Kehitys- ja integrointiympäristön kehittäminen

Olli Piipponen

Opinnäytetyö
Huhtikuu 2012
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautuminen

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautuminen

Olli Piipponen:
Kehitys- ja integrointiympäristön kehittäminen

Opinnäytetyö 49 sivua.
Huhtikuu 2012

Tässä opinnäytetyössä käsitellään ohjelmistointegrointia ja jatkuvaa integraatiota. Työ on toteutettu ja kirjoitettu Insta DefSec Oy -nimiselle yritykselle syksyn 2011 ja kevään 2012 aikana. Työn tavoitteena oli toteuttaa kyseiselle yritykselle toimiva ja nykyaikainen kehitys- ja integrointiympäristö. Samalla tavoitteena oli tutkia ja kehittää integrointiprosessia ja -menettelyjä.

Yrityksellä ilmeni tarve uudistaa sisäverkko, josta työ sai alkunsa. Tuon uudistuksen yhteydessä ohjelmistokehitysryhmän kehitys- ja integrointiympäristö jouduttiin siirtämään uuteen verkkoon. Siirron yhteydessä haluttiin kehittää ja nykyaikaistaa kehitys- ja integrointiympäristöä.

Vanha kehitys- ja integrointiympäristö oli ollut käytössä joitakin vuosia. Tätä ympäristöä oli kehitetty vähän kerrallaan tarpeiden ilmentyessä. Dokumentaatio vanhasta kehitys- ja integrointiympäristöstä ei ollut ajan tasalla. Yksi työn tärkeimmistä tavoitteista oli dokumentaation tuottaminen.

Työn aikana toteutettiin ja otettiin käyttöön uusi kehitys- ja integrointiympäristö. Ympäristöä suunniteltiin ja rakennettiin ottaen huomioon jatkuvan integraation perusperiaatteet ja viimeisimmät suuntaukset, kuten automatisoitu CI-ympäristö ja -prosessi sekä -työkalut. Lopputuloksena saatiin aikaan kehitys- ja integrointiympäristö, joka täyttää sille annetut vaatimukset ja helpottaa ympäristöä käyttävän ohjelmistokehitysryhmän päivittäistä työntekoa.

Jatkuvaa integraatiota noudattava kehitys- ja integrointiympäristö on suhteellisen monimutkainen kokonaisuus, koska siinä pitää ottaa huomioon koko integrointiprosessin automatisointi, suorituskyky ja palautemekanismit. Tämän monimutkaisuuden takia työn toteutusvaiheessa ilmeni asioita, joita ei oltu huomioitu määrittely- ja suunnitteluvaiheessa.

Asiasanat: ohjelmistointegrointi, jatkuva integraatio

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT engineering
Software engineering

Olli Piipponen:
Improving development and integration environment

Bachelor's thesis 49 pages.
April 2012

In this bachelor's thesis the main subject will be software integration and continuous integration. The work was done between fall 2011 and spring 2012 for a company called Insta DefSec Oy. The goal was to develop a modern integration environment that follows the principles of continuous integration. The other goal was to develop the integration procedure.

The need for this job emerged while Insta DefSec Oy was renewing their company network. In that renew process the software development group's integration environment had to be moved to the new company network. That move was a good opportunity to improve the old integration environment.

The old integration environment had been in use for some time. This integration environment had been developed step by step as new needs arose. Documentation about the old integration environment wasn't up to date. Making an up to date documentation about the new integration environment was also one of main the goals.

During the work a new integration environment was developed and deployed to use. The environment was developed keeping in mind the principles and newest trends of continuous integration. As an end result we got an integration environment that fulfills the requirements that it had. The new integration environment helps the software development group that uses it in their daily work.

Integration environment that uses continuous integration is a complex entity. There are many things that need to be taken under consideration such as automating the whole integration procedure, performance and feedback mechanism. All these things were hard to foresee in the planning phase and because of that in the development phase there were things that needed to be sorted.

Key words: software integration, continuous integration

SISÄLLYS

1. LYHENTEET JA TERMIT

AD	Microsoft Active Directory. Käyttäjätietokanta, johon pystytään tallentamaan tietoa käyttäjistä ja käyttöoikeuksista.
Ant	Apache Software Foundation -ryhmän kehittämä Java-pohjainen vastine make-työkalusta, joka on tarkoitettu automatisoimaan lähdekooditiedostojen lukemista ja niiden kääntämistä.
Apache2	Apache Http Server -palvelu on Apache Software Foundation -ryhmän kehittämä palvelinohjelmisto.
API	Application Programming Interface. Ohjelmointirajapinta, jonka kautta ohjelmistoa voidaan hyödyntää.
CI	Continuous Integration, jatkuva integraatio.
CVS	Concurrent Versions System, versionhallintajärjestelmä
IDE	Integrated Development Environment. Ohjelmistokehitysympäristö, joka pitää sisällään ohjelmistokehitykseen liittyviä työkaluja. Esimerkiksi Eclipse-IDE.
Ivy	Apache Software Foundation -ryhmän julkaisema riippuvuuksienhallintatyökalu.
JDK	Java Development Kit. Sun Microsystem -yrityksen kehittämä ohjelmistokehitysalusta Java-kielelle.
LDAP	Lightweight Directory Access Protocol. Yleiskäyttöinen protokolla tunnistautumiseen ja valtuuttamiseen tietovarastototeutusta vasten, kuten esimerkiksi Active Directory -palvelua vasten.
PKI	Public Key Infrastructure. Menettelytapa, jolla varmennetaan tiedostojen tai palveluiden alkuperä.
SCM	Source Code Management. Yleinen lyhenne, jolla tarkoitetaan versionhallintatyökalua, kuten esimerkiksi SVN tai CVS.
SPI	Service Provider Interface. Ohjelmointirajapinta, joka tarjoaa mahdollisuuden implementoida ja laajentaa tarjoajan palveluita.
SSH	Secure Shell. Protokolla, jolla voidaan avata suojattu yhteys toiselle tietokoneelle tai palvelimelle.

SVN	Subversion, versionhallintajärjestelmä.
RPM	Red Hat Packet Manager. Red Hat -yrityksen luoma pakettienhallintasovellus.
RSA	Vuonna 1978 julkaistu julkisten avaimien salausalgoritmi, jonka lyhenne tulee tekijöiden nimien alkukirjaimista. Kehittäjien nimet olivat Ron Rivest, Adi Shamir ja Len Adleman.
VMware	WMware inc -yrityksen tuottama virtualisointijärjestelmä.

2. JOHDANTO

Tässä opinnäytetyössä tullaan käsittelemään ohjelmistointegrointia ja jatkuvaa integraatiota. Työ on toteutettu ja kirjoitettu Insta Defsec Oy -nimiselle yritykselle syksyn 2011 ja kevään 2012 aikana. Kehitys- ja integrointiympäristön toteuttamisessa oma päävastuualueeni oli toteuttaminen ja testaaminen. Tämän lisäksi osallistuin ympäristön määrittelyyn ja suunnitteluun. Ympäristöä varten yrityksen ICT-tukipalveluilta saatiin käyttöön palvelimet, jotka ovat WMware-virtualisointijärjestelmän päälle perustettuja virtuaalipalvelimia. Suunnittelussa käytettiin hyväksi vanhan kehitys- ja integrointiympäristön dokumentaatiota ja itse ympäristöä.

Jatkuva integraatio on nouseva trendi ohjelmistotuotannossa. Jatkuva integraatio on vastaus ongelmiin, joiden kanssa ohjelmistokehittäjät ovat painineet integraation osalta vuosia. Jatkuvalla integraatiolla pyritään vastaamaan nimenomaan vanhan vesiputousmallin integraation ongelmiin ja poistamaan ne kokonaan.

Työn teoriaosuudessa käsitellään ohjelmistokehitystä yleisesti. Tämän osuuden on tarkoitus antaa lukijalle yksinkertainen peruskuva siitä, mitä ohjelmistokehitys on. Ilman perusymmärrystä ohjelmistokehityksestä on vaikea käsitellä jatkuvaa integraatiota ja siihen liittyviä asioita. Teoriaosuudessa kuvataan integroinnin ja jatkuvaa integraation periaatteet, joita sovelletaan työn myöhemmässä vaiheessa. Osuudessa vastataan tärkeimpään kysymykseen jatkuvasta integraatiosta eli miksi sitä tehdään?

Työn suurin osuus kuvaa yksittäisen jatkuvaa integraatiota toteuttavan kehitys- ja integrointiympäristön rakentamista. Tämä ympäristö rakennettiin noin 15 henkilöä sisältävän ohjelmistokehitysryhmän käyttöön. Ympäristön käyttöönotto tapahtui helmikuussa 2012. Ohjelmistokehitysryhmä oli tehnyt vanhassa kehitys- ja integrointiympäristössään töitä pitkään. Tämä aiheutti tiettyjä haasteita ja vaatimuksia uudelle ympäristölle. Toisaalta tämä oli hyvä asia, koska ympäristön tarpeet olivat tiedossa etukäteen. Kehitys- ja integrointiympäristö on toteutettu suorittamaan jatkuvaa integrointia ja kaikissa ratkaisuissa on pyritty ottamaan tämä huomioon. Tämä tarkoittaa, että kehitys- ja integrointiympäristön pitää pystyä tekemään kaikki integrointiin liittyvät toimenpiteet automaattisesti. Automaatiolla tässä yhteydessä tarkoitetaan sitä, että koko integrointiprosessi pitää pystyä suorittamaan ilman käyttäjän toimenpiteitä.

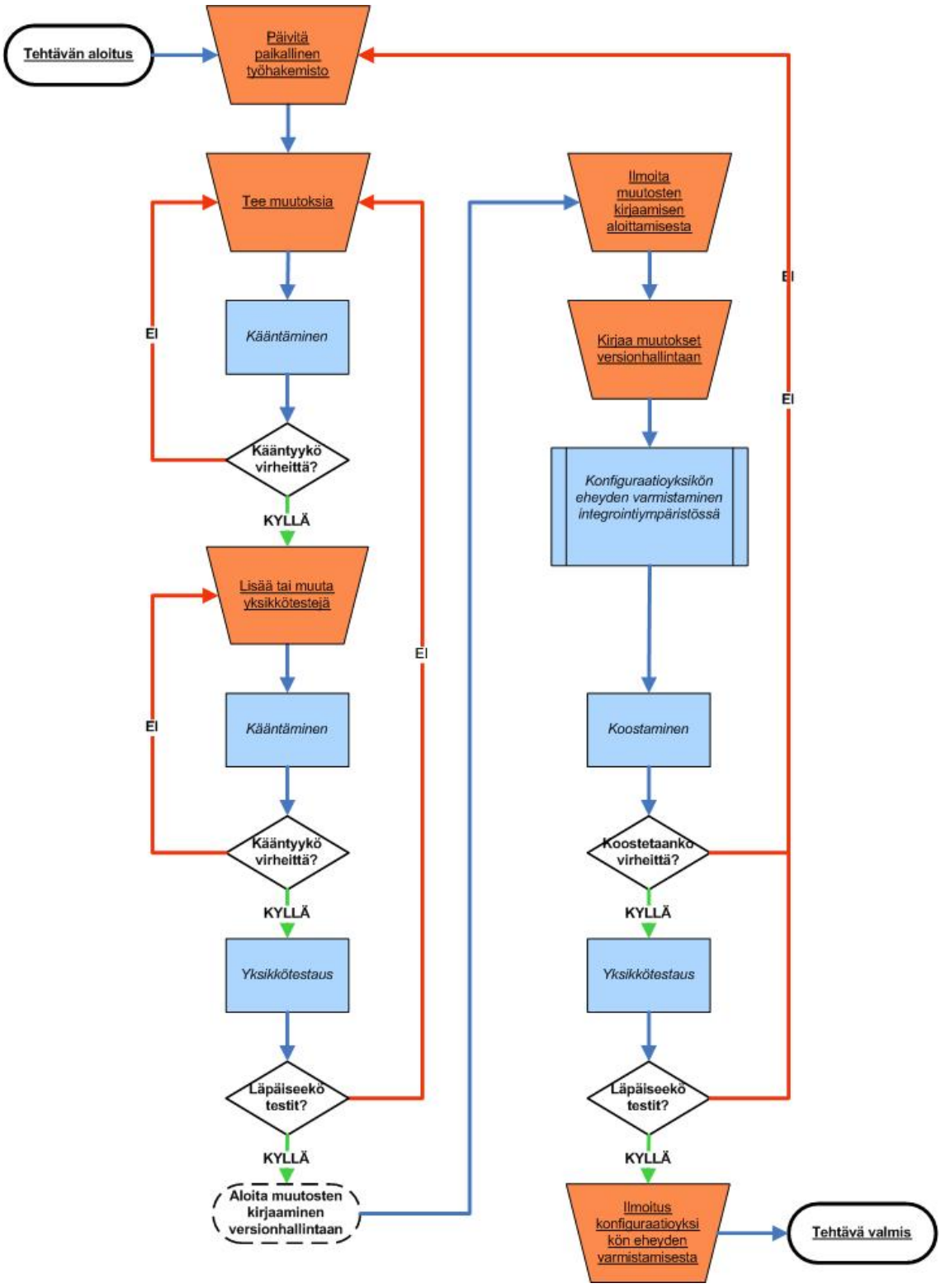
Lopuksi työssä pohditaan toteutettua kokonaisuutta. Saavutettiinko asetetut tavoitteet? Voiko ja miten kehitys- ja integrointiympäristöä sekä niihin liittyviä prosesseja kehittää jatkossa?

3. Ohjelmistointegrointi osana ohjelmistokehitystä

Tässä osuudessa tullaan antamaan lukijalle yleiskuva siitä, mitä ohjelmistokehitys ja ohjelmistointegrointi ovat. Kehitys- ja integrointiympäristön tarpeita on mahdoton hahmottaa, ellei ole peruskuvaa siitä mitä ohjelmistokehitys on. Asiaa on pyritty havainnollistamaan kuvien avulla. Luvussa ei tulla käsittelemään ohjelmistokehityksen koko prosessia määrittelystä testaukseen, vaan keskitytään kehitystyöhön ja integrointiin.

3.1. Kehitystyö

Kehitystyön etenemistä kuvataan tässä luvussa tehtäväpohjaisesti. Tällä tarkoitetaan sitä, että kehitettävä ohjelmisto on määritelty, suunniteltu ja jaettu toteutettaviksi osakokonaisuuksiksi. Tehtävät eivät ole kovin laajoja, vaan ne on jaettu pieniksi ja helposti hallittavaksi osakokonaisuuksiksi. Kuviossa 1 on kuvattu kehitystyön etenemistä tästä näkökulmasta.



KUVIO 1. Kehitystyöhön liittyvät toimenpiteet periaatteellisella tasolla (Helin, Peltola & Hietaranta 2009, 30, muokattu)

Kuviossa 1 yksittäinen kehittämistyö aloitetaan sillä, että haetaan kehitettävän ohjelmiston olemassa olevat lähdekooditiedostot versiohallintavarastosta. Versiohallinta on palvelu, joka mahdollistaa lähdekooditiedostojen samanaikaisen käytön useamman kehittä-

jän kesken. Lyhyesti selitettynä versionhallinnassa on lähdekooditiedostoja, joista pystyy lataamaan omaan käyttöön kopion. Näihin paikallisiin kopioihin tehdään muutoksia ja kun nämä muutokset halutaan antaa muiden käyttöön, ne kopioidaan takaisin versionhallintaan. Versiohallintaa tullaan käsittelemään tarkemmin luvussa 3.5.

