KÄÄNNÖS- JA INTEGROINTIYMPÄRISTÖ

Java-ohjelmistotuotteelle

LAHDEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2009
Jaakko Vähäniitty

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

JAAKKO VÄHÄNIITTY: Käännös- ja integrointiympäristö Java-ohjelmistotuotteelle

Ohjelmistotekniikan opinnäytetyö, 46 sivua

Kevät 2009

TIIVISTELMÄ

Tämä työ tehtiin Oy L M Ericsson Ab:lle suorittamaan ENIQ-nimisen tuotteen automatisoitua käännöstä ja jatkuvaa integrointia. ENIQ on sovellus, joka kerää tietoa tietoliikenneverkoista näiden optimointia ja raporttien koostamista varten. Tuotteen ydin on toteutettu Javalla, ja se perustuu modulaariseen arkkitehtuuriin. Työtä varten tutkittiin kolmea erilaista käännöstyökalua: Make-, Ant- ja Maven-ohjelmia sekä CruiseControl- ja Build Forge -työkaluja, jotka toteuttavat jatkuvaa integrointia.

Suuret ja monimutkaiset ohjelmistot ovat tuotteenhallinnan kannalta haastavia kehittää ja ylläpitää. Tähän haasteeseen on kehitetty ketterät ohjelmointimenetelmät, joiden tarkoituksena on parantaa kehitystyön ja sitä myötä tuotteen laatua ja tuottavuutta. Eräitä tällaisia menetelmiä ovat automatisoitu käännösprosessi ja jatkuva integrointi.

Automatisoidulla käännöksellä tarkoitetaan prosessia, jossa lähdekoodia sisältävät tiedostot käännetään ja linkitetään toimivaksi ohjelmaksi. Tähän prosessiin liitetään usein erilaisia verifiointimenetelmiä, kuten yksikkötestejä, ohjelman oikean toiminnan varmistamiseksi. Integrointi on ohjelman osan toiminnan verifioiminen yhdessä muiden osien kanssa. Menetelmää, jossa kehittäjät integroivat työtänsä usein, kutsutaan jatkuvaksi integroinniksi.

Työn toteuttamista varten valittiin käännöstyökaluksi Maven -ohjelmisto sen pitkälle viedyn käännöksen automaation, selkeän XML-syntaksin ja riippuvuuksien hallinnan vuoksi. Integroimistyötä suorittamaan valittiin CruiseControl, joka on vapaan lähdekoodin ohjelmisto, ja se on helppo käyttää sekä ylläpitää. Molempien työkalujen toimintakuntoon saattaminen vaati itse ohjelmien lisäksi erinäisiä XML -muotoisia asetustiedostoja, joissa määriteltiin ENIQ:n rakenne, ympäristö sekä muuta tietoa kääntämis- ja integrointiprosessiin liittyen.

Työn tuloksena syntynyt ympäristö toteuttaa ketterien menetelmien oppeja ja antaa välitöntä palautetta kehittäjille käännöksien onnistumisesta parantaen näin tuotteen ja sen kehitystyön laatua.

Avainsanat: Java, Kääntäminen, Jatkuva integrointi

Lahti University of Applied Sciences
Degree Programme in Information Technology

VÄHÄNIITTY, JAAKKO: Build and Continuous Integration Environment for Java software

Bachelor´s Thesis in Software Engineering, 46 pages

Spring 2009

ABSTRACT

This thesis was made for L M Ericsson Ab in order to create an automated build and continuous integration environment for ENIQ. It is a software product which collects data from a network for optimization and reporting purposes. The core of ENIQ is built using the Java programming language and modular architecture. Three different automated build tools were researched for this thesis: Make, Ant and Maven build tools. For continuous integration, the tools studied were CruiseControl and Build Forge.

Large and complicated software is challenging to develop and maintain from the product management point of view. Doctrines of extreme programming have been developed to meet these challenges. Automated build process and continuous integration are examples of these doctrines.

The automated build process involves compiling and linking of source code into an executable program. Different sorts of verification methods, such as unit testing, are included in this process, in order to test that the program works as it should. The integration process consists of a set of tests which check the correct interaction of two or more parts of the program. A doctrine where developers are supposed to integrate as often as possible is called continuous integration.

Maven and Cruise Control programs were chosen to create the automated build and continuous integration environment. Maven's automated build process, easy to understand XML format and dependency management were the key factors for choosing this tool. CruiseControl, on the other hand, is easy to manage and use. Creating the environment required installation of the previously mentioned tools, as well as a set of XML formatted configuration files. These files included all sorts of data about ENIQ and everyting related to its build and integration process.

The resulting environment builds and integrates ENIQ as is defined in the doctrines of automated build and continuous integration processes of extreme programming. This enhances the development processes and product management of ENIQ.

Key words: Java, build, continuous integration

SISÄLLYS

1 JOHDANTO

Ketterät ohjelmointimenetelmät ovat esitelleet useita uusia tekniikoita ja lähestymistapoja ohjelmistokehitykseen, kuten pariohjelmointi, testivetoinen ohjelmistokehitys ja jatkuva integrointi. Tarkoituksena on ollut raskaiden ja byrokraattisten prosessien muuttaminen joustaviksi ja lyhyiksi iteraatioiksi. Nopea reagointi jatkuviin muutoksiin on tärkeää ohjelmistokehityksessä, ja ketterät menetelmät ovat kehitetty vastaamaan tähän haasteeseen.

Työ tehdään Oy LM Ericsson Ab:lle, joka on maailman johtava telekommunikaatiolaitteiden ja näihin liittyvien palvelujen tarjoaja. Yritys työllistää maailmanlaajuisesti noin 65000 ihmistä, joista Suomessa työskentelee vajaa tuhat. Työ tehdään ENIQ:iä (Ericsson Network IQ) varten. Se on Ericssonin kehittämä sovellus, joka kerää tietoa tietoliikenneverkoista raportointia ja optimointia varten. (Suomen Ericsson 2009.)

Ericssonin ENIQ-tuotteen kehitysprosessit ovat siirtymässä käyttämään hyviksi havaittuja ketteriä menetelmiä. Yritys haluaa kehittää ENIQ:n tuotteen- ja laadunhallintaa ottamalla käyttöön jatkuvan integroinnin ohjelmistokehityksessä sekä tutkia erilaisia työkaluja kääntämisen automatisoimiseen. Ohjelmistojen kehitysvaiheessa löydetyt viat on helpompi ja halvempi korjata, mitä varten yritys haluaa tutkia edellä mainittuja teknologioita. Automatisoitu kääntäminen sekä jatkuva integrointi ovat työkaluja vikojen pikaiselle paikantamiselle.

Työn tavoitteena on tutkia erilaisia tekniikoita automatisoidun kääntämisen ja jatkuvan integroinnin toteuttamiseen sekä vertailla työkaluja, joilla nämä voidaan toteuttaa. Tarkoituksena on löytää yrityksen ja tuotteen tarkoituksiin sopivat työkalut automatisoidun käännöksen ja jatkuvan integrointiympäristön toteuttamiseksi. Tavoitteena on myös toteuttaa tämä ympäristö hyväksi havaituilla välineillä.

Automatisoidun käännöstyökalun tulisi kääntää Java lähdekoodia, suorittaa JUnit yksikkötestejä sekä olla helppokäyttöinen ja yhteensopiva valitun jatkuvaa integrointia suorittavan työkalun kanssa. Valmis ympäristö antaisi välitöntä palautetta kehittäjille tehtyjen muutoksien toimivuudesta erilaisten mittareiden avulla, kuten yksikkötestiraportit sekä kääntämisestä ja integroinnista saatu palaute. Tämän avulla virheet huomataan ja korjataan aikaisessa vaiheessa parantaen tuotteen ja kehitystyön laatua.

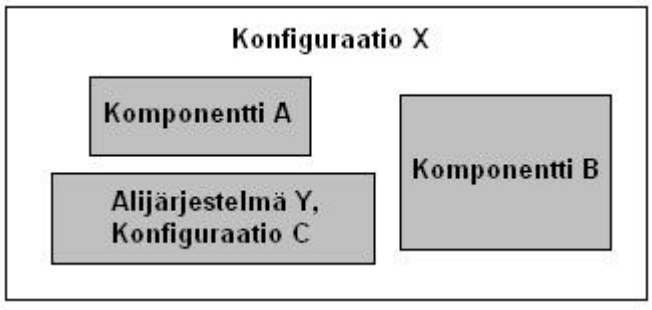
Työssä vertaillaan Make-, Ant- ja Maven-käännösohjelmia sekä CruiseControl- ja Build Forge -integrointityökaluja. Työn tuloksena syntyvä ympäristö käyttää siis joitakin näistä ohjelmista ENIQ:in automatisoidun käännöksen ja jatkuvan integroinnin suorittamiseen edellä mainittujen vaatimuksien mukaisesti.

2 TUOTTEEN HALLINTA

2.1 Tuotteenhallinnan osa-alueet

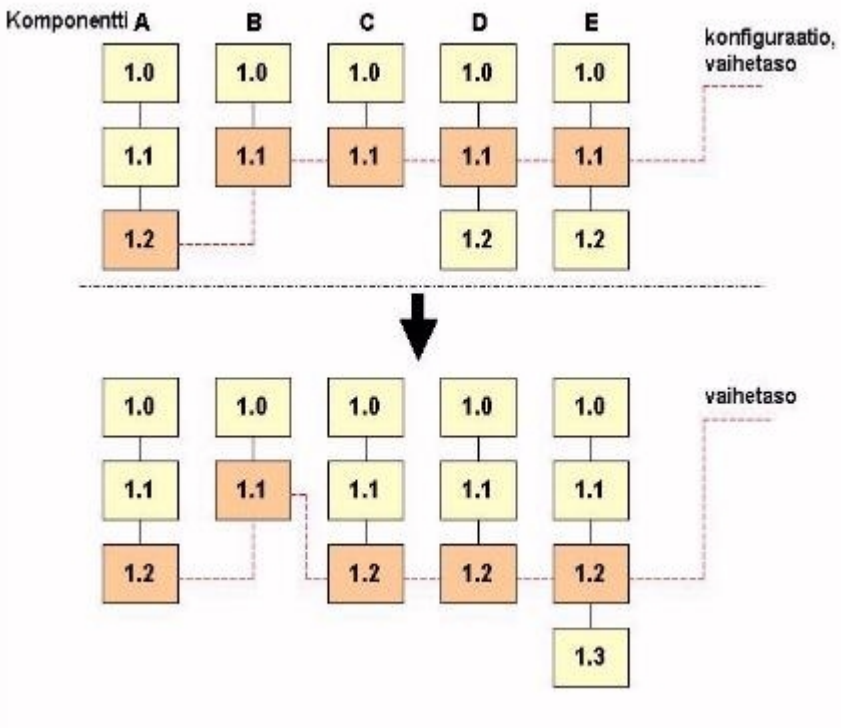
Ohjelmistotuote on kaikki toimivan ohjelman suunnitteluun, tekemiseen ja ylläpitämiseen vaadittavat osat, kuten lähdekoodia sisältävät ohjelmatiedostot, erilaiset vaatimusdokumentit ja testipetit. Näistä osista, komponenteista, kootaan suurempia kokonaisuuksia, joita kutsutaan konfiguraatioiksi. Tuotteen elinkaaren aikana sen komponentteihin ja konfiguraatioihin halutaan muutoksia, koska ne eivät ehkä toimi kuten pitäisi, tai kokonaan uusia ominaisuuksia ja toiminnallisuuksia halutaan lisää. Vanhojen osien jatkokehittämiseen saattaa tulla tarve, kuten myös asiakkaan ympäristö voi asettaa tuotteelle omat vaatimuksensa, mikä pitää ottaa huomioon komponenteissa tai konfiguraatioissa. Tuotteenhallinnan on tarkoitus hallinnoida kaikkea tätä. Se voidaan jakaa kolmeen osa-alueeseen, komponenttien-, konfiguraatioiden- ja muutoksienhallintaan. (Haikala & Märijärvi 2004, 255-256.)

2.2 Komponenttien ja konfiguraatioiden hallinta



KUVIO 1. Konfiguraatio ja komponentit (Haikala & Märijärvi 2004, 258)

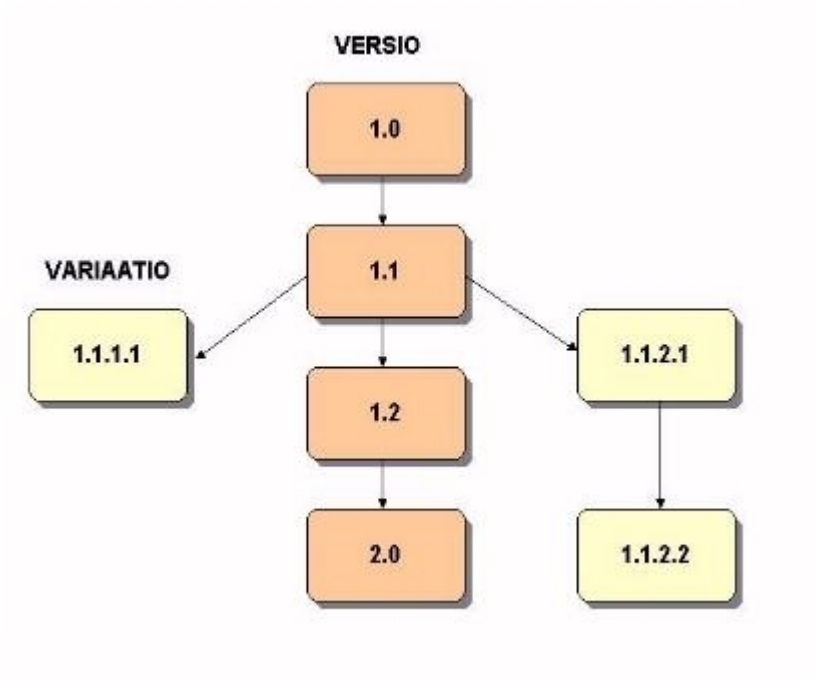
Komponentin voidaan sanoa olevan tuotteen perusyksikkö, joita ovat esimerkiksi ohjelmatiedostot ja dokumentit. Etenkin ohjelmatiedostojen osalta joitain komponentteja voidaan kutsua johdetuiksi komponenteiksi, jotka on lähdekoodia sisältävistä tiedostoista käännetty ohjelmatiedostoiksi. Konfiguraatiot ovat puolestaan näistä osista koostettuja kokonaisuuksia, eli toimivia ohjelmia tai sen osia. Komponentteja ja konfiguraatioita voidaan kutsua myös yhteisnimellä hallinta-alkio. Kuviossa yksi on kuvattu eräs konfiguraatio, joka koostuu kahdesta komponentista sekä toisesta konfiguraatiosta, joka on kyseisen konfiguraation alijärjestelmä. (Haikala & Märijärvi 2004, 257.)



KUVIO 2. Tuotteenhallinta (Haikala & Märijärvi 2004, 260)

Hallinta-alkiot versioituvat ohjelmistotuotteen elinkaaren aikana, kun niissä olevia virheitä korjataan tai ominaisuuksia lisätään. Version jäädyttämisellä tarkoitetaan sitä, ettei hallinta-alkion kyseiseen versioon enää tehdä muutoksia, vaan tätä varten tehdään hallinta-alkiosta uusi versio. Hallinta-alkion viimeisintä jäädytettyä versiota kutsutaan yleisesti vaihetasoksi. Kuviossa kaksi on esimerkki tuotteenhallinnasta, jossa on merkattuna useita komponentteja versioineen. Komponenttien ja niistä koostetun konfiguraation vaihetasot on merkattu

katkoviivalla. Viivan yläpuolella olevat versiot on myös jäädytetty, kun taas alapuolella olevat ovat kehitteillä. (Haikala & Märijärvi 2004, 257-264)



KUVIO 3. Versiopuu (Tuotteen Hallinta 2007)

Komponenttien ja konfiguraatioiden hallintaa kutsutaan yleisesti versionhallinnaksi. Sen tarkoituksena on taata kehittäjille vakaa ympäristö, missä he eivät pääse häiritsemään toistensa työskentelyä esimerkiksi muuttamalla komponentin lähdekoodia kertomatta muutoksista muille. Kun hallinta-alkion ensimmäinen vaihetaso on saavutettu, eivät kehittäjät pääse tekemään muutoksia tähän. Seuraavat muutokset tehdään uuteen versioon, eli revisioon, jotka nimetään usein lineaarisesti kasvavalla luvulla. Joskus jo jäädytettyyn komponenttiin joudutaan palaamaan tekemään muutoksia esimerkiksi asiakkaan vanhentuneen ympäristön vuoksi, minkä seurauksena kehitys joudutaan haarauttamaan. Tällaisia sivuhaaroja kutsutaan variaatioiksi. Haarauttamista tulisi kuitenkin välttää, mikäli mahdollista, sillä mistä tahansa haarasta löydetty vika on mahdollisesti myös muissa haaroissa, jolloin muutoksia on tehtävä useampaan haaraan. Kuviossa kolme on kolmeen haraan jakautunut versiopuu. Haarautuminen on tapahtunut version 1.1 kohdalla, jonka jälkeen kyseiset haarat ovat toisistaan merkittävästi erilaisia ja saattavat toimia vain tietyissä ympäristöissä. Lisäksi jatkokehitys ja

muutokset jatkuvat jokainen omassa haarassaan aiheuttaen pahimmassa tapauksessa kolminkertaisen työmäärän. Tämän vuoksi haarauttamista tulisi välttää. (Haikala & Märijärvi 2004, 260-263.)

