HAAGA-HELIA
ammattikorkeakoulu

Apache Maven -migraatio

Kai Kirjavainen

Opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

2013

HAAGA-HELIA
ammattikorkeakoulu

Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

12.5.2013

Tekijä tai tekijät Kai Kirjavainen	**Ryhmätunnus tai aloitusvuosi** 2008
Raportin nimi Apache Maven -migraatio	**Sivu- ja liitesivumäärä** 24 + 3
Opettajat tai ohjaajat Sauli Isonikkilä	

Tämän opinnäytetyön tavoitteena oli selvittää, kuinka Apache Maven -koontityökalu voidaan ottaa käyttöön osaksi ProCountor International Oy:n sovelluskehitystä. Työn lähtökohtana oli selvittää nykyisen työkalun, Apache Antin, toiminnallisuus ja käytössä oleva konfiguraatio. Koska tarkoituksena ei ollut tehdä suuria muutoksia yrityksessä käytössä oleviin prosesseihin, oli työn kannalta tärkeää ymmärtää lähtötilanne, josta Mavenin käyttöönotto alkoi.

Työn sisältö rajattiin käsittämään pelkästään Mavenin käyttöönottoon ja siihen liittyviin valmisteluihin. Maven mahdollistaa apuohjelmien kautta helposti mm. jatkuvan integraation (CI) ja automatisoidun yksikkötestauksen. Näiden käyttöönotto päätettiin kuitenkin jättää tämän työn ulkopuolelle, jotta kokonaisuus ei kasvaisi liian isoksi. Maveniin on myös mahdollista yhdistää palvelinohjelmisto, jolla yrityksen sisäisten Java-kirjastojen hallinta, päivitys ja jakelu yksinkertaistuu. Palvelimen asennus sisällytettiin projektiin, koska sen käyttöönotto oli yksinkertaista ja saadut lisähyödyt ilmeisiä.

Työn toteutukseen käytettiin yrityksen ohjelmistokehitysympäristöä, jonka tärkeimmät osat tämän projektin kannalta olivat ohjelmointiympäristö Eclipse, koontityökalu Ant ja SVN-versionhallinta. Apache Mavenista käyttöönottoa varten valittiin uusin versio 3, koska uuden työkalun käyttöönotossa on perusteltua valita versio, joka ei ole vanhentunut jo käyttöönottovaiheessa.

Käyttöönottoprojektin eteneminen eteni pääpiirteittäin tutustumalla nykytilanteeseen, eli käytössä olevaan Ant-työkaluun ja sen eri ominaisuuksiin sekä konventioihin. Tämän jälkeen hankittiin käsitys uuden- ja vanhan työkalun eroista ja yhtäläisyyksistä, jotta toteutukseen vaadittavista toimenpiteistä saataisiin selkeä käsitys. Seuraavaksi tehtiin varsinainen toteutus, jonka aikana ProCountor -ohjelmiston projektirakenne sovitettiin Mavenille sopivaksi. Lopuksi käyttöönotto ohjelmointiympäristössä dokumentoitiin, jotta siirtymä olisi koko kehitystiimin osalta mahdollisimman suoraviivaista.

Työn tuloksena oli selvitys siitä, mitä vaatimuksia edellä kuvattu siirtymä asettaa yrityksen ohjelmistokehitysprosesseille ja lisäksi toimiva produkti, joka voitiin luovuttaa yrityksen käyttöön.

Asiasanat
Apache Maven, ohjelmistokehitys, ohjelmointiympäristö, Java

HAAGA-HELIA
University of Applied Sciences

Abstract

Date of presentation

Degree programme

Authors Kai Kirjavainen	**Group or year of entry** 2008
Apache Maven migration	**Number of pages and appendices** 24 + 3
Supervisor(s) Sauli Isonikkilä	

The goal of this thesis was to find out how the Apache Maven build automation tool could be integrated into the software development process of ProCountor International Oy. Before that, it was necessary to examine how the current solution, Apache Ant, was configured and how it worked. The aim was to carry the integration process out with minimal changes to existing processes and it was important to properly understand the tools currently in use.

The scope of the thesis was limited to the basic deployment requirements of Apache Maven and to the preparations it required. While Maven makes it possible to integrate utility programs, such as continuous integration and automated unit tests into the development process, these were left out in order to not let the scope grow too wide. Maven also makes it possible to utilize a repository server, which makes updating, distributing and handling internal Java-libraries more convenient. This repository server was chosen to be included in the thesis due to its easy deployment and the benefits it offered.

The implementation was carried out using the software development environment used in the company. The most important tools regarding this implementation were the integrated development environment Eclipse, the software build automation tool Ant and the SVN version control system. Apache Maven 3, the newest version, was chosen so that the new tool would not be already old when implemented.

The project was carried out by gaining understanding of the conventions and properties of the current Apache Ant tool first. After that it was time to compare the two products in order to understand what needed to be done. When the comparison was done, it was followed by the actual implementation and finally the documentation of the process.

The outcome of the project was the documentation regarding how this kind of integration project affects the software development process of the company and also the final product which was handed over to the company.

Key words
Apache Maven, software development, IDE, Java

Sisällys

Käsitteitä

Artefakti

Artefakti on esim. Web Application Archive -tiedosto, joka syntyy Apache Mavenin suorittaessa POM.xml-tiedoston määrittelemät tehtävät. Tiedosto voi sisältää esimerkiksi käännetyt lähdekoodit, kirjastoja sekä muita Internet-sovelluksen ajamiseen tarvittavia resursseja. (Ching & Porter 2009, 13.)

Build.xml

Tiedosto, joka määrittelee yhden Apache Ant –projektin suorittamiseen tarvittavat tiedot, kuten lähdekoodin sijainnin yms.
(Holzner, S. 2009, 6-7.)

Kirjastopalvelin (repository server)

Palvelinsovellus, jonka kautta Maven-projektin riippuvuudet voidaan helposti jakaa sovelluskehittäjille ja jonka kautta sisäisten kirjastojen jakelu helpottuu. (Srirangan. 2011, 54)

POM (Project Object Model)

Apache Mavenin käyttämä XML-tiedosto, joka on projektin perusta. Tiedosto määrittelee mm. kirjastoriippuvuudet, projektin nimen ja artefaktin tyypin sekä projektin kääntämiseen ja ajamiseen tarvittavat tiedot. (Ching & Porter 2009, 13.)

1 Johdanto

Sovelluskehitystä tukevia työkaluja on olemassa lukematon määrä eri ohjelmointikielille. Kaikilla näillä on omat etunsa ja huonot puolensa, joten on lähes mahdotonta sanoa onko jokin työkalu parempi kuin toinen. Asiaa tulee tarkastella pitkälti käyttöympäristön ja käyttötarkoituksen kannalta, eli mitä tavoitteita ja hyötyjä käyttöönotolla halutaan saavuttaa?

Käännöstyökalu on luonnollisesti vain yksi osa koko sovelluskehitysprosessia, jota ilmankin olisi teoriassa mahdollista tulla toimeen. Isojen ohjelmistojen kääntäminen käsin ilman koontityökalua ei kuitenkaan ole järkevää, ja lisäksi työkalujen tarjoamat integraatiot muihin sovelluskehitystä tukeviin ohjelmistoihin ovat itsessään jo hyvä syy työkalujen käyttöön.

Koska eri työkalujen konventiot eroavat toisistaan, ei siirtyminen työkalusta toiseen ole aina yksinkertaista. Vaikka tässäkin työssä mukana olevat työkalut tekevät periaatteessa samoja asioita, on Maven enemmän kokonaisvaltainen projektinhallintatyökalu, jonka ydintoiminnan osa ohjelmakoodin kääntäminen on. Ant taas keskittyy pelkästään kääntämiseen, eikä sisällä projektin- tai kirjastonhallinnan osia. Tästä syystä Maven on ohjelmistokehittäjän kannalta houkutteleva vaihtoehto. Koska sen projektinhallinta on kohtalaisen tarkkaan määritelty, sitä varten luotujen automaatiotestaus yms. ohjelmistojen integroiminen on suhteellisen yksinkertaista – ohjelmistot osaavat olettaa tietynlaista rakennetta projektilta.

Tässä projektissa selvitetään Apache Antin korvaamista Apache Mavenilla, käydään läpi molempien työkalujen ominaisuuksia ja lopuksi toteutetaan siirtymä.

1.1 Työn tavoite

Työn tilaaja on kotimainen yritys ProCountor International Oy, joka on osa Accountor Group-konsernia. ProCountor International Oy tuottaa taloushallinnon ohjelmistoa palveluna (Software as a Service) ja on toiminut alalla vuodesta 2001 lähtien.