Versiohallinnasta saadulle paikalliselle kopiolle tehdään tehtävässä määriteltyjä muutoksia, joilla tehtävän tavoitteeseen päästään. Kun muutokset on tehty, kehittäjän ensimmäinen toimenpide tämän jälkeen on ohjelmiston kääntäminen. Kääntämisellä tarkoitetaan sitä, että ihmisen ymmärtämät lähdekooditiedostot muutetaan tietokoneen ymmärtämään muotoon. Tämän toimenpiteen suorittaa ns. kääntäjä. Kääntäjä on hyvin ”tyhmä” työkalu siinä mielessä, että kaikki lähdekooditiedostoihin kirjoitetut komennot pitää olla täysin määrätyllä tavalla kirjoitettu. Jos näihin komentoihin on kirjoitettu virheellisesti asioita, kääntäjä ei ymmärrä mitä sen pitäisi tehdä. Näitä virheitä kutsutaan käännösvirheiksi.

Jos käännöstyö saadaan suoritettua virheittä, voidaan siirtyä seuraavaan vaiheeseen, jolla varmistutaan tehdyn muutoksen toimivuudesta. Muutoksen loogisesta toimivuudesta pyritään varmistumaan yksikkötestien avulla. Kääntäjän ilmoittamat käännösvirheet eivät ota kantaa muuhun, kuin kirjoitetun koodin syntaksiin ja semanttiseen oikeellisuuteen. Vaikka syntaksi olisi täysin oikea, se ei tarkoita sitä, että ohjelma toimii silti oikein eli halutulla tavalla. Yksikkötestien avulla saadaan laajempi kuva muutoksen toimivuudesta, koska niillä pystytään havaitsemaan myös loogisia virheitä. Yksikkötestien toteuttaminen tapahtuu hyvin samalla tavalla, kuin itse muutoksen tekeminen ohjelmistoon. Kirjoitamme yksikkötestit lähdekooditiedostoon ihmisen ymmärtämään muotoon ja käskemme tämän jälkeen kääntäjää kääntämään ne tietokoneen ymmärtämään muotoon.

Yksikkötestien kirjoittamisen ja kääntämisen jälkeen, ne suoritetaan tehdylle muutokselle. Jos kaikkia yksikkötestejä ei läpäistä, joudutaan tarkistamaan lähdekoodeihin tehty muutos ja korjaamaan siellä mahdollisesti oleva looginen virhe. Kannattaa ottaa huomioon, että myös yksikkötestit on voitu kirjoittaa väärin ja tällöin ne testaavat väärin.

Jos yksikkötestit läpäistään virheettä, voidaan olettaa, että tehty muutos on toimiva. Tällöin voimme antaa tehdyn muutoksen myös muiden käyttöön viemällä sen versiohallin-

taan. Ihmisille, jotka tekevät samaan aikaan töitä saman osan kanssa, on hyvä ilmoittaa kun muutoksen viemisen aloittaa versionhallintaan. Tällä vältetään mahdollisia konfliktitilanteita eri versioisten lähdekooditiedostojen kanssa.

Ohjelmistot rakentuvat yleensä aina useista lähdekooditiedostoista, jotka vaikuttavat toisiinsa. Useasta lähdekooditiedostosta muodostuu osa, joka voi toteuttaa tietyn osan järjestelmän toiminnallisuudesta. Kun tämän tapaisia osia liitetään yhteen, saadaan ns. konfiguraatioyksikkö. Seuraavaksi pitää varmistua siitä, ettei tekemämme muutos ole vaikuttanut minkään toisen osan toimintaan. Tätä on kuvattu kuviossa 1 laatikolla ”Konfiguraatioyksikön eheyden varmistaminen integrointiympäristössä”. Tämä eheyden varmistaminen toteutetaan koostamalla koko konfiguraatioyksikkö. Koostamisella tarkoitetaan sitä, että useasta pienemmistä lähdekooditiedostokokoelmista muodostetaan suurempi kokonaisuus. Tämä koostaminen on osa integrointia, jota käsitellään tarkemmin seuraavassa luvussa.

Jos koostaminen onnistuu ilman virheitä, voidaan aloittaa konfiguraatioyksikön yksikkötestaus. Konfiguraatioyksikön yksikkötestaus toimii samalla tavalla kuin aikaisemmin kuvattu muutoksen yksikkötestaus. Tässä tapauksessa yksittäisen muutoksen sijaan, testattavana on suurempi kokonaisuus.

Mikäli myös konfiguraatioyksikön yksikkötestit läpäistään virheittä, voidaan todeta muutoksen olevan toimiva. Tästä asiasta on hyvä ilmoittaa muille, jotka tekevät töitä kyseisen konfiguraatioyksikön kanssa.

3.2. Ohjelmistointegrointi

Ohjelmistointegroinnista usein puhutaan yksikäsitteisenä asiana. Todellisuudessa ohjelmistointegrointi on laaja käsite, joka pitää sisällään monia tasoja. Lyhyesti selitettynä ohjelmistointegroinnilla tarkoitetaan pienempien osakokonaisuuksien liittämistä suuremmaksi kokonaisuudeksi. Jotta integrointia voidaan toteuttaa, näiden eri osien pitää osata keskustella toistensa kanssa. Tämä keskustelu hoidetaan usein API- ja SPI-rajapintojen kautta. Tässä luvussa tullaan käsittelemään ohjelmistointegrointia kahdella tasolla:

- Koostaminen eli konfiguraatioyksiköiden luominen
- Konfiguraatioyksiköiden integrointi

Koostamista käsiteltiin jo pintapuolisesti edellisessä luvussa. Lyhyesti selitettynä koostamisen tarkoitus on luoda konfiguraatioyksikkö käännetyistä lähdekooditiedostoista. Konfiguraatioyksikkö koostuu useista lähdekooditiedostoista ja kirjastoista. Konfiguraatioyksikön sisäiset osat integroituvat toisiinsa API- ja SPI-rajapintojen kautta. API- ja SPI-rajapinnan ero on se, että kun API-rajapinta tarjoaa suoraan rajapinnan takana olevan osan palveluita käyttöön SPI-rajapinta sisältää taas abstrakteja luokkia, jotka toteuttamalla päästään hyödyntämään rajapinnan takana olevia palveluita. Tässä integrointivaiheessa on mahdollista, että varsinaiseen lähdekoodiin tehdään muutoksia näiden rajapintojen kautta.

Konfiguraatioyksiköiden integrointi konfiguraationhallinnan kannalta on prosessi, jossa luodaan käytettävä ohjelmistokokonaisuus. Konfiguraatioyksiköt kommunikoivat keskenään API- ja SPI-rajapintojen kautta. Ne tarjoavat toisilleen palveluita, joista toiset konfiguraatioyksiköt tietävät.

Molempien tasoisten integrointien jälkeen on tärkeää varmistua integroinnin onnistumisesta. Tämä voidaan tehdä niin sanotuilla integrointitesteillä. Integrointitestausta voi tehdä kahdella tavalla. Alhaalta ylös -mekanismissa testataan hierarkian alimmat osat ensimmäisen ja edetään hierarkiaa ylöspäin. Ylhäältä alas -mekanismissa toimitaan päinvastoin, eli aloitetaan hierarkian yläpäästä ja edetään alaspäin. Molemmat tavat ovat toimivia ja käytettävä tapa valitaan yleensä testattavan ohjelmiston perusteella.

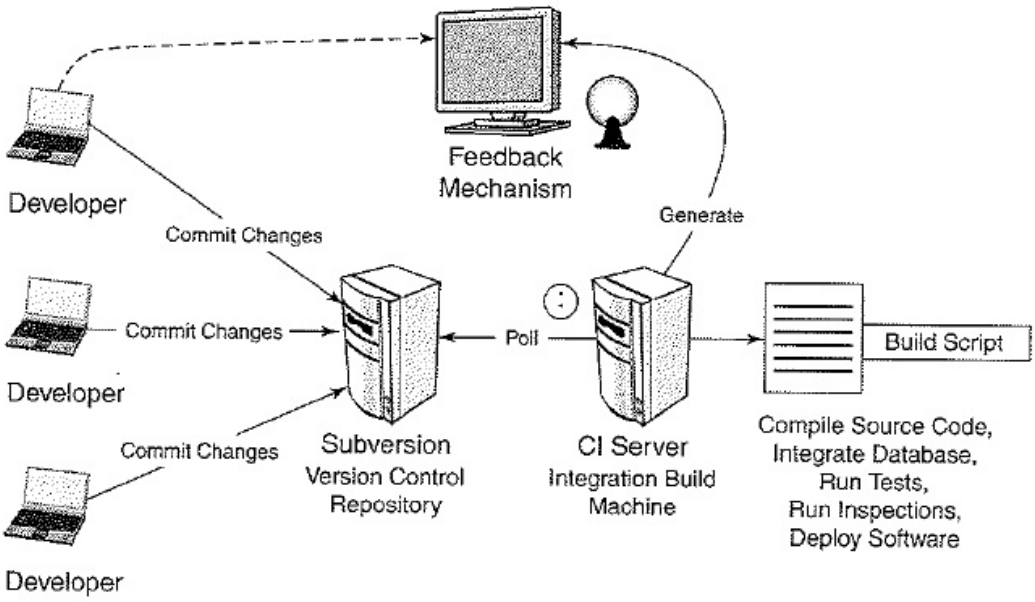
3.3. Jatkuva integraatio

Jatkuvalla integraatiolla tarkoitetaan nimensä mukaisesti prosessia, jossa integrointi on jatkuvaa. Jatkuvan integraation periaatteiden mukaisesti integrointi tulisi suorittaa aina, kun muutoksia tapahtuu. Tämä ajattelutapa on saanut alkunsa tarpeesta kehittää integrointiprosessia. Ennen kuin jatkuva integraatio esiteltiin ohjelmistokehitykseen, ohjelmiston integrointi suoritettiin yleensä vesiputousmallin mukaisesti ohjelmiston kehityksen loppupäässä. Tämä integrointi tapa oli hyvin työläs ja sen kestoa oli mahdoton arvioida. Projektin parissa oli voitu työskennellä vuosia kokeilematta kuinka ohjelmiston eri

osat toimivat keskenään lopputuotteessa. Jatkuva integraatio on alun perin yksi Extreme programming -kehityksen perusideoista, josta se on otettu yleisempään käyttöön. Extreme programming on yksi ketterän kehityksen tyyleistä.

Yksi jatkuvan integraation vaatimuksista on integrointiprosessin automatisointi. Koska integrointia suoritetaan todella usein, se pitää pystyä tekemään täysin automaattisesti. Duvall, Matyas ja Glover (2007, 67) kuvailevat tätä automatisointi osuvasti integration-painikkeella. Heidän mukaan integrointiprosessi pitää saattaa siihen tilaan, että koko prosessi voidaan suorittaa alusta loppuun yhtä nappia painamalla.

Käytännössä jatkuva integraatio toteutetaan integrointipalvelimen avulla. Tällä palvelimella suoritetaan integrointi. Kuviossa 2 kuvataan yksinkertaista jatkuvaa integraatiota suorittavaa integrointiympäristöä. Kuviossa vasemmalla laidalla olevat kehittäjät vievät tekemiään muutoksia versionhallintavarastoon, jota integrointipalvelin tarkkailee. Kun integrointipalvelin havaitsee muutoksia versionhallintavarastossa, se käynnistää automatisoidun integrointiprosessin. Tästä integrointiprosessista viedään välittömästä palaute muutoksen tehneelle kehittäjälle.



KUVIO 2. Jatkuva integraatio periaatetasolla (Duvall ym. 2007, 5)

Kuvion 2 tapahtumaketju toteutuu aina kun versionhallintavarastoon viedään muutoksia. Tämän takia jatkuvassa integraatiossa onkin tärkeää, että kehittäjien tekemät muu-

tokset viedään tarpeeksi usein versionhallintavarastoon. Duvall ym. (2007, 40) neuvovat, että muutoksia tulisi viedä versiohallintavarastoon pieninä osina. Työ tulisi jakaa osiin, jotka eivät riko kokonaisuutta eivätkä vaikuta kaikkeen samaan aikaan.

Duvall ym. (2007, 29) jakavat jatkuvan integraation höydyt viiteen osaan:

- Riskien väheneminen.
- Manuaalisen työn väheneminen.
- Ohjelmisto on aina tilassa, jossa se voidaan asentaa päätelaitteelle.
- Projektin tila on kaikkien tiedossa.
- Ohjelmistokehitysryhmän luottamus omaan tekemiseen kasvaa.

Jatkuvan integraation ensimmäinen hyöty on siinä, että kun integrointi suoritetaan mahdollisimman usein, saadaan myös kaikki integrointiin liittyvät virheet havaittua mahdollisimman aikaisin. Integrointiprosessista saatava välitön palaute myös pakottaa ohjelmistokehitysryhmää korjaamaan kaikki havaitut virheet mahdollisimman nopeasti. Pohjimmiltaan virheiden nopea havaitseminen ja korjaaminen vähentävät riskejä koko ohjelmistokehityksessä.

Integrointiprosessin jatkuva suorittaminen aiheuttaa myös sen, että kun muutokset integroidaan välittömästi, ne ovat käytössä myös välittömästi lopputuotteessa. Tällöin meillä on aina käytössä versio kehitettävässä ohjelmistosta, johon on tuoreimmat ominaisuudet lisätty. Tämä on yksi jatkuvan integraation hyödyistä, jolla vältämme vesiputous-mallin työlään integraation lopussa.

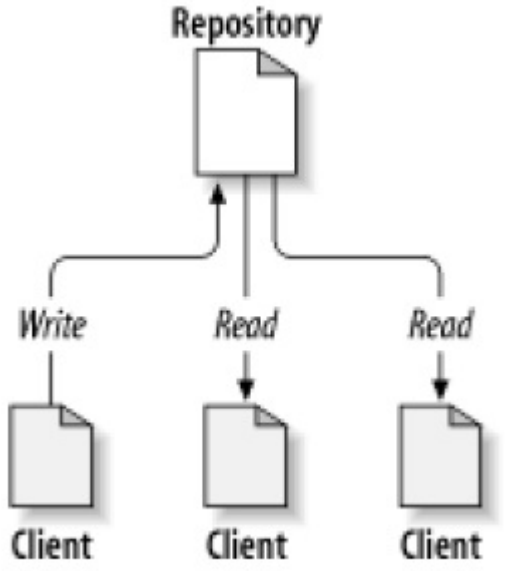
Projektin tila on jatkuvassa integraatiossa kaikkien tiedossa. Kaikki kehittäjät vievät tekemiään muutoksia jatkuvasti versiohallintaan, josta ne integroidaan välittömästi. Tällöin ei pääse syntymään tilanteita, jossa yksi kehittäjä pitää omassa paikallisessa työhakemistossa muutoksia, jotka vaikuttavat moneen muuhun osaan.

Kaikista edellisistä syistä johtuen ohjelmistokehitysryhmän luottamus tuotteeseen kasvaa. Tämä on suuri positiivinen asia ohjelmistokehityksessä, koska ohjelmistokehitysryhmä, joka luottaa tekemäänsä tuotteeseen ja sen laatuun, tekee yleensä parempaa jälkeä.

