjaa tietyn ongelman, laitetaan kommenttiin kyseisen ongelman tunnus Jira-järjestelmässä. Tämän jälkeen järjestelmä osaa näyttää kirjauksen sivulla, mihin on tehty muutoksia sitä korjatessa.

Testauksesta vastaavat henkilöt voivat järjestelmän avulla antaa tiettyjä kokonaisuuksia, tai yksittäisiä kirjauksia, tehtäväksi testaajille. Näin voidaan organisoida laajempaa testausta, esimerkiksi ennen uuden version julkaisua. Kirjaukset voivat olla myös rutiininomaisen testisekvenssin sisältäviä listauksia, joilla ohjeistetaan, mitä asioita tulee testata mistäkin osasta ohjelmistoa.

Integraatiotyökaluja vertaillessa piti selvittää, että työkalu osaa linkittää tekemänsä käännökset ja testit Jira-kirjauksiin ja että Jira-kirjauksiin saadaan linkki

niihin liittyviin käännöksiin. Kuvassa 3 on ruutukaappaus Jiran web-käyttöliittymästä.



KUVA 3. Jiran käyttöliittymä

3.1.3 Asennuksen paketoija InnoSetup

InnoSetup on ilmainen asennuspaketin tekoon tehty ohjelma, myös kaupalliseen käyttöön. InnoSetup käyttää paketin tekoon skriptitiedostoja, jotka sisältävät asennukseen liittyviä tietoja.

InnoSetup-skriptiin tulee tieto, mistä haetaan tiedostot asennuspakettiin, tiettyjä versiotietoja, asentaessa näytettävä ohjelman nimi ja muuta tarpeellista. Lisäksi asennuksen aikana valittavat asetukset, kuten kieli ja lisäosien asennus, määritellään tähän tiedostoon.

Automatisoinnin kannalta hyvä oli, että InnoSetup ohjelma toimii versionhallintaan asennettuna, ja se on käytettävissä suoraan komentoriviltä, jolloin sen ajamiseen on helppo tehdä komentoriviskripti.

3.1.4 Päivityspaketoija WyBuild

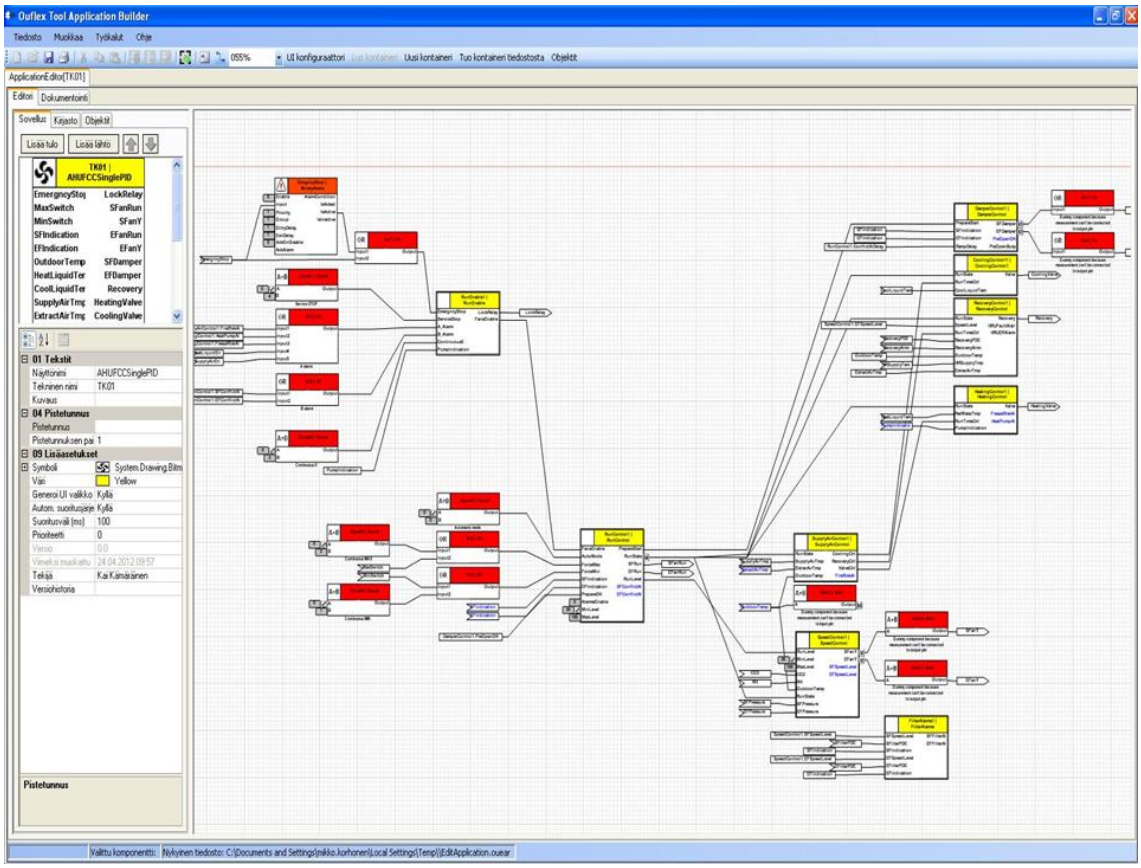
WyBuild on maksullinen päivityspakettien tekoon tarkoitettu ohjelma. Itse WyBuild tekee päivitystiedostot, jotka siirretään palvelimelle, missä ne ovat saatavilla päivittyvälle ohjelmalle. Ohjelmistoon, jonka halutaan päivittyvän, tulee lisätä WyUpdate-ominaisuus, joka käytännössä tarkoitti kahden tiedoston lisäämistä asennukseen. Tätä ennen ohjelmisto täytyy kuitenkin asentaa kansioon, josta WyBuild lukee sen ensimmäiseksi versioksi. Tästä eteenpäin uusi versio asennetaan aina uuteen kansioon, ja lisätään WyBuildiin.

Jokaiseen asennuspakettiin, jonka halutaan päivittyvän, lisätään myös nuo kaksi tiedostoa, jotka WyUpdate tekee. Tämä tekee päivityspakettien tekemisestä monimutkaista ja isotöistä. Onneksi myös WyBuild toimii komentoriviltä, ja erillisen xml-tiedoston avulla siihen voi lisätä automatisoidusti uuden version. Näin tämä monimutkainen vaihe saadaan myös automatisoitua, ja käyttäjän virheen mahdollisuus pienenee.

3.2 Kehitettävä OuflexTool-ohjelmisto

Ouman Ouflex on vapaasti ohjelmoitava automaatiojärjestelmä, jonka ohjelmointi tapahtuu helppokäyttöisellä OuflexTool-ohjelmointityökalulla. Työkalu sisältää valmiin sovelluskirjaston, joka nopeuttaa ja helpottaa järjestelmän ohjelmointia sekä vähentää virheiden määrää. (1, linkki Tuotteet -> Ouman Ouflex.)

OuflexTool on ohjelmointityökaluksi hyvin visuaalinen ja selkeä, kuten kuvassa 4 on nähtävissä, ja työkalulla ohjelmointi koostuukin erilaisten ohjelmalohkojen yhdistelemisestä, verrattuna perinteiseen koodin kirjoittamiseen. Työkalu vaatii toimiakseen pienen fyysisen USB-laitteen eli donglen, jolla estetään ohjelmiston luvaton käyttö.