Opinnäytetyön tavoite on hankkia Apache Maven-työkalun käyttöönottoon vaadittava tieto ja sen jälkeen toteuttaa se opitun tiedon avulla. Tähän sisältyy myös olemassa olevaan Apache Ant-työkaluun perehtyminen, jotta uusi toteutus vastaa myös olemassa oleviin vaatimuksiin. Teoriataustassa perehdytään olemassa olevaan ja valittuun työkaluun, sekä käännöstyökaluihin yleensä. Koottua tietoa hyödyntäen toteutetaan ja dokumentoidaan varsinainen käyttöönotto. Projektin rakenne kuvataan, jonka jälkeen tapahtuu toteutus, joka sisältää Mavenin käyttöönoton lisäksi myös Sonatype Nexus -palvelimen asennuksen ja konfiguroinnin. Lopuksi esitellään projektista saadut tulokset ja pohditaan mahdollisia jatkokehitystoimenpiteitä.

1.2 Tutkimusmenetelmät ja työn rajaus

Työn suoritukseen vaadittava tutkimustyö pohjautuu alan kirjallisuuslähteisiin sekä Internetistä saatavaan tietoon. Koska projektissa käytettävät työkalut ovat avoimen lähdekoodin sovelluksia, jotka ovat laajassa käytössä, tarvittavaa teoriatietoa on hyvin saatavilla. Täydentävää taustatietoa haettiin myös projektin edetessä ilmenneiden tarpeiden mukaan. Muita tutkimusmenetelmiä, kuten asiantuntijahaastatteluja ei katsottu projektin kannalta tarpeelliseksi, koska olemassa olevista lähteistä saatu tieto oli toteutuksen kannalta vähintäänkin riittävää.

Työn sisältö rajattiin koskemaan pelkästään Apache Mavenin käyttöönotta ja sen asettamia vaatimuksia, eikä tavoitteena ollut suorittaa laajempaa vertailua olemassa olevien työkalujen osalta. Rajaus oli työn laajuuden rajoittamisen kannalta perusteltua ja myös siksi, että esitellyt työkalut ovat projektin toimeksiantajan valitsemia ja yleisesti Java-projekteissa käytettyjä apuohjelmia.

2 Käännöstyökalut

Projektin aluksi on syytä selvittää mitä käännöstyökalut ovat ja mihin niitä käytetään. Koska tässä projektissa on tarkoitus korvata Ant-työkalu Mavenilla, on syytä myös tutkia tarkemmin kummankin työkalun perusteita ja toimintatapoja. Mavenin kutsuminen pelkäksi käännöstyökaluksi ei ole täysin oikein, koska se sisältää paljon muitakin ominaisuuksia. Näitä eroja käsitellään myöhemmin tässä luvussa.

2.1 Yleistä käännöstyökaluista

Käännöstyökalut ovat osa nykyaikaista ohjelmistokehitysprosessia. Yksinkertaistettuna niiden tehtävä on kutsua ohjelmointikielen kääntäjää, joka suorittaa ohjelmiston kääntämisen määritellyllä tavalla. Tämä ei kuitenkaan ole perimmäinen syy siihen, miksi käännöstyökaluja käytetään ja miksi ne alun perin kehitettiin. Ne mahdollistavat yksinkertaisen ohjelmistokoodin kääntämisen lisäksi monenlaisia muita operaatiota. Valmis ohjelmisto voidaan esimerkiksi paketoida Web Application Archive – pakettiin, siirtää sovelluspalvelimen hakemistoon ja käynnistää selain sekä sovelluspalvelin automaattisesti. Vastaavia sovelluskehitystä tukevia ja nopeuttavia toimintoja on lukematon määrä ja työkaluille on mahdollista tarvittaessa ohjelmoida omia lisäosia, mikäli omaan käyttöön sopivaa ei löydy valmiina. (Apache Maven Project 2013b; Sonatype 2008, 6-7.)

Koska molemmat työkalut ja lukuisat niitä varten tehdyt lisäosat ovat Apache Software Foundationin ilmaiseksi jakelemia sovelluksia, avoimen lähdekoodin projekteja, ei niiden osalta ole tarpeellista suorittaa kuluvertailuja. Ainoa käyttöönoton kustannus yritykselle on projektiin kulunut henkilötyöaika. (The Apache Software Foundation.)

2.2 Apache Ant

Projektin perustana on yrityksessä käytössä oleva koontityökalu Apache Ant, jonka kehitti alun perin James Duncan Davidson. Java-ohjelmointikieleen pohjautuva Ant julkaistiin virallisesti 2000-luvun alussa, jonka jälkeen siitä on tullut erittäin suosittu työkalu Java-ohjelmoijien keskuudessa. Antin alkuperäinen käyttötarkoitus oli kääntää Apache Tomcat -palvelinohjelmisto, mutta kun Ant julkaistiin vapaaseen levitykseen, monet muutkin kehittäjät huomasivat sen olevan tehokas työkalu ja siten Ant levisi nopeasti. Ant on lisäksi alustariippumaton, joten sama projekti voidaan kääntää missä tahansa käyttöjärjestelmässä, kunhan build.xml on määritelty oikein.
(Apache Ant Project 2013a; Holzner, S. 2009, 2)

2.3 Ant-projektin esimerkki

```
<project name="MyProject" default="dist" basedir=".">
    <description>
        simple example build file
    </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>
  <target name="compile" depends="init"
        description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>
  <target name="dist" depends="compile"
        description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>
    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file
-->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar"
basedir="${build}"/>
  </target>
  <target name="clean"
        description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
</target>
</project>
```

Kuvio 1 Yksinkertainen build.xml-tiedosto (Apache Ant Project 2012b)

Antin käytön perustana on build.xml -tiedosto, jossa projektin käännösprosessi, eri kohteet (build target) ja niiden sisältämät operaatiot on kuvattu (kuvio 1). Kuten esimerkistä käy ilmi, Ant ei sisällä mitään oletuksia esimerkiksi lähdekoodin sijainnista tai käännetyn ohjelmakoodin sijoituksesta, vaan XML-tiedostoon on tarkkaan määriteltävä mistä lähdekoodi löytyy ja mihin käännetyt ja mahdollisesti pakatut tiedostot lopulta sijoitetaan. Esimerkkiprojektin kääntäminen Antilla vaatisi käytännössä neljä Ant-kutsua (Apache Ant Project 2012b.):

1. ant init
2. ant compile
3. ant dist
4. ant clean

Antilla on myös mahdollista sisällyttää eri kohteita toistensa sisälle, jolloin esimerkiksi komento ”ant build”, suorittaisi kaikki edellä listatut kohteet järjestyksessä.

Mikäli ohjelmaan halutaan sisällyttää erillisiä kirjastoja, tulee ne myös määritellä erikseen. Antille on tosin luotu tätä varten Apache Ivy -projekti, jonka tarkoituksena on helpottaa erillisten kirjastojen hallintaa Ant-projekteissa hieman Maven-projekteja muistuttavalla tavalla. Vaikka Ant on komentorivipohjainen työkalu, se on mahdollista integroida ohjelmointiympäristöön, jolloin erilaisten kohteiden määrittely ja ajaminen yksinkertaistuu. (Apache Ant Project 2013c.)

2.4 Apache Maven

Apache Maven, jonka käyttöönotto projektin tavoite on, on hieman Antia uudempi työkalu, jonka kehityksen aloitti 2000-luvun alussa Jason van Zyl ja jonka ensimmäinen virallinen versiojulkaisu tapahtui vuonna 2004. Mavenin historia on samankaltainen kuin Antin, eli se luotiin alun perin tukemaan Jakarta Turbine -projektia, johon Ant ei soveltunut riittävän hyvin. (Apache Maven Project 2005c.)

Mavenin tärkeimmät ja kiinnostavimmat ominaisuudet ovat konvention määrittämä projektirakenne sekä automatisoitu kirjastojen hallinta.

Kuten edellisessä osiossa todettiin, Ant on hyvin joustava projektin rakenteen suhteen ja jättää kehittäjälle vapaat kädet kansioiden yms. määrittelemiseen. Antin toiminnallisuuden toistaminen Mavenissa on periaatteessa mahdollista, mutta ei järkevää käytännössä, koska sillä menetetään juuri se vakiintuneen konvention hyöty, jonka Maven tuo projektiin. Lisäksi konfiguraatiotiedostot kasvavat todella pitkiksi ja monimutkaisiksi, jos niihin määritellään poikkeussääntöjä. Tämä heikentää myös luonnollisesti projektin siirrettävyyttä alustalta tai koneelta toiselle, koska asetuksia tehdessä ei voida olla varmoja siitä, että kaikki kohdealustat olisivat samanlaisia esim. käyttöjärjestelmän osalta.