3.4. Versionhallinta

Tässä luvussa tullaan käsittelemään versiohallinnan periaatteita. Luvussa ei oteta kantaa, mikä käytettävä versionhallintajärjestelmä on. Tässä luvussa käsiteltävät asiat ovat yleiskäsitteisiä ja koskevat kaikkia keskusvaraston päälle rakentuviin versiohallintajärjestelmiin. Hajautetut versiohallintajärjestelmät ovat nykyään myös kasvattaneet suosiotaan. Hajautetulla versiohallintajärjestelmällä tarkoitetaan versionhallintajärjestelmää, jossa ei ole selvää keskusvarastoa, vaan tieto on hajautettu useampaan paikkaan. Versiohallinta on tärkeä kokonaisuus, joka on ehdoton edellytys jatkuvalle integraatiolle. Ilman toimivaan versiohallintajärjestelmää jatkuvaa integraatiota on mahdoton toteuttaa. Versionhallinta on yksi osa konfiguraationhallintaa ja se on tärkeä osa jatkuvan integraation kannalta.

Versiohallinnalla tarkoitetaan kokonaisuutta, joka mahdollistaa useamman henkilön samanaikaisen tiedostojen käytön. Kuviossa 3 on esitetty hyvin yksinkertainen käyttötapa versiohallintajärjestelmällä.



KUVIO 3. Versionhallintajärjestelmän perusidea (Collins-Sussman, Fitzpatrick & Pilato 2008, 1)

Kuviossa 3 ylhäällä oleva osa on ns. versiohallintavarasto. Versiohallintavarasto on paikka, josta käyttäjät lukevat tiedostoja ja tallentavat niitä. Versionhallintajärjestelmän yhtäaikainen käyttö perustuu siihen, että käyttäjät hakevat versiohallintavarastosta itselleen omat kopiot muokattavista tiedostoista. Kun käyttäjä on tehnyt omaan kopioonsa

halutut muutokset, hän kopio tiedoston versionhallintavarastoon, jolloin tehdyt muutokset ovat kaikkien käytettävissä.

Jokaisesta muutoksesta, joka viedään versiohallintavarastoon, perustetaan oma versio versionhallintajärjestelmään. Näiden versioiden avulla pystytään seuraamaan muutoksia, joita tiedostoille on tehty. Tarvittaessa versiohallintajärjestelmällä voidaan palauttaa tiedosto johonkin haluttuun aikaisempaan versioon.

Versionhallintajärjestelmissä on mahdollista perustaa haaroja. Haaroilla tarkoitetaan tässä yhteydessä kehityslinjaa, joka eroaa päähaarasta. Haara voidaan perustaa esimerkiksi tilanteessa, jossa sovellukseen kehitetään ominaisuus, jota ei haluta ottaa käyttöön päähaarassa. Haaroittaminen ei ole suositeltavaa jatkuvassa integraatiossa, vaan sitä tulisi tehdä ainoastaan pakosta. Haaroittamista ei suositella, koska jokaista haaraa pitää haaroittamisen jälkeen integroida erikseen. Tämä kasvattaa nopeasti integrointia vaativien kokonaisuuksien määrää. Samalla myös tuotteen integrointi muuttaa vaikeammaksi ymmärtää.

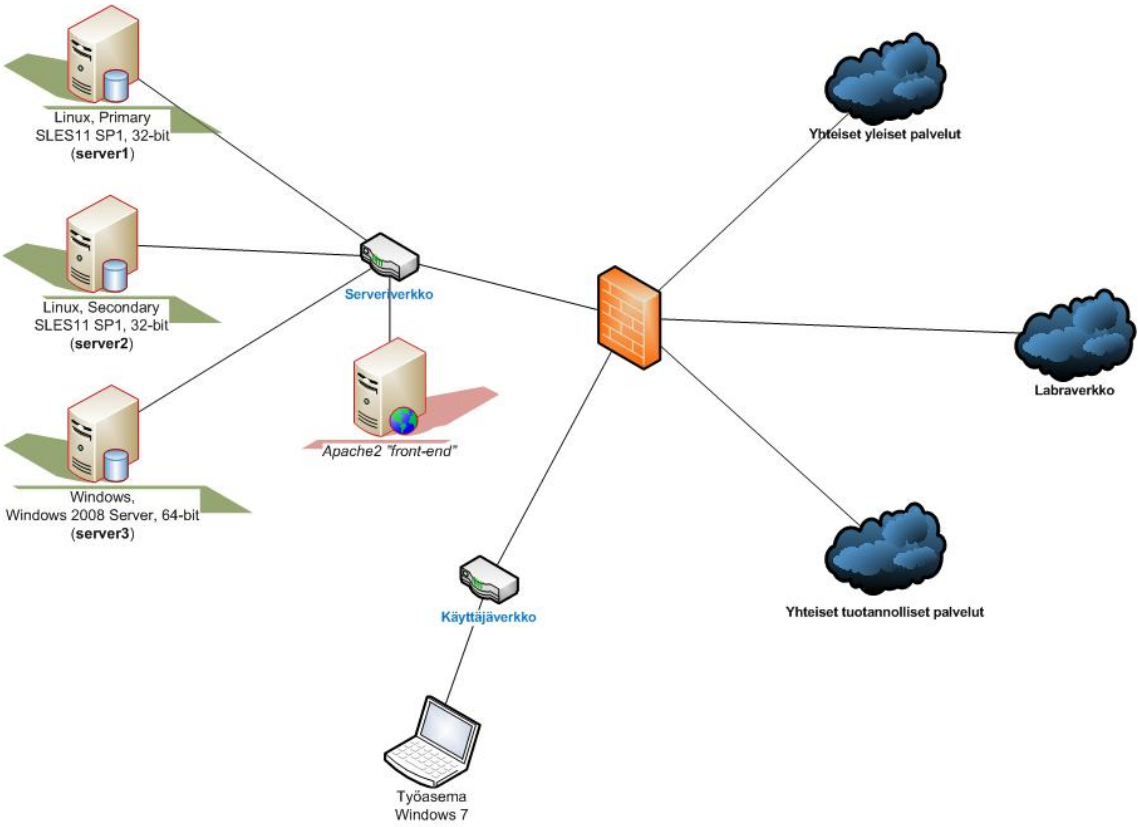
Versiohallinta on erityisen tärkeää jatkuvan integraation kannalta, koska koko jatkuvan integraation idea perustuu muutoksien aikaiseen ja jatkuvaan integrointiin. Versionhallintajärjestelmät tarjoavat jatkuvalle integraatiolle tavan havaita näitä muutoksia. Jos versiohallintajärjestelmää ei olisi käytössä jatkuvaa integraatiota suorittavassa integrointiympäristössä, joutuisimme jollain muulla tavalla kertomaan integrointiympäristölle koska muutoksia on tapahtunut.

4. Kehitys- ja integrointiympäristön kuvaus

Seuraavissa luvuissa tullaan kuvaamaan kehitys- ja integrointiympäristöä ja sen palveluita. Luvussa esitellään aluksi ympäristön rakenne. Kun alusta, jolle kokonaisuus on rakennettu, on lukijalle tuttu, kuvataan hieman yksityiskohtaisemmin ympäristön osia.

4.1. Rakenteellinen kuvaus

Kehitys- ja integrointiympäristö rakennettiin verkkoon, jossa on myös muiden hankkeiden projekteja. Verkon ylläpidosta vastaa yrityksen ICT-tukipalvelut. Kuviossa 4 on kuvattu tuota verkkoa rakennetun kehitys- ja integrointiympäristön kannalta. Muihin hankkeisiin liittyvät aliverkot ja palvelimet on jätetty kokonaan kuviosta pois.



KUVIO 4. Verkon rakenne (Kallio 2011, muokattu)

Kuviossa 4 nähdään vasemmassa reunassa palvelimet, joille kehitys- ja integrointiympäristö on toteutettu. Muut kuviossa olevat aliverkot ja palvelimet tarjoavat yleisiä pal-

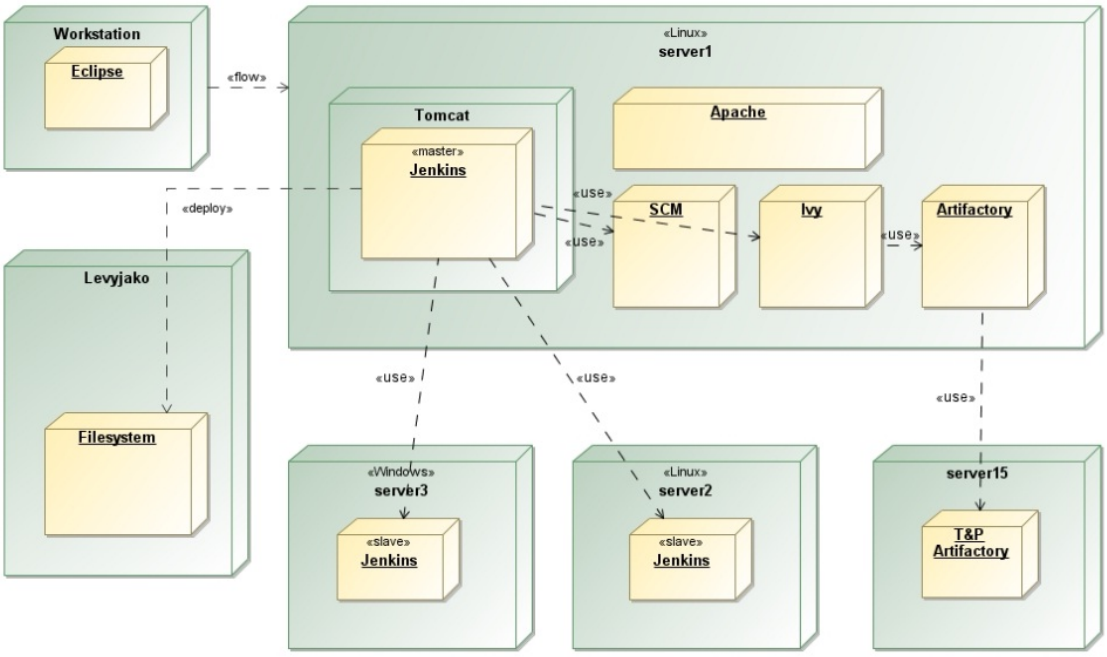
veluita koko yrityksen käyttöön. Osaa näistä palveluista on hyödynnetty kehitys- ja integrointiympäristössä.

Varsinainen kehitys- ja integrointiympäristö koostuu viidestä osasta: kehittäjän työasemasta, kolmesta palvelimesta ja verkkolevyistä. Kehittäjän työasemalla tapahtuu varsinainen kehitystyö eli manuaalinen muutosten tekeminen lähdekoodiin. Palvelimet hoitavat kaikki versionhallintaan, integrointiin ja tukipalveluihin liittyvät toimenpiteet. Viimeinen osa eli verkkolevyjako on paikka, johon julkaisuprosessissa julkaistaan tuotteita ja komponentteja.

Kaikki kolme palvelinta ovat ympäristössä virtuaalipalvelimia, joita ylläpitää yrityksen ICT-tukipalvelut. Kahdella palvelimella on käyttöjärjestelmänä Linux SUSE Enterprise Edition 11 SP 1 ja yhdellä on Windows Server 2008 R2. Palvelimia tullaan kutsumaan jatkossa nimillä:

- Server1, ensimmäinen Linux-käyttöjärjestelmän sisältävä palvelin
- Server2, toinen Linux-käyttöjärjestelmän sisältävä palvelin
- Server3, Windows-käyttöjärjestelmän sisältävä palvelin.

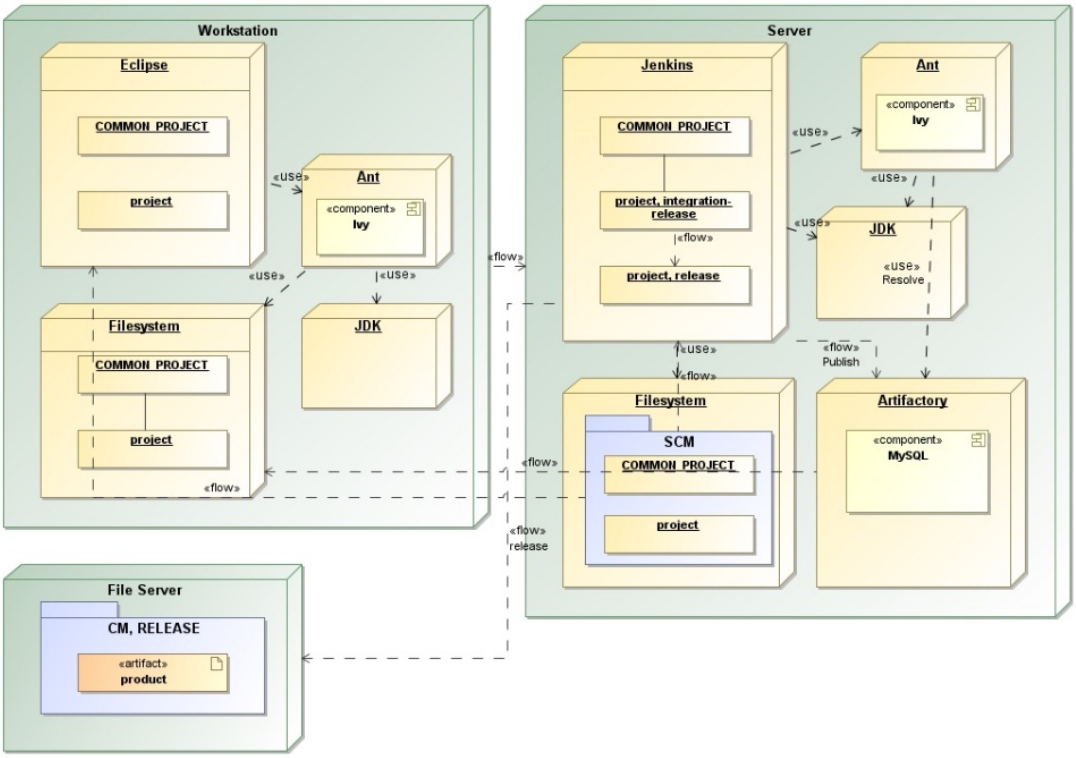
Seuraavassa kuviossa on esitetty kehitys- ja integrointiympäristön rakennetta. Kuvioon on myös merkitty eri palvelimilla sijaitsevia palveluita, joita tullaan tarkentamaan.



KUVIO 5. Kehitys- ja integrointiympäristön rakenne (Korhonen, Heinonen, Helin & Peltola 2011, muokattu)

Kuviosta 5 nähdään, että kehitys- ja integrointiympäristön pääpalvelin on Server1. Tällä palvelimelle sijaitsee tärkeimmät palvelut ja järjestelmät, kuten versionhallinnat, riippuvuuksienhallinta ja Jenkins-integrointipalvelun master-solmu. Näiden lisäksi palvelimella sijaitsee muita tukisovelluksia ja -palveluita, kuten testauksenhallinta, vikatietokanta ja koodianalysointipalvelu. Server2- ja Server3-palvelimet toimivat tässä vaiheessa ainoastaan Jenkins-integrointipalvelun slave-solmuina. Näille palvelimille on kuitenkin mahdollista siirtää osa palveluista, jos Server1-pavelimen kuorma alkaa nousta liikaa.