KUVA 4. OuflexTool-ohjelmisto

OuflexToolia kehitetään eteenpäin jatkuvasti, ja siitä julkaistaan säännöllisesti uusi versio. Ohjelmisto on uusi, joten versioissa tulee ohjelmointivirheiden korjausten lisäksi myös uusia ominaisuuksia ja pienempiä parannuksia toiminnallisuuteen. Yleensä useampi kehittäjä ei tee samoja osia ohjelmasta yhtäaikaisesti, mutta kun ohjelmakomponenttien välinen rajapinta muuttuu, vaikuttaa muutos muihinkin osiin, ja integrointivaiheessa on oltava tarkkana. Jos näitä muutoksia yhdistetään toisiinsa liian harvoin, usein jopa päivien jälkeen, on yhdistämis- eli integrointiprosessi monesti varsin työläs. Toisaalta pelkkä integrointi sisältää monta manuaalista vaihetta, jotka vievät aikaa, jos niitä toistetaan jatkuvasti. Tämän lisäksi jokaisen muutoksen vienti testattavaksi vaatisi paljon työtä.

Virallisia, asiakkaalle vietäviä versioita työkalusta tehdään harvemmin, yleensä useamman kuukauden välein. Nämä sisältävät paljon muutoksia, ja niitä myös testataan useamman ihmisen voimin ennen julkaisua. Vähemmän uusia muu-

toksia sisältävää, yrityksen sisäisessä testauskäytössä olevaa ns. beeta-versiota päivitetään tiheämmin. Lähempänä virallista julkaisua voi testiversio päivittyä useita kertoja päivässä. Päivityksen tekeminen sisältää ohjelman kääntämisen, asennuspaketin tekemisen, päivitystiedostojen tekemisen ja viemisen palvelimelle sekä vielä kaikille testaajille uudesta versiosta ilmoittamisen. Moni vaihe sisältää jopa kymmenien minuuttien odottelua. Tämän kaiken ohessa pitää muistaa päivittää korjatut ongelmat ja uudet ominaisuudet virheraportointityökalu Jiraan, jossa ne ovat listattuna.

Kun näitä päivityksiä alkaa olla useampia päivässä, tai edes viikossa, on järkevä automatisoida osa työtaakasta kehittämisen nopeuttamiseksi. Kun ohjelmistokehityksen eri vaiheita on automatisoitu, voidaan niitä toistaa tiheämmin ja saada nopeammin palautetta ohjelman toimivuudesta. (2.)

Kehittäjiä haastatellessa kävi ilmi, että OuflexToolin kehittäjille on tullut eteen tilanne, jossa muutoksia tehdessä jokin tiedosto tai muutos on jäänyt viemättä versionhallintaan. Ohjelmisto toimii tästä huolimatta kehittäjän itsensä tietokoneella moitteetta, mutta kun joku toinen kehittäjä hakee koodit versionhallinnasta, ei hän välttämättä saa ohjelmistoa edes käännettyä. Automaatio karsii tämän kaltaisia inhimillisiä virheitä pois ohjelmiston kehityksestä. (2.)

Pelkän OuflexTool-työkalun kehityksen lisäksi on olemassa myös sen käyttämä sovelluskirjasto, jota voi päivittää aivan erillään työkalun kehittäjistä. Nämä muutokset kuitenkin vaikuttavat työkalun toimivuuteen, mutta koska kirjasto päivittyy harvemmin, ei sitä ole ollut manuaalisesti järkevä aina ottaa mukaan jokaiseen testiin. Automaatiolla myös uusin kirjasto on joka kerta mukana testeissä.

Testausversion päivityspakettien tekeminen vei aikaisemmin jopa puoli tuntia työaikaa ja vaati koko ajan kehittäjän työpanosta. Nyt työn tuloksena integraatiojärjestelmä tekee sen huomattavasti lyhemmässä ajassa ja ilman ihmisen jatkuvaa valvontaa. Lisäksi kaikki tapahtuu aina samassa ympäristössä riippumatta siitä, kuka kehittäjistä uuden version julkaisee.

Jokaisen muutoksen jälkeen järjestelmä kääntää työkalun, asentaa sen kuten käyttäjä tietokoneelle ja testaa automaattitesteillä tärkeimpien osien toimivuuden. Lopuksi mahdollisesta ongelmasta ilmoitetaan muutoksen tehneelle kehittäjälle. Näin kehittäjä saa heti tiedon, jos hänen tekemänsä muutokset aiheuttivat joko työkalun kääntämisessä tai automaattitesteissä ongelmia, ja ongelman korjaaminen voidaan aloittaa välittömästi.

4 INTEGRAATIOTYÖKALUJEN VERTAILU

Työtä varten tutkittiin eri työkaluja, tarkemmin sanottuna integraatiopalvelimia. Käyttöön soveltuvia työkaluja oli useita, ja suurimmalla osalla oli lähes samat ominaisuudet. Tarkempaan kokeiluun ja vertailuun valikoitui kolme työkalua, joista kaksi oli kaupallisia ja yksi ilmainen avoimen lähdekoodin työkalu.

Jokainen kolmesta työkalusta täytti tarvittavat vaatimukset, ja erot syntyivät lähinnä käytettävyydessä ja määrittämisen helppoudessa. Kaupalliset työkalut olivat käytettävyydeltään selkeämpiä kuin avoimen lähdekoodin vastaava. Lisäksi kaupallisissa työkaluissa oli suurin osa ominaisuuksista valmiina, kun avoimen lähdekoodin työkaluun ominaisuuksia sai lisäosia asentamalla, eivätkä ne aina olleet täysin tuettuja.

Työkaluja vertaillessa oli tarkoitus selvittää, että työkalulla pystyy hakemaan koodin versionhallinasta ja kääntämään OuflexTool-ohjelmiston sekä automaattiset käyttöliittymätestit ohjelmistolle. Lisäksi piti pystyä tekemään asennuspaketti ohjelmasta InnoSetup-työkalulla ja asentaa ohjelma testattavaksi. Tämän jälkeen täytyi pystyä ajamaan automaattiset käyttöliittymätestit ja poistaa asennus. Lopuksi näistä kaikista kohdista piti saada mahdollisimman selkeä raportti.

4.1 Atlassian Bamboo

Ensimmäisenä työkaluista otettiin testikäyttöön Bamboo, joka on Atlassian-nimisen yrityksen tuote. Sama yritys oli virheraportointi työkalu Jiran takana, ja oli odotettavissa, että nämä kaksi ovat hyvin yhteensopivia jo valmiiksi. Bamboo oli kaupallinen tuote ja sen aloitusversio oli kymmenen dollarin hintainen, mikä teki siitä myös hyvin edullisen. Bamboosta oli saatavilla kokeiluversio, jota sai veloituksetta käyttää kuukauden ajan, ja testausta varten se asennettiin käyttöön.

Bamboo toimi yhteen kaikkien tarvittavien työkalujen kanssa suoraan perusasennuksella, eikä tarvinnut testin aikana mitään lisäosia. Versionhallinta Subversion oli tuettuna, kuten myös Atlassianin oma Jira-työkalu. Näin päästiin pian määrittämään automaattikäännöstä ja testien ajamista OuflexTool-ohjelmistolle.

Ohjelmien kääntäminen onnistui MSBuildin avulla helposti, ja työkalussa oli mahdollista käyttää Windows-yhteensopivia komentorivikäskyjä tai valmiiksi kirjoitettuja bat-tiedostoja monimutkaisempien toimintojen suorittamiseen.