Kirjastojen hallinnan osalta Maven tarjoaa niin sanotun keskusvaraston (repository), joka sisältää lukuisia yleisesti käytettäviä JAR-kirjastoja. Kirjastoista voi valita haluamansa version tai määrittää projektin käyttämään aina uusinta tarjolla olevaa versiota. Tuotantokäyttöön soveltuvaa sovellusta tehdessä on luonnollisesti syytä käyttää tiettyä versiota kirjastosta, koska uuden kirjaston tuominen projektiin voi aiheuttaa ongelmia ilman kattavaa testausta. Käytettävät kirjastot määritellään projektin pom.xml-tiedostoon, joka tämän projektin osalta esitellään myöhemmin. Keskitetyn kirjastojenhallinnan tärkeimpiä etuja on kuitenkin se, että kirjastot haetaan automaattisesti sovelluksen kääntövaiheessa ja lisäksi niille voidaan määritellä ominaisuuksia, kuten tarvitaanko kyseistä kirjastoa lopullisessa artefaktissa, esim. WAR-paketissa vai tarjoaako esimerkiksi sovelluspalvelin kyseisen kirjaston kun sovellus ajetaan. Koska kaikki projektin tarvitsemat kirjastot eivät luonnollisesti voi olla tarjolla julkisesti, on Mavenin tueksi tarjolla palvelinsovelluksia, jotka korvaavat tai täydentävät keskusvaraston toimintaa. (Sonatype 2011, 1-2.)

Kirjastojenhallintapalvelimeksi tähän projektiin valittiin Sonatype Inc. tekemä Sonatype Nexus. Valinnassa ei suoritettu varsinaisia vertailuja vaan painoarvoa annettiin ratkaisun ilmaisuudelle, sekä sille, että sovelluksen on kehittänyt yritys josta Maven on alun perin lähtöisin. Muut vastaavat tuotteet ovat todennäköisesti erittäin käyttökelpoisia, mutta tämän projektin laajuuden kannalta riittivät aiemmin mainittujen ominaisuuksien lisäksi helppokäyttöisyys ja mahdollisuus tallentaa omia tai kolmannen osapuolen kirjas-

toja palvelimelle, josta ne ladataan automaattisesti kehittäjän koneelle koodin kääntövaiheessa, kun kirjastot on määritelty oikein pom.xml-tiedostossa. Nexus-sovelluksen käyttöönotto kuvataan tarkemmin kappaleessa 4.

2.5 Mavenin perusteet

Apache Maven -projektin määrittely eroaa merkittävästi Antin vastaavasta. Kun Antissa on käytössä vain yksi build.xml, joka sisältää tiedot kaikista eri tavoista, joilla projekti halutaan kasata, Mavenissa koko toiminnan perustana on pom.xml. Yhdessä projektissa on yleensä useampi pom.xml, joista jokaisen kohteena (goal, vastaa Antin target-käsitettä) on yksi niin kutsuttu artefakti. Artefakti voi esimerkiksi olla käännetystä lähdekoodista luotu JAR-paketti, joka vuorostaan paketoidaan toisessa artefaktissa sovelluspalvelimelle asennettavaksi WAR-paketiksi. (Sonatype 2008, 7-10.)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
  </dependencies>
</project>
```

Kuvio 2 Yksinkertainen pom.xml

Kuvion 2 esimerkki toteuttaa käytännössä samat toimenpiteet kuin aiemmin esitelty build.xml. Kuten kuviosta käy ilmi, projektin oletusrakenteen käyttö vähentää merkittävästi tarvittavien asetusrivien määrää. Lisäksi etuna on projektin parempi siirreltävyys ja alustariippumattomuus, kun asetuksissa ei määritellä esim. käyttöjärjestelmäkohtaisia absoluuttisia polkuja. (Sonatype 2008, 9.)

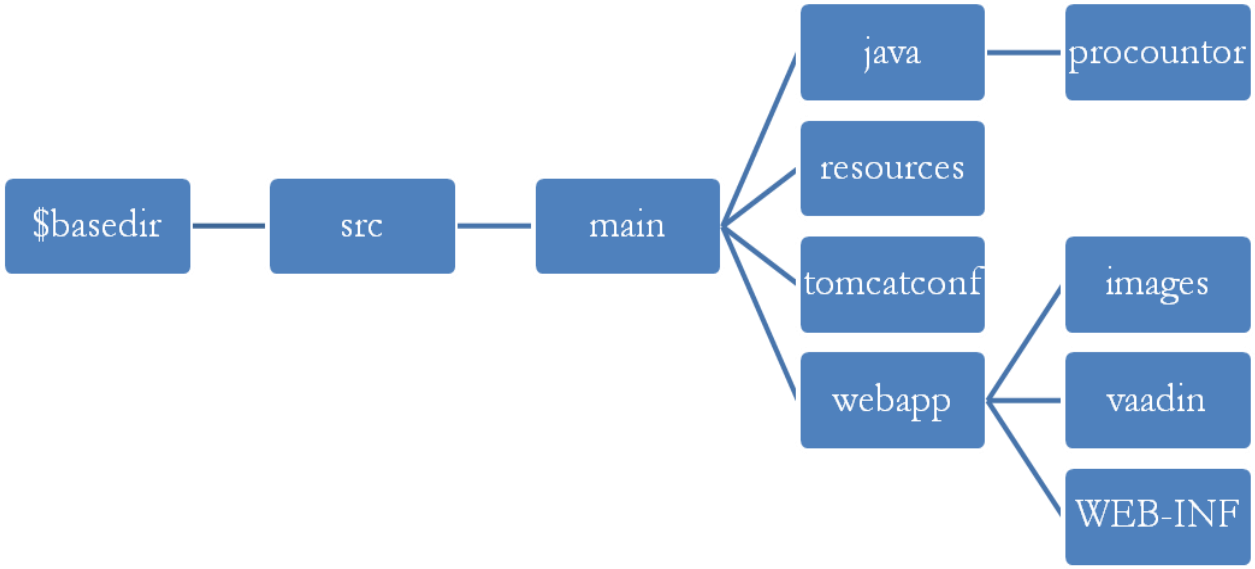
3 Maven käyttöönoton toteutus

Käyttöönoton lähtökohtana oli luoda Mavenilla käytössä olevaa Ant-työkalua vastaava toiminnallisuus Mavenin nuodattaen mahdollisimman hyvin Mavenin konventioita. Toteutuksen lähtökohtana oli ProCountorin nykyinen versio, joka on toteutettu Java Applet -tekniikalla. Tämä ei ollut lopulta täysin mahdollista, koska projekti oli alun perin suunniteltu täysin eri lähtökohdista, joihin Mavenin konventioiden sovittaminen koettiin haastavaksi. Tähän oli syynä projektissa mukana oleva jaettu koodi, jonka jakaminen Maven-moduuleihin ei ollut mahdollista ilman vaativampaa refaktorointityötä. Tähän taas ei liiketoiminnallisista syistä ollut mahdollista käyttää resursseja tämän projektin yhteydessä.

Työn edetessä avautui mahdollisuus toteuttaa Maven-integraatio täysin uuteen projektirakenteeseen. Sovelluksen Applet-toteutus päätettiin korvata vuoden 2013 aikana alustariippumattomalla Vaadin-versiolla, joka mahdollistaa sovelluksen käyttämisen ilman käyttöjärjestelmään asennettavia lisäosia. Koska uuden teknologian käyttöönotto oli joka tapauksessa vaativa työ, oli tämä projekti järkevää yhdistää siihen. Näin vältyttiin työltä, joka olisi vaadittu vanhan projektin sovittamiseen ja toteutus päästiin aloittamaan lähes puhtaalta pöydältä.

3.1 Projektin rakenne

ProCountorin lähdekoodista on tarkoitus luoda sovelluspalvelimella ajettava WAR-paketti. WAR-paketin sisältö koostuu sovelluksen ajamiseen tarvittavista luokista, JAR-kirjastoista, sekä WWW-sisällöstä, kuten tyylitiedostoista ja kuvista.



Kuvio 3 Projektin kansiorakenne

Projektin kansiorakenteessa päädyttiin käyttämään Maven-projektin arkkityypin mukaista vakiorakennetta, koska Maven tunnistaa rakenteen automaattisesti ja näin vältytään ylimääräiseltä konfiguroinnilta (Apache Maven Project, 2013a).

Projektin kansiorakenteen sisältö lyhyesti (kuvio 3):

- **$basedir** – projektin juurihakemisto, josta Maven etsii pom.xml – tiedostoa ja oletuskansioita
- **src** – projektin lähdekoodin (sisältäen testit) ja muiden resurssien juurikansio
- **main** – ohjelmiston varsinaisen lähdekoodin ja resurssien juurikansio
- **java** – ohjelmiston lähdekoodin kansio

- **resources** – projektiin liittyvät properties yms. tiedostot
- **tomcatconf** – projektin tarvitsemat Tomcat-sovelluspalvelimen konfiguraatiotiedostot
- **webapp** – verkkosivujen ja WWW-ohjelmiston resurssien juurikansio
- **images** – verkkosivuilla käytetyt kuvatiedostot
- **vaadin** – Vaadin – sovelluskehyksen tarvitsemat tiedostot
- **WEB-INF** – ohjelmiston ajamiseen tarvittavat kirjastot yms.

3.2 Projektin pom-tiedoston määrittely

Kuten Antin build.xml, Mavenin pom.xml on koko projektin keskeisin tiedosto, jonka avulla projektin riippuvuudet, paketointi, versionumerointi yms. tiedot määritellään. Tästä syystä on aiheellista käydä määrittelyiden olennaisimmat osat läpi.

Aluksi määritellään projektin id:t, versio ja paketointi. Id-elementtien perusteella projektista luotava artefakti löytyy yrityksen Nexus-palvelimelta polusta ”procountor/procountor_vaadin”, mikäli se palvelimelle asennetaan. Versionumerolla ei kehityksen kannalta ole merkitystä, joten se muutetaan vasta tuotantoon käännettävästä projektista.