Komponenttien versioiden hallintaan on kehitetty erillisiä ohjelmia, kuten CVS (Concurrent Versions System) ja ClearCase, jotka säilövät komponenttien eri versioita ja helpottavat niiden hallintaa. Konfiguraatiot voidaan hallinta-alkioina myös versioida määräämällä komponenttien versiot, jotka muodostavat kyseisen version konfiguraatiosta. Yleisesti käytettyjä työkaluja konfiguraatioiden kokoamiseen ovat erilaiset käännöstyökalut, kuten Make, Ant ja Maven. (Haikala & Märijärvi 2004, 263-264.)

2.3 Muutoksien hallinta

Muutospyyntöjä voivat tehdä asiakkaan tai toimittajan henkilökunta sekä mahdollisesti muut henkilöt. Muutoksia ovat pääasiassa havaittujen virheiden korjaus sekä uusien ominaisuuksien lisäys. Käytännössä muutospyynnöt tehdään kirjallisina raportteina, joissa käy ilmi viallisen tai puutteellisen ohjelmiston ja sen osien versiot, ongelmankuvaus, ongelman vakavuus ja mahdolliset korjausehdotukset havaitsijan puolelta. Muutoksien hallinnasta vastuussa oleva henkilö analysoi lähetetyn raportin ja päättää jatkotoimenpiteistä, joita voi olla vian korjaaminen seuraavaan tai tuleviin toimituksiin tai raportin hylkääminen väärin ymmärretyn toiminnon osalta. (Tuotteen Hallinta 2007.)

3 AUTOMATISOITU KÄÄNTÄMINEN JA JATKUVA INTEGROINTI

3.1 Kääntäminen Javassa

Erilaisilla korkean tason ohjelmointikielillä, kuten C:llä ja Javalla, kuvataan ohjelman toimintaa ihmiselle helposti ymmärrettävässä muodossa. Käytännössä nämä kuvaukset ovat tiedostoja, jotka sisältävät ohjelmointikielelle ominaista syntaksia ohjelman toimintojen suorittamiseen. Näillä kuvauksilla ei itsessään kuitenkaan tee mitään, sillä tietokone ei niitä ymmärrä. Tätä varten korkean tason kielillä on kääntäjiä, eli ohjelmia, jotka kääntävät ihmisen tuottaman kuvauksen koneen ymmärtämään muotoon. Kuviossa neljä on kuvattu kääntäjän toiminta mustalaatikkoperiaatteella, missä kääntäjä saa syötteenä korkean tason kielellä kuvatun ohjelmakoodin ja ulostulona ovat konekieliset ohjeet ohjelman suorittamiseksi. (Cooper & Torczon 2004, 1-2.)

KUVIO 4. Kääntäjän toiminta (Cooper & Torczon 2004, 2)

Javan toiminta perustuu tavukoodiin ja sitä suorittavaan virtuaalikoneeseen. Java-kielinen lähdekoodi käännetään tavukoodiksi Javan virtuaalikonetta varten. Virtuaalikoneen tehtävänä on taas tulkata tavukoodi alustan mukaisiksi ohjeiksi ohjelman suorittamiseksi. Tämä mahdollistaa Javan alustariippumattomuuden, sillä tavukoodi on kaikkien Java virtuaalikoneiden suoritettavissa. Teoriassa riittää, että käytetty alusta tukee Javaa. (Java Tutorial 2009.)

Javac on yleisesti käytetty kääntäjä Java-ympäristössä. Sen tehtävänä on kääntää .java-päätteiset lähdekoodia sisältävät tiedostot .class-päätteisiksi tavukoodia

sisältäviksi tiedostoiksi. Nämä tavukoodia sisältävät tiedostot pystytään suorittamaan koneissa, joihin Java on asennettu. Jos ohjelma koostuu useista tiedostoista, on nämä mahdollista pakata yhdeksi kirjastopaketiksi, JAR-tiedostoksi. (Schildt 2005, 85 & 248.)

Pois lukien kaikkein triviaaleimmat tapaukset, toimivan ohjelman kääntämiseen vaaditaan paljon muutakin kuin lähdekoodin kääntämistä konekielelle. Riippuvuudet ulkoisista ja kolmannen osapuolen kirjastoista on otettava huomioon ohjelmaa käännettäessä ja ajettaessa. Versionhallinnalliset ominaisuudet, kuten pakatun kirjastopaketin luominen uudesta versiosta ja niiden hakemistorakenne sekä mahdolliset laadunvarmistukseen liittyvät asiat, kuten yksikkötestaus, lisäävät käännöstyön määrää ja monimutkaisuutta. Siksi tätä varten on kehitetty työkaluja, jotka automatisoivat kyseiset toiminnot - automatisoidut käännöstyökalut. Tällaisia työkaluja Javalle ovat esimerkiksi Ant ja Maven. (Continuous Integration 2006.)

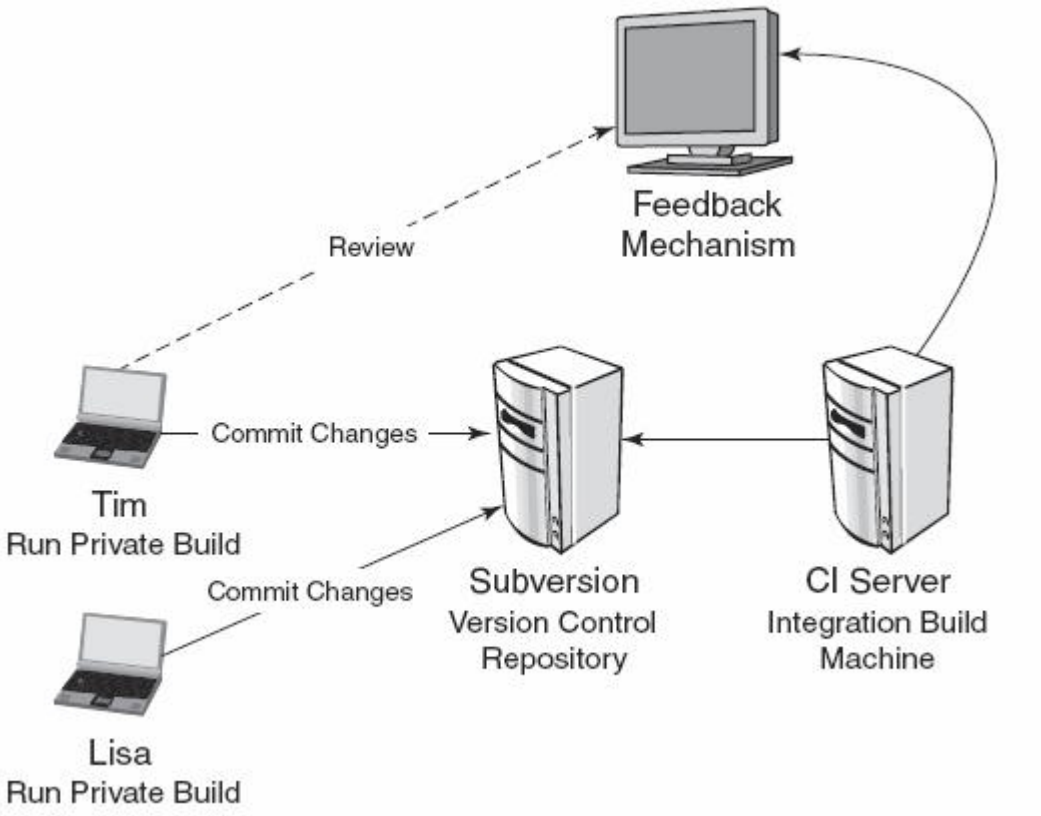
3.2 Jatkuva integrointi

Monimutkaisten järjestelmien kehittämistyössä asiakkaan vaatimukset voidaan jakaa toimintoihin, joita valmiin tuotteen tulee suorittaa täyttääkseen asiakkaan tarpeet. Näiden toimintojen perusteella tuote paloitellaan osiin suunnittelun, toteuttamisen ja ylläpidon helpottamiseksi. Yhteen koottuna nämä osat muodostavat asiakkaan tarpeet täyttävän järjestelmän. Integrointi on siis osien kokoamista yhdeksi toimivaksi kokonaisuudeksi. (Grady 1994, 3.)

Jatkuvalla integroinnilla tarkoitetaan menetelmää, jossa ohjelmistokehittäjät integroivat työtänsä usein. Parhaimmassa tapauksessa jokainen yksittäinen kehittäjä integroi vähintään kerran päivässä, mikä johtaa projektin useampaan integroimiseen päivässä. Prosessia, jossa käännetään lähdekoodi ja verifioidaan sen toiminta yksin ja muiden osien kanssa, voidaan kutsua kääntämisen ja integroinnin automatisoinniksi. (Continuous Integration 2006.)

3.3 Automatisoitu kääntäminen ja jatkuva integrointi käytännössä

Yksinkertaisimmillaan kääntäminen on lähdekoodin muuttamista suoritettavaksi ohjelmaksi. Vaikka ohjelma kääntyykin virheittä, ei se välttämättä toimi oikein yksin tai muiden ohjelmiston osien kanssa. Kääntämisprosessiin voi liittää useita verifiointimenetelmiä, kuten yksikkötestit ja jatkuvan integroinnin. Näiden menetelmien tarkoituksena on antaa välitöntä palautetta tehtyjen muutosten tai lisäysten toimivuudesta. Hyvin toteutettuna nämä metodit havaitsevat virheet aikaisessa vaiheessa, jolloin ne on helpompi ja halvempi korjata. Etenkin ketterät menetelmät käyttävät hyväkseen näitä työkaluja. (Continuous Integration 2006.)



KUVIO 5. Automatisoitu käännös- ja integrointiprosessi (Matyas M., Duvall, Matyas S., Schneider, Glover & Voit 2007, 26)

Automatisoitu kääntäminen ja integrointi toteutetaan pääasiassa kolmen työkalun voimin: versionhallinnan sekä käännöstä ja jatkuvaa integrointia suorittavien työkalujen avustuksella. Kuvion viisi mukaan versionhallinnasta kehittäjä (Tim tai Lisa) hakee halutun lähdekoodin ja tekee siihen tarpeelliset muutokset tai lisäykset

testeineen. Kun tarvittavat muutokset on tehty, käyttäjä suorittaa automatisoidun käännöksen (run private build), joka kääntää ja linkittää lähdekoodin suoritettavaksi ohjelmaksi ja ajaa tarvittavat testit. Koska kehittäjä ei välttämättä ole ainoa, joka tekee muutoksia kyseiseen lähdekoodiin, on hänen vastuullaan varmistaa, että lähdekoodin versio on uusin mahdollinen. Ennen muokatun lähdekoodin lähettämistä takaisin versionhallintaan (Commit Changes), toimiva suoritettava ohjelma on vielä integroitava ohjelmistoon, kokonaisuuden eheyden ja toiminnan varmistamiseksi. Versionhallinnan (Subversion) saadessa tiedon muutoksista, käynnistetään automatisoitu käännös- ja integrointiprosessi tälle omistetulla koneella (CI Server). Palautetta tämän prosessin onnistumisesta voi käyttäjä tarkastella sille tarkoitetun työkalun kautta (Feedback Mechanism). (Continuous Integration 2006.)

Automatisoitu kääntäminen ja integrointi tapahtuvat siis ensin ihmisen ja sitten koneen suorittamana. Tämän toimintatavan suhteen on oltava kurinalainen, eikä jättää testausta ja integrointia pelkästään koneen huoleksi. Suurissa ohjelmistotuotteissa on suuri määrä testattavaa, jolloin yksikkö- ja integrointitestaukseen voi mennä useita tunteja. Jos virhe huomataan vasta tässä vaiheessa, on kehitystyötä voitu jatkaa virheellisesti toimivan ohjelmiston parissa, jolloin myös virheen paikantamiseen ja korjaamiseen kuluu aikaa ja vaivaa. Automatisoidun kääntämisen ja jatkuvan integroinnin ei ole tarkoitus vähentää kehitystyöhön käytettävää aikaa suoranaisesti vaan parantaa ohjelmistotuotteen kehitysvaiheen laatua. (Continuous Integration Is an Attitude Not a Tool 2005.)

Jatkuva integrointi on eräs kommunikoinnin muoto ohjelmistokehityksessä. Mitä useammin toimivia muutoksia lähetetään versionhallintaan, sitä aikaisemmin mahdollisia samanaikaisesti kehitettävien osien välisiä virheitä havaitaan. Löydetyt virheet on myös korjattava mahdollisimman nopeasti. Kehittäjien on pystyttävä luottamaan siihen, että versionhallinnasta haetut koodit ovat toimivia. Avainasioita jatkuvassa integroinnissa ovat toimivien lähdekoodien lähettäminen versionhallintaan usein ja lähetyksen seurauksena saatu palaute. Kehittäjien tulisi lähettää muutoksia vähintään kerran päivässä, useammin, mikäli mahdollista, ja

palaute näiden seurauksena tehdyistä käännöksistä, testeistä ja integroinnista tulisi saada kehittäjille mahdollisimman nopeasti. (Continuous Integration 2006.)

4 KÄÄNNÖSTYÖKALUJA

4.1 Make

Make-ohjelman juuret ulottuvat 1970-luvulle, jolloin haluttiin helpottaa useista lähdetiedostoista koostuvan ohjelman kääntämistä ja linkittämistä. Tuloksena oli käännöksen automatisoiva ohjelma, joka käyttöjärjestelmäkohtaisia komentoja käyttäen kykenee suorittamaan kääntämisen lisäksi useita muita toimintoja. Make-ohjelma on yhä laajalti levinnyt ja käytetty työkalu, joka on ollut esikuvana monelle sen seuraajalle. (Wikipedia 2009.)

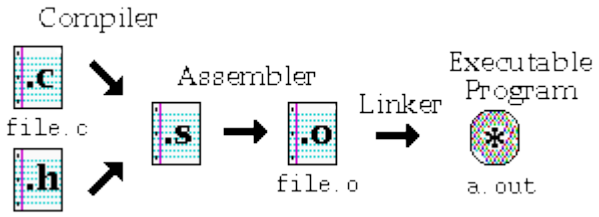
Komennolla make, Make-ohjelma käynnistyy etsimällä Makefile-tiedoston, joka sisältää ohjeet kehitettävän ohjelmiston kääntämiselle. Lähdekoodia sisältävät tiedostot käännetään objektikoodiksi, minkä jälkeen linkittäjä tarkistaa ja linkittää käännettävien lähdetiedostojen väliset riippuvuudet. Näin yhdestä tai useammasta lähdekoodia sisältävästä tiedostosta syntyy toimiva ohjelma. Make-ohjelman toiminta perustuu kehitettävään ohjelmaan tehtävien muutoksien seuraamiseen. Vain edellisen käännöksen jälkeen muuttuneet osat käännetään uudestaan. (Make – Tutorial 2009.)

4.1.1 Makefile

Makefile on tiedosto, joka sisältää ohjeet ohjelman kääntämiseen sekä mahdollisesti kyseisen prosessin ennen tai jälkeen tehtäviin toimenpiteisiin, kuten tiedostojen luominen tai poistaminen. Ohjeet alkavat yleisimmin muuttujien määrittelyllä, jota seuraa kohdealkiot. Muuttujat määritellään avain-arvo -pariperiaatteella. Avain kertoo muuttujan nimen, ja se on yleensä isoista kirjaimista koostuva merkkijono, arvo voi olla jokin komentoriville annettava komento tai hakemisto, jolloin sen pitää noudattaa käyttöjärjestelmän syntaksia.