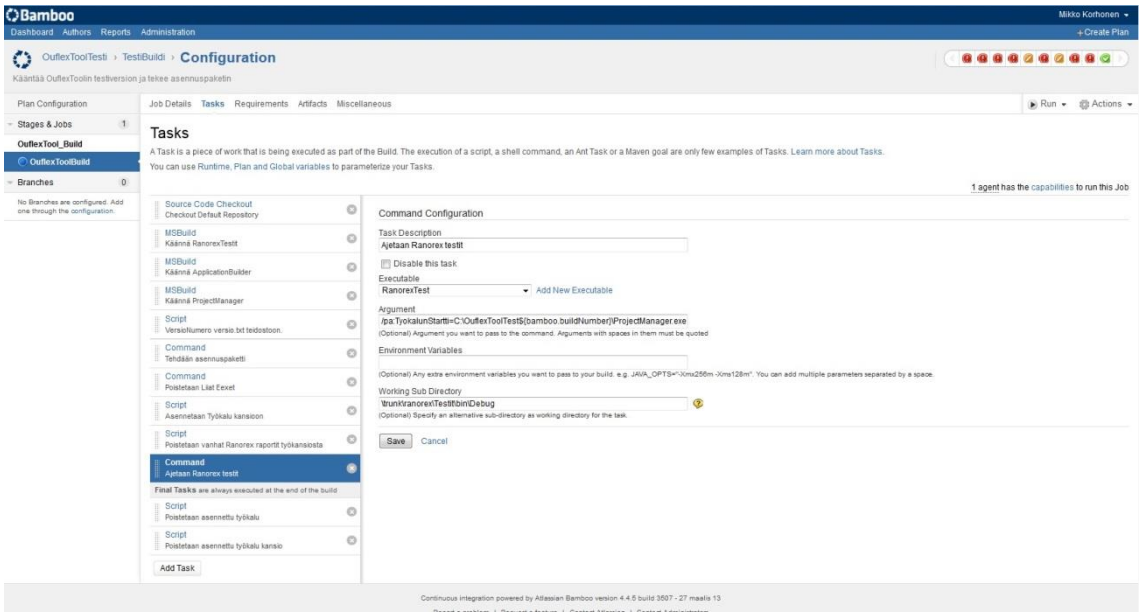
Bamboossa oli mahdollista määrittää käännös tapahtumaan ajastetusti tai silloin, kun versionhallintaan on tehty muutoksia. Tämä toimi kuten odotettu, ja kääntäminen alkoi minuutin sisällä siitä, kun muutoksia oli viety versionhallintaan. Myös versionhallintaan tallennettu tieto muutoksesta oli näkyvissä hyvin, kuten myös linkit Jira-järjestelmään.

Tarpeen vaatiessa Bamboolla olisi ollut mahdollisuus ajaa samaa käännöstä, tai sen eri vaiheita, useammalla koneella yhtä aikaa. Tälle ei kuitenkaan ainakaan vielä ollut tarvetta, kun käännös valmistui nopeasti yhdellä koneella.

Kun ohjelma oli saatu käännettyä, tehtiin siitä asennuspaketti ja asennettiin se koneelle, jossa kääntäjä sijaitsi. Sitten ajettiin ohjelmalle automaattiset käyttöliittymätestit, jonka jälkeen asennus poistettiin. Lisäksi siivottiin mahdolliset ylijäämäkansiot, joita uninstall-sovellus ei saanut poistettua.

Kun kaikki tehtävät oli ajettu, piti vielä siirtää asennuspaketti ja automaattisten testien raportti Bamboo palvelimelle tarkastelua varten. Näin nähtiin, jos automaattitestit ovat epäonnistuneet, ja lisäksi saatiin helposti juuri tämän käännöksen asennuspaketti haettua testaamista varten.

Käännöksen ja testauksen automatisointi Bamboolla oli melko yksinkertaista, ja ominaisuudet täyttivät lähes kaikki vaatimukset. Bamboon integraatio Jira-ympäristön kanssa oli paras testatuista työkaluista, mikä ei ollut yllätys, kun työkalut olivat molemmat samalta valmistajalta. Tulosten sekä käännöksen tuottamien raporttien ja tiedostojen välittämisen selkeydessä oli silti parannettavaa, mikä korostui, kun testattiin muita työkaluja, joissa asiat olivat paremmin. Esimerkiksi TeamCityyn verrattuna vaadittiin monta klikkausta enemmän, että päästiin näkemään määrittely. Raportit eivät olleet yhtä helposti löydettävissä kuin muilla työkaluilla. Kuvassa 5 näkyy Bamboon käännöksen määrittelyvalikko.



KUVA 5. Bamboon määrittely

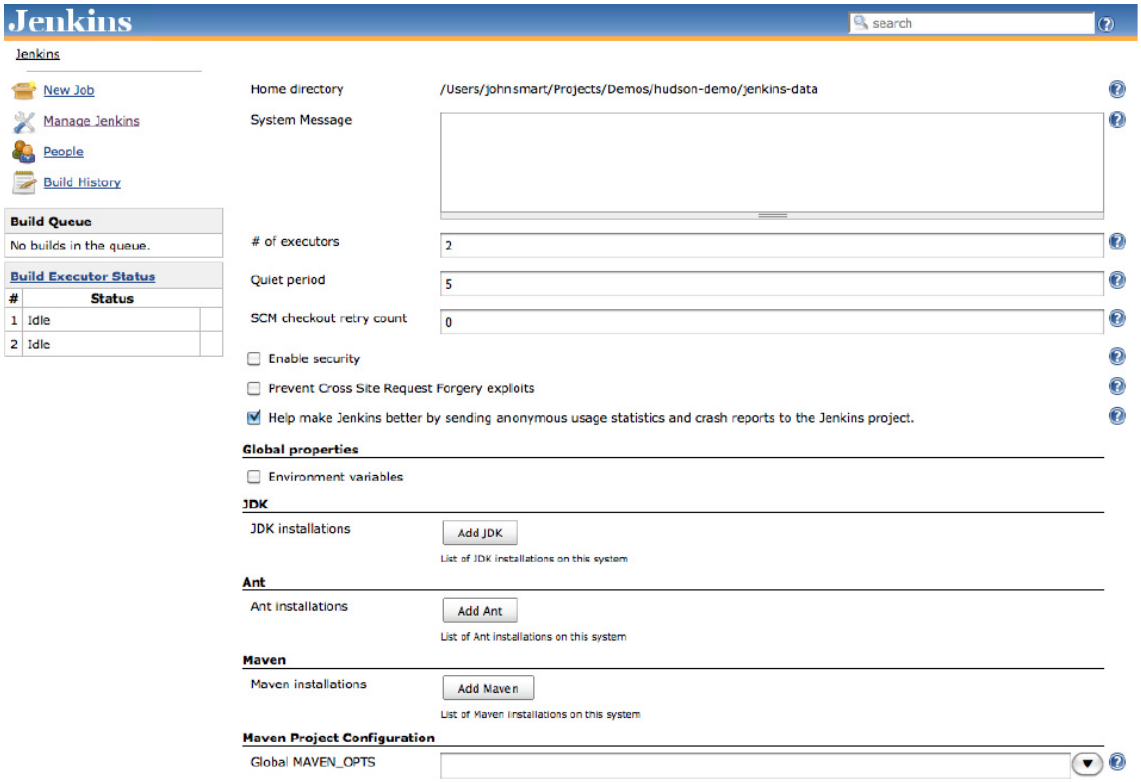
Vertailun lopussa, kun koostettiin yhteenvetoa eri työkaluista, kävi ilmi, että Bamboon edullinen perusversio ei antanut ajaa käännöksiä muulla koneella kuin sillä jossa palvelin toimi. Tätä testausympäristöä ajatellen se ei onnistunut, koska Ranorex-käyttöliittymätestit vaativat erillisen Windows-koneen, jossa testejä ajettiin, kun taas integraatiopalvelin määritettiin toimimaan yrityksen erillisessä palvelinkoneessa. Tästä rajoituksesta ei ollut mainittu missään tuotteen spesifikaatioissa, ja se kävi ilmi vasta sähköpostiviestillä kysyttäessä.