Kuviossa 6 on kuvattu kehitys- ja integrointiympäristö käyttäjän näkökulmassa. Vasemmalla on esitetty käyttäjän omalla työasemalla olevia palveluita. Oikeassa laatikossa on taas esitetty kehitys- ja integrointiympäristön palveluita.



KUVIO 6. Tarkempi kuvaus integrointiympäristön rakenteesta ja palveluista (Korhonen, Heinonen, Helin & Peltola 2011, muokattu)

Kuviossa 6 kuvataan integrointiympäristöä hieman tarkemmin. Siinä on kuvattu eri palveluiden riippuvuuksia toisiinsa ottamatta kantaa millä palvelimella ne sijaitsevat. Näiden palveluiden ja riippuvuuksien toimintaa käsitellään yksityiskohtaisesti seuraavissa luvussa.

4.2. Palvelut

Seuraavissa alaluvuissa tullaan käsittelemään kehitys- ja integrointiympäristöä sen palveluiden kautta. Palvelut on jaettu kategorioihin niiden käyttötarkoituksen mukaan.

4.2.1. Palvelinohjelmistot

Seuraavissa alaluvuissa käsitellään Linux-käyttöjärjestelmällä varustetuiden palvelimien hallintaan liittyviä palveluita. Nämä palvelut ovat palvelimien pohja ja niiden kautta hallitaan esimerkiksi seuraavia asioita:

- Pääsyä palvelimille http-liikenteen kautta.
- Välitystoiminnallisuuden tarjoaminen.
- Käyttäjien tunnistaminen ja käyttöoikeuksien hallinta osalla palveluista.

4.2.1.1. Apache Http -palvelin

Linux-käyttöjärjestelmällä varustetuiden palvelimien palvelinohjelmistona toimii Apache http -palvelin. Apache http -palvelin on open source -tuote, jolla pystytään hallinnoimaan http-liikennettä palvelimen ja asiakkaan välillä.

Apache http -palvelimella on toteutettu kehitys- ja integrointiympäristön palveluiden http-käyttöliittymien välitystoiminnallisuus. Välitystoiminnallisuudella tarkoitetaan sitä, että voimme välittää palvelimelle tulevaa http-liikennettä URL-osoitteesta toiseen. Tämä välitystoiminnallisuus jouduttiin toteuttamaan, koska palvelimella haluttiin avata http-liikenteelle ainoastaan 80- ja 8080-portit tietoturva syistä. Jokainen palvelu silti vaatii itselleen avoimen portin http-käyttöliittymän tarjoamiseen. Apache http -palvelimen välitystoiminnallisuudella tämä ongelma pystyttiin ratkaisemaan. Apache http -palvelin välittää liikennettä sisäisesti käyttäjälle siitä kertomatta oikeaan paikkaan. Esimerkiksi Jenkins-palvelu on kehitys- ja integrointiympäristössä asennettu siten, että sen http-käyttöliittymään saa yhteyden osoitteesta http://server1:8082/jenkins. Tuo 8082-portti on kuitenkin palomuurissa kiinni, eivätkä http-pyynnöt sitä kautta kulje perille. Apache http -palvelimen asetettiin välittämään kaikki liikenne http://server1/jenkins-osoitteesta palvelimen sisällä http://server1:8082/jenkins-osoitteeseen. Tämä tarkoittaa sitä, että työasemalta palvelimelle http-pyynnöt kulkevat normaalin 80-portin kautta, joka on palomuurissa auki. Kun pyyntö saapuu palvelimelle, Apache http -palvelin välittää pyynnön oikeaan osoitteeseen.

Apache http -palvelimen välitystoiminnallisuus ei rajoitu ainoastaan liikenteen ohjaamiseen portista toiseen, vaan sillä voi ohjata liikennettä myös palvelimelta toiselle. Kehitys- ja integrointiympäristössä kokeiltiin tilannetta, jossa palveluita siirrettiin Server1-palvelimelta Server2-palvelimelle ja Apache http -palvelin asetettiin Server1-palvelimella ohjaamaan liikenne Server2-palvelimelle. Tämä on hyödyllinen mahdollisuus, koska sen avulla palveluita voidaan siirtää kehitys- ja integrointiympäristössä palvelimelta toiselle ilman, että se näkyy käyttäjälle millään tavalla.

4.2.1.2. Apache Tomcat 6

Apache Tomcat on Apache Software Foundation -ryhmän tekemä open source WWW-sovelluspalvelin. Ohjelmisto tarjoaa palvelimelle ympäristön, jossa voidaan suorittaa Java-koodia ja esittää sitä selaimen kautta (Apache Tomcat Wiki 2012). Monet kehitys- ja integrointiympäristön palveluista ovat rakennettu käyttäen Java Servlet Pages -tekniikkaa. Käytössä oleva versio Tomcat-ohjelmistosta on 6.

Eri palveluille on rakennettu omat Tomcat-instanssit, jotta niiden hallittavuus pysyisi mahdollisimman helppona. Server1-palvelimella on kolme palvelua, joilla kaikilla on omat Tomcat-instanssit. Nämä palvelut ovat Artifactory, Jenkins ja Sonar. Monet palvelut sisältävät kevyen http-moottorin tai -palvelun, mutta Tomcat on tarkoitettu tuotantokäyttöön ja on siten luotettavampi, suorituskykyisempi ja skaalautuvampi.

4.2.2. Versionhallintajärjestelmät

Kyseisessä kehitys- ja integrointiympäristössä on kaksi versionhallintavarastoa käytössä. Käytössä olevat versionhallintajärjestelmät ovat SVN ja CVS. Ympäristössä on kaksi versionhallintaa käytössä, koska ympäristöä käyttävällä osastolla on kehityksessä tuotteita, joiden kehittäminen on aloitettu kauan sitten. Alun perin nuo lähdekoodiprojektit on perustettu CVS-versionhallintaan ja tätä ei ole haluttu vaihtaa kesken tuotteen elinkaaren. Ajan kuluessa kuitenkin huomattiin, että CVS-versionhallintajärjestelmä ei täytä kaikkia vaatimuksia. Näitä vaatimuksia täyttämään valittiin SVN-versionhallintajärjestelmä. Kaikki uudet projektit on lisätty kehitys- ja integrointiympäristössä jo jonkin aikaa SVN-versiohallintaan ja pääkäyttö onkin tällä.

4.2.2.1. SVN-versionhallintajärjestelmä

Suuremmassa käytössä oleva SVN-versionhallintajärjestelmä koostuu kahdesta versionhallintavarastosta. Jako kahteen varastoon on tehty loogisesti kahteen eri tuoteperheeseen kuuluvien projektien kanssa. SVN-versionhallintavarastot sijaitsevat server1-palvelimella seuraavissa hakemistoissa:

- /srv/svn/HANKE1
- /srv/svn/HANKE2

SVN-versionhallintavarastojen käyttöoikeuksien hallinta on hoidettu Apache2:sen avulla. Tästä syystä kaikki liikenne varastoihin kulkee http-liikenteenä. Käyttäjän kannalta versionhallintavarastot löytyvät osoitteista:

- http://server1/svn/HANKE1
- http://server1/svn/HANKE2

Käyttöoikeuksien hallinta molempiin varastoihin on toteutettu Apache2:sen dav_module-, dav_svn_module-, authz_svn_module-, authnz_ldap_module- ja ldap_module-nimisten kirjastojen avulla. Näitä moduuleita käyttämällä saatiin toteutettua kokonaisuus, jossa käyttöoikeudet versionhallintavarastoihin määritellään LDAP-rajapinnan kautta yrityksen AD:sta.

SVN-varaston selaamiseen selaimen kautta asennettiin server1-palvelimelle WebSVN-ohjelmisto. Tämä sovellus mahdollistaa helpon ja nopean versionhallintavaraston selaamisen. WebSVN on tarkoitettu ainoastaan versionhallintavaraston sisällön selaamisen; sillä ei voi tehdä normaaleja SVN-toimenpiteitä kuten commit tai checkout. Tämä palvelu otettiin käyttöön, koska usein tulee tarve päästä nopeasti tarkistamaan jostakin versionhallinnassa olevasta tiedostosta jokin pieni asia, kuten esimerkiksi kahden eri version eroavaisuus. WebSVN ei vaadi, että koneelle on asennettu mitään erillistä ohjelmistoa vaan pelkkä internet-selain riittää sen käyttöön.

4.2.2.2. CVS-versionhallintajärjestelmä

Kehitys- ja integrointiympäristössä on käytössä yksi CVS-versionhallintavarasto. Varasto sijaitsee server1-palvelimella /srv/cvs/HANKE1-hakemistossa. CVS-

versionhallintavarasto käyttää ext-protokollaa ja käyttöoikeuksienhallinta on hoidettu SSH-avainparien avulla.

SSH-avainparin käyttö tarkoittaa sitä, että jokaiselle CVS-versionhallintajärjestelmän käyttäjälle on luotu palvelimella oma käyttäjätunnus ja home-hakemisto. Home-hakemistoon luodaan käyttäjäkohtainen .ssh-hakemisto, esimerkiksi /home/olli.piipponen/.ssh/. Tähän hakemistoon tallennetaan SSH-avainparin julkinen osa eli id_rsa-tiedosto. Avaimen private-osa jää käyttäjälle itselleen. Tätä private-osaa voidaan käyttää palvelimelle tunnistautumiseen. Tätä tekniikkaa käytettäessä päästään myös eroon jatkuvasta salasanan syöttämisestä. SSH-avaimen private-osa voidaan tallentaa ns. SSH-agenttiin. SSH-agentti on ohjelmisto, joka käynnistetään työasemalle ja jätetään taustalle ajoon. SSH-agenttiin voidaan tallentaa SSH-avainparien private-osia. Tällöin aina kun avataan SSH-yhteys, SSH-asiakassovellus yrittää automaattisesti tarjota palvelimelle SSH-agentissa olevia avainparien private-osia, kun palvelin tekee tunnistautumispyynnön. Mikäli avain täsmää palvelimen julkisen avaimen kanssa, antaa palvelin suoraan pääsyn käyttäjälle. Tässä kehitys- ja integrointiympäristössä SSH-asiakassovelluksena käytetään PuTTY-ohjelmistoa. PuTTY-ohjelmiston SSH-agentti on nimeltään Pageant.

Työasemalla, jolla SSH-asiakassovellusta ja SSH-agenttia käytetään, pitää myös määrittää CVS_RSH-ympäristömuuttuja. Tämä muuttuja kertoo CVS:n ext-protokollalle mitä binääriä käytetään SSH-yhteyden avaamiseen CVS-toimenpiteissä. Tässä kehitys- ja integrointiympäristössä käytetään PuTTY plink -sovellusta, joka on komentoriviversio normaalista PuTTY-ohjelmistosta. Kun työasemalta kutsutaan jotakin CVS-komentoa, CVS huomaa, että käytettään ext-protokollaa. CVS käy tarkistamassa CVS_RSH-ympäristömuuttujasta, mitä ulkoista binääriä se käyttää yhteyden muodostamiseen. CVS käynnistää PuTTY plink -sovelluksen. PuTTY plink käy yhteyden muodostamisen yhteydessä tarkistamassa onko työasemalla käynnissä SSH-agent pageant. PuTTY plink löytää SSH-agentin ja tarjoaa palvelimelle tunnistautumiseen SSH-agentista löytyviä SSH-avaimia. Kaikki tämä on ennalta konfiguroitu ja tapahtuu automaattisesti. Käyttäjältä ei siis normaalissa käytössä vaadita mitään syötteitä normaalin CVS-komennon lisäksi.

CVS-versionhallintavaraston selaamiseen selaimen kautta on asennettu palvelimelle Viewvc-ohjelmisto. Viewvc on samantapainen ohjelmisto kuin WebSVN. Sen kautta ei

pysty tekemään mitään normaaleja CVS-toimenpiteitä; vaan se on tarkoitettu CVS-versionhallintavaraston sisällön selaamiseen. Tämä palvelu otettiin käyttöön, koska usein tulee tarve päästä nopeasti tarkistamaan jostakin versionhallinnassa olevasta tiedostosta jokin pieni asia, kuten kahden eri version eriäväisyyden tarkistus. Viewvc ei vaadi, että koneelle on asennettu mitään erillistä ohjelmistoa vaan pelkkä internet-selain riittää sen käyttöön.

4.2.3. Integrointipalvelut

Seuraavissa luvuissa käsitellään integrointiin liittyviä palveluita kehitys- ja integrointiympäristössä.

4.2.3.1. Apache Ant -työkalu ja COMMON_PROJECT-tukiprojekti

Ant-työkalua käytetään erilaisten Ant-kohteiden suorittamiseen. Näissä kohteissa määritellään erilaisia asioita, joita tehdään. Ant-kohteissa voidaan esimerkiksi koostaa lähdekoodiprojekti, suorittaa yksikkötestejä tai käynnistää Javadoc-dokumentaation luonti. Ant-työkalua käytetään osana integrointiympäristöä, mutta sitä voidaan käyttää myös yksittäiseltä työasemalta. (Apache Ant Wiki 2012)

COMMON_PROJECT-tukiprojekti on kokoelma kaikille lähdekoodiprojekteille yhteisiä ja yleisesti käytettyjä Ant-kohteita. Käyttämällä tämän tyylistä tukiprojektia, säästytään samojen yleisten Ant-kohteiden kirjoittamiselta useaan paikkaan. Toinen hyöty tukiprojektin käyttämisellä on se, että näiden Ant-kohteiden ylläpitäminen on helppoa. Tarvittaessa muutos pitää tehdä ainoastaan yhteen paikkaan. COMMON_PROJECT-tukiprojektin tapaisia tukiprojekteja löytyy avoimen lähdekoodin maailmasta monia valmiiksi, mutta tässä kehitys- ja integrointiympäristössä mikään noista valmiista tukiprojekteista ei täyttänyt kaikkia vaatimuksia. Tästä syystä päädyttiin rakentamaan oma tukiprojekti, jonka ylläpito on omalla vastuulla.