```
<groupId>procountor</groupId>
<artifactId>procountor_vaadin</artifactId>
<version>1.0</version>
<packaging>war</packaging>
```

Tässä osiossa projektille annetaan nimi ja osoite. Näiden merkitys ei sisäisen projektin kannalta ole kovinkaan suuri, vaan ne ovat lähinnä tiedoksi kehittäjille. Projektin kääntämisen kannalta tiedot eivät ole pakollisia. Tähän kohtaan olisi myös mahdollista lisätä erillisiä repository-palvelimia, mutta koska projektissa käytetään myöhemmin esiteltävää yrityksen sisäistä Nexus-palvelinta, määrityksiä ei tarvita.

```
<name>ProCountor Vaadin</name>
<url>http://www.procountor.com/</url>
```

Seuraavassa osiossa määritellään projektin riippuvuudet, joita tässä projektissa on toista kymmentä. Alla on esimerkki MySQL-ajurin tietyn version liittämisestä projektin riippuvuudeksi.

Riippuvuuksille voidaan määritellä <scope> elementti, jonka pois jättäminen merkitsee samaa kuin ”compile”. Tällä arvolla Maven paketoi kirjaston mukaan valmiiseen projektiin. Kirjastot, joita esim. sovelluspalvelin tarjoaa projektin käyttöön, määritellään scopella ”provided”. Tällöin tiedostoa ei pakata mukaan lopulliseen artefaktiin, vaan sitä hyödynnetään ainoastaan projektin kääntämiseen.

```
<dependencies>
 <dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <version>x.x</version>
</dependencies>
```

Osiossa <build> määritellään <finalName>, joka on paketoidun artefaktin nimi. Mikäli määritystä ei tehdä, Maven käyttää valmiille tiedostolle oletusnimeä, joka sisältää versionumeron. Maven-war-plugin tarvitaan projektin paketoimiseksi.

```
<build>
 <finalName>procountor_vaadin</finalName>
  <pluginManagement>
   <plugins>
    <plugin>
     <artifactId>maven-war-plugin</artifactId>
     <version>2.3</version>
    </plugin>
```

Maven-compiler-plugin -määrittely sisältää kääntämiseen tarkoitetut parametrit, kuten versionumeroinnit. <configuration>-osion määritykset annetaan suoraan ajettavalle javac-kääntäjälle. Näistä ”verbose” on hyödyllinen kehitysvaiheessa, koska se tulostaa mahdolliset virheet kattavammin. <fork> ja kaksi viimeistä muistiasetusta varmistavat, että javac ajetaan erillisenä prosessina ja sille varataan tässä tapauksessa maksimissaan gigatavu muistia. Ilman näitä määrityksiä kääntäminen saattaa kaatua muistin loppumiseen.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.5.1</version>
   <configuration>
    <verbose>true</verbose>
    <fork>true</fork>
    <source>1.7</source>
    <target>1.7</target>
    <meminitial>512m</meminitial>
    <maxmem>1024m</maxmem>
   </configuration>
 </plugin>
```