4.2 Jenkins CI

Jenkins CI oli avoimen lähdekoodin työkalu, mikä tarkoitti, että se oli ilmainen käyttää ja sitä oli mahdollista muokata oman tarpeen mukaan. Jenkins CI:n etu oli se, että siihen oli paljon lisäosia, jotka mahdollistivat lähes minkä tahansa projektin automatisoinnin. Samalla lisäosat olivat myös heikoin kohta, sillä kaikki ominaisuudet eivät olleet vakiona käytössä eivätkä saman tekijän tuotosta, jolloin yhteensopivuus ja dokumentointi olivat usein vajavaista. Toisaalta, kirjoitushetkellä jenkins-ci.org-sivuston mukaan oli tarjolla jo 786 lisäosaa, joten mahdollisuuksia kyllä riitti.

Kun alettiin määrittää Jenkins CI:tä käyttöön, vaati se enemmän asetusten säätöä jo pelkästään versionhallinnan ja MSBuildin käyttöönotossa. Subversion oli tuettuna jo vakioasennuksella, mutta MSBuild vaati oman lisäosan, ja lisäksi sille piti määrittää asennuspolut. Kaikki tämä oli ylimääräistä verrattuna toisiin työkaluihin.

Jenkins CI:llä toteutettiin sama toiminnallisuus kuin Bamboolla, ja vaikka nämä samat asiat oli tehty ennenkin, oli määrittäminen silti vaikeampaa. Koska yhtenä tärkeänä kriteerinä oli käytön helppous, pidetään tätä monimutkaisuutta ehdottomana negatiivisena puolena. Kuvassa 5 on Jenkins CI:n määrittelynäkymä.



KUVA 5. Jenkins CI:n määrittely

Siinä, missä kaksi kaupallista työkalua loivat oman välilehden Jiraan tehtyihin kirjauksiin, Jenkins CI:n lisäosa lisäsi vain kommentin kyseiseen kirjaukseen.

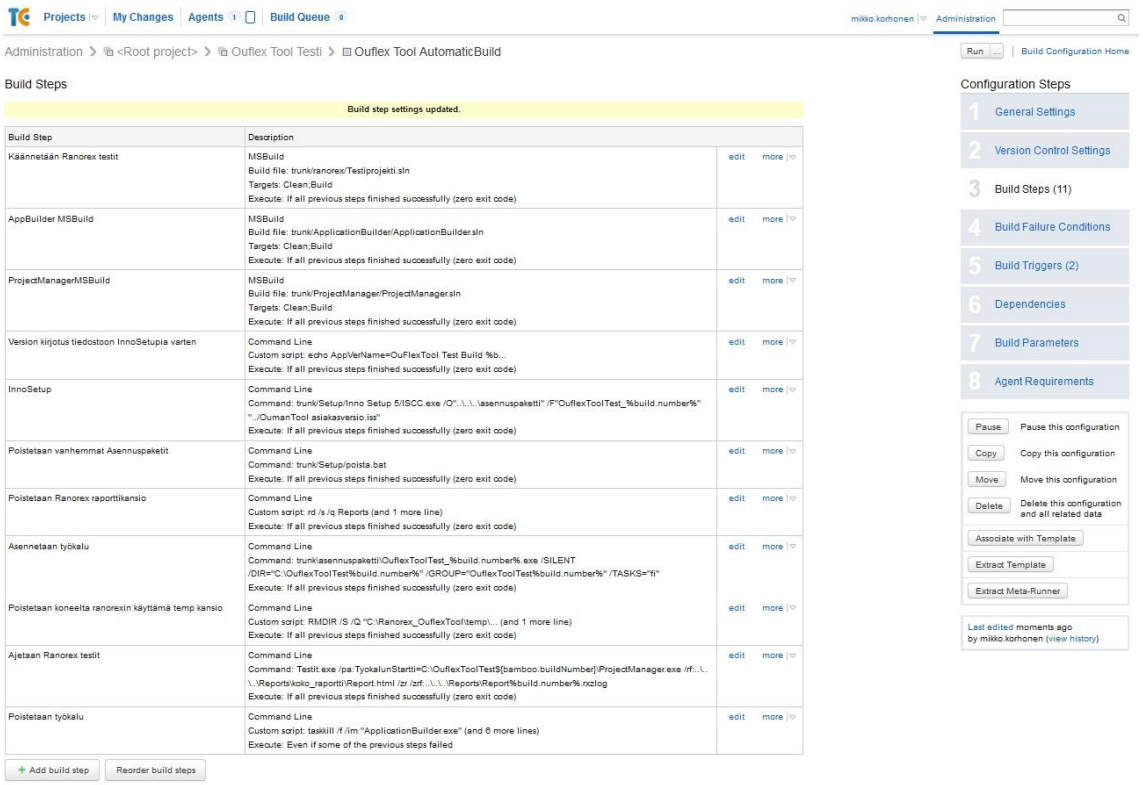
Tämä oli huomattavasti epäselvempi vaihtoehto eikä tuonut lisäarvoa kovinkaan paljon. Linkit Jenkins CI:n käyttöliittymästä Jiraan olivat melko selkeästi esillä.

Jenkins raportoi viimeisen käännöksen tilan lisäksi myös viiden viimeisen tilanteen sääennusteen kaltaisella kuvakkeella, joka muuttui aurinkoisesta ukkoseksi, kun useampi käännös oli epäonnistunut. Muutoinkin raportointi poikkesi kaupallisista hyvin paljon, esimerkiksi lokitiedostot olivat paljon vaikeammin löydettävissä, mikä hidasti erityisesti järjestelmän määritysvaiheessa virheiden etsimistä. Toisaalta Jenkins CI:n raportointia oli mahdollista, tiettyyn pisteeseen asti, muokata mieleisekseen erillisillä lisäosilla.

4.3 JetBrains TeamCity

Kolmas testattavista työkaluista oli JetBrains nimisen yrityksen tekemä TeamCity. TeamCity muistutti heti ulkoisesti hyvin paljon Bamboota. Yksi uniikki ominaisuus oli palvelimen verkkokäyttöliittymään kirjautuessa mahdollisuus rekisteröidä uusi käyttäjä. Näin olisi voitu etukäteen luoda oikeudet, jolloin käyttäjä, jolla ei ollut tunnuksia, pystyi rekisteröimään itsensä ja kirjautumaan sisälle. Tätä ei kuitenkaan tässä ympäristössä haluttu käyttää, ja ominaisuus onnistui ottaa helposti pois käytöstä.

TeamCity osasi hakea ohjelman koodit versionhallinnasta suoraan ja antoikin testatuista työkaluista parhaat mahdollisuudet määritellä haettava sisältö. Käännös- ja testausprosessin eri vaiheet oli helppo määritellä, kun MSBuildille oli oma valmis pohja ja monimutkaiset komennot suoritettiin yksinkertaisesti komentorivikoodina. Joka paikassa pystyi käyttämään valmiita muuttujia, kuten käännöksen numero, tai sitten itse määriteltyjä parametreja. TeamCityllä toteutettiin myös sama toiminnallisuus kuin Bamboolla, ja se osoittautui jopa Bamboota helpommaksi käyttää. Kuvassa 6 on TeamCityn määrittelynäkymä.



KUVA 6. TeamCityn määrittely

Vaikkei Jiran kanssa integroituminen ollut aivan samalla tasolla kuin Bamboolla, toi se silti huomattavan lisäarvon TeamCityn käyttöön. Linkit TeamCitystä Jiraan olivat hyvin esillä ja jokaiselle käännösmäärittelylle oli oma Issue log -välilehti, jossa oli kaikki siihen määritykseen liittyvät kirjaukset. Linkitys Jirasta TeamCityyn vaati lisäosan Jiraan, mutta sen avulla integraatio oli lähes samalla tasolla Bamboon kanssa. Ainoana erotuksena oli se, että kun Bamboossa pystyi Jiran kautta aloittamaan käännöksen, TeamCity vaati käytännössä yhden napin painamisen omasta käyttöliittymästään.