Avain-arvo parit erotetaan toisistaan rivinvaihdolla. Kohdealkiot määritellään antamalla sille kohteen suorittamaa toimintoa kuvastava nimi, minkä jälkeen luetellaan riippuvuudet muista kohteista välilyönnein erotettuna. Nämä riippuvuudet ovat myös kohdealkioita, jotka suoritetaan ennen kyseessä olevan alkion suorittamista. Kohdealkion varsinainen toiminta määritellään nimen ja riippuvuuksien määrittelevästä rivistä seuraavassa, sisennettynä tabulaattorilla. Komennon nimi annetaan aivan kuten se annettaisiin kyseisen käyttöjärjestelmän komentoriville, alussa määriteltyjä muuttujia mahdollisesti hyväksi käyttäen. (How To – Makefiles 2009.)

Komentoriviltä annettuna komento make etsii tämän hetkisestä hakemistosta makefile nimistä tiedostoa. Mikäli kyseinen tiedosto löytyy, suoritetaan järjestyksessä ensimäinen tiedoston kohdealkio, johon tulisi määrittää oletusarvoisena suoritettavat tehtävät. Kohdealkioita voi myös suorittaa komennolla make, jota seuraa suoritettavan kohteen nimi. (How To – Makefiles 2009.)



KUVIO 6. C-kielisen ohjelman kääntäminen (Make – Tutorial 2009)

Kuviossa kuusi on kuvattu yksinkertaisen C-kielisen ohjelman kääntäminen lähdekoodista suoritettavaksi ohjelmaksi. Prosessi voidaan jakaa kolmeen osaan, C-kääntäjä (Compiler), konekielinen kääntäjä (Assembler) ja linkittäjä (Linker). C-kääntäjän tehtävänä on nimensä mukaisesti kääntää C:llä tehty ohjelmakuvaus (file.c) alemman tason konekieliseksi (Assembly) kuvaukseksi. Konekielinen kääntäjä muuttaa Assembly-koodin objektikoodiksi. Viimeinen vaihe on linkittää objektikoodin käyttämät kirjastot linkittämisen tuloksena syntyvän suoritettavan ohjelman käytettäväksi. Linkittämisvaiheeseen kuuluu myös ohjelman sisäisten riippuvuuksien tunnistaminen. Kuviossa kuusi käännettävä ohjelma syntyy vain

yhdestä C-kielisestä tiedostosta, mutta Make -ohjelma kykenee kääntämään useita tiedostoja sisältäviä projekteja. Toiminta on käytännössä sama kuin yhdenkin lähdetiedoston kanssa, jokaista lähdetiedostoa kohti syntyy yksi objektikoodia sisältävä tiedosto, jotka linkittäjä taas koostaa yhdeksi tai useammaksi suoritettavaksi ohjelmaksi. (Make – Tutorial 2009.)

Make-ohjelma ei sinällään ole kieli- tai alustariippuvainen. Se toimii useimmissa käyttöjärjestelmissä, kuten Windows ja Unix järjestelmät, ja sillä voi kääntää eri lähdekielisiä tiedostoja, kunhan kyseisen kielen kääntäjä löytyy koneelta. Itse Makefile-tiedostossa käytettävä syntaksi poikkeaa kuitenkin edellä mainittujen seikkojen vuoksi. Windows- ja Unixjärjestelmät eivät välttämättä tunnista samoja käskyjä, eikä kaikkien kielien käännösprosessi ole samankaltainen. Make-ohjelma soveltuu käytettäväksi myös muissakin yhteyksissä kuin kääntämisessä. (How To – Makefiles 2009.)

4.1.2 Make Javassa

```
1 JCC = javac
2 JFLAGS = -g
3 CLASSPATH = -cp buildlib/jaxen-1.1.1.jar:buildlib/dom4j-1.6.1.jar
4 DEST = -d classes
5
6 default: init Example.class dist
7
8 Example.class: Example.java
9     $(JCC) $(JFLAGS) $(CLASSPATH) src/*.java $(DEST)
10
11 init:
12     mkdir -p classes
13
14 dist:
15     jar cf TestProject.jar classes/*.class
16
17 clean:
18     rm classes/*.class
```

KUVIO 7. Esimerkki Makefil- tiedostosta

Koska Java sovelluksien suorittaminen tapahtuu virtuaalikoneen ja tavukoodia sisältävien tiedostojen avulla, eroaa Javakielen kääntäminen C:stä jonkin verran.

Javac-kääntäjä muuntaa .java -päätteiset lähdekieliset tiedostot .class -päätteisiksi tavukoodia sisältäviksi tiedostoiksi, joita taas Javan virtuaalikone pystyy tulkkaamaan. Näin ollen C:n tapaan objektitiedostoja tai niistä linkitettyjä suoritettavia ohjelmia ei synny, mikä aiheuttaa pieniä muutoksia Make tiedoston sisällössä.

Kuviossa seitsemän on esimerkki Makefile-tiedostosta, joka kääntää Java-lähdekoodin tavukoodiksi, paketoi projektin .JAR tiedostoksi sekä suorittaa tarvittavat kansioiden luomiset ja poistamiset, mikäli tarpeellista. Oletuksena on, että tämä tiedosto on Java projektin juurihakemistossa ja käytössä on Unix-pohjainen käyttöjärjestelmä. Riveillä 1-4 on määritelty tiedoston sisällä käytettävät muuttujat (makrot). Etenkin isoissa ja monimutkaisissa tapauksissa muuttujien käyttäminen selkeyttää tiedoston luettavuutta. Tässä tiedostossa on määritelty muuttujina käytettävä kääntäjä, kääntäjän käyttämät optiot, projektin riippuvuussuhteet ulkoisista kirjastoista sekä kohdehakemisto, minne käännetyt tiedostot sijoitetaan. Rivillä kuusi on määritelty oletustoiminto make komentoa kutsuttaessa. Se suorittaa järjestyksessä kaksoispisteen jälkeen määriteltyjä alkioita. Rivin kahdeksan kohde suorittaa käännöksen, ja siihen tarvittavat komennot on määritelty seuraavalla rivillä, tabulaattorin aloittaessa kyseisen rivin. Tässä kohtaa käytetään tiedoston alussa määriteltyjä muuttujia. Rivin 11-12 alkio luo kansion, johon käännetyt tiedostot luodaan ja rivin 14-15 paketoi projektin jar paketiksi. Viimeisien rivien 17-18 alkio poistaa kaikki käännetyt tiedostot. Tätä alkiota pitää kutsua erikseen komennolla make clean, koska sitä ei ole määritelty suoritettavan oletuksena.

Kuvion 7 Makefile-tiedostossa on käytetty vain osaa kaikista mahdollisista komennoista ja ominaisuuksista, mihin käyttöjärjestelmä ja kääntäjä pystyvät. Esimerkiksi tiedostojen kopiointi ja siirtäminen sekä yksikkötestit ovat mahdollisia toteuttaa, mutta monimutkaistavat Makefile-tiedoston toteutusta. Koska tiedosto käyttää käyttöjärjestelmäkohtaisia komentoja, sitä ei välttämättä voi käyttää eri järjestelmissä ilman muutoksia, kuten esimerkiksi Unix-järjestelmästä siirryttäessä Windows-maailmaan. Lisäksi jokainen Java -projekti

tarvitsee oman Make tiedostonsa, mikä isoissa ohjelmistoprojekteissa tarkoittaa useiden tiedostojen luomista ja ylläpitämistä.

4.2 Ant

Ant-ohjelma on Apache Software Foundationin kehittämä Make-ohjelman kaltainen automatisoitu käännöstyökalu pääasiassa Javaprojektien kääntämiseen. Se on toteutettu Javalla ja vaatii siis Javan kehitysympäristön toimiakseen. (Wikipedia 2009.)

Alkuperäisen luojansa mukaan Ant on lyhenne sanoista Another Neat Tool, eli karkeasti käännettynä taas yksi näppärä työkalu. Se on alun perin ollut osa Apachen Tomcat projektia helpottamassa kääntämisprosessia. Sieltä Ant-ohjelma on kuitenkin levinnyt käytännöllisyytensä vuoksi laajempaan käyttöön ja aina omaksi projektikseen. (Apache Ant FAQ 2009.)

Ant-ohjelman luomisen liikkeellepanevana voimana on ollut Make-ohjelman kaltaisten käännöstyökalujen alustariippuvaisuuden poistaminen. Missä Make-ohjelman komennot ovat komentoriville annettavia käyttöjärjestelmäkohtaisia käskyjä, kuten haluttujen lähdetiedostojen kääntäminen ja linkittäminen, Ant-ohjelma käyttää XML-muotoista konfiguraatiotiedostoa. Tätä tiedostoa Ant-ohjelma käy läpi ja suorittaa siinä määriteltyjä toimintoja, jotka vastaavat kyseisen käyttöjärjestelmän vastaavia komentoja. Koska Ant -ohjelma on toteutettu Javalla, on se käytännössä alustariippumaton, riittää, että käyttöjärjestelmässä toimii Java. Näin ollen sama konfiguraatiotiedosto toimii useissa eri ympäristöissä, toisin kuin Make-tiedostot. Lisäksi Ant-ohjelman konfiguraatiotiedoston käyttämä XML-formaatti poistaa Makefile-tiedostojen sisennys- ja välilyöntimuotoilusta koituvat sekaannukset. (Apache Ant Manual 2009.)

4.2.1 Konfiguraatiotiedosto

Konfiguraatiotiedosto on Ant-ohjelman sielu, ja se koostuu projektialkiosta, johon kuuluu yksi tai useampi kohdealkio. Projektialkiolla on kolme määrettä, nimi, oletus sekä perushakemisto. Nimimääre kertoo projektin nimen, oletusmääre suoritettavan kohteen nimen siltä varalta, ettei sitä ole komentoriviltä erikseen määritelty ja perushakemisto määrittelee projektin juurihakemiston osoitteen. Projektialkiolla voi olla myös useita ominaisuusalkioita. Nämä avain-arvo parit määrittelevät projektin yleisiä ominaisuuksia, kuten hakemisto-osoitteita, ja niitä voidaan käyttää muualla projektissa viittaamalla kyseiseen avaimeen. Projektialkion määreitä tai ominaisuuksia ei ole pakollista määrittää. (Apache Ant Manual 2009.)

Kohdealkiot määrittelevät erilaisia kääntämiseen liittyviä tehtäviä. Niillä voi olla määreinä nimi, riippuvuus, ehto ja kuvaus, joista vain kohteen tunnistava nimi on pakollista määrittää. Riippuvuus kertoo, mitkä kohteet tulee suorittaa ennen kyseessä olevan kohteen suorittamista, ja ehtomääreellä voi tarkistaa esimerkiksi kriittisen hakemiston olemassaolon. Kuvauksessa voi lyhyesti kertoa, mitä kyseinen kohde tekee. (Apache Ant Manual 2009.)

Kohdealkiossa olevat tehtäväalkiot suorittavat varsinaiset toiminnot, mitä kääntämisessä voi tarvita. Ant-ohjelmassa on joukko valmiita komentoja, minkä lisäksi käyttäjä voi tarvittaessa itse ohjelmoida niitä lisää. Esimerkkejä tehtäväalkioiden valmiista komennoista ovat tiedostojen käsittelystä tutut poisto, kopiointi, luonti ja pakkaus sekä erilaiset kääntämiskomennot. (Apache Ant Manual 2009.)

Käännösaikaiset riippuvuudet määritellään luokkapolkualkiossa. Määrittely on mahdollista tehdä tiedoston tarkkuudella tai tarvittaessa hakea kaikki tietyn nimiset tiedostot annetusta kansiosta. Luokkapolkualkio sijoitetaan kohdealkioon, joka tukee sen käyttöä, kuten esimerkiksi kääntämiskomennon suorittava javac-kohdealkio. Ant-ohjelmalla on myös oma kirjastohakemisto kotihakemistossaan, johon voi lisätä pakattuja Java-kirjastoja. Tässä tapauksessa kannattaa kuitenkin ottaa huomioon, että kyseinen hakemisto on kaikille Ant-ohjelmaa käyttäville projekteille yhteinen. (Apache Ant Manual 2009.)

4.2.2 Ant-ohjelma käytännössä

Ant-ohjelman käyttämiseen tarvitaan itse ohjelman lisäksi yhteensopiva Java-kehitysympäristö, sekä XML-parsija. Ennen kuin Antia voi ajaa, pitää järjestelmän ympäristömuuttujiin lisätä Antin sekä Javan kotitiedostot. Nyt käännöstyökalu käynnistyy komentoriville annettaessa komento ant. Se ei kuitenkaan käännä mitään, sillä konfiguraatiotiedosto build.xml puuttuu. (Apache Ant Manual 2009.)

```
1  <?xml version="1.0"?>
2
3  <project name="TestProject" default="dist" basedir=".">
4
5      <property name="version" value="1-0-0" />
6
7      <target name="init">
8          <mkdir dir="classes" />
9          <mkdir dir="dist" />
10     </target>
11
12     <target name="compile" depends="init">
13         <javac srcdir="src" destdir="classes">
14             <classpath>
15                 <pathelement path="${classpath}" />
16                 <fileset dir="buildlib">
17                     <include name="**/*.jar" />
18                 </fileset>
19             </classpath>
20         </javac>
21     </target>
22
23     <target name="dist" depends="compile">
24         <jar jarfile="dist/TP-${version}.jar" basedir="classes" />
25     </target>
26
27     <target name="clean">
28         <delete dir="classes" />
29         <delete dir="dist" />
30     </target>
31
32 </project>
```

KUVIO 8. Konfiguraatiotiedosto build.xml

Kuviossa kahdeksan on kuvattuna yksinkertainen konfiguraatiotiedosto kokonaisuudessaan. Kuvion kolmannella rivillä on projektialkio, project, joka kietoo sisäänsä kääntämisprosessin toiminnallisuuden. Huomionarvoisia määreitä ovat default ja basedir, joista ensimmäinen kertoo oletusarvoisena suoritettavan

kohteen nimen ja jälkimmäinen projektin hakemistojuuren. Eli komentoriviltä ajettaessa, Ant-ohjelma käynnistää kohteen dist, ja juurihakemistona käytetään hakemistoa, jossa kyseinen tiedosto tällä hetkellä on. Rivillä viisi on koko projektille yhteinen ominaisuusalkio, jonka avain-arvo pari määrittelee versionumeron.

Riviltä seitsemän alkaen on määritelty init niminen kohdealkio, jonka tehtävänä on luoda hakemistorakenne kääntämistä varten. Kaksi tiedostoa, classes ja dist, tehdään tätä kohdetta kutsuttaessa. Rivin 12 kohde kääntää Java projektin. Se on riippuvainen init-kohteesta, mikä tarkoittaa, että kyseinen kohde on suoritettava ennen tämän kohteen suorittamista. Rivillä 13 on määritelty käytettävä kääntäjä, käännettävien Java-tiedostojen sijainti sekä hakemisto, minne käännetyt tiedostot sijoitetaan. Projektin riippuvuudet ulkoisista kirjastoista määritellään luokkapolkualkiossa riviltä 15 alkaen. Kaikki JAR -päätteiset tiedostot buildlib hakemistosta sisällytetään käännösaikaisiin riippuvuuksiin. Kolmas kohdealkio, nimeltään dist, alkaa riviltä 23 ja se on riippuvainen käännösalkiosta. Tämän kohteen tarkoitus on pakata käännettävä projekti JAR-paketiksi classes nimiseen hakemistoon. Viimeisenä kohteena on clean, joka nimensä mukaisesti siivoaa käännöksen jälkeen jättämät tiedostot ja hakemistot.

Tämä konfiguraatiotiedosto sijoitettuna käännettävän Java-projektin juurihakemistoon mahdollistaa Ant-ohjelman käyttämisen. Komennolla ant oletuskohde dist suoritetaan. Koska dist-kohde on riippuvainen compile kohteesta, joka puolestaan on riippuvainen init:stä, suoritetaan viimeksi mainittu ensimmäisenä. Kokonaisuudessaan ant komento siis luo tarvittavat hakemistot, kääntää Java-tiedostot tavukoodiksi ja tallentaa ne vastaaviin tiedostoihin sekä luo JAR-pakatun paketin näistä. Yksittäisiä kohteita, mikäli ne eivät ole riippuvaisia muista kohteista, on mahdollista suorittaa komennolla ant ja antamalla suoritettavan kohteen nimi tämän perään. Kuvion kuusi konfiguraatiotiedoston clean kohde on siis mahdollista suorittaa komennolla ant clean, jolloin kyseinen kohde suoritetaan, minkä seurauksena classes- ja dist-kansiot sisältöineen poistetaan.