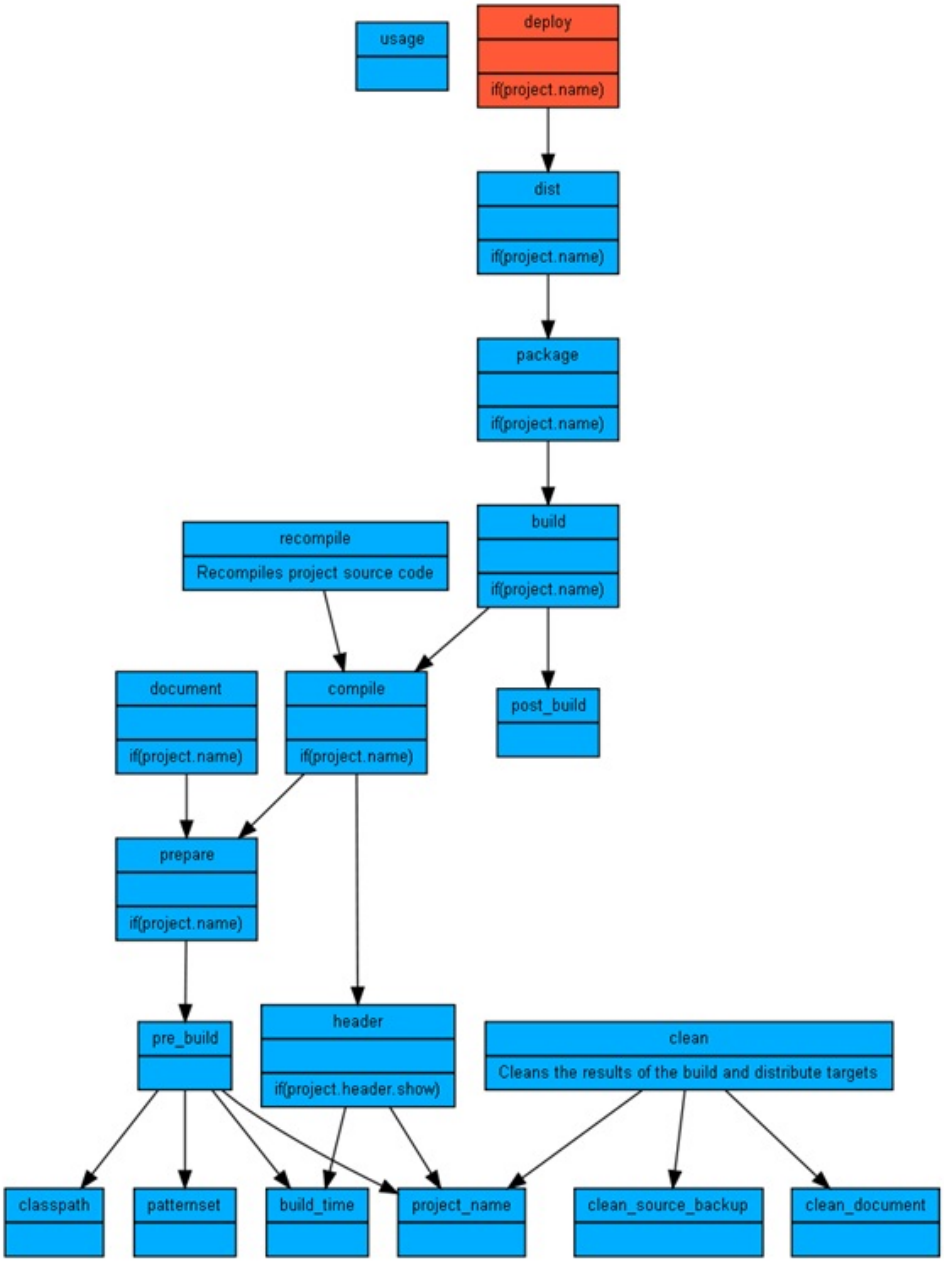
Taulukossa 1 listataan kehitys- ja integrointiympäristön COMMON_PROJECT-tukiprojektin sisältämät tehtävät ja niiden kuvaukset. Kuviossa 7 on esitetty tehtävien

riippuvuuksia toisiinsa. Kuvio 7 on tuotettu Graphviz-työkalulla, jota käytetään kehitys- ja integrointiympäristössä riippuvuuksien graafiseen havainnollistamiseen.

TAULUKKO 1. Konfiguraatioyksikön yleisimmät apu- ja päätason tehtävät kuvauksineen (Helin, Peltola & Hietaranta 2009, 17, muokattu).

Tehtävä	**Tarkoitus**
usage	Tehtävä tulostaa `build.xml`-tiedostossa määritellyn merkkijonopohjaisen käyttökuvauksen eri tehtäville. Lisäksi voi käyttää komentoa `ant -p` tai `ant -projecthelp`, joka tulostaa päätason tehtävät kuvauksineen.
classpath	Tehtävä asettaa erilaiset luokkapolkuviittaukset (`ref`) ja tärkeimmät luokkapolkumuuttujat (`property`), joiden kautta asetetaan eri ympäristö ja riippuvuudet tehtävien suorittamista varten.
patternset	Tehtävä muodostaa eri tiedosto- ja hakemistojoukot, joiden perusteella tehtävien tiedostojärjestelmän käyttö (esim. kopiointi) suoritetaan.
build_time	Tehtävä muodostaa Ant-työkalun suorituksen aloitusaikaleiman ja sijoittaa sen tiettyyn muuttujaan (`property`) muita tehtäviä varten.
project_name	Tehtävä muodostaa lähdekoodiprojektin nimen, joka perustuu erikseen annettavaan muuttujaan (`property`) tai lähdekoodiprojektin ylimmän tason hakemistonimeen.
clean_source_backup	Tehtävä poistaa kaikki tilapäiset tiedostot ja varmuuskopiot lähdekoodi- ja resurssihakemistoista, esim. ~-päätteiset tiedostot.
clean_document	Tehtävä poistaa kaikki dokumentointiin ja muiden työkalujen tuloksiin liittyvät hakemistot `doc`-hakemiston alta.
Clean	Tehtävä poistaa kaikki Ant-työkalun avulla suoritettujen tehtävien luomat tilapäiset hakemistot ja tiedostot sekä kutsuu muita erillisiä poistotehtäviä, esim. `clean_document` ja `clean_source_backup`.
Header	Tehtävä tulostaa erilaisia lähdekoodiprojektiin liittyviä tietoja kuten projektin nimen ja Java- ja Ant-ympäristön tiedot.
pre_build	Tehtävä sisältää konfiguraatioyksikön rakentamiseen liittyviä alustavia toimenpiteitä, joita voi tarkentaa projektikohtaisesti.
Prepare	Tehtävä tekee kääntämiseen ja rakentamiseen liittyviä yleisiä alustuksia kuten luo tarvittavat tilapäiset hakemistot käännös- ja koostamistuloksille.
document	Tehtävä vastaa javadoc-dokumentaation luomisesta javadoc-työkalun avulla. Tehtävän suorittamiseen voidaan liittää muita dokumentaatioon liittyviä toimenpiteitä kuten UML-luokkakaavioiden luomisen erikseen määritellyn kuvauksen perusteella.

compile	Tehtävä vastaa lähdekooditiedostojen kääntämisestä `build`-hakemiston alle.
recompile	Tehtävä suorittaa lähdekooditiedostojen uudelleen kääntämisen poistamalla aluksi käännöstulokset ja sen jälkeen suorittaa kääntämisen (`compile`).
post_build	Tehtävä sisältää konfiguraatioyksikön rakentamisen jälkeisiä toimenpiteitä, joita voi tarkentaa projektikohtaisesti.
Build	Tehtävä suorittaa kääntämisen ja rakentamisen sekä koostaa konfiguraatioyksiköstä suoritettavan ohjelmakoodihakemiston `build`-hakemistoon.
package	Tehtävä paketoi koostamistehtävän muodostaman ohjelmakoodihakemiston `jar`-paketiksi, joka on yleensä suoritettava Java-ohjelma (`executable jar`).
Dist	Tehtävä suorittaa paketoidulle jar-paketille ylimääräisiä ja konfiguraatioyksikkökohtaisia toimenpiteitä kuten tarkistussummatiedostojen luomisen eri tiivistealgoritmeilla ja digitaalisen allekirjoituksen.
Deploy	Tehtävä riippuu edellisistä tehtävistä ja muodostaa päätason tehtävän, jonka vastuulla on kopioida (sijoittaa) lopullinen konfiguraatioyksikön muodostava paketti tarkistussummatiedostoineen valittuun hakemistoon. Tämän tehtävän suorittaminen vaatii integrointiympäristön asentamisen Ant-työkalun ja lisäkirjastojen sekä muiden työkalujen osalta ja vastaa manuaalista integrointia, esim. kehittäjän työasemalla tai koostamispalvelimella.



KUVIO 7. Konfiguraatioyksikön yleisimmät apu- ja päätason tehtävät sekä niiden väliset riippuvuudet (Helin, Peltola & Hietaranta 2009, 19).

Ant-työkalun käyttö perustuu siihen, että yhden työkalun kautta voidaan määritellä ja kutsua kaikkia konfiguraatioyksikön tarvitsemia toimenpiteitä. Toinen tärkeä asia Ant-työkalussa on se, että Ant-työkalu on käyttöjärjestelmäriippumaton. Kun Ant-työkalu on asennettu työasemalle tai palvelimelle, sen käyttö toimii aivan samalla tavalla aina käyttöjärjestelmästä riippumatta. Tämä on tärkeä asia kehitys- ja integrointiympäristössä, koska jos esimerkiksi koostamisessa pitäisi ottaa huomioon käyttöjärjestelmä, se aiheuttaisi omien integrointitöiden tekemisen jokaiselle käytettävälle käyttöjärjestelmälle.

Ant-työkalun alkuperäinen käyttötapa on komentoriviltä kutsuminen, esimerkiksi Windows-käyttöjärjestelmän komentokehoiteelta. Nykyään Ant-työkalua voidaan käyttää myös IDE-työkaluista löytyvien käyttöliittymien kautta. Työkalua käytetään työaseman komentoriviltä syöttämällä komentoriville käsky ant ja Ant-kohteen nimi, joka on kuvattu build.xml-tiedostossa. Komento tulee antaa hakemistossa, jossa build.xml-tiedosto sijaitsee.

4.2.3.2. Jenkins-integrointipalvelu

Jenkins-palvelun tarkoitus on toimia integrointiympäristön keskuksena. Tämä palvelu mahdollistaa yksittäisten integrointiprosessien keskitetyn ja organisoidun suorittamisen. Jenkins-palvelu tarjoaa ympäristön, jossa voidaan määritellä eri tuotteille ja projekteille omat integrointityöt. Näiden töiden sisältö pystytään hyvin vapaasti määrittelemään, koska Jenkins-palvelu ei sinänsä ota kantaa tai rajoita sitä, mitä töissä tehdään (Smart 2011, 79). Kokonaisuuden kannalta Jenkins-palvelu on hyvin tärkeä osa integrointiympäristöä. Tästä palvelusta ollaan yhteydessä melkein jokaiseen integrointiympäristön osaan, kuten versionhallintoihin, riippuvuuksienhallintavarastoihin ja julkaisupalvelimiin.

Käytetyin käyttötapa tässä integrointiympäristössä on erilaisten Ant-kohteiden suorittaminen.

Jenkins-palvelu on rakennettu kehitys- ja integrointiympäristössä master-/slave-arkkitehtuurin päälle. Tämä tarkoittaa sitä, että palvelulla on yksi master-solmu ja useampia slave-solmuja. Master-solmu käskyttää slave-solmuja. Master-/slave-arkkitehtuurin hyöty on siinä, että sillä saadaan hajautettua raskaita käännöstöitä useammille palvelimille. Kaikkia töitä ei ole järkevää ajaa yhdellä palvelimella, koska hyvin nopeasti yksittäisen palvelimen suorituskyky tulee vastaan. Hajauttamalla raskaita töitä useammalle palvelimelle saadaan yksittäisen palvelimen kuormaa laskettua. Tällöin koko kehitys- ja integrointiympäristön suorituskyky paranee huomattavasti. Master-/slave-arkkitehtuuri on erityisen tärkeä myös siitä syystä, että kehitys- ja integrointiympäristössä kehitetään ja integroidaan tuotteita, jotka vaativat integroinnin useammalla käyttöjärjestelmällä.

Jenkins-palvelun master-solmu on asennettu server1-palvelimelle ja sillä on slave-solmut server2- ja server3-palvelimilla.

Seuraavaksi käsitellään Jenkins-työn sisältöä. Uutta työtä luotaessa on muutamia tärkeitä asioita, jotka pitää ottaa huomioon.

- Milloin suoritetaan?

  Työlle pitää määritellä suorittamisajankohta. Vaihtoehtoja laukaisimille on:

  - Aja aina kun, jokin toinen työ on suoritettu

    Tämän tyyppisellä laukaisimella voidaan ketjuttaa töitä. Esimerkiksi, jos suoritamme jonkin core-moduulin käännöstyön, on hyvä tämän jälkeen ajaa myös kaikkien projektien käännöstyöt, joissa kyseistä core-moduulia käytetään.

  - Käynnistäminen etänä

    Tämä laukaisin mahdollistaan työn käynnistämisen etänä, esimerkiksi käskynä jostakin muusta palvelusta.

  - Ajastettu suorittaminen

    Työn suorittaminen aloitetaan tiettyinä ajankohtina. Tämän ajan pystyy määrittelemään minuuteista päiviin.

  - Versionhallinnan tarkkaileminen

    Jenkins-palvelu voidaan asettaa tarkkailemaan määriteltyä versionhallintavarastoa. Kun se havaitsee, että versionhallinnassa on tapahtunut muutos, aloitetaan työn suorittaminen.

  - Riippuvuuksienhallinnan tarkkaileminen

    Jenkins-palvelu voidaan asettaan tarkkailemaan konfigraatioyksikön riippuvuuksia. Jos konfiguraatioyksikön jossakin riippuvuudessa havaitaan muutos, käynnistetään integrointityö.

- Missä suoritetaan?

  Määritellään mikä solmu suorittaa kyseisen työn. Monimutkaisemmissa Multi-configuration -töissä voidaan määritellä työ suoritettavaksi useammissa solmuissa. Tämä on hyödyllinen ominaisuus esimerkiksi tapauksissa, jossa käännettävää ohjelmistoa käytetään useammalla käyttöjärjestelmällä.

- Mitä tehdään?

Työlle pitää kertoa mitä se tekee. Tämä voidaan määritellä useampana vaiheena. Erilaisia vaiheita on komentorivikomennon suorittaminen, Ant-kohteen suorittaminen, Maven-kohteen suorittaminen tai jonkin toisen Jenkins-työn käynnistäminen.

- Mitä tehdään ajon jälkeen?
  Julkaistaanko työntuloksia johonkin? Julkaistaan generoituja testiraportteja tai Javadoc-dokumentteja? Käynnistetäänkö lähdekoodin analysointi? Ilmoitetaanko käännöksen tuloksista jollekin?

4.2.3.3. Jatkuva integraatio ympäristössä

Jatkuva integraatio kehitys- ja integrointiympäristössä on toteutettu Jenkins-integrointipalvelun avulla. Ympäristössä suoritetaan jatkuvaa integrointia kahdella tasolla. Ensimmäinen tasoista on snapshot-taso ja toinen on integration-taso. Merkittävin ero näillä kahdella tasolla on, että snapshot-tason integrointia suoritetaan useammin ja se on kevyempää. Jokaiselle projektille on määritetty Jenkins-integrointipalvelussa molemmille tasoille oma työ. Näissä Jenkins-töissä on määritelty milloin tehdään ja mitä tehdään.

Snapshot-tason Jenkins-töissä on määritelty aloituskriteeriksi versionhallinnan tarkkailu. Tämä tarkoittaa sitä, että Jenkins-työ tarkkailee omia lähdekooditiedostojaan versionhallinnasta. Kun se havaitsee, että tiedostoissa on tapahtunut jokin muutos, aloitetaan snapshot-työn suorittaminen.

Integration-tason Jenkins-töitä suoritetaan ajastetusti. Tällä tarkoitetaan sitä, että Jenkins-työ käy kerran vuorokaudessa tarkistamassa versionhallinnasta, onko omiin lähdekooditiedostoihin tullut muutoksia edellisen vuorokauden integrointityöhön nähden. Jos muutoksia havaitaan, aloitetaan työn suorittaminen.

Molempien tasojen integroinnin sisältö on kuvattu luvussa 5.