Raportoinnissa TeamCity oli selkein ja monipuolisin kolmesta testatusta työkalusta. Heti kun käännös aloitettiin, oli saatavilla helppo pikavalinta mennä tarkastelemaan lokia, joka päivittyi lähes reaaliajassa. TeamCity osoitti selkeästi, missä oli vika, jos käännös ei onnistunut, mikä auttoi varsinkin määrittämisvaiheessa paljon. Lisäksi TeamCityyn pystyi luomaan oman välilehden Ranorex-

testien tuottamalle xml-muotoiselle raportille, joka oli sitten helposti nähtävissä aina käännöksen valmistuttua suoraan web-käyttöliittymästä.

TeamCitylle toivat lisäarvoa myös sen sisältämät lisäosat eri työkaluille kuten Visual Studiolle, sekä Windowsin työkalupalkkiin asennettava Tray Notifier -sovellus. Tray Notifier ilmoitti käännösten tilan ja näytti ponnahdusilmoituksen aina kun käännös alkoi. Tämä oli esimerkiksi järjestelmästä tai projektista vastaavalle hyvä työkalu, josta näki, kun ohjelmaa käännettiin palvelimella ja olivatko viimeiset käännökset onnistuneet.

4.4 Työkalun valinta

Kun työkalujen testaus alkoi, eivät kaikki kriteerit olleet vielä täysin selvillä, vaan osa tarkentui vertailun edetessä, kun tutustuttiin ohjelmistoihin paremmin. Tärkein yksittäinen tekijä oli helppokäyttöisyys, sillä työkalua tulisivat käyttämään paitsi ohjelmistokehittäjät, myös testaajat ja projektinvetäjät. Samankaltainen kriteeri oli myös selkeys, niin käytössä kuin tulosten raportoinnissa. Koska jokaisella työkalulla pystyi tekemään aluksi määritellyt toiminnot, korostui käytettävyys ja lopulta myös monipuolisuus vertailussa.

Loppujen lopuksi valittiin työkaluksi TeamCity, suurimmaksi osaksi sen ylivoimaisen selkeyden ja helppouden vuoksi. Työkalun valintaa esitettiin yrityksen vastuuhenkilöille torstaina 8.8.2013 palaverissa, jossa käyttöön otettavaksi valittiin yksimielisesti TeamCity. Taulukossa 1 on nähtävissä työkalujen ominaisuuksien vertailua, jota myös tuossa palaverissa esitettiin.

TAULUKKO 1. Työkalujen ominaisuuksien vertailu

	TeamCity	Jenkins CI	Atlassian Bamboo
Visual Studio integraatio	Kyllä	Ei	Kyllä
JIRA integraatio	Lisäosalla	Lisäosalla	Kyllä
Käännöksen määrittäminen	Helposti	Vaikeampaa	Helposti
Ranorex testien ajo	Helposti	Vaikeampaa	Helposti
Tulosten selkeys	Selkeä	Melko selkeä	Melko selkeä
Käytön helppous	Helppo	Vaikeampi	Hieman vaikeampi

Kuten taulukosta näkee, oli suurin osa kriteereistä hyvin subjektiivisia, koska kaikki työkalut kuitenkin suoriutuivat valituista tehtävistä ja erot olivat käytettävyydessä. Tämän takia työkaluja testattiin yhtä aikaa ja moneen kertaan, jotta ei totuttu yhteen ja suosittu sitä erityisesti.

Työkalujen hinnan vertailu jätettiin tarkoituksella viimeiseksi, ettei se vaikuttanut käytännön kokemukseen. Näin saatiin vertailtua kaikki työkalut samalta viivalta riippumatta hinnasta. Yksi oli kuitenkin ilmainen, avoimen lähdekoodin työkalu, ja kaksi muuta kaupallisia, vaikka JetBrains tarjosi TeamCity-ohjelmaansa ilmaisena Professional versiona. Taulukossa 2 on vertailu työkalujen kustannuksista. Kustannukset on laskettu edullisimmalle Ouman Oy:n käyttöön sopivalle lisenssimäärälle, mikä tarkoittaa kunkin työkalun kohdalla vähintään yhtä integraatiopalvelinta ja yhtä erillistä integraatiokonetta.

TAULUKKO 2. Työkalujen hankinta- ja 3 vuoden käyttökustannusten vertailu, tiedot valmistajien verkkosivuilta elokuussa 2013

	Ostohinta €	Lisäosat €	3v. ylläpito €	Kokonaisuus €
Bamboo	800	0	1200	2000
TeamCity	0	275	0	275
Jenkins CI	0	0	0	0

Kaikkien työkalujen hinnat olivat kohtuullisia, ja hinta muutoinkin tässä vertailussa oli pienemmässä asemassa. Vaikka hinta olisi ollut tärkeä kriteeri, olisi valinta silti osunut todennäköisimmin TeamCityyn. Bamboo oli huomattavasti kalliimpi, ja Jenkins CI kuitenkin käyttömukavuudeltaan sen verran jäljessä kaupallisista työkaluista, ettei sen ja TeamCityn hintaero ollut merkitsevä missään tilanteessa.

5 INTEGRAATIOPALVELIMEN KÄYTTÖÖNOTTO

Integraatiopalvelimen käyttöönotto sujui enimmäkseen ripeästi, kun työkaluja vertaillessa oli jo tehty samoja toiminnallisuuksia työkalulla. Käyttöönottovaiheessa järjestelmän vaatimukset kuitenkin täsmentyivät, ja mukaan otettiin uusia toimintoja, joita piti automatisoida. Vaikka näiden tekemiseen menikin odotettua enemmän aikaa, pysyttiin enimmäkseen aikataulussa.

Pois jouduttiin jättämään Ouflex-laitealustan mukaan ottaminen, joka jäi sitten jatkokehitysmahdollisuudeksi. OuflexTool-työkalun integraation määrittämiseen ja käyttöönottoon meni lopulta sen verran aikaa, että pitäydyttiin ainoastaan siinä. Näin ehdittiin testata järjestelmän toimintaa kunnolla käytännössä, eikä käyttöön jäänyt mitään keskeneräistä määrittelyä. Laitealustan mukaan ottaminen on suunnitelmien mukaan tehtävänä heti tämän työn valmistuttua, ja tarkoitus on muutenkin laajentaa järjestelmän käyttöä.

5.1 Järjestelmän määrittely

Käyttöönoton alussa käytiin läpi ohjelmiston kehittäjien kanssa vaatimukset järjestelmälle ja suunniteltiin, millä tasolla automaatio sekä integraatio otetaan käyttöön. Lisäksi selvitettiin, mihin integraatiokone sijoitetaan sekä mitä ohjelmistoja ja työkaluja siihen tarvitsee asentaa.

Yhdessä kehittäjien kanssa päätettiin alustavasti automatisoida seuraavat asiat ohjelmiston kääntämisestä ja testaamisesta: itse testattava ohjelmisto käännetään kahdessa osassa, ensin Application Builder ja sitten Project Manager, jonka jälkeen käännetään automaattitestit, tehdään OuflexToolista asennuspaketti InnoSetupilla ja asennetaan se integraatiokoneelle, ajetaan automaattitestit, ja lopuksi poistetaan asennettu OuflexTool-ohjelmisto koneelta. Käytännössä edellisten askelten väliin vaadittiin tiettyjä tehtäviä myös raporttien ja tiedostojen siirtelemistä varten, mutta ne käydään tarkemmin läpi seuraavassa luvussa.