4.3 Maven

Alun perin osana Apachen Jakarta projektia Maven-ohjelma on Ant-ohjelman tavoin kehittynyt itsenäiseksi käännöstyökaluksi. Sen on tarkoituksena suorittaa automatisoitua käännöstä yleisemmällä tasolla. Ant-ohjelman projektikohtaiset konfiguraatiotiedostot saattavat olla hyvin samankaltaisia keskenään, jolloin se on haluttu työkalu, joka kuvaa yleisellä tasolla automatisoidun käännöksen ja siinä suoritettavat tehtävät. Käännösaikaiset projektien ja ulkopuolisten kirjastojen väliset riippuvuudet on myös haluttu ottaa huomioon Maven-ohjelmassa. Tässä työssä käsitellään Maven-ohjelman toista versiota, jossa on merkittäviä uudistuksia ensimmäiseen versioon verrattuna. (Apache Maven: What is Maven? 2009.)

4.3.1 Project Object Model -tiedosto

Maven-ohjelman vastine Ant-ohjelman konfiguraatiotiedostolle on Project Object Model -tiedosto, eli lyhyemmin POM-tiedosto. Se on XML -muotoinen tiedosto ja sisältää tietoa projektin hakemistorakenteesta, riippuvuuksista muihin projekteihin ja ulkopuolisiin kirjastoihin sekä yleistä tietoa projektista. Valmiiksi määritellyt maalit suorittavat kaikki toimenpiteet, kuten kääntämisen, yksikkötestauksen sekä pakkaamisen, käyttäen avukseen POM-tiedoston tietoa käännettävästä projektista. Nämä maalit ovat verrattavissa Antin kohteisiin, paitsi että ne ovat valmiiksi toteutettu Maven-ohjelmassa. Mikäli valmiiksi määritellyt maalit eivät riitä, voi niitä kirjoittaa itse lisää. (Apache Maven Simplifies the Java Build Process 2003.)

Yksi Maven-ohjelman merkittävimmistä ominaisuuksista ovat POM-tiedostossa määriteltävät riippuvuudet, jotka ladataan ulkoisesta tietovarastosta. Maven-ohjelma hakee tarvittavat ulkoiset kirjastot ennen suoritusta ja tallentaa ne paikalliseen hakemistoon sekä päivittää niitä tarpeen vaatiessa. Myös omat käännetyt projektit paketoitaessa siirretään tähän yleiseen paikalliseen kirjastohakemistoon, jotta muut projektit voivat tätä käyttää. Tämä helpottaa kehittäjän työtä, koska hänen ei tarvitse etsiä itse oikeita versioita tarvittavista

kirjastoista Internetistä, versionhallinnasta tai yksittäisten ihmisten kovalevyiltä. (Apache Maven: Getting Started 2009.)

POM kuvaa projektia sen laajemmassa merkityksessä. Se ei siis ole vain joukko koodia sisältäviä tiedostoja vaan kaikki ohjelman toimintaan liittyvät asiat, kuten riippuvuudet, kehittäjät, organisaatio, lisenssit ja laadunhallinta. Tämän myötä POM-tiedosto saattaa olla suuri ja monimutkainen kokonaisuus. Hahmottamisen helpottamiseksi se voidaan paloitella neljään osaan, projektin välisiin suhteisiin, projektin tunnistetietoihin, käännösasetuksiin ja -ympäristöön. (The Maven 2 POM demystified 2006.)

Projektin välisissä suhteissa on pakollisina määritteinä projektin osoitetiedot, joita kutsutaan myös koordinaateiksi. Näillä erotetaan eri projektit ja niiden versiot toisistaan. Projektien välisissä suhteissa määritellään myös riippuvuudet toisiin projekteihin, käytettävät ulkoiset kirjastot sekä osat, joista kyseinen projekti koostuu. Projektin tunnistetiedoissa on yleistä tietoa projektista, kuten projektin nimi, yrityksen nimi, kehittäjät ja kuvaus. Käännösasetuksissa määritellään kaikki kääntämiseen ja sen raportointiin liittyvät asiat. Näihin kuuluvat esimerkiksi käytettävä kääntäjä versioineen, yksikkötestit sekä paketointitapa. Viimeisenä lohkona ovat ympäristön asetukset, missä määritellään projektin käyttämät Maven-ohjelman ulkoiset sovellukset, kuten versionhallinta ja jatkuvan integroinnin työkalut sekä tietovarastot. (The Maven 2 POM demystified 2006.)

4.3.2 Maven käytännössä

Kun Maven -ohjelma on asennettu ja tarpeelliset ympäristömuuttujat asetettu, voidaan ohjelma käynnistää kutsumalla mvn komentoriviltä, jonka perään määritellään argumenttina haluttu toiminto. Keskeisimpänä suoritettavana osana toimii käännös, joka jakaantuu eri vaiheisiin. Oletuksena nämä vaiheet ovat järjestyksessä validointi, käännös, yksikkötestaus, paketointi, integrointitestaus, verifiointi, asennus sekä käyttöönotto. Kyseiset vaiheet käydään alusta loppuun aina siihen vaiheeseen asti, minkä käyttäjä antaa komentoriviargumenttina.

Esimerkiksi jos halutaan paketoida haluttu projekti, annetaan komento mvn package, jolloin käännösprosessi alkaa koodin oikeellisuuden tarkistuksella. Tätä seuraa kääntäminen yksikkötesteineen, minkä jälkeen käännetyt tiedostot paketoidaan kirjastopaketiksi. Kaikkien edellisten vaiheiden suorittamiseen vaadittavat tiedot Maven -ohjelma saa kyseisen projektin POM-tiedostosta. Käännös on vain yksi monista mahdollisista maaleista, mitä Maven-ohjelma tarjoaa, ja mikäli haluttua toiminnallisuutta ei löydy, voi sellaisen luoda. (Apache Maven: Introduction to the Lifecycle 2009.)

```
1  <project>
2      <!-- POM Relationships -->
3      <modelVersion>4.0.0</modelVersion>
4      <groupId>com.ericsson.eniq</groupId>
5      <artifactId>Eniq</artifactId>
6      <version>1-0</version>
7      <dependencies>
8          <dependency>
9              <groupId>dom4j</groupId>
10             <artifactId>dom4j</artifactId>
11             <version>1.6.1</version>
12         </dependency>
13         <dependency>
14             <groupId>jaxen</groupId>
15             <artifactId>jaxen</artifactId>
16             <version>1.1.1</version>
17         </dependency>
18     </dependencies>
19
20     <!-- Project Information -->
21     <name>TestProject</name>
22
23     <!-- Build Settings -->
24     <packaging>jar</packaging>
25     <build>
26         <plugins>
27             <plugin>
28                 <groupId>org.apache.maven.plugins</groupId>
29                 <artifactId>maven-compiler-plugin</artifactId>
30                 <version>2.0.2</version>
31                 <configuration>
32                     <source>1.6</source>
33                     <target>1.6</target>
34                 </configuration>
35             </plugin>
36         </plugins>
37     </build>
38
39     <!-- Build Environment -->
40 </project>
```

KUVIO 9. Esimerkki POM-tiedosto

Kuviossa yhdeksän on esimerkki yksinkertaisesta POM-tiedostosta. Riveillä 2, 20, 23 ja 39 on kommentoituna eri lohkojen aloitusrivit. Ensimmäisessä lohkossa

määritellään projektin koordinaatit sekä suhteet muihin projekteihin ja kirjastoihin. Rivin kolme alkio modelVersion kertoo tässä projektissa Maven-ohjelman käyttämän oliomallin version. Seuravilla riveillä olevat groupId-, artifactID- ja version -alkiot ilmoittavat projektin koordinaatit. Riviltä seitsemän alkaen on lueteltuna projektin riippuvuudet, tässä tapauksessa kaksi ulkoista kirjastoa. Molemmilla on määriteltynä koordinaattitunnisteet, minkä mukaan tarvittavat tiedostot haetaan paikallisesta tai ulkoisesta tietovarastosta. Projektin tunnisteissa tässä esimerkissä rivillä 21 on määriteltynä vain projektin nimi. Muita mahdollisia määrittelyjä ovat esimerkiksi yrityksen nimi, lisenssitietoa tai projektin kuvaus.

Käännösasetukset alkavat riviltä 24 pakkaustyypin määrittelyllä. Pakkauskomennon suorituksen tuloksena on projektin tavukooditiedostoista koottu JAR kirjastopaketti. Rivin 25 build alkion sisään on määritelty liitännäisenä kääntäjä, joka käyttää Javan 1.6 versiota kääntämiseen. Myös muuttujien ja raportoinnin määrittelyt kuuluvat tähän lohkoon. Viimeisenä, puuttuvana lohkona on käännösympäristön määrittely. Tässä esimerkissä käytetään oletusasetuksia, mutta tähän kohtaan voitaisiin määritellä versionhallintaan tai jatkuvaan integraatioon käytettävät ohjelmat.

Esimerkin POM-tiedosto sijoitetaan projektin juureen, josta Maven-ohjelman maaleja voidaan kutsua. Lähdekoodeja etsitään oletushakemistosta, koska tiedostorakennetta ei ole erikseen määritelty. Suoritettaessa käännöstä ensimmäistä kertaa käyttäen tätä POM-tiedostoa komennolla mvn package, Maven aloittaa käännösprosessin lataamalla tarvittavat tiedostot. Näihin kuuluvat käännösprosessissa suoritettavat vaiheet validate, compile, test ja package sekä ulkoiset kirjastot jaxen ja dom4j, mistä esimerkin projekti on riippuvainen. Kun tiedostot on ladattu suoritetaan maalit aina package vaiheeseen asti, minkä tuloksena syntyy JAR-kirjastopaketti käännetyistä lähdetiedostoista.

4.4 Käännöstyökalujen vertailu

Käännöstyökaluja vertailtaessa on määriteltävä käännettävän tuotteen asettamat vaatimukset. Tässä työssä käsiteltävän tuotteen, ENIQ:n, kehityksessä tarvittavia ominaisuuksia ovat ainakin yksikkötestaus, jatkuva integrointi sekä yhteensopivuus ClearCase ja CVS -versionhallintasovellusten kanssa. Itse käännökseltä vaaditaan käännösskriptin selkokielisyyttä, jotta kaikki kehittäjät sitä ymmärtävät, käännöksen eri vaiheiden dokumentointia sekä etenkin selkeää riippuvuuksien hallinnointia. Taulukossa yksi on kuvattu Maken, Antin sekä Mavenin ominaisuuksia.

TAULUKKO 1. Käännöstyökalujen ominaisuuksia

	Make	**Ant**	**Maven**
Syntaksi	Komentoskripti	XML	XML
Siirrettävyys	Käyttöjärjestelmäkohtainen	Java	Java
Käännös	Suoritetaan kohdealkioissa määritellyt toimenpiteet	Suoritetaan kohdealkioissa määritellyt toimenpiteet	Valmiiksi määritelty käännösprosessi
Käännöstiedosto	Moduulikohtainen makefile -tiedosto	Moduulikohtainen build.xml -tiedosto	Projektia kuvaava POM.xml -tiedosto
Yksikkötestaus	Ei suoranaisesti tuettu	Määriteltävä suoritettava kohde	Osana käännösprosessia
Dokumentointi	Ei suoranaisesti tuettu	Määriteltävä suoritettava kohde	Osana käännösprosessia, liitännäiset
Riippuvuudet	Määriteltävä moduulikohtaisesti Makefile -tiedostossa	Määriteltävä moduulikohtaisesti classpath -alkiossa build.xml -tiedostossa	Määriteltävä projektin POM.xml -tiedoston depedencies -alkiossa
Kirjastopakettien hallinta	Määriteltävä hakemisto(t), jossa paketit sijaitsevat	Määriteltävä hakemisto(t), jossa paketit sijaitsevat	Kaikille Maven projekteille yhteinen paikallinen kirjasto, sekä ulkoinen kirjasto, josta voi hakea kolmannen osapuolen paketteja

Make-ohjelma on hyvä käännöstyökalu, jolla on myös muita käyttökohteita. Java -ohjelmien kääntämiseen ja hallinnointiin, etenkin isojen ja monimutkaisten projektien kohdalla, löytyy kuitenkin parempia vaihtoehtoja. Käyttöjärjestelmäkohtaiset komennot sekä hankala skriptikieli tekevät Makesta huonon valinnan ENIQ:n käännöstyökaluksi. Periaatteessa kaikki ENIQ:n tarvitsemat ominaisuudet on mahdollista toteuttaa Makella, mutta käytännön toteutus on monimutkaista. Todellinen valinta tehdään Antin ja Mavenin välillä,

mitkä ovat suosittuja Java-ohjelmistojen käännöstyökaluja. Vaikka Maven-ohjelma on uudempi ja pyrkii paikkaamaan Ant-ohjelman puutteita, ovat ne käyttötarkoituksiltaan ja toiminnallisuuksiltaan erilaisia. Ant-ohjelma on erikoistunut moduulikohtaiseen käännökseen ja pystyy suorittamaan monimutkaisiakin toimintoja yksinkertaisella XML -muotoisella skriptillä. Ongelmana on, että suuri projekti tarvitsee useita skriptitiedostoja, joiden ylläpitäminen voi olla työlästä. Lisäksi Ant-ohjelman riippuvuuksien hallinnoiminen ilman ulkoisia ohjelmia voi osoittautua hankalaksi tehtäväksi. Kaikki tarvittavat ominaisuudet, kuten yksikkötestaus sekä raportointi hoituvat, ja Ant-ohjelma on laajennettavissa käyttämään muita työkaluja, jotka esimerkiksi kuvaavat projektin rakennetta Maven-ohjelman tapaan.

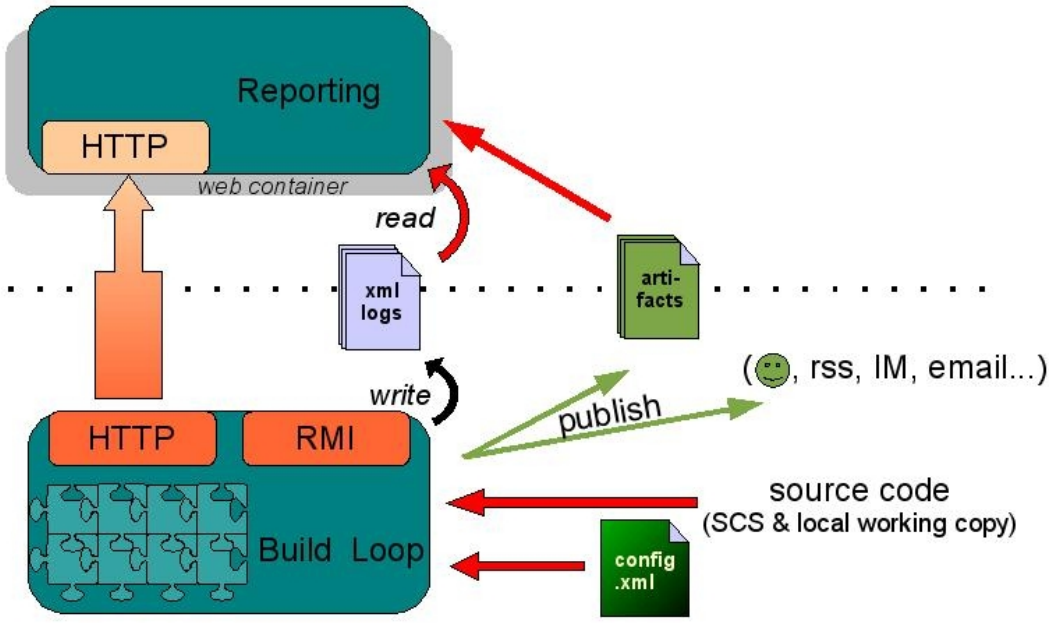
Maven-ohjelma suorittaa käännöksen yleisemmällä tasolla, ja se ottaa enemmän kantaa koko projektin luonteeseen. Tuotteen osat ja riippuvuudet on määritelty POM-tiedostossa, minkä mukaan käännösprosessi suoritetaan. Käännösprosessiin kuuluu oleellisimmat ominaisuudet, kuten yksikkö- ja integrointitestaukset sekä raportointi. Maven-ohjelma on pitkälle automatisoitu, jolloin kehittäjän tarvitsee määritellä vain korkean tason käännösohjeet koko projektille. Tämä pakottaa projektin noudattamaan yleisesti hyväksi katsottuja tapoja Java projektin koostumuksesta. Kääntöpuolena Maven-ohjelma ei tämän takia ole niin joustava kuin Ant -ohjelma.