4.2.4. Riippuvuuksienhallinta

Riippuvuuksienhallinta on kokonaisuus, jossa säilötään ja tarjotaan riippuvuuksia niitä tarvitseville komponenteille ja palveluille. Riippuvuuksienhallinnan toteutus kehitys- ja integrointiympäristössä koostuu kahdesta osasta: Artifactory-riippuvuuksienhallintavarastosta ja Ivy-riippuvuuksienhallintatyökalusta.

4.2.4.1. Apache Ivy -riippuvuuksienhallintatyökalu

Riipuvuuksienhallintaan käytetään kehitys- ja integrointiympäristössä Apache Ivy -palvelua. Ivy-palvelun käyttö tapahtuu Ant-työkalun kautta. Ant-työkalu kutsuu Ivy-palvelua, joka ratkaisee konfiguraatioyksikön riippuvuudet. Konfiguraatioyksikön riippuvuudet määritellään lähdekooditiedostossa nimeltä dependencies.xml. Konfiguraatioyksikön koostamisvaiheessa Ant-työkalu käy lukemassa dependencies.xml-tiedoston ja ratkaisee riippuvuudet ennalta määrätyiden ratkaisimien kautta (Apache Ivy Tutorial 2012). Näihin ratkaisimiin on määritelty Artifactory-palvelun varastoja, johon eri komponentteja on julkaistu. Kun Ivy löytää oikealla versiolla varustetun riippuvuuden, se lataa kyseisen riippuvuuden Artifactory-riippuvuuksienhallintavarastosta.

Koostamisen lopuksi Ivy-palvelu julkaisee aina ivy.xml-tiedoston, johon se kirjaa tarkkaan kaikkien sen lataamien riippuvuuksien tiedot, kuten versionumerot. Tästä tiedostosta pystytään jälkikäteen jäljittämään konfiguraatioyksikön eri komponenttien versionumerot. Tämä on erityisen tärkeää, jotta aina pystytään tarkasti jälkikäteen tuottamaan tietty konfiguraatioyksikkö oikeilla komponenteilla.

4.2.4.2. Artifactory-riippuvuuksienhallintavarasto

Riippuvuuksienhallintavarastona kehitys- ja integrointiympäristössä käytetään Artifactory-palvelua. Samaan tarkoitukseen löytyy myös muita vastaavia palveluita, kuten esimerkiksi Nexus-työkalu. Tässä tapauksessa kuitenkin päädyttiin käyttämään Artifactory-palvelua, koska yrityksessä se on käytössä myös muilla kehitysryhmillä. Artifactory-palvelussa on hyviä ominaisuuksia, joilla pystyy käyttämään hyväksi muita Artifactory-palveluita riippuvuuksien hakemiseen (Artifactory Wiki 2012).

Artifactory-palvelu asennettiin server1-palvelimelle Tomcat-palvelun päälle. Artifactory-palvelulla on oma Tomcat-instassi, jotta sen hallittavuus ja seuranta olisivat mahdollisimman helppoja. Artifactory-palvelun asennus jakautuu kolmeen paikkaan server1-palvelimella: /srv/artifactory/-hakemistoon, /srv/tomcat/artifactory/-hakemistoon ja MySQL-tietokantaan. /srv/tomcat/artifactory -hakemistoon on asennettu Artifactory-palvelun Tomcat. /srv/artifactory-hakemisto on taas paikka, johon Artifactory-palvelu tallentaa lokeja ja konfigurointitiedostoja. Varsinainen säilöttävä data tallennetaan Artifactory-palvelun MySQL-tietokantaan.

Artifactory-palvelun käyttöoikeuksienhallinta hoidetaan LDAP:lla yrityksen AD:ta vasten.

Artifactory-palvelu rakentuu varastojen päälle. Varastoihin julkaistaan komponentteja ja näitä komponentteja haetaan varastosta kun riippuvuuksien ratkaistaan. Varastoja on kolmen typpisiä: local, remote ja virtual. Local-varastolla tarkoitetaan paikallista varastoa, joka on tallennettu palvelimen levyllä sijaitsevaan MySQL-tietokantaan. Remote-varasto on varasto, joka sijaitsee jonkin muun palvelimen Artifactory-palvelussa. Yhteys remote-varastoihin muodostetaan http-yhteyden kautta. Kun remote-varasto on konfiguroitu Artifactory-palveluun, voidaan sitä käyttää samalla tavalla kuin normaalia paikallista varastoa. Artifactory hoitaa kaiken liikenteen eri Artifactory-palveluiden välillä. Virtual-varasto on nimensä mukaisesti virtuaalinen varasto. Virtuaaliseen varastoon voidaan kerätä yhden varaston alla useampia erityyppisiä varastoja. Tällöin käyttäjä pystyy käyttämään ainoastaan yhtä varastoa, joka taas viittaa sisäisesti useampaan paikkaan. Tämä nopeuttaa ja helpottaa Artifactory-palvelun käyttöä, kun ei tarvitse kaikkia eri varastoja erikseen muistaa, vaan riittää, että muistaa virtuaalivaraston nimen. Tämä tietysti vaatii, että virtuaalivarasto on konfiguroitu oikein.

Artifactory-palveluun luotiin seuraavat varastot:

- local
    - ext-gbad-commercial-release-local
    - ext-gbad-commercial-snapshot-local
    - ext-gbad-opensource-release-local
    - ext-gbad-opensource-snapshot-local
    - gbad-installation-local
    - gbad-integration-local

    - gbad-release-local
    - gbad-snapshot-local
- remote
    - ic2p-bundle-repo
    - ic2p-common-repo
    - ic2p-developer-repo
    - ic2fdf-releases
    - ic2fdf-snapshots
- virtual
    - gbad-all-virtual, viitaa gbad-virtual ja ic2p-virtuaali varastoihin
    - gbad-virtual viittaa paikaillisiin gbad varastoihin
    - ic2p-virtual viittaa kaikkii ic2p remote varastoihin

Paikallisten varastojen rakenne on selkeä ja nimi kertoo niiden käyttö tarkoituksen. Ext-alkuiset varastot ovat ulkoisia komponentteja varten ja ne ovat jaettu kaupallisiin ja open source -komponentteihin. Release-varastoihin tulee kyseisten komponenttien julkaistut versiot ja snapshot-varastoihin kehitysaikaiset versiot. Gbad-installation-local on paikallinen varasto, johon julkaistaan eri tuotteiden asennusmediat, kuten RPM-paketit, add-on-paketit ja windows installer -paketit. gbad-integration-local on varasto, johon julkaistaan CI-prosessin integration-tasolla integroidut paketit. gbad-snapshot-local on varasto johon julkaistaan snapshot-käännökset eri tuotteista. Integration- ja snapshot -pakettien ero on se, että integration varastoihin tallennetaan kattavamman integrointi-prosessin läpi kulkeneet tuotokset. Integration-töitä ajetaan ajastetusti kerran vuorokaudessa, kun taas snapshot-työt ajetaan versionhallintakyselyjen perusteella. Integration-varastoon tallennetaan kattavammat testiraportit ja Javadoc-dokumentit kuin snapshot-varastoon. gbad-release-local on varasto, johon tallennetaan release-töiden tuotokset lukuun ottamatta asennusmedioita, jotka menevät gbad-install-local-varastoon. Tänne tallentuvat kaikki muut julkaisun tulokset.

Artifactory-palvelun ic2p remote -varastot ovat toisella palvelimella sijaitsevan artifactory-palvelun virtuaalivarastoja. Niiden käyttö tapahtuu aivan samalla tavalla kuin paikallisten varastojen.

Artifactory-palveluun luotiin kolme virtuaalivarastoa. Näillä niputettiin olemassa olevia local- ja remote-varastoja yhden nimen alle. gbad-virtual pitää sisällään kaikki gbad-

local-varastot ja ic2-virtual pitää sisällään kaikki ic2p-virtuaalivarastot. Lisäksi luotiin kolmas virtuaalivarasto gbad-virtual-all, joka pitää sisällään kaksi muuta virtuaalivarastoa.

4.2.5. Muut tukipalvelut

Näiden tulipalveluiden tarkoitus on helpottaa kehitys- ja integrointiympäristössä tehtävää kehitystyötä. Tässä luvussa ei esitellä käytettäviä tukisovelluksia tarkasti, vaan mainitaan lähinnä niiden nimet ja käyttötarkoitukset.

4.2.5.1. Bugzilla-vikatietokanta

Kehitys- ja integrointiympäristössä käytetään vikatietokantana Bugzillaa. Vikatietokanta on paikka, johon kaikki kehityksessä ja testauksessa havaitut ohjelmistovirheet kirjataan. Palvelun tarkoitus on kerätä kaikki ohjelmistovirheet yhteen paikaan ja antaa ne kaikkien saataville. Bugzilla-palvelusta on myös liitos Jenkins-palveluun. Tämän liitoksen avulla Bugzilla-palveluun kirjatut virheet saadaan näkyviin Jenkins-palvelussa.

4.2.5.2. Testlink-testauksenhallintatyökalu

Testauksenhallintaan on kehitys- ja integrointiympäristöön asennettu Testlink-palvelu. Testlink-palvelulla pystytään organisoimaan testausta. Palveluun kerätään tietoa siitä, mitä on testattu milläkin ohjelmistoversiolla.

4.2.5.3. Sonar-analysointityökalu

Kehitys- ja integrointiympäristössä otettiin käyttöön Sonar-palvelu. Sonar-palvelu on tarkoitettu lähdekoodin analysointiin. Sonar-palvelulla pystytään tekemään monenlaista analysointia. Vakioasennuksen jälkeen mukana tulevat seuraavat sääntö-lisäosat: Checkstyle, Findbugs, PMD ja sonarway. Näitä sääntö-lisäosia löytyy myös muita ja niitä pystyy ottamaan käyttöön oman käyttötarpeen mukaan. Kyseiseen kehitys- ja integrointiympäristöön otettiin käyttöön asennuksen mukana tuleva vakioprofiili nimeltä Sonarway with findbugs. Sonar suorittaa pääosin staattista analyysiä analysoitavan ohjelmiston lähdekoodille.

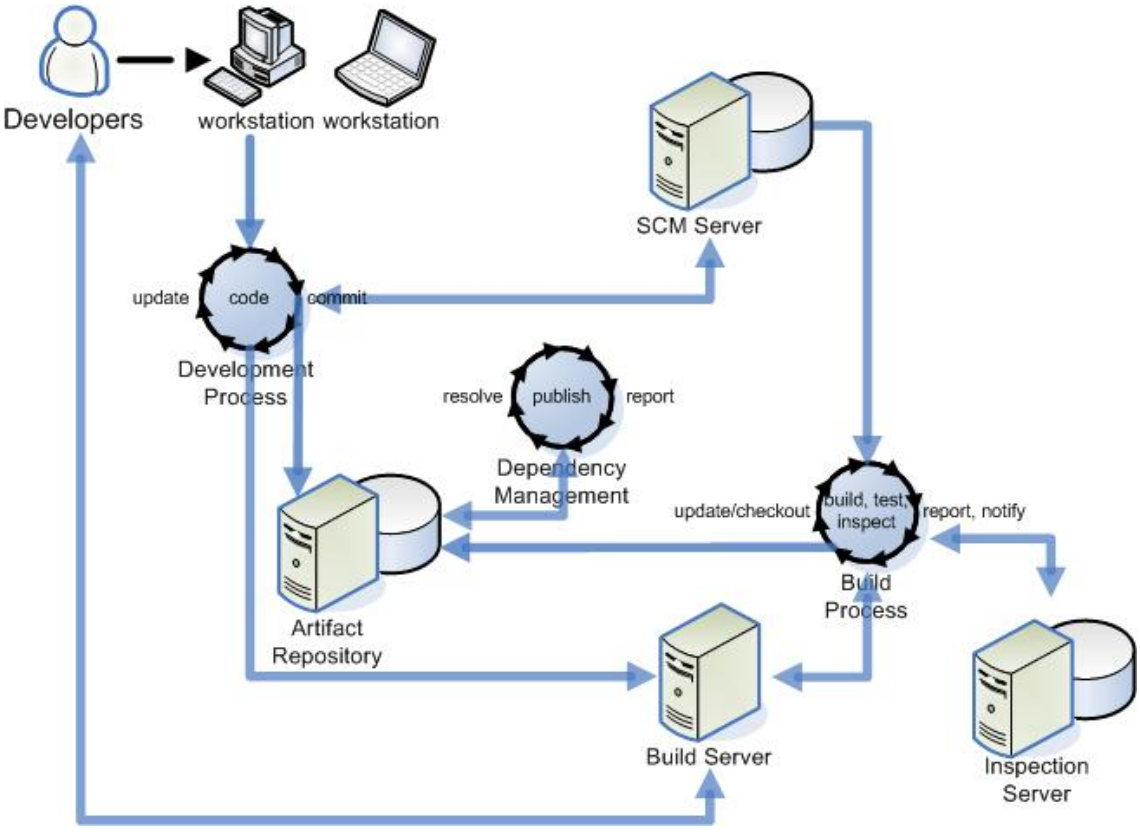
5. Kehitys- ja integrointiympäristön kokonaisuuden toiminnallinen kuvaus

Edellisissä luvuissa kuvatuista palveluista muodostuu kokonaisuus, jota kutsutaan kehitys- ja integrointiympäristöksi. Tätä ympäristöä pystyy parhaiten kuvaamaan sen toiminnan kautta. Tässä luvussa on kuvattu muutaman sekvenssikaavion avulla erilaisia tilanteita kehitys- ja integrointiympäristöstä.

Kehitys- ja integrointiympäristön integrointityöt jaetaan kolmeen luokkaan:

- Snapshot
- Integration
- Release

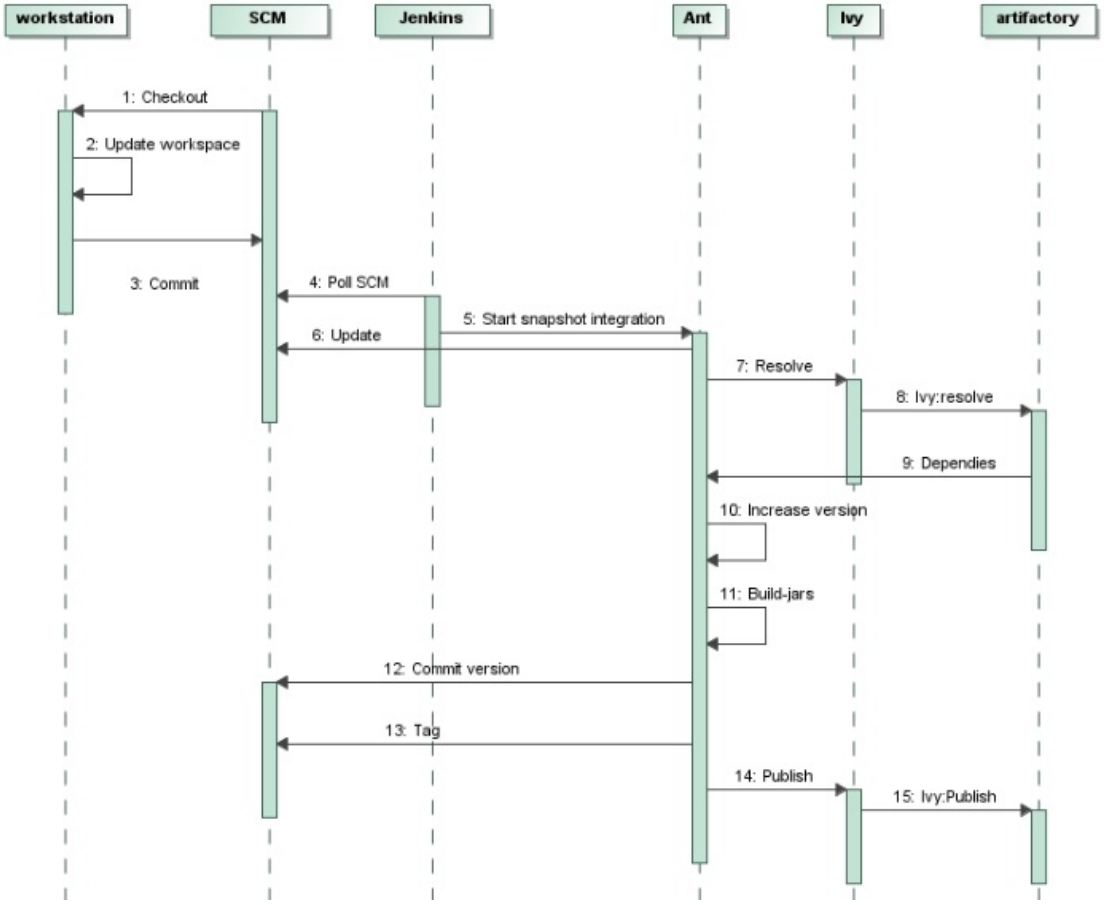
Kuviossa 8 on kuvattu ympäristöä, jossa kaiken tasoiset integroinnit tapahtuvat.