Edellä selvitetyn itsensä testaavan käännöksen lisäksi piti tehdä pelkästään työkalun kääntäminen automaattisesti sekä päivityspakettien teko. Näistä päivi-

tyspaketin teko vaati enemmän aikaa, koska sen mukana tuli uusia tehtäviä ja työkalu, jota ei aikaisemmin ollut käytetty, WyBuild.

Itse käyttöönoton alussa asennettiin Ouman Oy:n tuotantopalvelimelle TeamCityn integraatiopalvelin, jonka asentaminen kävi jopa yllättävän helposti. Kun TeamCity oli asennettu ja sen vaatima tietokanta määritetty, pääsi palvelimeen käsiksi helposti web-käyttöliittymän kautta. Näin yrityksen tuotekehityksen jokaiselta koneelta pääsi tarvittaessa näkemään, mitä palvelimella tapahtuu.

TeamCityyn pystyi määrittämään mahdollisuuden luoda uudet tunnukset kirjautumisikkunassa, mutta tämä otettiin pois käytöstä koska jokainen käyttäjä, ja heidän käyttöoikeutensa, määritettiin käsin. Näin pystyttiin rajaamaan tietyt käyttäjät vain tiettyihin projekteihin. TeamCity palvelimelle määritettiin selkeät ryhmät, joihin käyttäjiä voidaan liittää. Ryhmien avulla käyttöoikeuksien ja sähköposti-ilmoitusten määrittäminen usealle käyttäjälle käy nopeammin. Esimerkiksi projektista vastaaville henkilöille ilmoitetaan jokaisesta käännöksestä, jota tehdään. Projektin kehittäjälle vuorostaan tulee ilmoitus vain sellaisista käännöksistä, joissa hänen oma muutoksensa aiheutti epäonnistumisen.

Seuraavaksi määritettiin TeamCity palvelimelle linkitys Jiraan, joka vaati Jiran osoitteen sekä käyttäjätunnukset, joilla pääsee näkemään kirjausten tiedot. Lisäksi palvelimelta voidaan määrittää, mihin Jira-projekteihin linkitetään, jos halutaan linkitys vain osaan projekteista. Jiran puolelta linkitys vaati lisäosan asentamisen, joka kävi nopeasti. Lisäosan määrittäminen oli yhtä suoraviivaista kuin TeamCityn puolelta, eli osoite ja käyttäjätunnukset olivat ainoat vaatimukset.

Sitten määritettiin ns. agentti eli integraatiokone. TeamCity-järjestelmässä tuota konetta kutsutaan nimellä ”agent”. Integraatiokoneeksi valittiin yksi käyttämättömänä ollut pöytäkone, joka asetettiin erääseen työtilaan ja merkittiin asianmukaisilla varoituksilla. Kone liitettiin verkkoon, ja TeamCityn verkkokäyttöliittymästä ladattiin agenttiohjelmisto koneelle. Tämän asennuksen jälkeen kone oli palvelimen komennettavissa. Tarkempia määrityksiä ei tarvinnut tehdä, kun käytössä oli vain yksi agentti.

5.2 Jatkuvan integraation käyttöönotto

Käyttöönottovaihe koostui TeamCity-projektien määrittämisestä. Yhteensä määritettiin käyttöön kolme eri projektia eri käyttötarkoituksiin. Ensimmäinen oli jatkuvan integraation perusta, jokaisesta muutoksesta ajettava ohjelmiston käännös. Seuraava oli käännös, jossa ajettiin myös automaattiset käyttöliittymätestit, ja kolmas oli järjestelmän hyötykäyttöä päivityspaketin automatisoinnin tekemiseen.

5.2.1 Pelkän työkalun käännös -projekti

Ensimmäisenä tehtiin pelkän työkalun kääntämiseen projekti. Tähän määritettiin ensin versionhallinnan tiedot, joka sisälsi osoitteen versionhallintaan, käyttäjätunnukset sekä hakemiston, joka haetaan koneelle. Koska tämä käännös tehtiin uusimpaan versioon koodista, haettiin trunk-hakemisto, jossa uusin koodi sijaitsi.

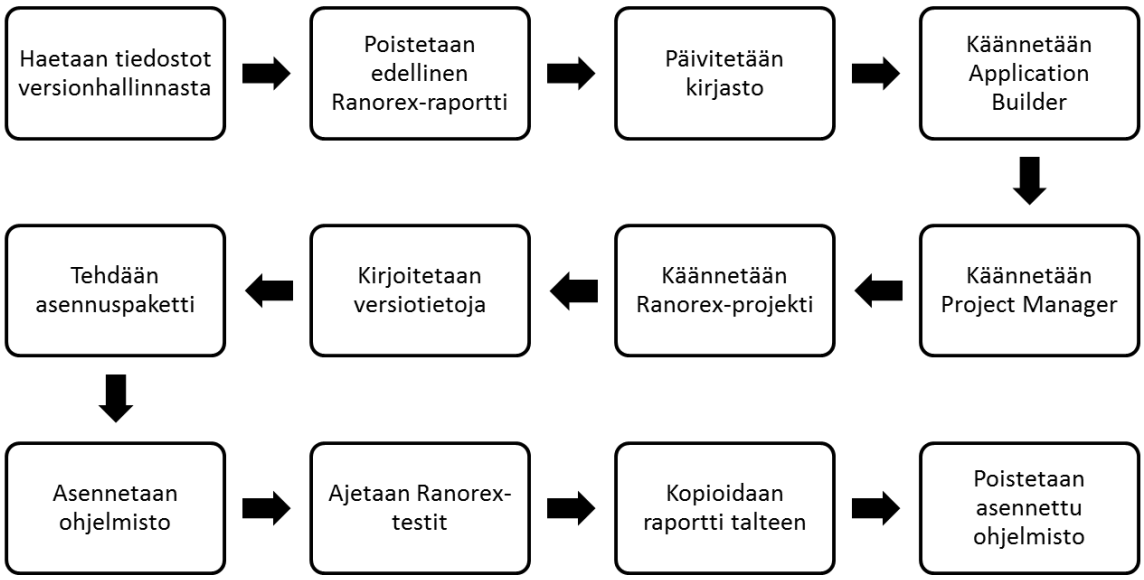
Kun hakemisto oli haettu koneelle, käännettiin ensin Application Builder, jonka jälkeen Project Manager. Tätä varten määritettiin TeamCitylle käännettävät projektit, sekä MSBuild versio, jolla ne käännettiin. Lisäksi määritettiin, että ensin tehdään ns. ”clean” ja sitten ”rebuild”, eli puhdistettiin mahdolliset aiemmin käännetyt tiedostot, ja sitten käännettiin kaikki alusta asti uudelleen. Näin saatiin aikaiseksi aina puhdas ja kokonaan integraatiokoneella tehty käännös.

Sitten määritettiin tuo käännös ajettavaksi joka kerta, kun kehittäjä vie muutoksia versionhallintaan, tosin pienellä viiveellä jos muutoksia tuleekin useampia peräkkäin. Jos käännös epäonnistui, lähetettiin sähköpostissa varoitus ja tiedot virheestä sille kehittäjälle, jonka muutos käännöksen laukaisi.