Ant- ja Maven -ohjelmat ovat molemmat hyviä käännöstyökaluja ja täyttävät ENIQ:n asettamat vaatimukset. Maven-ohjelma on kuitenkin riippuvuuksien hallinnalla, kaikille projekteille yhteisellä kirjastolla sekä yhdenmukaistetulla käännösprosessillaan ENIQ:n kääntämiseen valittava työkalu.

5 INTEGROINTIYMPÄRISTÖJÄ

5.1 CruiseControl

ThoughtWorksin kehittämä CruiseControl on ilmainen vapaan lähdekoodin työkalu ohjelmistotuotteen automatisoidun käännöksen ja jatkuvan integroimisen toteuttamiseen. Tämä tapahtuu oletuksena Antin ja versionhallinnan yhteistyönä. CruiseControl on laajennettavissa liitännäisillä tukemaan muita kieliä ja kääntämistyökaluja, kuten esimerkiksi Mavenia. Kääntämisen ja integroinnin tulokset kerätään keskitetylle sivustolle, josta kehittäjät voivat nähdä sen hetkisen tilanteen tuotteen eheydestä. (Wikipedia 2009.)



KUVIO 10. CruiseControl arkkitehtuuri (CruiseControl overview 2009.)

Kuviossa 10 on kuvattu CruiseControlin toiminta. Se voidaan jakaa kolmeen osaan: käännössilmukkaan (Build Loop), JSP- sekä kojelauta raportointisivustoihin (Reporting). Käännössilmukka tarkkailee versionhallintaa ja aloittaa käännös- ja integrointiprosessin, kun muutos rekisteröidään tai tietyin väliajoin, riippuen käyttäjän asetuksista. Tulokset käännöksestä lähetetään

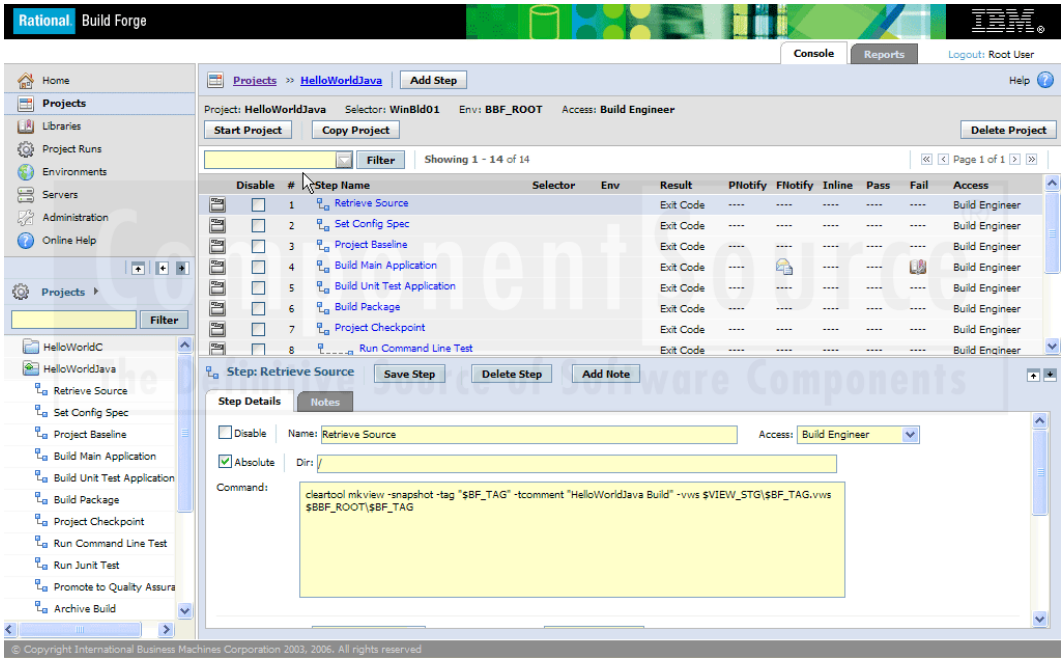
raportointisivustoille (xml logs & artifacts), minkä lisäksi erilaiset liitännäiset mahdollistavat esimerkiksi sähköpostipalautteen ja käännettyjen tiedostojen siirtämisen toiseen kohteeseen (publish). Kojelaudan konfigurointi tapahtuu sille tarkoitetusta asetustiedostosta, mutta oletusarvoisesti se on varsin kattava nykyisen ja aikaisempien käännöksien onnistumisen tarkastamiseen. JSP-raporttisivu antaa puolestaan hieman tarkempaa tietoa eri käännöksistä. Nimensä mukaisesti JSP (JavaServer Pages) teknologiaa käyttävä raportointi sivu koostuu useista JSP tiedostosta, jotka dynaamisesti luovat verkkosivuja. Tämän sivuston konfiguroimiseksi joudutaan kirjoittamaan XML:n kaltaista syntaksia sisältäviä tiedostoja, mitkä julkaisevat haluttuja raportteja. (CruiseControl overview 2009.)

Ennen CruiseControlin asentamista, tarvitaan olemassa oleva käännöstyökalu skripteineen, joka kääntää halutun projektin. Tämän lisäksi kyseisen projektin tulee olla jossakin versionhallinnassa. On huomioitavaa, että käännös ja siihen liittyvä verifiointi on täysin käännöstyökalun vastuulla. CruiseControl on käytännössä työkalu, joka kuuntelee versionhallintaa muutoksien varalta, käynnistää kääntämisprosessin aina kun on tarpeellista ja kerää tästä tiedon yhteen keskitettyyn paikkaan kehittäjien nähtäville. Se ei siis itsessään käännä, suorita integrointia, tai muita vastaavia toimintoja vaan jakaa komentoja eri työkaluille, jotka suorittavat nämä toiminnot. CruiseControlin käännössilmukkaa suoritetaan taustalla jatkuvasti ja se toimii konfiguraatiotiedostossa määriteltyjen tietojen mukaan. Tässä tiedostossa on siis määritelty kaikki suoritettavat komennot, tiedostojen ja hakemistojen sijainnit, muutoksien tarkastusajankohtien ajastus sekä paljon muita mahdollisia toimintoja. (Hightower, Onstine, Visan, Payne, Gradecki, Rhodes, Watkins, Meade 2004, 481-497.)

5.2 Build Forge

CruiseControlin tapaan IBM:n Rational perheeseen kuuluva Build Forge on käännöksen automatisointiin ja jatkuvaan integrointiin soveltuva kaupallinen työkalu. Se tukee eri kieliä sekä niitä kääntäviä työkaluja ja onkin suunniteltu käytettäväksi useiden erilaisten projektien keskitettynä käännöstyökaluna.

Ohjelmistotuotteen kääntämisen, testauksen ja raportoinnin lisäksi Build Forge tarjoaa työkalut käyttäjien hallinnointiin sekä yhdenaikaisiin käännösprosesseihin. Näiden ansiosta kehittäjät pystyvät suorittamaan eri projektien kääntämistä ja siihen liittyviä tehtäviä mistä tahansa. (CM Crossroads 2009.)



KUVIO 11. Build Forge käyttöliittymä (IBM Rational Build Forge Express Edition 2009)

Kaikki kääntämisestä, käyttäjäoikeuksiin ja käytettäviin servereihin hoidetaan kuviossa 11 esitetyn Build Forgen WWW-käyttöliittymän avulla, johon voi selaimen avulla päästä käsiksi mistä vain. Kuvion käyttöliittymän vasemman laidan valikon kautta navigoidaan käyttäjä-, projekti-, ympäristö-, ja palvelinvalikoihin. Näiden muokkaamiseen avautuu yksi tai useampi valikko, mistä voidaan hallinnoida erilaisia asioita. Esimerkiksi käyttäjävalikosta (Administration) voi lisätä käyttäjiä ja antaa näille oikeuksia tiettyjen projektien raporttien tarkasteluun tai projektin kääntämisohjeiden muokkaamiseen. Palvelinvalikosta (Servers) voidaan määritellä kääntämistä sekä siihen liittyviä tehtäviä suorittavat palvelimet. Automatisoitua käännöstä ja jatkuvaa integrointia varten annettavat ohjeistukset ja suoritettavat toimenpiteet määritetään projektikohtaisesti projektivalikosta (Projects). Jokainen projekti koostuu yhdestä tai useammasta askelmasta, jotka jakavat suoritettavia tehtäviä, kuten kääntämisen

tehtäväksi sille tarkoitetulle palvelimelle. Käyttäjät voivat käynnistää näitä tehtäviä manuaalisesti tai ajoittaa ne käynnistyväksi tietyin väliajoin ja tarkastella tuloksia sille tarkoitetusta valikosta. (Rational Build Forge Enterprise Edition 2009.)

5.3 Integrointiympäristöjen vertailua

Kuten CruiseControl, Build Forge ei itsessään suorita käännöstä tai siihen liittyviä tehtäviä vaan jakaa kyseiset tehtävät eteenpäin suoritettaviksi. Tämän takia käännösskriptit, testit, versionhallinta ja muut vastaavat on oltava olemassa, joille edellä mainitut työkalut tehtävänsä jakaa. Tässä prosessissa ei siis ole suuria eroja näiden tuotteiden välillä, ja ne molemmat tukevat Mavenia käännöstyökaluna sekä ClearCasea ja CVS:ää versionhallintana. Isoin ero löytyy kohderyhmässä, jolle nämä tuotteet on tarkoitettu. CruiseControl on hyvin soveltuva kehitystiimin sisäiseen käyttöön käännöksen automatisoimiseen ja jatkuvan integroinnin työkaluksi, joka antaa palautetta ryhmän jäsenille tehdyistä muutoksista. Build Forge tarkentaa tätä palautteenantoa ja kertoo tarkemmin, kuka on tehnyt, mitä ja miksi. Näin ollen kehitystiimin ulkopuolisetkin voivat saada hyödyllistä tietoa projektin etenemisestä. Tämän lisäksi Build Forge kykenee tehokkaasti hallinnoimaan useampia projekteja, jotka voivat olla esimerkiksi eri kielisiä, ja jotka voivat käyttää erilaisia käännöstyökaluja ja eri käännöspalvelimia. (Build Forge Briefing 2007.)

Build Forgen käyttöliittymän kautta voidaan asettaa kaikki työkalun toiminnan kannalta oleelliset asiat, kuten käyttäjät, serverit ja käännösohjeet. CruiseControlissa asetukset määritetään XML -muotoisessa konfiguraatiotiedostossa. Käytettävyydeltään työkalut eroavat siis jonkin verran ja käyttäjäkohtaisesti riippuu luonnistuuko XML sujuvasti vai onko käyttöliittymän kautta käyttö helpompaa. Oletuksena on hyvä ajatella, ettei käyttäjien tarvitsisi opetella mitään uutta syntaksia pystyäkseen käyttämään uutta työkalua. Build Forgen käyttöliittymä vaatii kuitenkin jonkin verran opettelua, sillä valikkoja ja nappeja on paljon, suoritetaanhan kaikki toimenpiteet tämän kautta.

Pelkästään jatkuvaa integrointia ja käännöksen automatisointia varten CruiseControl on riittävä työkalu. Build Forge pystyy paljon muuhun, ja sitä voi käyttää täyttämään usean eri projektin ja kehitystiimin tarpeet. ENIQ:n tai sen kehittäjien kannalta ei kuitenkaan ole tarpeellista pystyä hallitsemaan useita erilaisia projekteja, sillä kehitystyö tapahtuu vain kyseisen tuotteen parissa. Riittää, että tekijät näkevät tuotteen tilan ja saavat palautetta tekemistään muutoksista. Ilmaisena, yksinkertaisena ja helppokäyttöisenä tuotteena CruiseControl on valittu tuote suorittamaan ENIQ:n jatkuvaa integrointia.

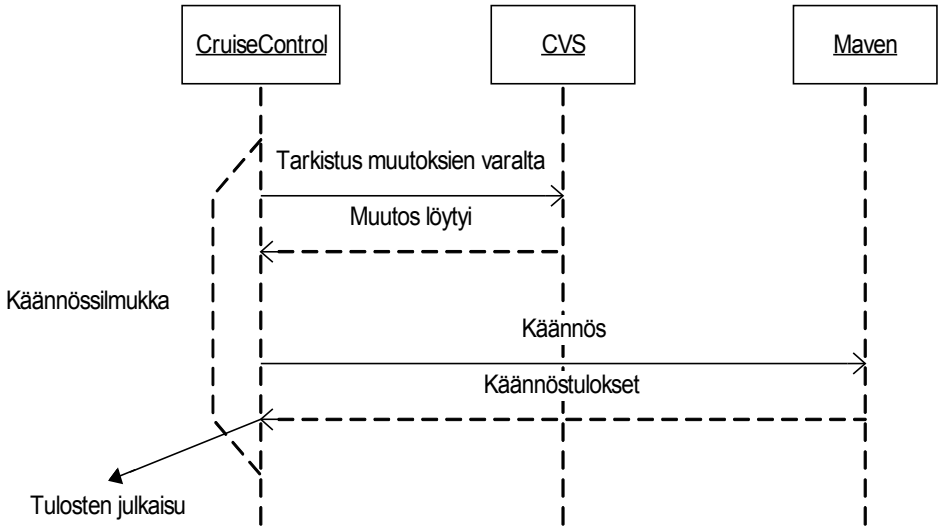
6 KÄÄNNÖS JA INTEGROINTIYMPÄRISTÖN TOTEUTTAMINEN

6.1 Toteutusympäristön määrittely

Automatisoitu käännös ja integrointiympäristö suunnitellaan ja toteutetaan ENIQ-järjestelmälle, joka kerää tietoa tietoliikenneverkosta analysointia ja raportointia varten. ENIQ on pääasiassa toteutettu Javalla ja koostuu useasta eri osasta, eli noudattaa modulaarista arkkitehtuuria. Näiden osien sekä niiden muodostaman kokonaisuuden käännöstä ja integrointia toteutettavan ympäristön tulee tukea. ENIQ:n osat löytyvät CVS-versionhallinnasta erilliseltä palvelimelta, jossa myös kääntäminen tapahtuu. Käännöstyökaluksi on valittu Maven -ohjelma, joka kääntää, testaa, pakkaa ja julkaisee yksittäisiä osia tai koko ohjelmiston. Käännetyt osat siirretään erilliselle palvelimelle, missä tapahtuu integrointitestaus. Tätä kaikkea hallinnoi CruiseControl, joka seuraa versionhallintaan tehtäviä muutoksia, kääntää muuttuneet osat ja siirtää ne integrointiserverille testausta varten. Kääntämisestä ja testauksesta syntyneet raportit kerätään yhteen, ja niitä voi tarkastella CruiseControlin WWW-käyttöliittymän kautta.

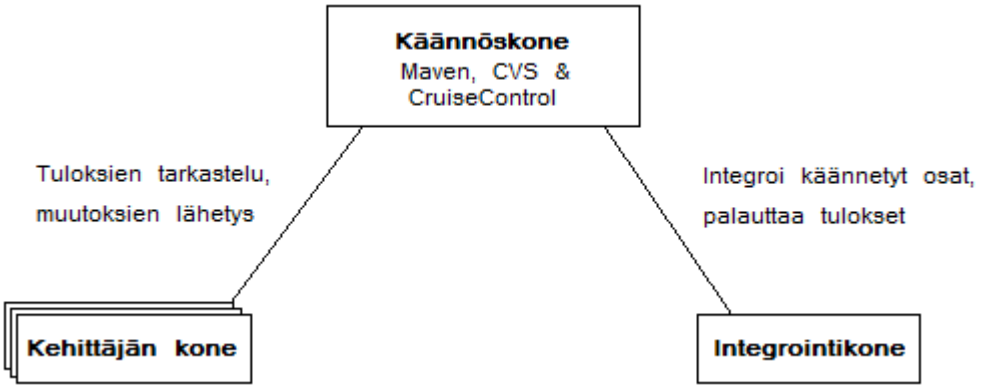
Toteutusympäristön vaatimat sovellukset ovat Maven (versio 2.1.0) sekä CruiseControl (versio 2.8.2), jotka asennetaan samalle palvelimelle käytössä olevan CVS-versionhallinnan kanssa. Tarkoituksena on keskittyä pääasiassa ENIQ:n kehitysversion kääntämiseen ja jatkuvaan integrointiin. Mahdollisuus yksittäisten moduulien sekä tuotteen vanhempien versioiden kääntämiseen on kuitenkin pidettävä mielessä erityisesti käännösskriptejä kirjoitettaessa. Raportoinneista tärkeimpinä ovat käännösaikaiset lokit sekä koodikattavuutta mittaavat Cobertura-raportit. Näitä raportteja, käännöksen tuloksia sekä tämän seurauksena syntyneitä tiedostoja käyttäjät pääsevät tarkistelemaan CruiseControlin käyttöliittymän kautta.

