Lab Manual for MVN-101 and MVN-201

Maven

Sonatype Maven Training

Sonatype

Introduction

This workbook is designed to accompany Sonatype's Maven training course, it consists of a sequence of lab exercises which will introduce the concepts of Maven. These lab exercises should be completed in sequence in the presence of an instructor. The instructor will distribute the software and code necessary to complete these exercises. If, at any time during the lab exercise, you encounter problems with your software installation or if you do not understand any of the instructions, please ask your instructor for help.

1. Conventions

1.1. Typographic Conventions

This workbook uses typographic conventions to provide cues to identify Java class names, filenames, and other variables that are used throughout this book. Table 1, "Typographic Conventions in this Workbook" lists some of these conventions and provides example values using the inline typographic styles that you will encounter throughout this workbook.

Table 1. Typographic Conventions in this Workbook

Type	Example
Environment Variables	`M2_HOME`, `PATH`
Filenames and Directories	`C:\Users\John`, `/usr/local/maven-2.2.1`

1.2. Projects Working Directory

The lab examples in this workbook use Windows paths and filename conventions. While this lab manual uses C\:\\Projects as a working directory, you may wish to create your working directory under your own home directory or in an alternate location. If you are working on a Mac OSX or Linux machine, you should also consider creating a Projects directory under your own home directory.

Table 2. Suggested Locations for Projects Working Directory

Operating System	Directory
Windows	C:\Projects
Mac OSX	~/Projects
Linux	~/Projects

Training Materials and Software

This lab manual and training materials assume that you have been supplied with the necessary software prerequisites. Your instructor will distribute installation media or provide instructions for downloading this material from the Internet. The supporting software required for Sonatype's Maven training are:

- Java Development Kit version 1.6.0_21 (JDK)
- Eclipse 3.5
- Apache Maven 2.2.1
- Sonatype Nexus 1.4.0
- Hudson

All software prerequisites are contained in the `${training.software.dir}` directory in the training software distribution. Table 3 provides a listing of the contents of this directory.

Table 3. Training Software Prerequisites

Filename	Platform
`JavaForMacOSX10.5Update2.dmg`	Mac OSX 10.5
`jdk-1_5_0_19-linux-amd64.bin`	64-bit Linux
`jdk-1_5_0_19-linux-amd64-rpm.bin`	64-bit Linux
`jdk-1_5_0_19-linux-i586.bin`	32-bit Linux
`jdk-1_5_0_19-linux-i586-rpm.bin`	32-bit Linux
`jdk-1_5_0_19-windows-i586-p.exe`	32-bit Windows
`jdk-1_5_0_19-windows-amd64.exe`	64-bit Windows
`eclipse-jee-galileo-SR1-macosx-cocoa.tar.gz`	Mac OSX 10.5
`eclipse-jee-galileo-SR1-win32.zip`	32-bit Windows
`eclipse-jee-galileo-SR1-linux-gtk.tar.gz`	32-bit Linux
`eclipse-jee-galileo-SR1-linux-gtk-x86_64.tar.gz`	64-bit Linux
`nexus-webapp-1.4.0-bundle.tar.gz`	All platforms
`nexus-webapp-1.4.0-bundle.zip`	All platforms
`apache-maven-2.2.1-bin.tar.gz`	All platforms
`hudson-1.331.war`	All platforms

1. Downloading Prerequisites

The following files can be downloaded from Sonatype's training repository:

1.1. Lab Solutions

The lab-solutions.zip archive contains the solutions for the babble example projects.

- http://sonatype-train.s3.amazonaws.com/lab-solutions.zip

1.2. Java Development Kit

Download the JDK from one of the following URLs:

- Linux 32-bit
  - http://sonatype-train.s3.amazonaws.com/jdk-1_5_0_19-linux-i586-rpm.bin
  - http://sonatype-train.s3.amazonaws.com/jdk-1_5_0_19-linux-i586.bin
- Linux 64-bit
  - http://sonatype-train.s3.amazonaws.com/jdk-1_5_0_19-linux-amd64-rpm.bin
  - http://sonatype-train.s3.amazonaws.com/jdk-1_5_0_19-linux-amd64.bin
- Windows 32-bit
  - http://sonatype-train.s3.amazonaws.com/jdk-1_5_0_19-windows-i586-p.exe
- Windows 64-bit
  - http://sonatype-train.s3.amazonaws.com/jdk-1_5_0_19-windows-amd64.exe

1.3. Apache Maven

Download Maven from one of the following URLs:

- http://sonatype-train.s3.amazonaws.com/apache-maven-2.2.1-bin.tar.gz
- http://sonatype-train.s3.amazonaws.com/apache-maven-2.2.1-bin.tar.gz

1.4. Eclipse 3.5 (Galileo)

Download Eclipse 3.5 from one of the following URLs:

- http://sonatype-train.s3.amazonaws.com/eclipse-jee-galileo-SR1-linux-gtk-x86_64.tar.gz

- http://sonatype-train.s3.amazonaws.com/eclipse-jee-galileo-SR1-linux-gtk.tar.gz
- http://sonatype-train.s3.amazonaws.com/eclipse-jee-galileo-SR1-macosx-cocoa.tar.gz
- http://sonatype-train.s3.amazonaws.com/eclipse-jee-galileo-SR1-win32.zip

1.5. Nexus Open Source

Download Nexus Open Source from one of the following URLs:

- http://sonatype-train.s3.amazonaws.com/nexus-webapp-1.4.0-bundle.tar.gz
- http://sonatype-train.s3.amazonaws.com/nexus-webapp-1.4.0-bundle.zip

1.6. Hudson

Download Hudson from one of the following URLs:

- http://sonatype-train.s3.amazonaws.com/hudson-1.331.war

1.7. Subversion

Download the Subversion Client and Server from one of the following URLs:

- http://sonatype-train.s3.amazonaws.com/CollabNetSubversion-server-1.6.3-3.win32.exe
- http://sonatype-train.s3.amazonaws.com/Subversion%201.6.2%20Universal.dmg

Lab 1. JDK Installation

1.1. Goal

This lab exercise is designed to walk you through the process of installing the Java Development Kit (JDK) on your own workstation. After completing this lab exercise you should be able to install the JDK on any workstation and configure the appropriate environment variables.

> Note
>
> While this training was designed to work with Java 5.0, everything in this training manual will work with newer releases of Java. If you already have a JDK installed on your workstation, you can safely skip this lab and move on to the next lab exercise.

1.2. Agenda

- Download the Java Development Kit 5.0
- Install the Java Development Kit 5.0
- Set the `JAVA_HOME` Environment Variable
- Add Java to your `PATH`

1.3. Lab Exercise

In this exercise, you are going to download and install the Java Development Kit on your local workstation. This process involves downloading the JDK, unpacking the distribution, and performing a number of platform specific steps to set the appropriate environment variables.

> Note
>
> The following exercises make reference to a web site and download URL for the Sun Java download website. If you do not have access to the Internet, please use the files that were distributed with this training material and referenced in Table 3, “Training Software Prerequisites”. When your instructor handed out these training materials you should have received a CD containing these files or instructions for downloading them from a web site.

1.3.1. Step 1: Download the Java Development Kit

> Note
>
> If you do not have access to the Internet from your training workstation, or if you want to use the JDK distribution contained in the training software distribution, you can safely skip this step.

A. Go to http://java.sun.com/javase/downloads/index_jdk5.jsp. You should see the following web site:

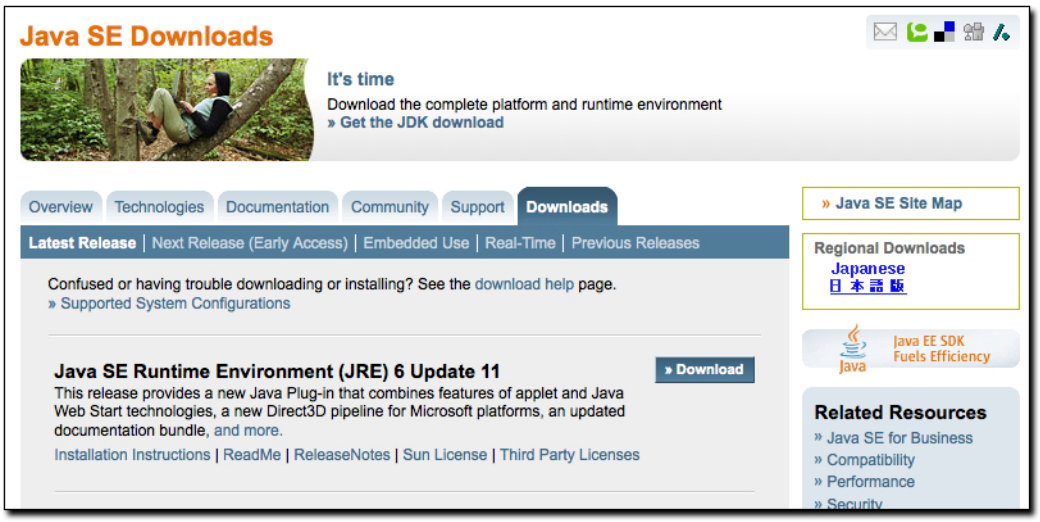

Figure 1.1. Sun Java Download Site

B. Under the Downloads tab, you should see a link for "Previous Releases". Click on "Previous Releases".

Figure 1.2. Previous Java Releases

C. Once you click on "Previous Releases", click on "J2SE 5.0 Downloads" to see a list of downloaded Java distributions as shown in Figure 1.3, “Downloading JDK 5.0”.