Jotta projekti on mahdollista käynnistää sisäisellä sovelluspalvelimella, määritellään tiedostoon Tomcat6-palvelin. Kun palvelimelle on annettu nimi ja portti, sekä <goal> ”deploy>, projekti ajetaan komennoilla ”mvn tomcat6:deploy” ja ”mvn tomcat6:run”. Projektin ajamiseen tarvittavat Tomcatin oletuskonfiguraation muutokset tuodaan mukaan elementillä <tomcatWebXml>.

```
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat6-maven-plugin</artifactId>
                <version>2.1</version>
                <goals>
                <goal>deploy</goal>
                </goals>
                    <configuration>
                    <server>tomcat</server>
                    <port>8080</port>
                    <tomcatWebXml>
                    ${basedir}/src/main/tomcatconf/web.xml
                    </tomcatWebXml>

                    </configuration>
            </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

Tässä läpikäydyt asetukset olivat projektin kannalta olennaisimmat. Pom.xml:n lopullinen versio on esillä liitteessä 1.

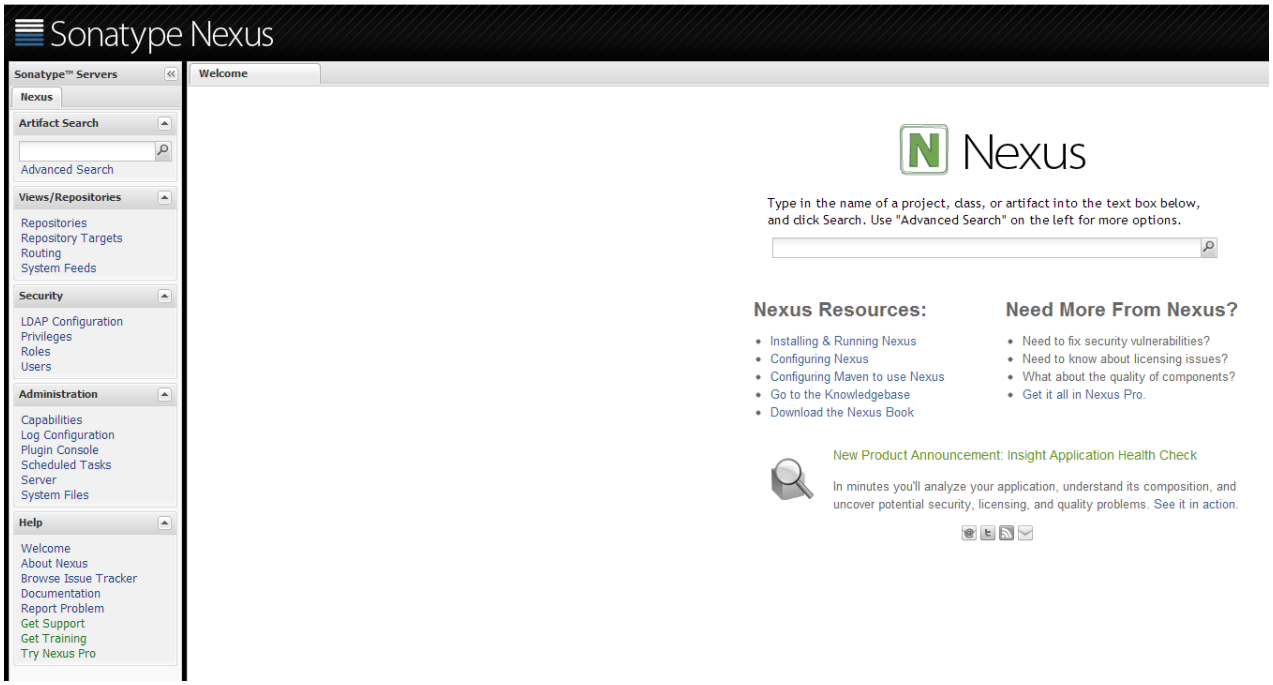
4 Sonatype Nexus -palvelin

Kuten aiemmin todettiin, kaikki projektin tarvitsemat kirjastot eivät ole tarjolla Mavenin julkisessa kirjastossa. Tästä syystä oli tarpeellista asentaa yrityksen sisäinen kirjastopalvelin, Sonatype Nexus. Omien kirjastojen lisäksi palvelin voidaan määritellä toimimaan välityspalvelimena Mavenin keskuspalvelimelle, jolloin mahdolliset käyttökatkot tai muut tietoliikenneongelmat eivät vaikuta kehittäjien työhön, kunhan palvelin on kertaalleen tallentanut vaaditut kirjastot välimuistiin.

4.1 Nexus käyttöönotto ja konfigurointi

Sonatype Nexus on mahdollista ladata joko valmista sovelluspalvelinta varten luotuna WAR-kirjastona tai täysin itsenäisenä versiona, joka sisältää kaiken tarvittavan palvelimen käyttöönottoon. Vaikka yrityksellä oli jo valmiiksi olemassa oleva sovelluspalvelin ProCountor-sovelluksen testaamista varten, projektissa päädyttiin kuitenkin asentamaan erillinen itsenäinen sovelluspalvelin. Ratkaisuun päädyttiin mm. siksi, että mikäli kirjastopalvelin tarvitsisi erillisiä huoltotoimenpiteitä tai olisi epävakaa, se ei häiritsisi sovelluksen testaamista. Käyttöönoton aikana ei kuitenkaan ilmennyt mitään ongelmia, joiden takia WAR-paketin ajamista jaetulla sovelluspalvelimella ei voisi harkita.

Asennus aloitettiin lataamalla Sonatypen kotisivuilta ohjelman sisältämä paketti. Tämän jälkeen paketti purettiin ja ajettiin käytössä olevalle käyttöjärjestelmälle sopiva install-nexus.bat, joka asentaa palvelimen Windowsin palveluksi. Tämän jälkeen palvelin käynnistyy automaattisesti jos palvelin käynnistetään uudelleen ja se on lisäksi helppo pysäyttää / käynnistää Windowsin Services-valikosta, mikäli tarvetta ylläpitotoimenpiteille on. Tämän jälkeen asennus oli valmis ja palvelimen toiminta pystyttiin todentamaan avaamalla osoite selaimella.

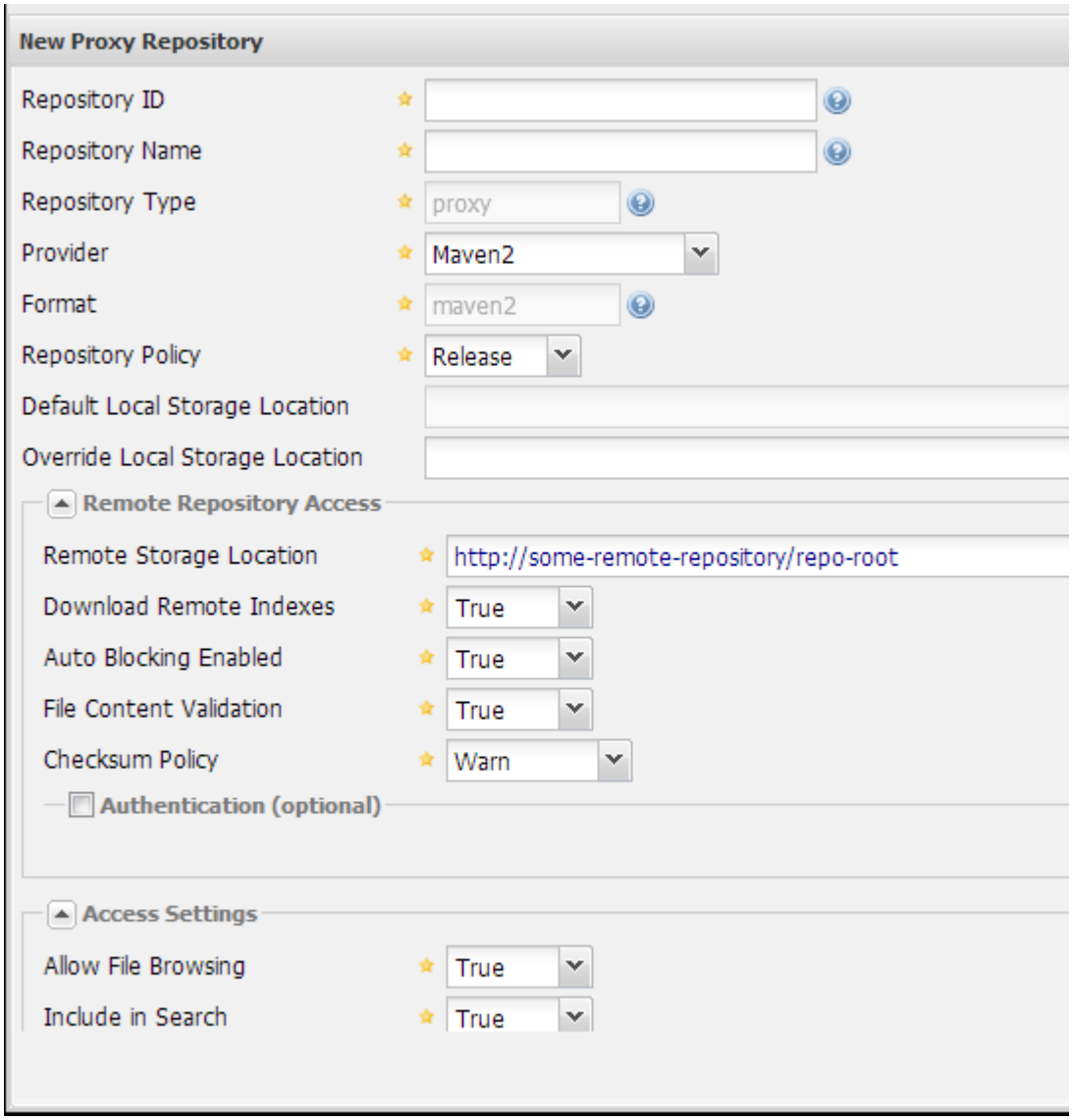


Kuvio 4 Sonatype Nexus hallintapaneeli

Oletustunnuksilla kirjautumisen jälkeen avautui kuviossa 4 näkyvä hallintapaneeli. Nexus-palvelin mahdollistaa mm. käyttöoikeuksien hallinnan LDAP-tuella ja muita kehittyneitä ominaisuuksia, joita ei kuitenkaan tämän projektin puitteissa katsottu tarpeelliseksi ottaa käyttöön, koska palvelimen ensisijainen tarkoitus oli tarjota kehittäjille tarvittavat kirjastot. Palvelimen muihin oletusasetuksiin ei myöskään tarvinnut koskea, vaan kyseessä oli erittäin toimiva valmis kokonaisuus.

Ennen sisäisten kirjastojen käyttöönoton valmistelua palvelimesta tehtiin välityspalvelin Mavenin keskuskirjastopalvelimelle.

Välityspalvelimen luonti tapahtui Repositories-valikon sisältä Add->Proxy Repository-valinnalla, joka on esillä kuviossa 5.



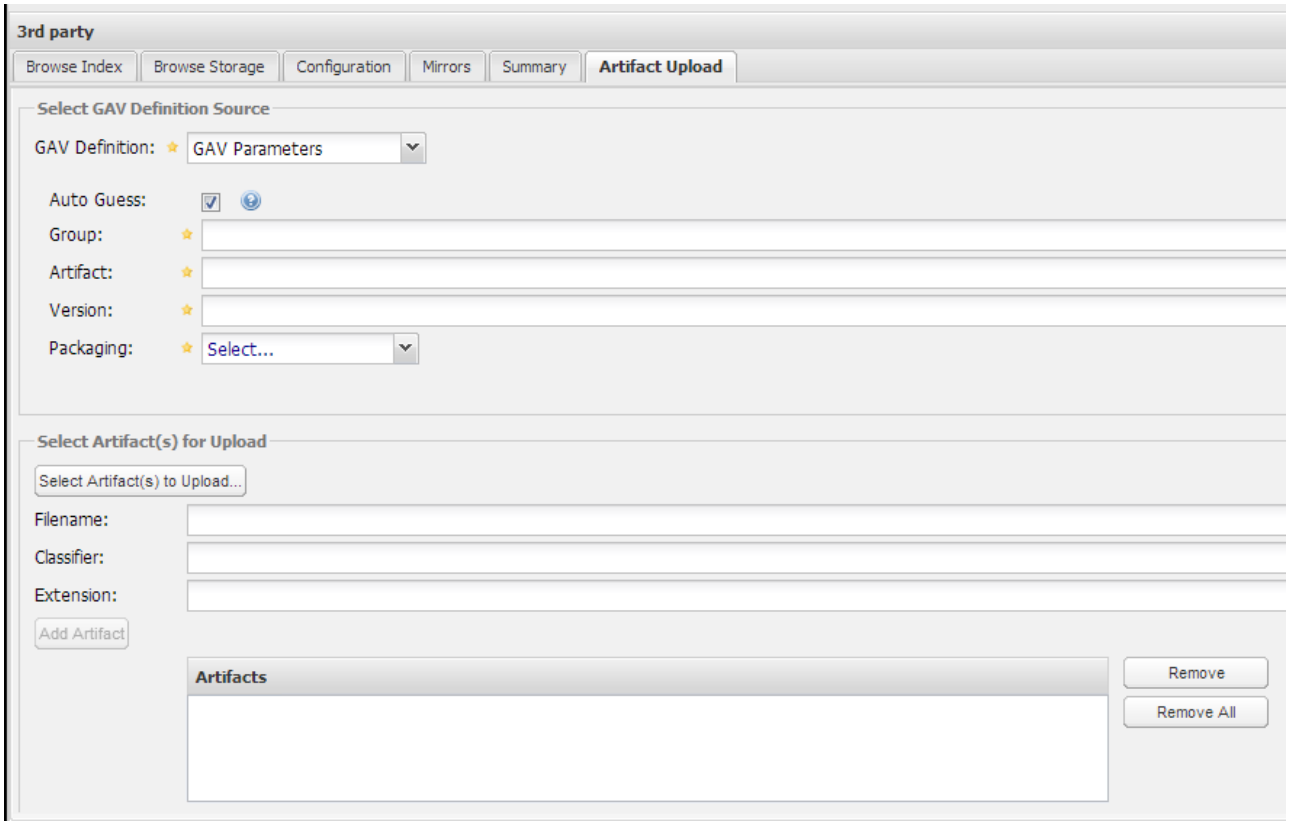
Kuvio 5 Välityspalvelimen määrittely

Välityspalvelimelle määriteltiin konfiguraatiovaiheessa ID ja nimi, joiden avulla se tunnistetaan myöhemmin Mavenin asetustiedoston kautta. Asetusten tekeminen oli yksinkertaista, central-nimen ja ID:n asettamisen jälkeen Remote Storage Location-kenttään asetettiin Mavenin keskuskirjasto ja muut asetukset jätettiin oletuksiksi. Välityspalvelimelle päädyttiin antamaan ID-tunnukseksi ja nimeksi ”central”, jotta nimestä kävisi selvästi ilmi minkä palvelimen sisältöä on tarkoitus tarjota eteenpäin.

Kun välityspalvelin oli määritelty, oli vuorossa tallentaa sinne projektissa tarvittavat JAR-kirjastot. Ennen tätä oli kuitenkin lisättävä erillinen kirjasto kolmannen osapuolen JAR-kirjastoja varten. Kirjaston luominen ei eronnut välityspalvelimen luomisesta merkittävästi, sille annettiin nimeksi ”3rd party” ja tyypiksi valittiin ”hosted”, joka viittaa siihen, että kyseinen kirjasto on vain tämän palvelimen tarjoama, eikä kopio julkisesta palvelimesta.

4.2 Kirjastojen tallentaminen Nexus-palvelimelle

Kirjastojen tallentamiseen Nexus-palvelimelle käytetään alla olevassa kuviossa 6 näkyvää käyttöliittymää:



Kuvio 6 Kirjastojen tallennusnäkymä

Pom.xml mukaisesti palvelimelle tallennettavalle kirjastolle määritellään parametrit group, artifact, version ja packaging. Tämän jälkeen tallennettava kirjasto valitaan käytettävän koneen kovalevyltä, lisätään listaan Add Artifact-napilla ja tallennetaan painamalla Upload Artifact(s).

Osa projektissa käytettävistä kirjastoista olivat niin vanhoja, ettei Mavenin vaatimia tietoja ollut kaikissa tapauksissa tarjolla. Tästä syystä kirjastoille pyrittiin valitsemaan mahdollisimman kuvaavat tiedot, varmistan samalla, etteivät ne mene sekaisin julkisesti tarjolla olevien tiedostojen kanssa. Joistain kirjastoista puuttui myös kokonaan versionumerointi, vaikka ne olivat vanhoja versioita julkisesti saatavilla olevista tiedostoista. Tämä ongelma ratkaistiin antamalla kirjastojen versionumeroksi ”1.0-PROCOUNTOR”, jolloin oli selvää mitkä tiedostoista olivat sisäisiä ja mitkä ulkoisia. Kuvaavien nimien valitseminen varmisti myös sen, että pom.xml:stä pystyi tarvittaessa näkemään mitkä kirjastot olivat sisäisiä ja mitkä eivät. Kun projektissa tarvittavat kirjastot oli ladattu palvelimelle, korjattiin määritykset pom.xml-tiedostoon ja sen jälkeen suoritettiin tarvittavat muutokset Mavenin asetustiedostoon, jotta ohjelmaa käännettäessä kirjastoja osattiin hakea oikealta palvelimelta.

4.3 Nexuksen määritys Mavenin asetuksiin

Mavenin vakioasennus tekee tiettyjä oletuksia mm. käytössä olevan kirjastopalvelimen osalta. Asetukset sijaitsevat asennushakemiston alla sijaitsevan conf-hakemiston settings.xml-tiedostossa. Suurin osa vakioasetuksista jätettiin huomiotta, koska niillä ei ollut vaikutusta tämän projektin toimintaan. Nexuksen käyttöönottoon tarvittavat muutokset olivat seuraavat:

```
<mirror>
  <id>nexus</id>
  <url>http://proctest-02:8081/nexus/content/groups/public/</url>
  <mirrorOf>*</mirrorOf>
</mirror>
</mirrors>
```

Yllä olevassa XML-määrittelyssä Maven muutetaan käyttämään keskuspalvelimen sijaan yrityksen sisäistä palvelinta, jonka <mirrorOf>*</mirrorOf> -määrittely ohjaa Mavenin hakemaan kaikkia tarvittavia kirjastoja kyseiseltä palvelimelta. ID-kenttä ei vaikuta etsittävän kirjastopalvelimen hakuun vaan haku perustuu täysin <url> elementin sisältöön. Tästä syystä ID-kenttään määriteltiin tässä tapauksessa arvo ”nexus” vaikka aiemmin määritelty välityspalvelin olikin nimeltään ”central”. Nimeämisellä haluttiin tehdä selvä ero varsinaisen keskuspalvelimen ja välityspalvelimen välille. Lisäksi ”central”-nimi on varattu seuraavaa asetusmuutosta varten, joka osoittaa Mavenille keskuspalvelinta korvaavan profiilin käytön.

”Central”-kirjastopalvelimen korvaamiseksi ja käyttöönottamiseksi seuraava askel oli asettaa Maven käyttämään seuraavanlaista profiilia:

```
<profile>
  <id>nexus</id>
  <repositories>
    <repository>
      <id>central</id>
      <url>http://</url>
      <layout>default</layout>
    </repository>
  </repositories>
<pluginRepositories>
    <pluginRepository>
      <id>central</id>
      <url>http://central</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

Profiilin asetetut ”nexus”-ja “central”-asetukset sekä olemattomat URL-osoitteet varmistavat, että Maven käyttää aiemmin määriteltyä ”nexus” mirror-palvelinta. <snapshots>-tagin lisääminen kertoo, että kyseiseltä palvelimelta voidaan myös hakea keskeneräisiä kirjastoja. Tämä ei projektin kannalta ollut välttämätöntä, mutta jatkokehitystä silmällä pitäen asetus päätettiin määritellä. Jatkossa olisi siis mahdollista ladata keskeneräisiä ja/tai kehityksessä olevia kirjastoja SNAPSHOT-versiotunnuksella palvelimelle ja Maven osaisi etsiä myös kyseiset tiedostot tältä palvelimelta.

Lopuksi Mavenille tarvitsi yksinkertaisesti määritellä, että käytössä oleva kirjastopalvelimen profiili on aiemmin luotu ”nexus”:

```
<activeProfiles>
    <activeProfile>nexus</activeProfile>
  </activeProfiles>
</settings>
```

Näiden muutosten jälkeen Maven haki automaattisesti kirjastot yrityksen omalta Nexus-palvelimelta, jonka välimuistiin tiedostot myös jäivät. Projektin käyttöönotto millä tahansa alustalla oli näiden muutosten myötä mahdollista, koska kaikki tarvittavat julkiset sekä yksityiset kirjastot olivat saatavilla sisäiseltä palvelimelta. Jotta kehitystyö ja kääntäminen olisi mahdollista myös ilman verkkoyhteyttä, Maven lataa kirjastojen kopiot myös kehittäjän koneelle ensimmäisellä kääntökerralla. Tämän jälkeen yhteys palvelimeen ei ole välttämätöntä, ellei projektiin lisätä kirjastoja.