CruiseControl-, CVS- ja Maven -ohjelmat muodostavat ympäristön, joka toteuttaa tuotteen kääntämisen ja jatkuvan integroinnin. Kuvion 12 sekvenssikaavio kuvaa edellä mainittujen työkalujen välistä toimintaa. Kuviossa CruiseControlin käännössilmukka tarkkailee CVS:ää muutoksien varalta halutun määräajan välein. Mikäli muutoksia on tapahtunut, haetaan muuttuneet lähdetiedostot ja suoritetaan käännös. CruiseControlin käynnistämä Maven-skripti kääntää, testaa, raportoi, paketoi ja julkaisee haetuista lähdekoodeista syntyneet ENIQ:n moduulit testipalvelimelle integrointia varten. Koko käännösprosessista saatu palaute tulee CruiseControlille, minkä jälkeen tulokset julkaistaan webbikäyttöliittymässä.



KUVIO 12. Kääntämistä ja jatkuvaa integrointia kuvaava sekvenssikaavio

Yllä olevan toiminnan toteuttamista varten tarvitaan ENIQ:iä kuvaava Maven skripti, minkä tarkoituksena on suorittaa koko projektin kääntäminen testeineen aina asennettavaksi paketiksi asti. Myös koodikattavuutta laskevat Cobertura-raportit suoritetaan Maven-ohjelman toimesta. CruiseControlin yhteistoiminta Mavenin ja CVS:n kanssa on kuvattuna määrittelytiedostossa. Periaatteesa työskentely tapahtuu kahden XML-syntaksia käyttävän tiedoston voimin, mutta käytännössä muita erilaisia asetustiedostoja ja skriptejä tullaan tarvitsemaan.



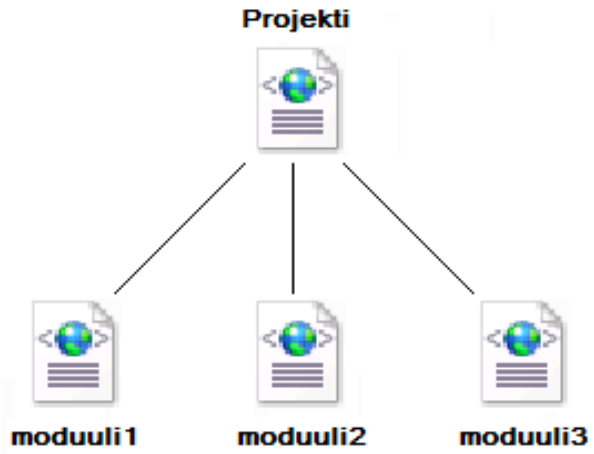
KUVIO 13. Käännöstä ja jatkuvaa integrointia toteuttavat koneet

Kuviossa 13 on kuvattu automatisoidun käännöksen ja jatkuvan integroinnin ympäristöä loogisten koneiden kannalta. Työssä ympäristön tarvitsemat ohjelmat ovat jaettuna kahdelle koneelle, minkä lisäksi kehittäjät voivat tarkastaa koneiltansa käännöksen tilan. Kuvion keskimäinen laatikko (käännöskone) sisältää kääntämisprosessin kannalta tärkeimmät ohjelmat. Tälle koneelle on asennettu Maven, CruiseControl sekä CVS. Versionhallinnan ei ole tarpeellista olla samalla koneella kääntämisympäristön kanssa, mutta tässä työssä toteutuksen yksinkertaistamiseksi ja vähempien resurssien käyttämiseksi näin on tehty. Tällä koneella on myös WWW-palvelin käännöstulosten julkaisua varten, joihin kehittäjät voivat päästä käsiksi. Kehittäjien lähettämät muutokset versionhallintaan käynnistävät kääntämisprosessin, joka kuten edellä mainittu, suoritetaan suurimmaksi osaksi käännöskoneella. Integrointia varten on oma koneensa, johon on asennettu ENIQ, sekä mahdolliset testausohjelmat eheyden varmistamiseksi. Käännetyt ja paketoidut osat lähetetään integrointikoneelle asennettavaksi ja testattavaksi. Kyseiseltä koneelta lähetetään tulokset takaisin käännöstä, jotka julkaistaan kehittäjien tarkasteltavaksi.

6.2 Maven-skripti

Maven-skriptissä on määritelty käännösohjeet yksittäiselle projektille. Näiden ohjeiden mukaan suoritetaan erilaisia käännösvaiheen toimenpiteitä, kuten yksikkötestit ja paketointi. Oletuksena Mavenin käännöksen elinkaareen kuuluvat

kaikki ENIQ:n osien kääntämiseen tarvittavat vaiheet, koodikattavuuden raportointia ja yksittäisen moduulin asennuspaketin kokoamista lukuun ottamatta. Edellä mainitut puutteet korvataan liitännäisillä. Tämän lisäksi oletus käännöselinkaareen kuuluviin vaiheisiin joudutaan tekemään pieniä muutoksia, koska ENIQ:n tiedostorakenne eroaa Mavenin olettamasta muodosta. Tarkoituksena on pitää skripti mahdollisimman yksinkertaisena käyttäen Mavenin oletustoimintoja ja -arvoja ymmärrettävyyden ja ylläpitämisen helpottamiseksi.



KUVIO 13. Käännösskriptin tiedostorakenne

Käännösskripti muodostuu ENIQ:iä kuvaavasta POM-tiedostosta. Tässä tiedostossa on määritelty kyseisen projektin yleiset tiedot ja riippuvuudet. ENIQ:n modulaarisen arkkitehtuurin vuoksi käännösskriptin tulee pystyä kääntämään tuote osissa. Jokaiselle moduulille on siis tehtävä oma POM-tiedostonsa. Maven tukee perintää, jolloin käännösskripti muodostuu yhdestä koko projektia kuvaavasta POM-tiedostosta ja tästä periytyvistä moduulikohtaisista POM-tiedostoista. Kuviossa 13 on havainnollistettu tämä tiedostorakenne. Projektia kuvaava POM-tiedosto sisältää ENIQ:iä yleisesti koskevien tietojen lisäksi sen moduulien yhteiset ominaisuudet sekä riippuvuudet eri liitännäisiin ja ulkoisiin kirjastoihin. Moduulikohtaisissa POM-tiedostoissa määritellään kyseisen osan riippuvuudet toisiin projektin moduuleihin ja mahdolliset ylimääräiset tehtävät, joita projektikohtaisessa tiedostossa ei ole määritelty. Maven selvittää käännösprosessin alussa automaattisesti moduulien väliset riippuvuudet ja

järjestelee osat käännösjärjestykseen. Projektin on näin ollen noudatettava hyväksi havaittuja ohjelmointikäytäntöjä eikä esimerkiksi ristikkäisriippuvuuksia sallita.

```
19    <!-- POM Relationships -->
20
21    <groupId>com.ericsson.eniq</groupId>
22    <artifactId>eniq</artifactId>
23    <version>2.0</version>
24    <modules>
25        <module>../common_utilities/dev</module>
26        <module>../etl_controller/dev/engine</module>
27        <module>../mediation/dev/export</module>
28    </modules>
29    <dependencies>
30        <dependency>
31            <groupId>dbunit</groupId>
32            <artifactId>dbunit</artifactId>
33            <version>2.1</version>
34        </dependency>
35        <dependency>
36            <groupId>junit</groupId>
37            <artifactId>junit</artifactId>
38            <version>4.3.1</version>
39        </dependency>
40        <dependency>
```

KUVIO 14. ENIQ:n POM –tiedoston riippuvuussuhteita

ENIQ:iä kuvaava POM-tiedosto muodostuu neljästä osasta: projektin yleiset tiedot, riippuvuussuhteet, käännösasetukset ja -ympäristö. Projektin yleiset tiedot sisältävät tietoa yrityksestä ja tuotteesta, jotka eivät vaikuta itse käännökseen. Käännösympäristön määrittelyjä, kuten versionhallinnan määrittelyt, ei käytetä, koska ne tehdään CruiseControlin puolelta. Varsinainen tominnallisuus on määritelty riippuvuussuhteissa ja käännösasetuksissa. Kuviossa 14 on ote ENIQ:n POM-tiedoston riippuvuussuhteiden määrittelyosasta. Riveillä 21-23 on kyseessä olevan POM-tiedoston tunnistealkiot. Näiden jälkeen määritellään modules-alkiossa tämän POM-tiedoston aliprojektit, eli käännettävät moduulit. Jokaisella osalla on oma POM-tiedostonsa, jossa on määritelty moduulikohtaisia asetuksia ja riippuvuuksia. Riviltä 29 alkaen on määritelty projektin yleisiä riippuvuuksia ulkoisista kirjastoista. Tämän lisäksi moduulikohtaisissa POM-tiedostoissa on määritelty kyseisen osan riippuvuudet ulkoisista ja ENIQ:n muista osista. Maven-ohjelma selvittää käännösjärjestyksen näiden riippuvuuksien perusteella.

```
63    <!-- Build Settings -->
64
65    <packaging>pom</packaging>
66    <build>
67        <plugins>
68            <plugin>
69                    <groupId>org.apache.maven.plugins</groupId>
70                    <artifactId>maven-jar-plugin</artifactId>
71                    <version>2.2</version>
72                    <configuration>
73                        <outputDirectory>./dclib/</outputDirectory>
74                    </configuration>
75                </plugin>
76            <plugin>
```

KUVIO 15. ENIQ:n POM-tiedoston käännösasetuksia

Kuviossa 15 on ote ENIQ:n käännösasetuksista. Riviltä 67 lähtien on määritelty liitännäiset, joita Maven-ohjelma käyttää käännöksessä. Liitännäiset ovat ikään kuin pieniä ohjelmia, joita suoritetaan käännöksen aikana. Eräät liitännäiset ovat valmiina oletus käännöselinkaaressa, toiset pitää erikseen määritellä suoritettavaksi johonkin elinkaaren vaiheeseen. Tämän lisäksi useita liitännäisiä pystyy muokkamaan configuration-alkiossa tai vastaavassa. Kuvion 15 rivien 68-75 liitännäinen määrittelee .jar paketointioperaation. Erikoisasetuksena rivillä 73 on määritelty hakemisto, jonne valmis paketti siirretään. Muita tällaisia liitännäisiä ovat muun muassa testaukseen, raportointiin ja siivoukseen liittyvät liitännäiset.

Toimiakseen Maven-skripti tarvitsee muutaman asetustiedoston sekä tietovaraston kolmannen osapuolen ja käännettyjen kirjastojen varalle. Viimeksi mainitun Maven luo itsenäisesti käyttäjän juurihakemistoon. Tietovarasto olisi mahdollista tehdä keskitetysti yhteen paikkaan, jota kaikki käyttäjät pystyisivät käyttämään. Näin säästettäisiin yksittäisen käyttäjän kannalta tilaa, kun jokaisen ei tarvitsisi säilöä kirjastoja omalla koneellaan. Keskitetty tietovarasto mahdollistaa myös yrityksen sisäisten kirjastojen julkaisun ja jakelun kehittäjille. Käyttäjäkohtainen tietovarasto on helpommin muokattavissa esimerkiksi kirjastojen eri versioiden kokeiluun ja on aina käyttäjän tavoitettavissa. Jatkuvan integroinnin kannalta ei

tietovaraston tyypillä ole väliä, mutta jatkokehityksen kannalta yhteinen tietovarasto on harkinnan arvoinen ajatus.

Mavenin asetustiedostossa on mahdollista määrittää erilaisia käyttäjäkohtaisia- sekä yleisiä asetuksia. Tärkeimpinä asetuksina ovat tietovarastojen sijainnit, proxyt sekä ulkoisten palvelimien määrittelyt. ENIQ:n kääntämiseen riittää oletuksena Apachen ylläpitämästä Mavenin keskitetystä tietovarastosta paikalliseen käyttöön haetut kirjastot. Tähän keskitettyyn tietovarastoon pääsy yrityksen verkosta tapahtuu proxyn kautta. Myös integrointipalvelin, missä ENIQ:n integrointi tapahtuu, määritellään asetustiedostossa.

Käännösskripti asetustiedostoineen tulee suorittaa oletus käännöselinkaaren kaikki vaiheet aina julkaisuvaiheeseen asti. Näihin vaiheisiin liitetään myös koodikattavuusraportointi sekä asennuspaketin kokoaminen, jolloin tuloksena on ENIQ:n vaatimukset täyttävä käännösskripti jatkuvaa integrointia varten. Tällä Maven -skriptillä on myös mahdollista kääntää manuaalisesti käyttäen edellä mainittuja komentoja. Versionhallintaliitännäisen avustuksella myös vanhempien versioiden kääntäminen uudelleen on mahdollista. Versionhallinassa olevat eri ohjelmiston versiot tarvitsee leimata tämän toteuttamiseksi. Näiden leimojen avulla on siten mahdollista kääntää haluttu vanhempi versio.

6.3 CruiseControl asetukset

Jatkuvaa integrointia suorittava CruiseControl vaatii toimiakseen projektin käännöstyökaluineen sekä asetustiedoston. Tässä tiedostossa määritellään yhteys versionhallintaan, käännöstyökalulle annettavat komennot, käännöksien ajoitus sekä käännöstulosten julkistaminen. ENIQ:n muodostavat moduulit käännöstiedostoineen haetaan versionhallinnasta paikalliseen hakemistoon, missä kääntämisprosessi tapahtuu. Tätä paikallista kopiota CruiseControl tarkkailee viiden minuutin välein ja jos muutoksia versionhallinnassa tapahtuu, käännös laukaistaan pienen viiveen jälkeen. Kyseisen viiveen tarkoituksena on odottaa, että kehittäjä on varmasti lähettänyt kaikki tiedostot versionhallintaan. Periaatteessa

tarkistuksien välisellä ajalla ei ole väliä, sillä aloitettu käännösprosessi viedään loppuun vaikka tämän aikana muutoksia lähetettäisiinkin versionhallintaan. Uusi käännös aloitetaan tämän jälkeen edellisen käännöksen aikana lähetettyjen muutoksien kera. Jatkuvan integroinnin määritelmien mukaan integrointia on suoritettava mahdollisimman usein, jolloin valittiin aika, minkä sisään muutoksia on realistista odottaa.

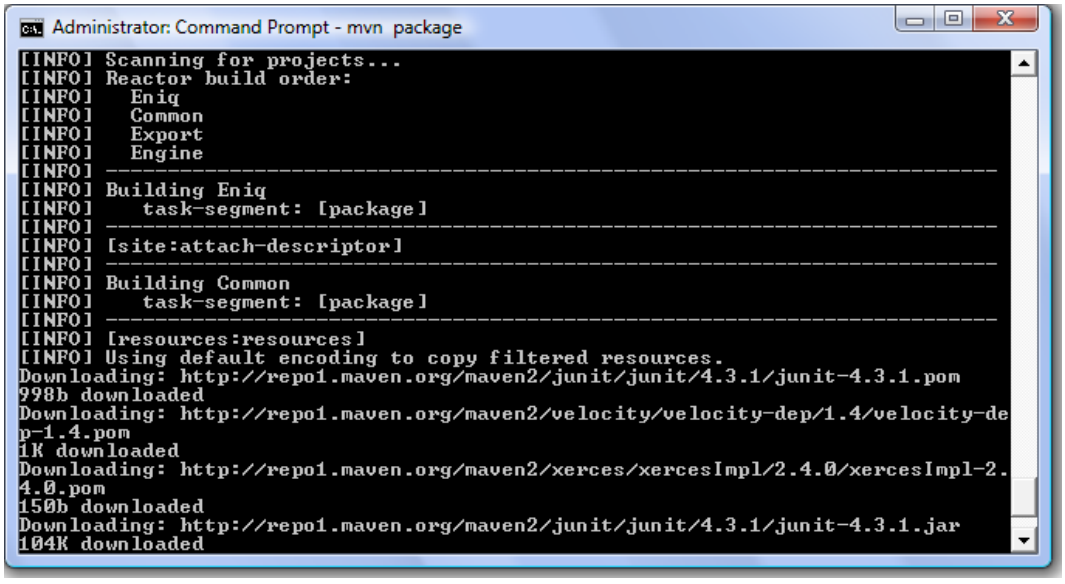
Asetustiedostossa määritellään myös, mitä Maven-komentoja, eli maaleja, käännöksessä suoritetaan. ENIQ:n tapauksessa ensin suoritetaan siivous, joka poistaa edellisestä käännöksestä syntyneet tiedostot, minkä jälkeen kutsutaan julkaisu maalia. Koska julkaisu on viimeisin oletus käännöselinkaaren maaleista, ei sitä edeltäviä maaleja, kuten testausta ja pakkausta, tarvitse erikseen kutsua. Erinäiset liitännäiset voidaan yhdistää suoritettaviksi johonkin elinkaaren vaiheeseen, jolloin myös nämä suoritetaan ilman erillistä komentoa. Näin on asennuspaketin kokoamisen tapauksessa. Tämän jälkeen ajetaan käännöselinkaareen kuulumaton maali, site, joka luo erilaisia raportteja projektista verkkojulkaisua varten. Tähän maaliin liitetään myös Cobertura koodikattavuusraportointi, joka luo HTML-muotoon tehdyn moduulikohtaisen raportin. Jotta moduulikohtaiset kattavuusraportit saadaan koottua yhdeksi raportiksi, kutsutaan viimeisenä kojelauta –liitännäistä (dashboard).