KUVIO 8. Integrointiympäristön toimijoiden väliset suhteet (Helin, Peltola & Hietaranta 2009, 10, muokattu).

Kuvion 8 tarkoitus on selvittää kehitys- ja integrointiympäristön toimijoiden väliset riippuvuudet integrointiprosessissa. Tärkeimmät asiat kuvassa ovat nuoliympyröillä kuvatut prosessit. Näiden prosessien syötteitä ja tuloksia on kuvattu nuolilla ympyrään tai ympyrästä ulos. Jos nuoli on 2-suuntainen, se tarkoittaa, että kyseinen prosessi saa toimijalta syötteitä ja antaa toimijalle syötteitä. Esimerkkinä voidaan käyttää kuvion Build Process -nuoliympyrää. Tämä prosessi käynnistetään Build Server -toimijalta ja se antaa tuloksia samalle toimijalle. Prosessin aikana käytetään Artifact Repository -, Inspection Server -, ja SCM Server -toimijoita. Versiohallintajärjestelmästä haetaan koostettavan lähdekoodiprojektin lähdekoodit. Artifactory-palvelua pyydetään ratkaisemaan riippuvuudet. Inspection Server -toimija tekee lähdekoodille staattista analyysia ja säilöö näitä tuloksia myöhempää käyttöä varten. Seuraavaksi käydään läpi tarkemmin mitenkä eri tasojen integroinnit käyttävät kyseistä ympäristöä.

Snapshot-integrointi on osuus, joka toteuttaa ympäristön jatkuvan integroinnin. Tämä tarkoittaa sitä, että sitä suoritetaan kaikista useimmin ja se on kaikista suppein. Snapshot-töiden tarkoitus on jatkuvan integroinnin periaatteiden mukaisesti integroida pieni muuttunut osa mahdollisimman aikaisessa vaiheessa suurempaan kokonaisuuteen ja antaa palaute integroinnista muutoksen tekijälle. Kuviossa 9 on esitetty snapshot-integrointia sekvenssikaaviona.



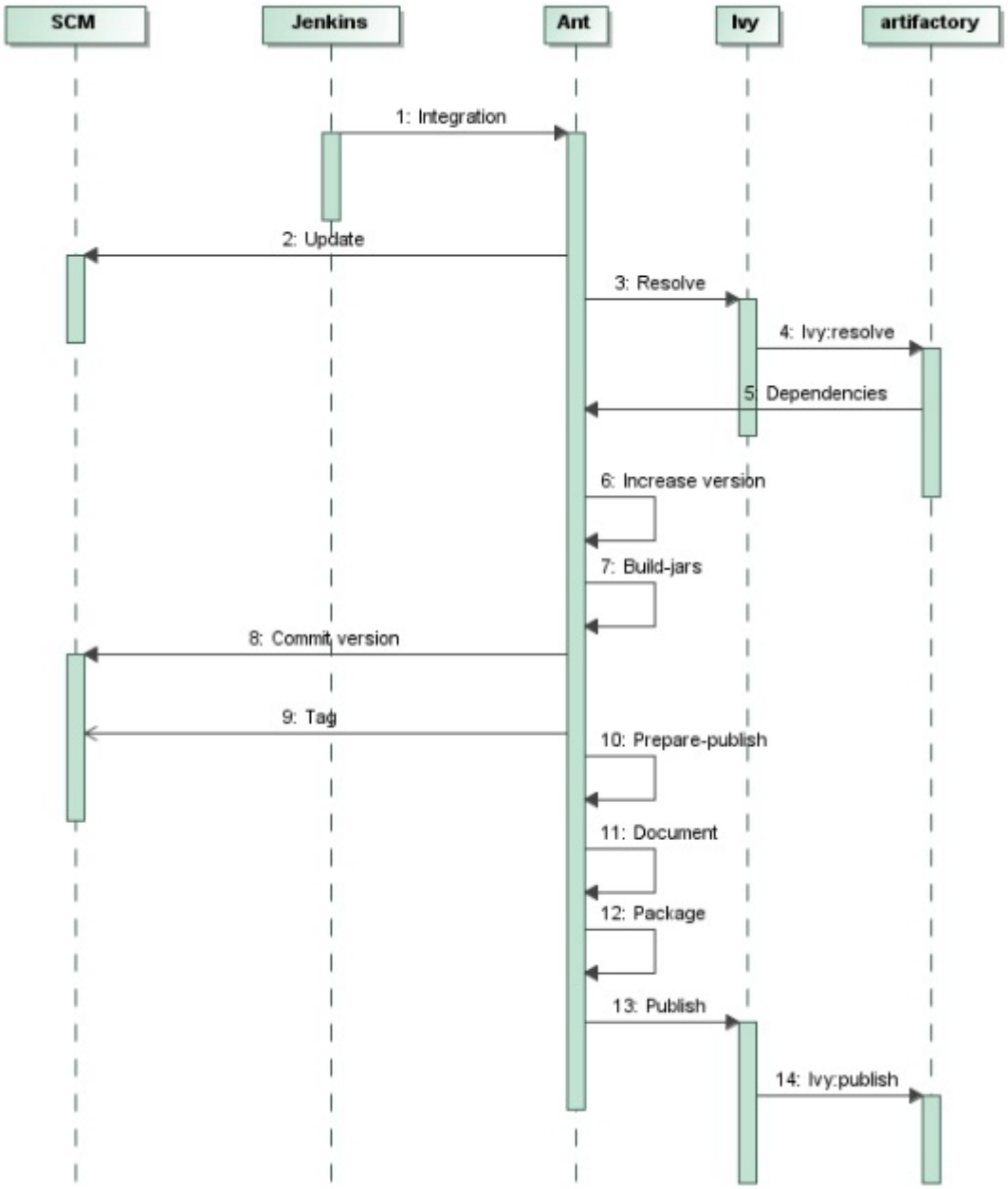
KUVIO 9. Snapshot-integraatio sekvenssikaaviona

Kuviossa 9 on esitetty tarkka kuvaus toimenpiteistä, jotka tehdään snapshot-integroinnin yhteydessä. Snapshot-integraation luonteen vuoksi kuvioon 9 on kuvattu aluksi tilanne, jossa kehittäjä muuttaa jonkin projektin lähdekooditiedostoja. Käytännössä Jenkins-integrointipalveluun luodut projektien snapshot-integrointityöt tarkkailevat kokoajan versionhallintaa omien lähdekooditiedostojen osalta. Kun Jenkins-integrointipalvelu havaitsee, että versionhallinnassa on tapahtunut muutos lähdekooditiedossa, se käynnistää Ant-työkalun ja kutsuu tälle snapshot-nimistä Ant-kohdetta. Tämä Ant-kohde suorittaa nopean ja suppean integroinnin muuttuneelle osalle. Tämä tarkoittaa sitä, että aluksi ratkaistaan työn riippuvuudet, jonka jälkeen koostetaan ja käännetään konfiguraatioyksikkö.

Koostamisen ja kääntämisen jälkeen uusi versio viedään versionhallintaan ja sille annetaan tag-versiotieto. Tätä tag-versiotietoa voidaan käyttää myöhemmin identtisen version luomiseen projektista. Viimeiseksi Ant-työkalu julkaisee integroinnin tuloksen Artifactory-riippuvuuksienhallintavarastoon. Tämä tehdään Ivy-riippuvuustyökalun avulla.

Huomion arvoista snapshot-tason integroinnissa on se, että siinä ei tehdä ollenkaan Javadoc-dokumentaatiota tai suoriteta yksikkötestejä. Virheitä tarkkaillaan ainoastaan Java-kääntäjän virheistä. Näin toimitaan, jotta koko snapshot-integrointiprosessi saadaan pidettyä nopeana ja kevyenä toimenpiteenä. Snapshot-integrointiprosessi ei saa olla raskas, koska sitä tehdään useille projekteille useita kertoja päivässä.

Snapshot-integroinnin suppeuden vuoksi tarvitaan integration-tasoksi nimetty integrointi. Tämä integrointi suoritetaan ajastetusti joka projektille kerran vuorokaudessa. Käytännössä tämä integrointi lisää edellä kuvattuun snapshot-integrointiin yksikkötestauksen ja javadoc-dokumentaation luomisen. Kuviossa 10 on kuvattu tämä kokonaisuus sekvenssikaaviolla.

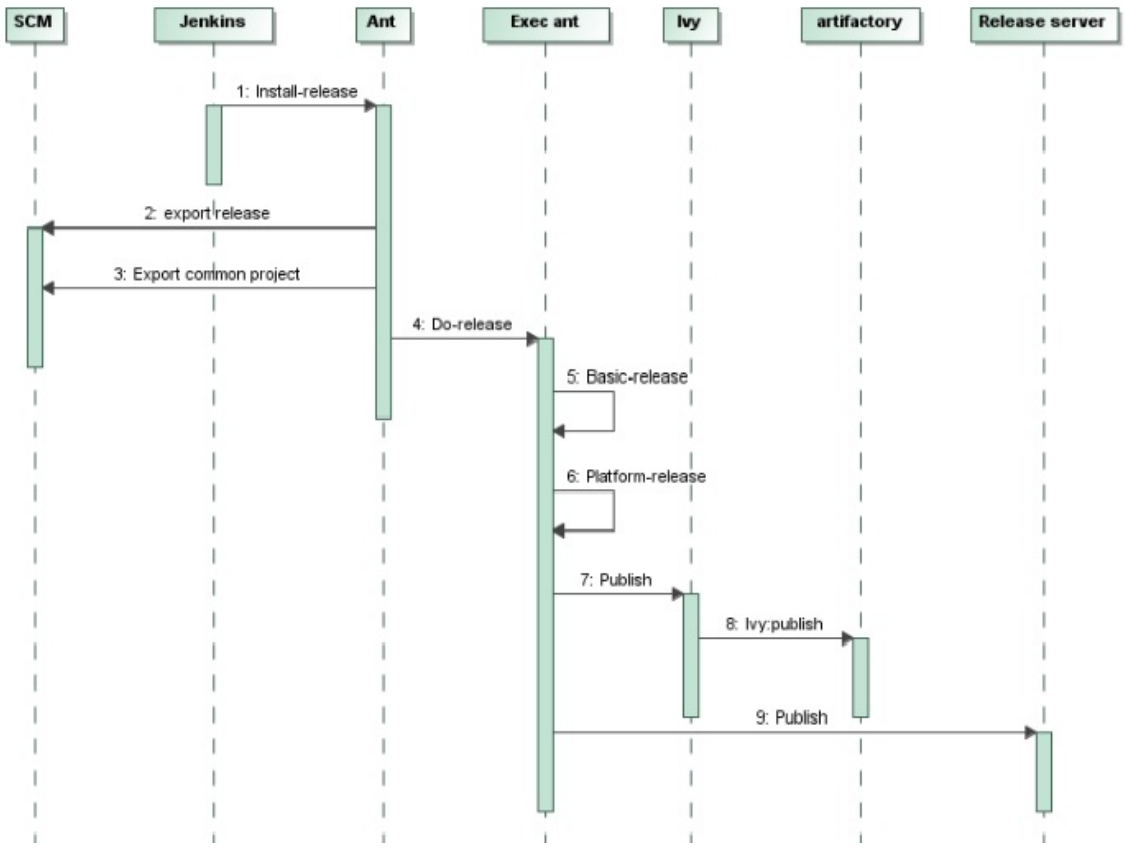


KUVIO 10. Integraation sekvenssikaavio

Erona kuvion 9 ja kuvion 10 välillä on se, että ennen kuin jar-paketit julkaistaan Artifactory-riippvuuksienhallintavarastoon, suoritetaan Javadoc-dokumenttien generointi ja yksikkötestien suorittaminen. Nämä toimenpiteet tehdään prepare-publish-nimisessä Ant-kohteessa versionhallintaan viemisen jälkeen. Nämä luodut dokumentit ja yksikkötestien tulokset pakataan ja viedään publish-toimenpiteen yhteydessä Artifactory-riippuvuuksienhallintavarastoon myöhempää käyttöä varten.

Kolmatta integrointitasoa kehitys- ja integrointiympäristössä kutsutaan nimellä release-integrointi. Tämä on kolmesta tasosta kaikista kattavin ja samalla myös raskain ja hitain

suorittaa. Release-integrointia ei suoriteta jatkuvasti tai automaattisesti, vaan se on tarkoitettu nimensä mukaisesti julkaisemiseen. Tämän tason tehtävässä tuotetaan kaikki tarvittavat tiedostot, jotta kyseisestä projektista voidaan toimittaa asennuspaketti asiakkaalle. Kuviossa 11 on kuvattu release-integrointia sekvenssikaaviolla.



KUVIO 11. Release-integraatio

Kuviossa 11 kuvataan julkaisutoiminnon toimenpiteitä kehitys- ja integrointiympäristössä. Tässä kuvassa on nostettu abstraktiotasoa, eikä siinä kuvata yksityiskohtaisesti kaikkia julkaisuun liittyviä toimenpiteitä. Julkaisu tapahtuu aina manuaalisesti käyttäjän toimesta. Julkaisu käynnistetään Jenkins-integrointipalvelusta. Julkaisun käynnistyessä Jenkins käynnistää Ant-työkalun ja kutsuu tuolle työkalulle install-release-nimistä kohdetta. Install-release-kohde on kokoelma muita Ant-kohteita ja tuo kokonaisuus hoitaa koko julkaisun.

Ensimmäisenä Ant-työkalu käy hakemassa versiohallinnasta julkaistavan tuotteen lähdekooditiedostot ja COMMON_PROJECT-tukiprojektin. Nämä tiedostot haetaan versionhallinnasta ns. puhtaina. Tämä tarkoittaa sitä, että mitään Jenkins-integrointipalvelun työhakemistossa valmiiksi olevia tiedostoja ei käytetä, vaan ver-

sionhallinnasta otetaan kaikki tiedostot täydellisenä. Erona tässä versiohallintatoimenpiteessä on myös se, että versionhallinnasta ei oteta tuoreinta tietoa, vaan haluttu symbolisen nimen mukainen versio eli tag-versio.