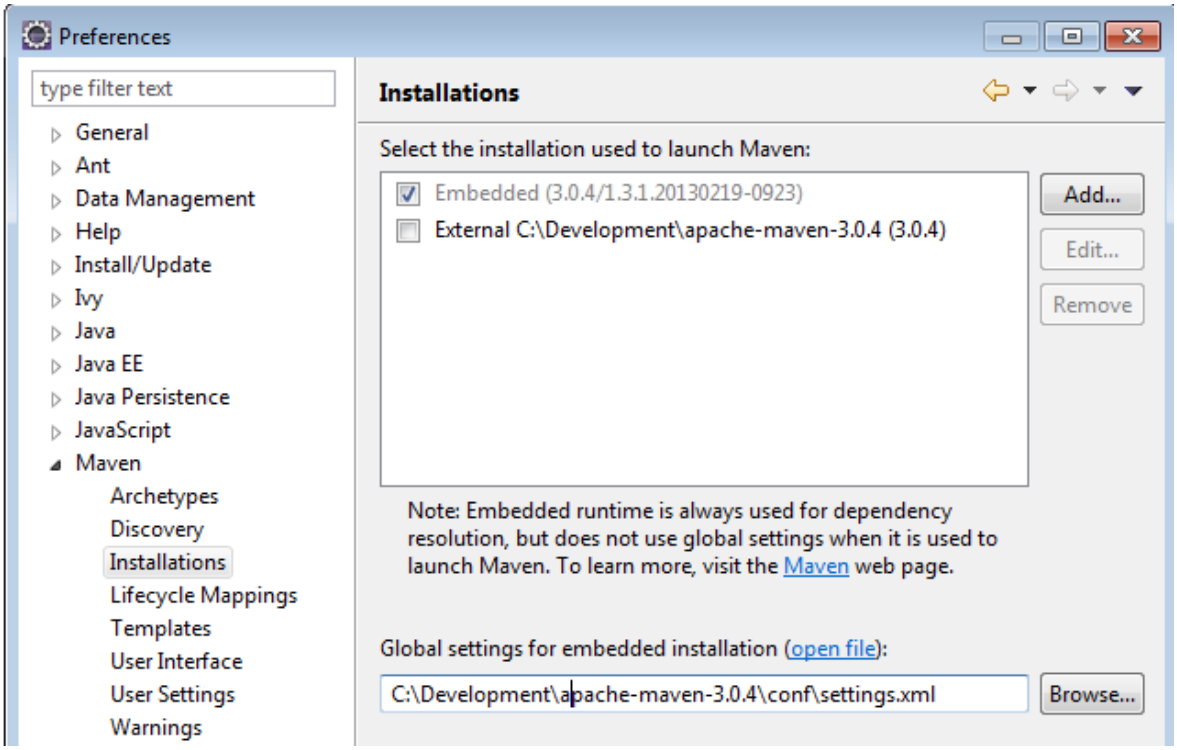
5 Integrointi kehitysympäristöön

Kun Maven-projekti on luotu ja käyttövalmis, sen käyttöönotto on erittäin suoraviivaista. Käytössä olevaan ohjelmointiympäristöön, Eclipseen, on tarjolla M2Eclipse-lisäosa. Lisäosa ladataan Eclipsen Juno-versioon Eclipse Marketplace -palvelusta. Lisäosan avulla on mahdollista käyttää Mavenia suoraan graafisella käyttöliittymällä ilman tarvetta komentorivikomentojen kirjoittamiseen. Kun projekti on ladattu versionhallinnasta ja lisätty Eclipseen, löytyy kyseisen projektin kontekstivalikosta Maven-osio, jossa kaikki kääntämiseen ja ajamiseen tarvittavat komennot on listattu.

5.1 Eclipsen asetusten määrittely

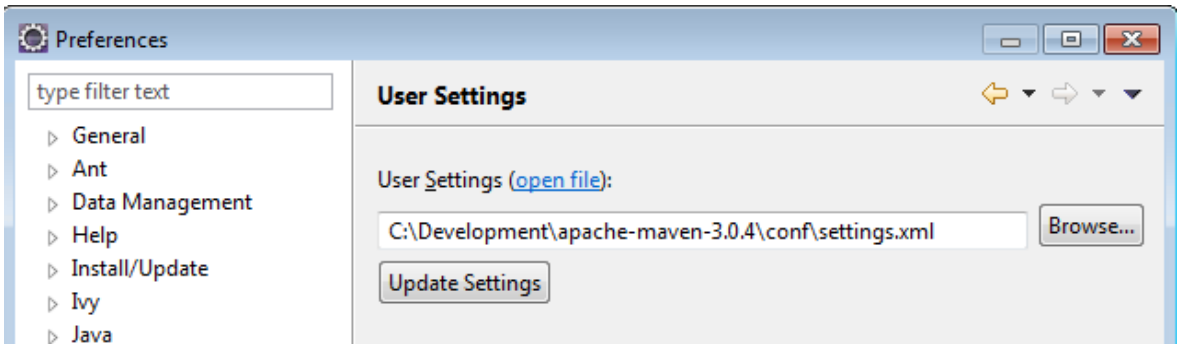
Eclipse on Javalla toteutettu avoimen lähdekoodin kehitysympäristö, jota ylläpitää useista suuryrityksistä koostuva Eclipse Foundation. Eclipseen on mahdollista asentaa monenlaisia lisäosia, joiden avulla kehitysympäristö on helposti muokattavissa oman projektin tarpeisiin. Tästä toiminnallisuudesta on esimerkkinä mm. tässä projektissa käytetty M2Eclipse-lisäosa. M2Eclipse-lisäosa ei vaadi merkittäviä määrittelyjä, mutta koska projektissa käytetään muokattua settings.xml-tiedostoa, se pitää määritellä erikseen.

Kun projekti on valmiina Eclipsessä ja Maven-lisäosa asennettu, tapahtuu asetusten tekeminen Window->Preferences->Maven -valikon kautta. Muokattu settings.xml asetetaan ensin Installations-valikon alla olevaan polkuun, kuten kuviosta 6 käy ilmi.



Kuvio 6

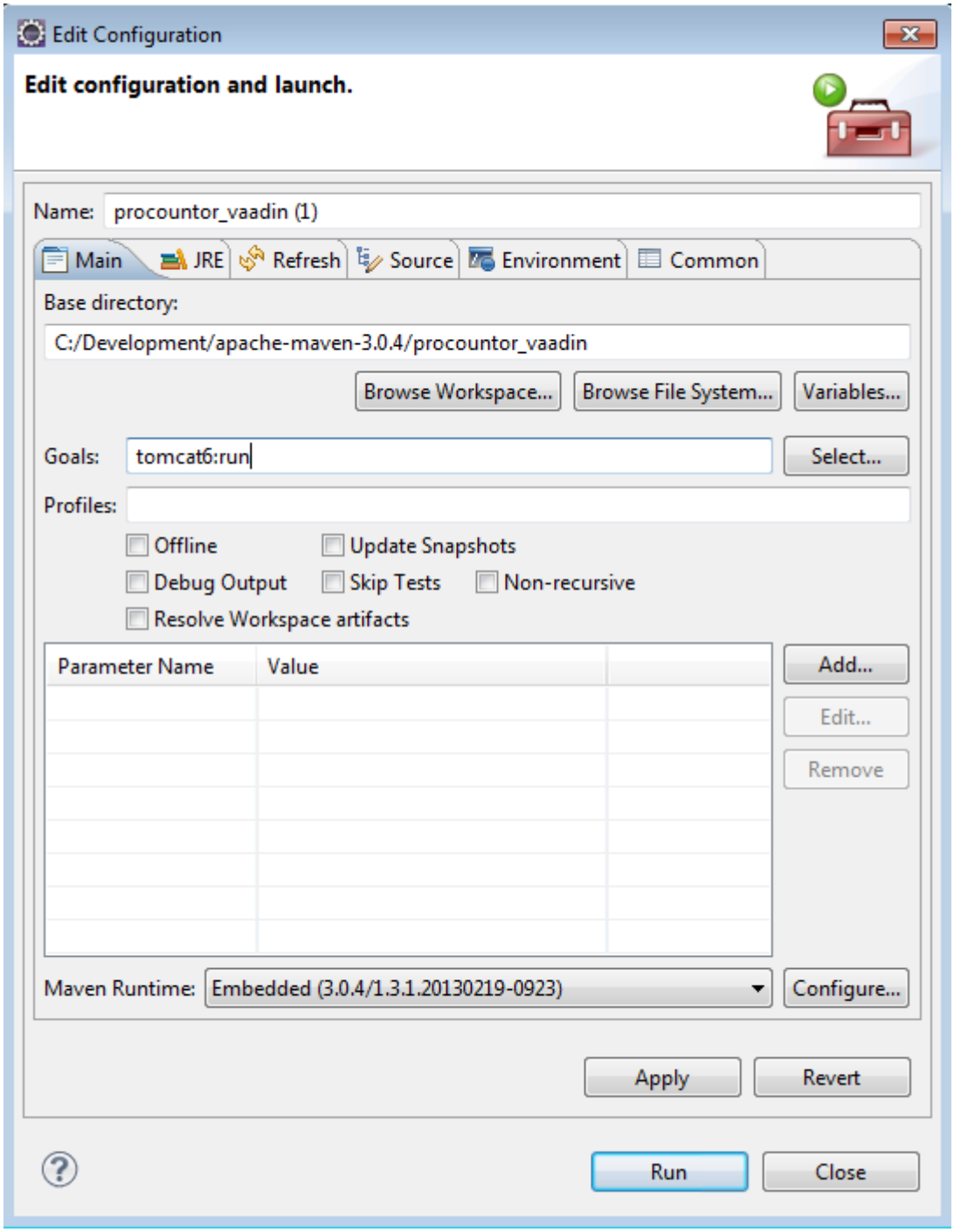
Tämä määritys ei kuitenkaan riitä kirjastopalvelimen asetuksia varten. Siksi tarvitaan vielä toinen muutos User Settings-valikkoon (kuvio 7).



Kuvio 7

Kun nämä asetukset on määritelty, on projekti siinä tilassa, että se voidaan ajaa suoraan Eclipse-kehitysympäristöstä.

Mavenilla on mahdollista suorittaa lukuisia erilaisia tehtäviä, joista tässä esitellään vain kyseisen projektin kannalta olennaiset vaihtoehdot. Projektin kannalta tärkeimmät käskyt ovat: compile, tomcat6:run ja package. Esimerkki määrittelyistä löytyy kuviosta 8.