Käännöstuloksien julkaisu tapahtuu sille tarkoitetulla sivustolla. Jotta kattavuusraportit saadaan näkyville, määritellään asetustiedostossa halutut tiedostot tai hakemistot julkaisua varten. CruiseControl:ssa olevat JSP- ja kojelauta-raportointisivut ovat tuloksien julkaisua varten, joista edellä mainittu julkaisee tarkempaa tietoa käännösprosessista ja jälkimmäinen antaa hyvän yleiskuvan projektin tilasta. Kojelautaan muutoksia ei tehdä, sillä oletusarvoisesti se antaa hyvän kuvan projektin tilasta. JSP-raportointisivulle puolestaan tehdään muutama JSP-sivu lisää Cobertura-koodikattavuutta sekä projektin yleistä tietoa sisältävää sivua varten. Asetustiedostoon määritellään myös kohde, joka poistaa vanhat raportit, jotta CruiseControlia ajavan koneen levy ei täyty.

6.4 Ympäristön toiminta

6.4.1 Maven käännös

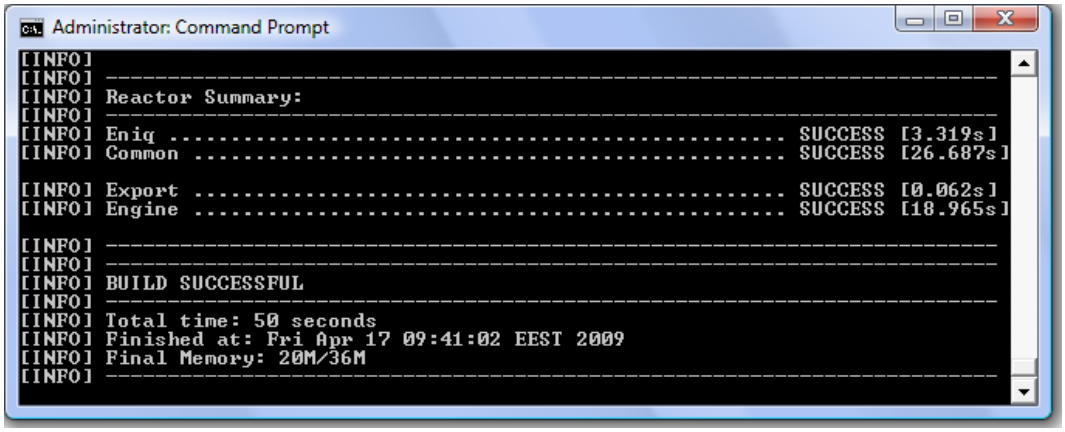
Kun tarvittavat ohjelmat on asennettu, niiden asetustiedostot tehty ja projekti haettu versionhallinnasta, käynnistetään CruiseControl, mikä laukaisee ensimmäisen käännöksen. Maven-ohjelma selvittää ENIQ:n moduulien kääntöjärjestyksen ja aloittaa tarpeellisten kolmannen osapuolten kirjastojen sekä erilaisten liitännäisten lataamisen Maven-ohjelman keskitetystä tietovarastosta paikalliseen tietovarastoon. Tämä lataus tapahtuu vain kertaalleen, sillä Maven pyrkii hakemaan riippuvuudet paikallisesta tietovarastosta ja vain mikäli niitä ei sieltä löydy, otetaan yhteys ulkoiseen tietovarastoon. ENIQ:n keskinäiset riippuvuudet käsitellään kääntöjärjestyksellä, jolloin moduulit, jotka ovat täysin riippumattomia toisista, käännetään ensimmäisenä. Näiden käännetyt ja paketoidut JAR-paketit viedään paikalliseen tietovarastoon seuraavien kyseisistä moduuleista riippuvaisten osien käytettäviksi. Tämä tarkoittaa, että ristikkäisriippuvuuksia ei saa olla tai käännös keskeytetään.



KUVIO 16. Maven loki käännöstä aloittaessa

Kuviossa 16 on esitetty ensimmäinen käännös Maven-ohjelman pakkaus komennolla. Käännös alkaa etsimällä käännettävä projekti, eli käytännössä

Maven-ohjelma etsii POM-tiedostoa hakemistosta, josta komento suoritettiin. ENIQ:iä kuvaava projektitiedosto löytyy, minkä perusteella käännösjärjestys tehdään. Ensimmäisenä käännetään koko projektia kuvaava ENIQ-tiedosto, mitä seuraa itse lähdekoodia sisältävät moduulit Common, Export ja Engine. Koska ENIQ on abstrakti kuvaus koko projektista, eikä se siis sisällä varsinaista lähdekoodia, eli varsinaista käännöstä ei tapahdu. Tämän jälkeen siirrytään seuraavaksi käännettävään moduuliin, nimeltään Common. Kuvion 16 alaosassa on lokia ladatuista riippuvuuksista, joita kyseinen osa tarvitsee. Kun tarvittavat riippuvuudet on ladattu tapahtuu itse käännös lähdekoodista tavukoodiksi niin varsinaisen koodin osalta, kuin testiluokista. Kaikki testit suoritetaan, tavukoodi pakataan JAR-paketiksi ja lähetetään paikalliseen tietovarastoon seuraavien moduulien käytettäviksi. Tätä toistetaan, kunnes koko projekti on käännetty.



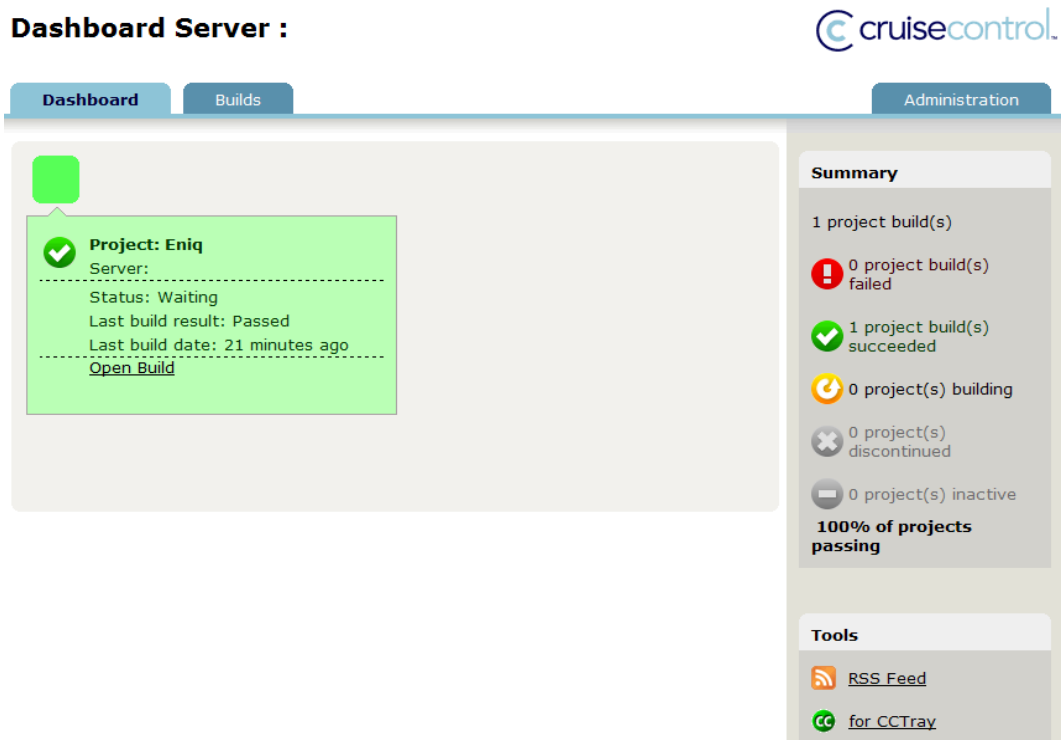
KUVIO 17. Maven loki päättyneestä käännöksestä

Päättyneen käännöksen loki on kuvattuna kuviossa 17. Projektin käännöstulokset luetellaan moduulikohtaisesti järjestyksessä käännöstuloksineen ja -aikoineen. Lokin lopussa on yhteenveto koko käännöksestä. Tässä yhteydessä käännöksestä puhuttaessa tarkoitetaan myös yksikkö- ja integrointitestien suorittamista, eikä pelkästään lähdekoodien kääntämisestä ja linkittämisestä toimivaksi ohjelmaksi. Kuviosta voidaan huomata, että abstraktin ENIQ-projektin ja Export-nimisen moduulin käännökseen ei juuri ole kulunut aikaa. Tämä johtuu testien puutteesta. Erilaiset testit ja niiden raportointi kuluttuvat eniten aikaa käännösprosessissa, mikä kannattaa pitää mielessä testejä luodessa. Testauksen laadun sekä määrän ja jatkuvan integroinnin välillä tasapainoilu on avainasemassa automatisoidun

käännöksen ja jatkuvan integroinnin ympäristöä luodessa. Testauksessa ei missään nimessä pitäisi säästellä, mutta liian pitkät käännökset eivät toteuta jatkuvan integroinnin periaatteita, missä palautetta pitäisi saada kehittäjille mahdollisimman nopeasti. Hyvä keskitien ratkaisu on määritellä kriittisimmät testit suoritettaviksi jatkuvan integroinnin aikana ja ajaa koko testiarsenaali kerran vuorokaudessa niin kutsutuissa ”nightly build” -käännöksissä.

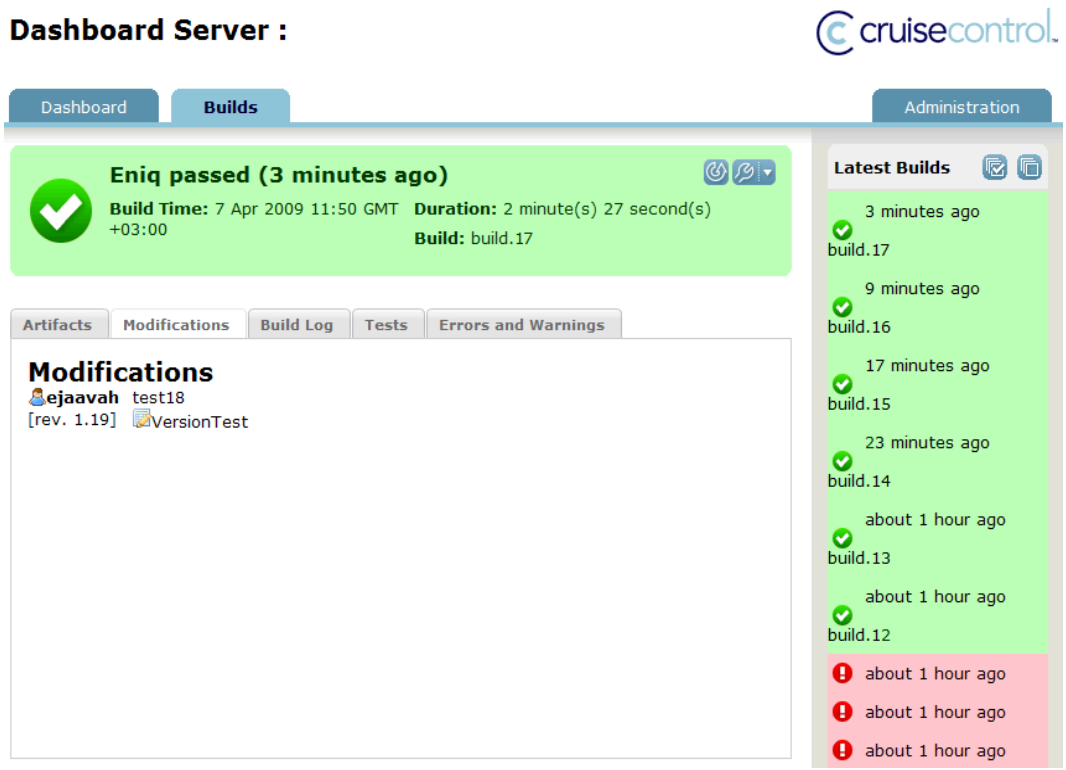
6.4.2 Tuloksien tarkastelu

CruiseControlin käynnistyksen yhteydessä tehdyn ensimmäisen käännöksen tulokset löytyvät nyt raportointisivustolta asetustiedostoissa tehtyjen määrittelyiden mukaan. Jatkuva integrointi jatkuu tämän jälkeen normaalisti, eli versionhallintaa seurataan muutoksien varalta viiden minuutin välein. Mikäli muutoksia on tapahtunut, käynnistyy käännösprosessi uudestaan. CruiseControl tallentaa kaikkien käännöksien tulokset, jolloin kehittäjät pääsevät käsiksi sekä viimeisimmän, että tätä edeltävien käännöksien raportteihin. Kyseisiä raportteja säilytetään levyllä tietyn aikaa, kunnes ne automaattisesti poistetaan CruiseControlin toimesta.



KUVIO 18. Kojelaudan raportointisivu

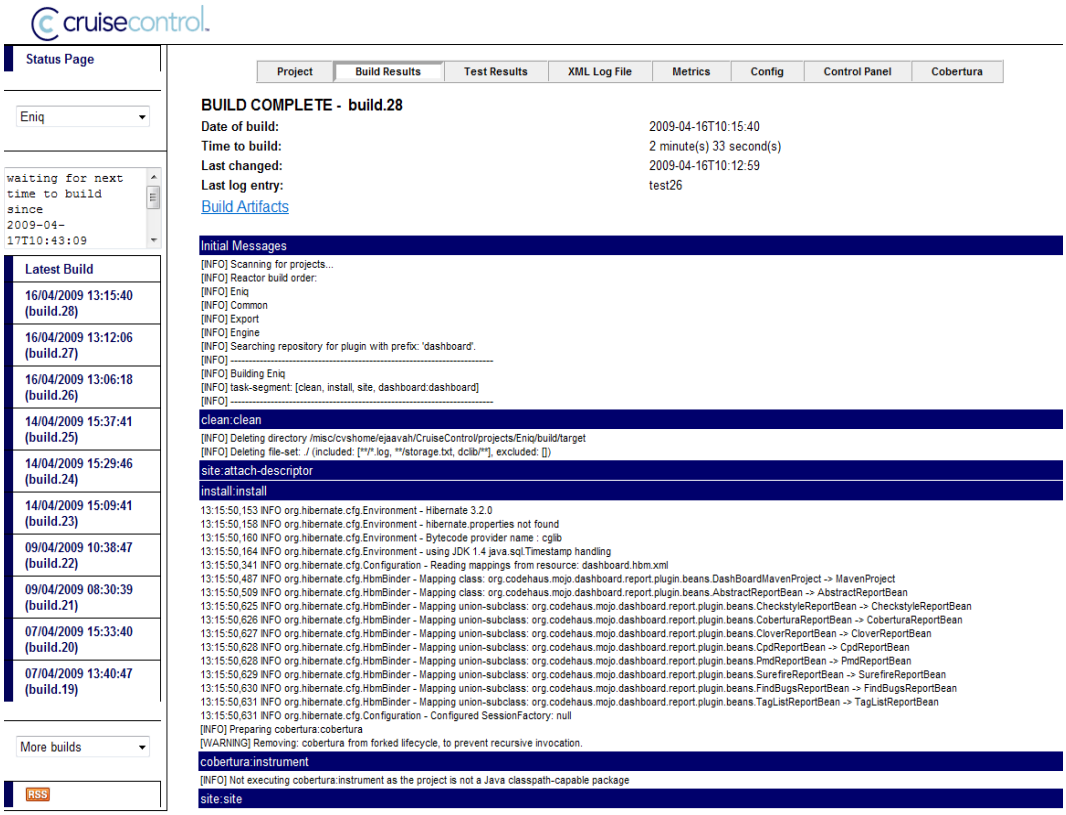
Hyvän yleiskatsauksen käännöksien tilaan saa CruiseControlin kojelauta (Dashboard) raportointisivulta. Kuviossa 18 on kojelaudan etusivu, mistä nähdään käännöksien yleistila. Vasemman yläkulman vihreä laatikko tarkoittaa käännöstilan olevan hyvä, eli viimeisin käännös on onnistunut. Mikäli viimeisin käännös olisi epäonnistunut, väri olisi punainen. Näin kehittäjät saavat nopealla vilkaisulla tietää, mikä on käännöksen tila sillä hetkellä. Siirtämällä kursori laatikon ylle, avautuu puhekuplamainen laatikko, mikä kertoo tilasta hieman tarkemmin, kuten milloin viimeisin käännös on tapahtunut. Mikäli projekteja olisi useampia, myös näitä värilaatikoita olisi rivissä enemmän. Lisätietoa vimmeisimmästä sekä tätä aikaisemmista käännöksistä saa klikkaamalla laatikkoa tai Builds-välilehdestä.