Lähdekooditiedostojen ja COMMON_PROJECT-tukiprojektin hakemisen jälkeen, Jenkins-integrointipalvelun käynnistämä Ant-työkalu käynnistää itsestään uuden suorituksen. Tätä uutta Ant-sovellusta kutsutaan kuviossa 11 nimellä Exec Ant. Tämä Ant-työkalun uudelleen käynnistäminen tehdään, jotta varmistutaan siitä, että Ant-työkaluun ei ole jäänyt mitään vanhoja muuttujatietoja.

Uusi Ant-työkalu käynnistää vasta varsinaisen julkaisun. Nämä toimenpiteet tehdään basic-release- ja platform-release-nimisissä Ant-kohteissa. Basic-release-niminen Ant-kohde tekee samoja toimenpiteitä kuin aiemmin kuvattu Integration-tason integrointi. Näihin toimenpiteisiin kuuluu seuraavia asioita: riippuvuuksien ratkaiseminen, koostaminen, kääntäminen, jar-pakettien luonti, yksikkötestien suorittaminen ja javadoc-dokumentaation luominen.

Plarform-release-nimisen Ant-kohteen tehtävä on hoitaa kaikki käyttöjärjestelmäriippuvaiset tehtävät. Ensimmäisenä platfrom-release-tehtävä tarkistaa, mikä on käyttöjärjestelmä, jossa tällä hetkellä toimitaan. Käyttöjärjestelmän ollessa Linux, Ant-kohde aloittaa RPM-paketin luomisen. RPM-paketin luomiseen käytetään basic-release-kohteessa käännettyjä tiedostoja. RPM-paketin luomisen jälkeen kohde käyttää tuota RPM-pakettia Add-on-asennusmedian luomiseen. Käyttöjärjestelmän ollessa Windows, Ant-kohde luo jar-paketin, johon pakataan käännetyt tiedostot Windows-käyttöjärjestelmän vaatimalla tavalla.

Lopuksi julkaisun tuotokset julkaistaan Ivy-riippuvuustyökalun kautta Artifactory-riippuvuuksienhallintavarastoon. Artifactory-riippuvuuksienhallintavarastoon julkaistaan ainoastaan platform-release-kohteen tuottamat tiedostot. Tässä tapauksessa toimitaan näin, koska alun perin tämä julkaisu tehtiin versiosta, joka oli jo aikaisemmin käännetty ja julkaistu Artifactory-palveluun.

Viimeisenä toimenpiteenä Jenkins-integrointipalvelu lähettää sähköpostin määrätyille ihmisille julkaisun lopputuloksesta jatkuvan integraation periaatteiden mukaisesti.

6. Varmuuskopiointi

Kehitys- ja integrointiympäristöllä on tiukat vaatimukset varmuuskopiointiin. Nämä vaatimukset johtuvat ympäristössä kehitettävien ohjelmistojen luonteesta. Suurin osa näistä vaatimuksista tulee suoraan asiakkailta. Jotta nämä kaikki varmuuskopiointivaatimukset täytetään, on varmuuskopiointiin jouduttu kiinnittämään erityistä huomiota ja tarkkaan määrittelemään varmuuskopioinnin toteutus.

Varmuuskopiointia suoritetaan kolmessa tasossa.

- Palvelimien yleiset varmuuskopioinnit
- Palveluiden varmuuskopioinnit
- Julkaisujen varmuuskopioinnit

Palvelimien yleinen varmuuskopiointi on hoidettu yrityksen ICT-tukipalveluiden toimesta. Palvelimet on rakennettu VMware-virtualisointijärjestelmän päälle ja tämä helpottaa varmuuskopioiden ottamista palvelimista. Palvelimien varmuuskopiot otetaan arkipäivisin aamuyöllä. Ajankohdaksi valittiin aamuyö, koska varmuuskopioinnin piiriin haluttiin aina edellisen päivän tuotokset kokonaisina. Varmuuskopiot tallennetaan siten, että normaalisti varmuuskopio tallennetaan kuvana kiintolevylle ICT-tukipalveluiden toimesta. Näitä kovalevylle menneitä varmuuskopiota säilytetään erikseen sovittu aika. Osa varmuuskopiosta menee kuitenkin ns. nauhavarmistukseen. Tämä nauhalle menevä varmuuskopio menee pidempään säilytykseen kassakaappiin.

6.1. Varmuuskopion ottaminen

Varmuuskopio otetaan palvelimista snapshot-tyylisesti. Tämä tarkoittaa sitä, että palvelimen sen hetkisestä tilasta otetaan ”kuva”. Tämä kuva voidaan palauttaa palvelimelle haluttaessa milloin vain. Tämän tyylinen varmuuskopiointi aiheuttaa tiettyjä haasteita erityisesti tilanteissa, joissa varmuuskopioitavalla tiedolle voi tapahtua muutoksia. Näitä haasteita voi verrata valokuvan ottamiseen kameralla. Liikkuvasta kohteesta valokuvan ottaminen voi olla haastavaa. Pyrimme tarkoissa kuvissa ottamaan kuvan mahdollisimman paikoillaan olevista kohteista. Yleensä valokuvaan jää liikkeen seurauksena jotain häiriötä. Samalla tavalla snapshot-varmuuskopiointi ”kuvan” ottamisen yhteydessä kaikki liike voidaan tulkita huonoksi asiaksi. Liike tässä tapauksessa voi olla mitä tahansa levylle kirjoittamista, lukemista tai tiedon muuttamista. Mikäli levyllä tapahtuu

liikettä kuvanotto hetkellä, voi se aiheuttaa häiriöitä tai tiedon korruptoitumista. Tätä liikettä tapahtuu pääsiassa eri palveluiden, tietokantojen tai versionhallintojen tiedoissa. Tämän takia jouduttiin määrittelemään palveluiden varmuuskopiointi erikseen.

Jotta kaikkien palveluiden tiedot saadaan vakaassa tilassa varmuuskopioitua, piti näiden palveluiden tiedot ottaa erikseen talteen ja pakata ennen varsinaista palvelimen varmuuskopion ottamista. Tämä tarkoittaa sitä, että ennen jokaista ICT-tukipalveluiden ottamaa varmuuskopiota suoritamme palvelimilla varmuuskopiointitoimenpiteet. Näissä toimenpiteissä kopioimme ja pakkaamme kaikki tärkeimpien palveluiden tiedot. Nämä tiedot tallennetaan palvelimen kovalevylle, jolloin kun koko palvelimen varmuuskopio otetaan, menee tämä pakattu paketti samaan kuvaan vakaassa tilassa. Taulukossa 2 on esitetty erikseen pakattavat palvelut ja niiden sijainnit.

TAULUKKO 2. Varmuuskopioitavat palvelut

Palvelu	Sijainti levyllä	Kopionti tapa
SVN	/srv/svn/HANKE1	svnadmin hotcopy & tar
	/srv/svn/HANKE2	svnadmin hotcopy & tar
CVS	/srv/cvs/HANKE1	cp & tar
Käynnistystiedostot	/etc/init.d	cp & tar
MySQL-tietokannat	MySQL:artifactory	mysqldump
	MySQL:jenkins	mysqldump
	MySQL:testlink	mysqldump
Jenkins	/srv/jenkins	cp & tar
Artifactory	/srv/artifactory	cp & tar
Sonar	/opt/sonar/sonar-2.13.1	cp & tar
Tomcat	/srv/tomcat/jenkins	cp & tar
	/srv/tomcat/sonar	cp & tar
	/srv/tomcat/artifactory	cp & tar

Taulukossa 2 on esitetty myös kopiointitapa. Kaikki tiedot SVN-varastoja ja MySQL-tietokantoja lukuun ottamatta kopioidaan tavallisella cp-komennolla. Kopioinnin jälkeen kopioitu hakemisto pakataan tar-työkalulla. SVN-varastot kopioidaan svnadmin hotcopy -komennolla. Tätä komentoa käytettään tässä yhteydessä, koska se mahdollistaa versionhallinnan käytön samaan aikaan. Muillakin komennoilla päästäisiin samaan lopputulokseen, mutta ne vaatisivat versionhallinnan käytön lopettamisen kopioinnin ajaksi. MySQL-tietokannat kopioidaan komennolla mysqldump. Kyseinen komento kopio kaiken tietokannan sisällön .sql päätteiseen tiedostoon, joka pystytään tarvittaessa lukemaan tietokantaan. Lopuksi kaikista kopioiduista tiedoista pakataan yksi paketti, joka pakataan tar- ja gzip-työkaluilla. Tämä on se paketti, joka kopioidaan palvelimen varmuuskopioinnin yhteydessä talteen.

Palveluiden varmuuskopioinnin ajastus on hoidettu käyttämällä crontab-työkalua. Kaikki palveluiden varmuuskopiointitoimenpiteet on kirjoitettu tiedostoon. Tämä tiedosto on asetettu crontab-työkalulla suoritettavaksi seuraavilla asetuksilla:
0 2 * * 2-6 /opt/sysman/server1-backup.sh > /dev/null 2 >&1
Tämä tarkoittaa sitä, että /opt/sysman/server1-backup.sh -varmuuskopiotiedosto suoritetaan tiistaisin, keskiviikkoisin, torstaisin, perjantaisin ja lauantaisin kello 02:00.

Varmuuskopioinnin valvonnan parantamiseksi varmuuskopion ottamisen jälkeen lähetetään sähköpostiviesti ympäristön ylläpitäjille. Sähköpostiviestissä kerrotaan varmuuskopion onnistuneesta ottamisesta, sekä annetaan tarkat tiedot kestosta, koosta ja lokeista. Tämä helpottaa varmuuskopiontiprosessin seuraamista.

Julkaisujen varmuuskopiointi tapahtuu myös ICT-tukipalveluiden toimesta. Kaikki julkaisut kopioidaan verkkolevylle yleiseen ohjelmistokehitysryhmän käyttöön. Tämä verkkolevy on ICT-tukipalveluiden yleisen varmuuskopioinnin piirissä. Tämä varmuuskopiointi on nauhavarmistuksen piirissä.

6.2. Varmuuskopion palauttaminen

Varmuuskopion palauttamisen hoitaa yrityksen ICT-tukipalvelut. Palauttaminen hoidetaan WMware-virtualisointijärjestelmän avulla. Tämä tarkoittaa sitä, että varmuuskopiosta perustetaan virutalisointijärjestelmään uusi virtuaalipalvelin. Jos varmuuskopiosta halutaan palauttaa ainoastaan jokin tietty osa, perustetaan varmuuskopiosta uusi virtuaa-

lipalvelin. Tältä uudelta virtuaalipalvelimelta kopioidaan halutut osat palautusta vaativalle palvelimelle. On myös mahdollista palauttaa koko varmuuskopio kerrallaan palvelimen tilalle. Tämä tarkoittaa, että palvelin palautetaan täsmälleen samaan tilaan, kun se on ollut varmuuskopion ottamisen aikaan.

7. POHDINTA

Työn tavoitteena oli suunnitella ja toteuttaa jatkuvaa integraatiota suorittava kehitys- ja integrointiympäristö. Kehitys- ja integrointiympäristön vaatimukset pyrittiin määrittelemään riittävän tarkasti tiedossa olevien parannuskohtien ja ongelmien kautta.

Kehitys- ja integrointiympäristöä lähdettiin rakentamaan yrityksen ICT-tukipalveluiden tarjoaman WMware-virtualisointijärjestelmän päälle. Jatkuvan integraation periaatteet pidettiin kokoajan mielessä ja kaikissa ratkaisuissa pyrittiin ottamaan ne huomioon parhaalla mahdollisella tavalla.

Työlle asetetut tavoitteet saavutettiin, koska lopputuloksena saatu kehitys- ja integrointiympäristö täyttää sille annetut vaatimukset. Tavoitteeseen päädyttiin yritys ja erehdys -taktiikalla. Tätä helpotti suuresti se, että kun kehitys- ja integrointiympäristön toteutus aloitettiin, sen käyttöönotolla ei ollut kiire. Kehityksen aikana käytössä oli vanha kehitys- ja integrointiympäristö. Näin ollen uuden ympäristön palvelimia ja palveluita ehti rauhassa toteuttamaan. Jos jokin ratkaisu ei ollut hyvä, pystyi helposti sen poistamaan ja kokeilemaan jotakin toista ratkaisua tilalle. Myös se, että kehitys- ja integrointiympäristö otettiin käyttöön porrastetusti, helpotti asiaa.

Käyttöönotto ympäristöllä tapahtui helmikuussa 2012. Käyttöönotto sujui ilman suurempia ongelmia, koska kaikkien palveluiden käyttöönotto oli testattu etukäteen. Käyttöönotto tapahtui käytännössä yhden päivän aikana. Seuraavan viikon aikana kehitystyö sujui jo normaaliin tapaan uudessa ympäristössä.

Tätä työtä tehdessä oppi valtavasti uutta asiaa. Tietämys kasvoi kahdelta alueelta erityisesti: integrointi ja siihen liittyvät asiat sekä yleinen Linux-palvelintietämys. Linux-tietämys kasvoi erilaisten palveluiden ja palvelimien konfiguroinnissa.

Kehitys- ja integrointiympäristöä on tarkoitus kehittää jatkossa suorittamaan myös continous release -toiminallisuutta. Tällä tarkoitetaan esimerkiksi julkaistavien ohjelmistojen automaattista asennusta päätelaitteille. Tämä toiminnallisuus on pidemmälle vietyä jatkuvaa integrointia, jonka pitäisi entisestään ennalta ehkäistä kaikkia integrointiin liittyviä ongelmia.

LÄHTEET

Apache Ant Wiki. 2012. Apache Ant -työkalun käyttäjien keräämiä ohjeita. Luettu 6.4.2012. http://wiki.apache.org/ant/

Apache Ivy Tutorial. 2012. Apache Ivy -työkalun käyttöohje. Luettu 6.4.2012. http://ant.apache.org/ivy/history/latest-milestone/tutorial/start.html

Apache Tomcat Wiki. 2012. Apache Tomcat -työkalun yleisiä käyttöohjeita. Luettu 6.4.2012. http://tomcat.apache.org/tomcat-6.0-doc/index.html

Artifactory Wiki. 2012. Artifactory-palvelun yleisiä käyttöohjeita. Luettu 6.4.2012. http://wiki.jfrog.org/confluence/display/RTF/Artifactory+User+Guide

Collins-Sussman, B. Fitzpatrick, B. & Pilato, M. Version Control with Subversion: For Subversion 1.5: (Compiled from r3305). Luettu 6.4.2012. http://ebooksgo.org/computer/svn-book.pdf

Duvall, P. Matyas, S. & Glover, A. 2007. Continuous integration improving software quality and reducing risk. 4. painos. Boston: Pearson Education, Inc

Helin, M. Peltola, T & Hietaranta, E. 2009. Kehittäjän opas. Insta DefSec Oy sisäinen dokumentaatio.

Korhonen, J. Heinonen, T. Helin, M. & Peltola, P. Integrointiympäristö-, linux-, testaus- ja labrakoulutus. Insta DefSec Oy sisäinen dokumentaatio.

Smart, J. 2011. Jenkins: The Definite Guide. 1. painos. Sebastol: O`Reilly Media, Inc