5.2.2 Automaattiset testit -projekti

Seuraava projekti oli automaattitestit ajava käännös. Tässä tapauksessa määritettiin versionhallinnasta haettava hakemisto niin, että haettiin versionhallinnan juuresta ja määritettiin muuttujalla haluttu hakemisto. Oletuksena oli, että testit ajetaan uusimmalle versiolle, joten oletusvalinta oli trunk-hakemisto. Kun päivitysversiota tehdessä tehtiin hakemisto, jossa sen version tiedostot olivat, tuli se

tags-hakemistoon asianmukaisella versionimellä. Jos haluttiin ajaa testit tuossa hakemistossa olevalle versiolle, määritettiin ”tags/versio”-polku muuttujaan. Lisäksi määritettiin sovelluskirjasto haettavaksi omasta versiohallintapolustaan, ja se vietiin samaan kansioon kuin muutkin tiedostot tässä käännöksessä. Koko prosessi eteni kuvan 7 lohkokaavion mukaisesti.



KUVA 7. Käyttöliittymätestien automaatioprosessi

Seuraava askel oli poistaa Ranorex-testien jälkeensä jättämä raporttikansio koneelta, jotta vain uusin raportti oli siellä testien ajon jälkeen. Tämä onnistui yksinkertaisella komentorivikäskyllä.

Tämän jälkeen päivitettiin koneella oleva sovelluskirjasto uusimmalla kirjastolla, joka oli haettu versionhallinnasta. Ensin poistettiin vanha kirjasto koneelta, vietiin tilalle uusi ja poistettiin sovelluksista lähdekoodit sitä varten tehdyllä ohjelmalla. Tämä kirjasto tuli mukaan tehtävään asennuspakettiin.

Seuraavaksi käännettiin OuflexTool-ohjelmisto kuten edellisessäkin projektissa. Tämän lisäksi käännettiin automaattitestit, jotka myös olivat MSBuild-ohjelmalla käännettävissä.

Sitten aloitettiin asennuspaketin teko. Ensin kirjoitettiin komentorivikäskyllä tekstitiedostoon tämän käännöksen numero ja muita tietoja, jotka InnoSetup pakettia tehdessä sieltä luki. Sen jälkeen pystyttiin tekemään asennuspaketti, joka tehtiin ajamalla InnoSetup komentoriviltä ja antamalla sille kolme parametria: kansio johon asennuspaketti tallennettiin, asennuspaketin nimi sekä polku ISS-tiedostoon. ISS-tiedosto sisälsi kaikki asennuspakettiin tulevat tiedostot ja asetukset. Tämän tuloksena saatiin haluttuun kansioon asennustiedosto, josta pystyttiin asentamaan työkalu.

Seuraavana asennettiin tuo edellä tehty paketti koneelle ja ajettiin automaattiset Ranorex-testit asennetulle OuflexTool-ohjelmalle. Näin saatiin testattua käytännön toimintojen toimivuus uusimmassa versiossa. Asennus sekä testien ajo onnistui yksinkertaisella komentorivikäskyllä, jolla myös määritettiin mihin OuflexTool asennettiin ja Ranorex-testien raportti tallennettiin.

Tämän jälkeen kopioitiin Ranorex-raportti talteen integraatiokoneella olevaan kansioon, johon oli pääsy verkon kautta. Kansiosta sai haettua helposti kaikki raportit myös myöhemmin. Sitten tehtiin yksinkertainen index.html niminen verkkosivu, jota käytettiin TeamCity palvelimella ohjaamaan ”Ranorex Testit Raportti” -välilehdeltä tuohon uusimpaan raporttiin. Näin saatiin jokaiseen käännökseen sen oma raportti näkyviin helposti verkkokäyttöliittymästä.

Lopuksi, vaikka jokin edellisistä askeleista olisi epäonnistunut, poistettiin asennettu OuflexTool integraatiokoneelta, jottei sinne alkaisi kertyä asennuksia epäonnistuneiden testien johdosta. Tätä varten suljettiin ensin pakolla tietyt prosessit, jotta oikealla poisto-ohjelmalla onnistui poistaa koko OuflexTool-asennus vaikka OuflexTool olisi jäänyt päälle. Viimeisenä vielä poistettiin kansio, johon OuflexTool oli asennettu.

Tämä käännös määritettiin ajettavaksi joka yö, koska käyttöliittymätestit kestivät liian kauan ajettavaksi joka muutoksesta. Näin käännöstä ajettaessa mukana saattoi olla useampia muutoksia useammilta kehittäjiltä. Mahdollisista epäonnistumisista ilmoitettiin jokaiselle mukana olevalle kehittäjälle sekä projektin vastuuhenkilöille. Näin mahdollisimman moni asianosainen sai tiedon virheestä ja pääsi sitä korjaamaan.

Lisäksi määritettiin vielä tuo käännöksen aikana tehty asennuspaketti sekä Ranorex-raportti tallennettavaksi palvelimelle. Näistä pidettiin tiedot palvelimella vain viiden viimeisen onnistuneen käännöksen ajalta. Vanhat Ranorex-raportit olivat kuitenkin tallessa integraatiokoneella. Muut lokit säilyivät palvelimella pidempään.

5.2.3 Päivityspaketti-projekti

Päivityspaketin tekoa varten määritettiin versionhallinnasta haettavakasi tags-hakemistosta annetun parametrin mukainen versio. Tämä parametri oli pakollinen antaa tämän käännöksen alussa. Lisäksi haettiin uusin sovelluskirjasto samaan työkansioon.

Projekti eteni ohjelmiston asentamiseen asti kuten Automaattiset testit -projekti, lukuunottamatta Ranorex-raporttien poistoa ja Ranorex-projektin kääntämistä. Työkalu kuitenkin asennettiin verkkoasemalle, johon kaikki uudet versiot asennettiin, ja sen jälkeen poistettiin tuolta asennuskansiosta tietyt tiedostot, jotka eivät saaneet tulla mukaan päivitykseen.

Seuraavaksi täytyi tehdä xml-tiedosto, joka sisälsi kaikki tiedot uuden version lisäämiseksi WyUpdate-projektiin. Tuo projekti sisälsi kaikki päivitysten tiedot. Sitten tuon xml-tiedoston avulla lisättiin uusi versio, ja poistettiin WyUpdaten hakemistosta vanhan päivityksen tiedostot.

Tämän jälkeen käännettiin WyUpdatella sekä palvelimelle menevät päivitystiedostot että uuteen asennuspakettiin mukaan tulevat tiedostot. Näiden avulla ohjelmisto osasi automaattisesti päivittää itsensä palvelimelta. Sitten kopioitiin uutta asennuspakettia varten tarvitut tiedostot oikeaan paikkaan, jonka jälkeen tehtiin ohjelmistosta uusi asennuspaketti. Tuo asennuspaketti siirrettiin vielä TeamCity palvelimelle ja tiettyyn verkkosijaintiin, josta se oli testaajien saatavissa.

Lopuksi kopioitiin itse päivitystiedostot päivityspalvelimelle. Tätä varten piti ensin avata yhteys palvelimelle, ja tietoturvasyistä yhteys katkaistiin heti kopioinnin päätyttyä. Ennen uusien tiedostojen kopiointia poistettiin kaikki edelliset tiedostot palvelimelta.

Tätä projektia ei laukaistu ollenkaan automaattisesti, vaan kehittäjä ajoi sen käsin, kun halusi julkaista uuden version päivityksineen palvelimelle. TeamCityn käyttöliittymästä määritettiin, mistä kansiosta uuden version tiedostot haettiin, ja annettiin päivitykselle versionumero.

5.3 Järjestelmän toimivuuden testaus

Järjestelmän toimivuutta testattiin käyttöönoton aikana kokoaikaisesti kaksi viikkoa, ja sen valmistuttua vielä viikon ajan todellisessa käytössä. Näin varmistuttiin siitä, että automaatio toimi odotetulla tavalla. Käyttöönoton ajan keskusteltiin kehittäjien kanssa järjestelmän toimivuudesta ja heidän näkökulmastaan sen käyttöön, ja sen perusteella tehtiin muutamia muutoksia.