KUVIO 19. Kojelaudan tarkempi näkymä käännöksistä

Kuviossa 19 näkyvät tarkemmat tiedot kertovat viimeisimmän ja sitä aiempien käännöksien käännöstuloksia, kuten luodut tiedostot (Artifacts -välilehti), muutokset (Modifications), käännösloki (Build Log), testit (Tests) sekä virheet ja varoitukset (Errors and Warnings). Luoduissa tiedostoissa on listattuna kaikki

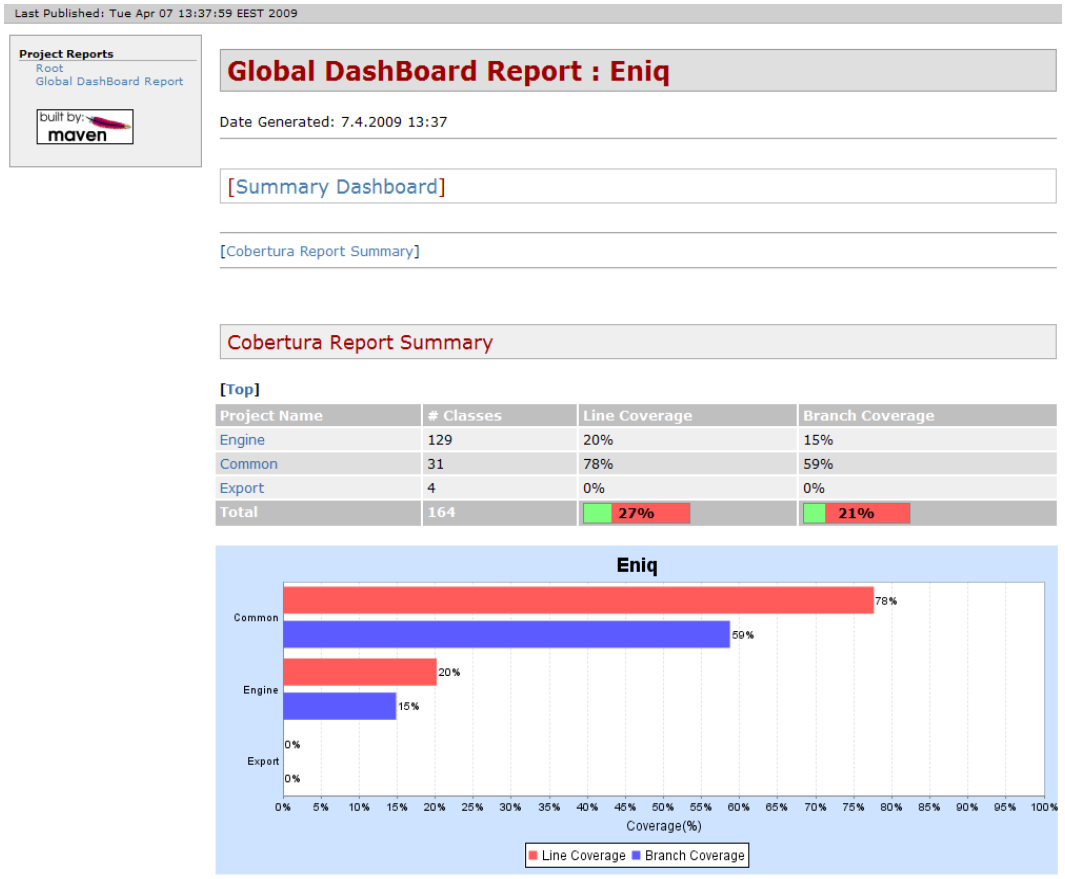
käännöksen aikana syntyneet tiedostot, mitkä on CruiseControlin asetuksissa määritelty julkaistaviksi. ENIQ:n julkaistaviin tiedostoihin kuuluvat muun muassa koodikattavuusraportit. Kuviossa avoinna oleva muutosvälilehti kertoo versionhallintaan tehdyt muutokset, jotka laukaisivat kyseisen käännöksen. Muutoksien tiedoissa näkyvät muutoksen tehneen kehittäjän tunnus, versionhallintaan päivitetty tiedosto viesteineen sekä versioineen. Käännösloki-, testi- sekä virheet ja varoitukset -välilehdet selittävät itsensä. Oikealla palstalla (Latest Builds) löytyy viimeisimmät käännökset. Myös näiden tarkemmat käännöstiedot ovat kehittäjien tarkasteltavissa hiiren painalluksella.



KUVIO 20. JSP-raportointisivuston käännösloki

CruiseControlin JSP-sivut antavat tarkempaa tietoa käännöksistä. Kuviossa 20 on kuvakaappaus JSP-raportointisivustosta, minkä muokattavuus on kojelautaa parempi. Ylhäällä näkyvän valikon nappeja voi käyttäjä lisätä kirjoittamalla uusia JSP-sivuja. ENIQ:lle on tässä kuvassa lisätty yleistietoa antava Project -sivu sekä koodikattavuusraportin sisältävä Cobertura-sivu. Muut sivut ovat oletuksena

käytössä, kuten kuvassa esillä oleva Build Results -sivu, mikä sisältää täydellisen raportin käännöksestä.



KUVIO 21. JSP-raportointisivun koodikattavuusraportti

Testauksen laatua on yleisesti hankala mitata, eikä vedenpitävää tapaa ole. Yksikkötesteissä on yleistynyt mittaustapa, jossa lasketaan koodirivit, jotka testikoodi käy läpi. Cobertura on eräs tällaista mittausta suorittava teknologia, ja se on liitettynä ENIQ:n laadunvarmistuksen mittariksi. Mitä enemmän koodia testien toimesta käydään läpi, sitä todennäköisemmin mahdolliset viat löydetään. Näitä raportteja voi tarkastella CruiseControlin JSP-raportointisivulta, mikä on kuvattuna kuviossa 21. Kuvassa on moduulikohtaisesti kattavuuksien prosenttiluvut sekä näistä tehty pylväsdiagrammi, joista punainen pylväs kertoo rivikattavuuden ja sininen ilmoittaa haarakattavuuden moduuleittain.

7 YHTEENVETO

Tässä työssä on käsitelty automatisoidun käännöksen ja jatkuvan integroinnin ympäristön toteuttamista Java -ohjelmistotuotteelle. Kyseinen tuote on Ericssonin työkalu verkkoliikenteen seurantaan ja raportointiin, nimeltään ENIQ. Tarkoituksena on ollut tutkia tämän ympäristön toteuttamiseen soveltuvia teknologioita ja toteuttaa kyseinen ympäristö näiden avulla.

Ohjelmistotuotteen kääntäminen ja integrointi liittyy oleellisesti tuottenhallintaan kokonaisuudessaan. Ohjelmisto koostuu moduuleista, jotka yhdessä muodostavat erilaisia konfiguraatiota. Moduulit ja konfiguraatiot versioituvat sitä mukaa kun niihin tehdään muutoksia. Usein muutokset näihin osiin voivat johtaa kehityksen haarauttamiseen, jotta tuote vastaa paremmin esimerkiksi jonkun käyttöliittymän tai asiakkaan tarpeisiin. Versionhallinnan vastuulla on eri versioiden ja haarojen tunnistaminen. Näiden avulla kääntäminen myöhemmissä vaiheissa on mahdollista.

Kääntäminen on yksinkertaisimmillaan lähdekoodin muuntamista suoritettavaksi ohjelmaksi. Tähän vaiheeseen liitty erilaisia vaiheita, kuten testejä ja linkittämistä. Kääntämistyö on manuaalisesti vaivalloista isoissa ohjelmistoissa, jolloin on syntynyt erilaisia kännöstyökaluja, jotka ovat automatisoineet käännösprosessin. Kääntämiseen usein liitetty vaihe on integrointi, missä varmistetaan ohjelmistotuotteen eri osien yhteistoiminta. Toimintatapa, jossa kehittäjät integroivat työtänsä usein, kutsutaan jatkuvaksi integraatioksi.

Useista erilaisista automatisoiduista käännösteknologioista on vertailtu Makea, Antia sekä Mavenia, joista soveltuvin valitaan ENIQ:n käännöstyökaluksi. Make on perinteinen työkalu ohjelmistojen kääntämiseen, mutta sen alustariippuvaisuuden sekä hankalan syntaksin vuoksi se ei ole soveltuva työkalu käännöksen suorittamiseen. Todellinen valinta on tehty Ant ja Maven -ohjelmien välillä, joista molemmat käyttävät XML-syntaksia käännösskripteissään ja ovat

Javalle suunnattuja käännöstyökaluja. Maven-ohjelma on valittu näistä kahdesta soveltuvammaksi käännöstyökaluksi riippuvaisuuksien hallinan sekä koko projektia kuvaavan käännösskriptinsä vuoksi.

CruiseControl ja Build Forge ovat olleet vertailussa ENIQ:n jatkuvaa integrointia suorittaviksi tuotteiksi. Molemmat toimivat pääpiirteittäin samaa tapaan välittäen komentoja eri työkaluille ja keräten näistä saatua tietoa kehittäjien tarkasteltaviksi. Build Forgessa on tuki useiden käyttäjien ja projektien hallintaan, mutta CruiseControlin helppokäyttöisyys sekä ilmainen lisenssi tekivät CruiseControlista houkuttelevamman valinnan.

Toteutusympäristön versionhallintana toimii CVS, jonka kanssa samalle koneelle asennetaan CruiseControl sekä Maven-ohjelma. Edellämainittu tarkistaa versionhallinan muutoksien varalta tietyin väliajoin, ja mikäli muutoksia on tehty, Maven käännös käynnistyy. Tässä prosessissa ENIQ:n osat käännetään tavukoodiksi, testataan, pakataan jar -paketiksi, julkaistaan integrointiserverille testattavaksi sekä raportoidaan koko prosessi. Käännöksen tulokset voi tämän jälkeen tarkastaa CruiseControlin raportointisivujen kautta.

Ympäristön toteuttaminen kokonaisuudessaan on onnistunut hyvin. Käytetyistä teknologioista löytyy kattavasti tietoa ja erilaisia esimerkkejä sekä ongelmiin paljon ratkaisuehdotuksia. Varsinaiset ongelmat ovat liittyneet hieman ohi aiheen, suurimpana ENIQ:n ristikkäisriippuvuudet, joiden takia Maven -ohjelman käännös ei onnistunut ilman lähdekoodin uudellenjärjestelyä. Toteutettu ympäristö on hyvä pohja jatkokehitykselle, kuten esimerkiksi sähköposti-ilmoituksille ja nightly build -kokonaiskäännöksille. Nykyisellään toteutus mahdollistaa kuitenkin ennalta määritellyn toiminnan jatkuvaa integrointia ja siitä raportointia varten.

LÄHTEET

Apache Ant. 2009. Wikipedia. [viitattu 29.1.2009]. Saatavissa: http://en.wikipedia.org/wiki/Apache_Ant

Apache Ant FAQ. 2009. Apache Ant. [viitattu 29.1.2009]. Saatavissa: http://ant.apache.org/faq.html

Apache Ant Manual. 2009. Apache Ant. [viitattu 29.1.2009]. Saatavissa: http://ant.apache.org/manual/index.html

Apache Maven: Getting Started. 2009. Apache Maven. [viitattu 13.2.2009]. Saatavissa: http://maven.apache.org/guides/getting-started/index.html

Apache Maven: Introduction to the Lifecycle. 2009. Apache Maven. [viitattu 13.2.2009]. Saatavissa: http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

Apache Maven Simplifies the Java Build Process. 2003. DevX, Dave Ford. [viitattu 12.2.2009]. Saatavissa: http://www.devx.com/Java/Article/17204

Apache Maven: What is Maven?. 2009. Apache Maven. [viitattu 12.2.2009]. Saatavissa: http://maven.apache.org/what-is-maven.html

Build Forge Briefing. 2007. Redmonk. [viitattu 12.3.2009]. Saatavissa: http://www.redmonk.com/cote/2007/01/10/build-forge-briefing/

Continuous Integration. 2006. Martin Fowler. [viitattu 5.2.2009]. Saatavissa: http://www.martinfowler.com/articles/continuousIntegration.html

Continuous Integration Is an Attitude Not a Tool. 2005. James Shore. [viitattu 5.2.2009]. Saatavissa: http://jamesshore.com/Blog/Continuous-Integration-is-an-Attitude.html

Cooper K. & Torczon L. 2004. Engineering a Compiler. Elsevier.

CruiseControl. 2009. Wikipedia. [viitattu 3.3.2009]. Saatavissa: http://en.wikipedia.org/wiki/CruiseControl

CruiseControl overview. 2009. SourceForge. [viitattu 3.3.2009]. Saatavissa: http://cruisecontrol.sourceforge.net/overview.html

Grady J. 1994. System Integration. 2. painos. CRC Press

Haikala I. & Märijärvi J. 2004. Ohjelmistotuotanto.10. painos. Talentum, Helsinki.

Hightower R., Onstine W., Visan P., Payne D., Gradecki J., Rhodes K., Watkins R., Meade E. 2004. Professional Java Tools for Extreme Programming: Ant, Xdoclet, JUnit, Cactus, and Maven. John Wiley and Sons.

How To – Makefiles. 2009. Swathmore College. [viitattu 10.2.2009]. Saatavissa: http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html

IBM Rational Build Forge. 2007. CM Crossroads. [viitattu 12.3.2009]. Saatavissa: http://www.cmcrossroads.com/content/view/8048/570/

IBM Rational Build Forge Enterprise Edition. 2009. IBM. [viitattu 12.3.2009]. Saatavissa: http://www-01.ibm.com/software/awdtools/buildforge/enterprise/index.html

IBM Rational Build Forge Express Edition. 2009. Component Source. [viitattu 29.5.2009]. Saatavissa: http://www.componentsource.com/products/ibm-rational-build-forge-express/index-es.html

Java Tutorial. 2009. Sun Microsystems, Java Tutorial. [viitattu 4.2.2009]. Saatavissa: http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

Make. 2009. Wikipedia. [viitattu 10.2.2009]. Saatavissa: http://en.wikipedia.org/wiki/Make_(software)

Make – Tutorial. 2009. Hawaii University, College of Engineering. [viitattu 10.2.2009]. Saatavissa: http://www.eng.hawaii.edu/Tutor/Make/1-1.html

Matyas S, Duvall P., Matyas S., Schneider N., Glover A. & Voit A. 2007. Continuous Integration: Improving Software quality and Reducing Risk. Addison-Wesley.

Schildt H. 2005. Java: The Complete Reference. 6. painos. McGraw-Hill Professional

Suomen Ericsson. 2009. Oy LM Ericsson. [viitattu 29.5.2009]. Saatavissa: http://www.ericsson.com/fi/ericsson/summary.shtml

The Maven 2 POM demystified. 2006. Javaworld, Eric Redmond. [viitattu 13.2.2009]. Saatavissa: http://www.javaworld.com/javaworld/jw-05-2006/jw-0529-maven.html

Tuotteen Hallinta. 2007. Joensuun Yliopisto. [viitattu 9.2.2009]. Saatavissa: http://cs.joensuu.fi/tSoft/tuothall.htm