Kuvio 8 Määrittely projektin ajamisesta Tomcat 6-palvelimella

Goals-kenttään määritellään haluttu lopputulos ja Name-kenttään tätä kuvaava nimi. Aiemmin mainituista ”compile” kääntää ohjelmiston lähdekoodin, ”tomcat6:run” kääntää ja ajaa sovelluksen Tomcat 6 -palvelimella ja ”package” tekee lähdekoodista WAR-paketin.

Kehityksen kannalta mielenkiintoisen ja hyödyllisen vaihtoehdon tarjoaa myös Eclipsen ”Debug As”-valinta, jolla ohjelmisto voidaan käynnistää niin kutsutussa debug-tilassa sovelluspalvelimella. Tämä mahdollistaa ns. breakpointien käyttämisen koodissa, jossa sovelluksen suoritus pysäytetään halutussa kohdassa ja sen tilaa on mahdollista tarkastella erillisessä debug-näkymässä. Tätä kautta kyseisen tilan muuttujien sisältö on nähtävissä selkokielellä ja pysäytetystä tilasta seuraavaan siirtyminen onnistuu helposti. Lisäksi ohjelmakoodiin on mahdollista tehdä muutoksia suorituksen aikana, jonka avulla virheiden korjaus ja tarkistus on nopeampaa, koska koko ohjelmakoodia ei tarvitse kääntää uudestaan yksittäisen virheen selvittämiseksi.

6 Tulokset ja johtopäätökset

Projektin tarkoituksena oli suunnitella ja toteuttaa Apache Ant -käännöstyökalun korvaaminen Apache Mavenilla. Vaikka molemmat työkalut tarjoavat samoja toiminnallisuuksia, ei siirtyminen ollut täysin ongelmatonta. Osasyynä tähän oli aiemmin kuvattu projektin rakenne, joka ei toteutustekniikaltaan vastaa Mavenin konventiota. Uuden käyttöliittymätekniikan mukaantulo kuitenkin mahdollisti projektin loppuunsaattamisen siinä skaalassa kuin alun perin oli suunniteltu. Muutos voi ensisilmäyksellä vaikuttaa työläältä, mikä onkin totta migraation osalta. Kun Maven on saatu käyttöön, ovat sen tuomat hyödyt kuitenkin käyttöönottoon käytetyn ajan arvoiset.

Pilvipalveluiden kehityssuunta on vahvasti poispäin asiakaskoneelta vaadittavista ohjelmistoista, joista hyvänä esimerkkinä toimivat muun muassa Google Docs ja Microsoftin Office 365. Apache Maven on suunnattu selvästi enemmän arkkitehtuuriin, jossa palvelin tarjoilee sivuston ja sen toiminnallisuuden asiakaspäätteelle esim. Javascript- ja HTML-yhdistelmällä toteutettuna. Tämä kävi ilmi kun käyttöönottoa tehtiin alkuperäiseen Applet-pohjaiseen toteutukseen, johon Mavenin EJB-tuki ei soveltunut ongelmitta.

Palvelimella erikseen ajettavan sovelluksen ja sitä kutsuvan asiakasohjelman kääntäminen Mavenin tarjoamilla konventioilla olisi todennäköisesti ollut mahdollista, koska se onnistuu myös Antilla. Vaativuudeltaan tämä olisi kuitenkin ollut haasteellista ja työläs-

tä, joten projektin kannalta Vaadin-ympäristöön siirtyminen oli looginen ja luonteeltaan moderni kehitysaskel. Lisäksi useassa lähdeteoksessa varoitettiin yrittämästä sovittaa Mavenia omaan projektimalliin ja mieluummin sovittamaan oma projekti Maveniin.

Toteutuksen aikana kävi myös selväksi, että avoimen lähdekoodin ratkaisujen kanssa tarvitsee paljon itsenäistä ja kattavaa tutkimista hyödyntäen Internetissä käytäviä keskusteluja, koska sovelluksilta puuttuu kaupallisista vastakappaleista löytyvä tekninen tuki. Toisaalta oppimiskokemuksen kannalta selvitystyön tekeminen ja hankitun tiedon soveltaminen olivat erittäin tärkeitä asioita. Henkilökohtaisen oppimiskokemuksen lisäksi projekti toi yritykselle paljon arvokasta tietoa ja osaamista, jota ei ilman tätä työtä olisi välttämättä osattu etsiä.

7 Jatkokehitysehdotukset

Koska Mavenin käyttö ohjelmistoprojektissa mahdollistaa monenlaisten apuohjelmien käytön, olemassa olevat jatkokehitysmahdollisuudet ovat erittäin laajat. Tärkeintä on pyrkiä löytämään kyseistä yritystä parhaiten palvelevat ratkaisut. Työkalujen tarkoitus on loppujenlopuksi tehostaa prosesseja ja sitä kautta hyödyttää yrityksen liiketoimintaa. Ohjelmistokehityksen tehostamisen kannalta kiinnostavimmat apuohjelmat ovat ohjelmiston laadunvarmistukseen, kirjastojenhallintaan ja jatkuvaan integraatioon liittyvät ratkaisut. Näiden työkalujen käyttöönotto ei välttämättä vaadi Maven-pohjaista projektia, mutta sen yleiset konventiot helpottavat työtä huomattavasti. Tilannetta voisi verrata esim. Applen iOS-käyttöjärjestelmään, jolle ohjelmien kehitys on suoraviivaista, koska käytössä oleva laitekanta ja sen ominaisuudet ovat hyvin tunnettuja.

Ohjelmistoprojektin laadunvarmistusta, sekä yleistä ohjelmistokoodin laadunvarmistusta varten on olemassa suosittu SonarSourcen kehittämä sovelluskokonaisuus Sonar. Sonar tarjoaa mm. koodin staattisen testauksen, yksikkötestauksen automatisoinnin (joka tosin on mahdollista myös tehdä Mavenin koodinkääntöprosessin aikana), koodikonventioiden käytön tarkkailun ja mahdollisten ohjelmointivirheiden tunnistamisen. Nämä tiedot ovat saatavilla raportilla, josta yrityksen sovelluskehityksen tilaa ja työn laatua on yksinkertaista seurata. Sonar voidaan ottaa tarvittaessa käyttöön omalla sovelluspalvelimellaan ja se tunnistaa projektin lukemalla Mavenin pom.xml-tiedostoa.

Koska tässäkin projektissa on käytössä kolmannen osapuolen kirjastoja, jotka eivät ole saatavilla julkisesti, oli oman kirjastonhallintapalvelimen käyttöönotto loogista. Palvelimen käytön ei kuitenkaan tarvitse rajoittua pelkästään tähän käyttöön, vaan Nexusta on mahdollista käyttää myös mm. omien kirjastojen jakamiseen.

Projektin kautta hankittua osaamista on myös mahdollista soveltaa ja hyödyntää yrityksen tulevissa projekteissa. Tätä työtä kirjoittaessa yksi käynnissä olevista projekteista liittyy palvelun tarjoaman liiketoiminnan raportoinnin kehittämiseen, joka jaetusta tietosisällöstä huolimatta on täysin irtonainen projekti. Mavenin hyödyntäminen tällaisessa uudessa projektissa olisi luontevaa, koska projekti voitaisiin sovittaa alun perin oikeiden konventioiden mukaiseksi.

Lähteet

Apache Ant Project 2013a. Frequently Asked Questions.
Luettavissa: http://ant.apache.org/faq.html
Luettu 12.1.2013

Apache Ant Project 2012b. Using Apache Ant.
Luettavissa: http://ant.apache.org/manual/using.html
Luettu 12.1.2013

Apache Ant Project 2013c. The Official Ivy documentation.
Luettavissa: http://ant.apache.org/ivy/history/latest-milestone/index.html
Luettu 12.1.2013

Holzner, S. 2009. Ant: The Definitive Guide. 2nd Edition. O'Reilly Media Inc., Sebastopol

The Apache Software Foundation. Foundation Project.
Luettavissa: http://www.apache.org/foundation/
Luettu 12.1.2013

Srirangan. 2011. Apache Maven 3 Cookbook. Packt Publishing Ltd. Birmingham.

Ching, M., Porter, B. 2009. Apache Maven 2 Effective Implementation. Packt Publishing Ltd. Birmingham.

Apache Maven Project 2013a. Introduction to the Standard Directory Layout.
Luettavissa: http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
Luettu 12.1.2013

Apache Maven Project 2013b. Available Plugins.
Luettavissa: http://maven.apache.org/plugins/
Luettu 12.1.2013

Apache Maven Project 2005c. History of Maven by Jason van Zyl
Luettavissa: http://maven.apache.org/background/history-of-maven.html
Luettu 12.1.2013

Sonatype. 2008. Maven: The Definitive Guide. O'Reilly Media Inc., Sebastopol.

Sonatype. 2011. Repository Management with Nexus. Sonatype Inc,. Silver Spring.

Liitteet

Liite 1. Salainen