Käyttöönoton aikana ilmeni joitain ongelmia, esimerkiksi kun ohjelmisto ei yhtäkkiä kääntynyt oikein integraatiokoneella. Tämä johtui kuitenkin siitä, että kehittäjällä oli jäänyt viemättä yksi tiedosto versionhallintaan, ja ilman sitä ohjelmistoa ei voitu kääntää. Tämä varmisti sen, että järjestelmästä oli hyötyä, ja tämän kaltaiset virheet saatiin nopeasti kiinni.

Testausvaiheen aikana myös kirjoitettiin järjestelmää varten yrityksen sisäiseen käyttöön ohjekirja, jossa oli selitetty kattavasti eri toiminnot mitä järjestelmä teki ja mitä sillä pystyi tekemään. Lisäksi ohjekirjassa oli selitetty tarkemmin jokaisen eri projektin määrittely TeamCityssä. Ohjekirja sisälsi yrityksen kannalta luottamuksellista tietoa, joten sitä ei voitu laittaa liitteeksi tähän työhön.

Lisäksi testauksen aikana, kun järjestelmä oli siinä muodossa kuin se tulee olemaan, perehdytettiin yrityksen henkilöstöä tarkemmin järjestelmän ylläpitoa varten. Näin järjestelmää pystyttiin ylläpitämään myös ilman opinnäytetyön tekijän läsnäoloa.

Järjestelmä todettiin toimivaksi ja se otettiin käyttöön Ouman Oy:n ohjelmistokehityksessä. Järjestelmän ylläpitoa jatkoi siihen nimetty henkilö. Mikäli tarvetta muutoksille ilmeni, ylläpitäjän oli helppo päivittää määritykset integraatiopalvelimelle.

6 TULOKSET JA POHDINTA

Työn tarkoituksena oli tuoda jatkuva integraatio käyttöön Ouman Oy:llä sekä etsiä sopiva työkalu integraation automatisointiin. Työn ensimmäinen osa oli enimmäkseen työkalujen etsimistä ja vertailua. Vertailun tuloksena käytettäväksi valikoitui JetBrains yrityksen tekemä TeamCity, suurimmalta osin käytön helppouden takia. Toinen osa koostui työkalun käyttöönotosta ja jatkuvan integraation sisällyttämisestä ohjelmistokehitykseen.

Ensimmäinen vaihe aloitettiin tutustumalla tarkemmin jatkuvaan integraatioon sekä eri tapoihin toteuttaa se yrityksissä. Lisäksi selvitettiin integraatiopalvelimelta vaadittavat ominaisuudet ja liitynnät nykyisiin käytössä oleviin työkaluihin. Ensimmäisen vaiheen aikana asennettiin ja testattiin monta työkalua, mikä oli selvästi kaikkein vaativin osa työstä. Koska vertailun työkalut muistuttivat paljon toisiaan ja toimivat lähes samalla tavalla, täytyi vertailu suorittaa subjektiivisten kriteereiden kuten käytön helppouden mukaan. Tätä varten työkaluja täytyi käyttää yhtä paljon, jotta tottumus ei vaikuttaisi tulokseen.

Toinen vaihe oli käyttöönotto, jossa asennettiin integraatiopalvelin yrityksen palvelinkoneelle sekä otettiin se käyttöön. Lisäksi määritettiin integraatiokone, joka tekee käännökset, ja määritettiin OuflexTool-ohjelmistolle kolme erilaista käännöstä eri käyttötarkoituksiin. Tämä vaihe oli suoraviivaisempi, koska määrityksiä oli tehty jo integraatiopalvelimia vertaillessa. Käyttöönoton aikana kuitenkin ilmeni uusia vaatimuksia järjestelmälle, kun haluttiin automatisoida myös päivityspakettien teko, joka vaati uusien työkalujen sisällyttämistä järjestelmään. Tämä viivästytti käyttöönottoa muutamia päiviä, mutta onnistui kuitenkin lopulta suhteellisen helposti. Suuri osa ajasta käytettiin järjestelmän toiminnan testaamiseen ja varmistamiseen. Kun järjestelmä oli lopulta määritetty, oli sitä jo kattavasti kokeiltu käytännössä eikä niin paljoa testausta vaadittu.

Työn tuloksena Ouman Oy:llä on käytössään valmiiksi määritetty integraatiopalvelin, joka tekee automaattisesti käännökset OuflexTool-ohjelmistosta sekä testaa sen toimintaa automaattisilla käyttöliittymätesteillä. Lisäksi aikaisemmin paljon manuaalista, tarkkaa työtä vaatinut päivityspakettien tekeminen on automa-

tisoitu lähes kokonaan. Nyt kehittäjän tarvitsee enää määrittää, missä kansiossa olevasta versiosta uusi päivitys tehdään.

Järjestelmän käyttöönoton ja testaamisen aikana tehtiin yhteensä 36 sivua laaja ohjekirja järjestelmän käytöstä sekä yleisimmistä toiminnoista, joita sillä tehdään. Ohjekirjassa sen lisäksi käytiin läpi ja selitettiin järjestelmään tehdyt määrittelyt, jotta ylläpito olisi helpompaa ja järjestelmän toiminta ymmärrettävämpää. Tuo ohjekirja sisältää koodilistauksia ja yrityksen kannalta salassa pidettävää tietoa, joten sitä ei voitu liittää tähän työhön.

Tarkoituksena oli aluksi automatisoida OuflexTool-ohjelmiston sekä Ouflex-laitealustan ohjelmistojen kääntäminen ja testaus, mutta työ jouduttiin rajaamaan pelkästään OuflexTool-ohjelmistoon. Syynä rajaamiseen oli Ouflex-laitealustan kehittäjän poissaolo työn ensimmäisen osan aikana ja lopulta ajan puute myös käyttöönottovaiheessa. Laitealustan ohjelmiston kääntämisen automatisointiin ei ehditty, mutta se on heti ensimmäisenä jatkosuunnitelmissa, kun järjestelmän käyttöä laajennetaan. Kun alun perin aikataulutettiin työtä, olisi voitu ottaa huomioon, että saatetaan tarvita enemmän aikaa kaikkien tavoitteiden saavuttamiseen.

Yrityksestä henkilöstöä on perehdytetty tarkemmin järjestelmän ylläpitoon, ja OuflexTool-ohjelmiston kehittäjien kanssa on käyty läpi järjestelmän toimintaa. Lisäksi järjestelmän toimintaa ja tarkoitusta on yleisellä tasolla esitelty koko tuotekehitykselle. Näin kaikki tietävät, minkälainen järjestelmä on käytössä ja mitä sillä tehdään, sekä voivat jakaa ideansa järjestelmän kehittämisen suhteen.

Työn tuloksena saatiin yrityksen käyttöön toimiva järjestelmä, joka kattaa kuitenkin tässä vaiheessa vasta osan yrityksen tuotteiden kehityksestä. Tarkoitus on jatkossa sisällyttää järjestelmään lisää tuotteita, ja koska alun perin suunniteltua Ouflex-laitealustaa ei ehditty siihen sisällyttää, on se ensimmäisenä vuorossa.

LÄHTEET

1. Ouman Oy:n kotisivut. Saatavissa: http://www.ouman.fi/. Hakupäivä 2.9.2013.

2. Fowler Martin. 2006. Continuous Integration. Saatavissa: http://martinfowler.com/articles/continuousIntegration.html. Hakupäivä 20.6.2013.

3. Shore J. & Warden S. 2008. The Art Of Agile Development. Yhdysvallat: O'Reilly Media Inc.

4. Lindström, Jukka. 2011. Scrum. Reaktor. Saatavissa: http://reaktor.fi/osaaminen/scrum/. Hakupäivä 6.10.2013.