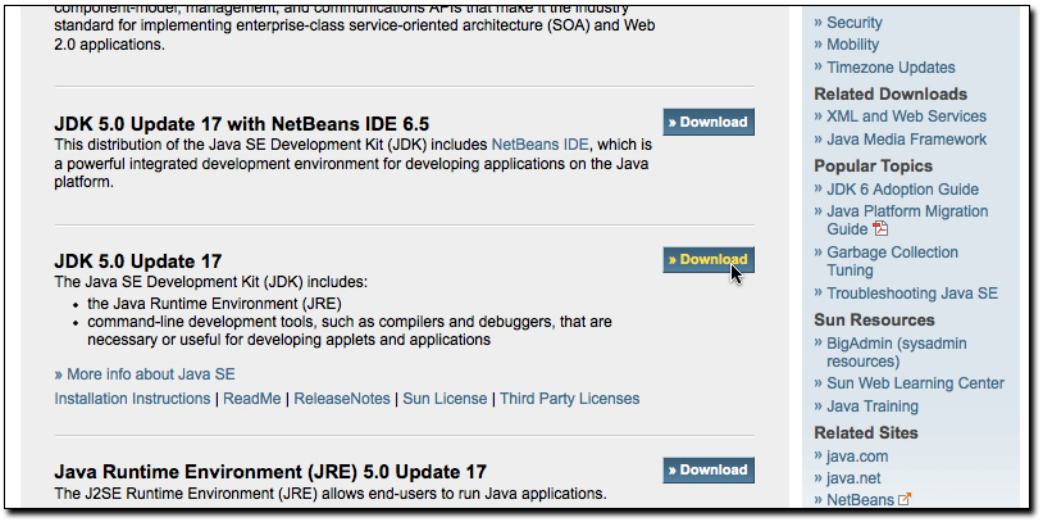

Figure 1.3. Downloading JDK 5.0

D. Select your platform and your language and click on the check box to agree to Sun Microsystem's Terms of Use, then click on Continue.

E. Select the appropriate distribution archive for your selected language and platform, and download this distribution to your local workstation.

Note

To download the JDK for a Mac OSX platform, visit http://developer.apple.com/java/. For the purposes of this training classes, you will need to download "Java for Mac OS X 10.5 Update 2".

1.3.2. Step 2: Install the JDK

The installation process for the JDK varies by platform. The following sections provide a general outline of the installation platform for selected platforms. If you experience any problems installing Java, please ask your instructor to help with the installation process.

1.3.2.WIN. Installation on Windows (32/64-bit)

To install the JDK on a Windows platform:

A. Run the executable distribution.

B. Follow the instructions on the step-by-step installation wizard.

Once the installation is completed Java will be installed in `c:\Program Files\Java\jdk1.5.0_19\`.

1.3.2.MAC. Installation on Mac OSX

To install the JDK on a Mac OSX platform:

A. Mount the "Java for Mac OS X 10.5 Update 2" disk image.

B. Double-click on the installer for Java.

Once completed, Java 5.0 will be available in `/System/Library/Frameworks/JavaVM.framework/Home`.

1.3.2.LINUX_RPM. Installation on Linux (32/64-bit) (RPM)

To install the JDK on a Linux platform (RPM):

A. Make the binary distribution executable with the chmod command.

B. Execute the binary distribution.

C. Copy the JDK to the appropriate installation directory.

The following screen listing assumes that you have sudo (or administrative rights) to install Java.

```
$ chmod 700 jdk-1_5_0_19-linux-i586.bin
$ ./jdk-1_5_0_19-linux-i586.bin
```

```
$ rpm -i java-1.5.0-sun-compat-1.5.0.19-1jpp.noarch.rpm
```

1.3.2.LINUX. Installation on Linux (32/64-bit) (non-RPM)

To install the JDK on a Linux platform (non-RPM):

A. Make the binary distribution executable with the chmod command.

B. Execute the binary distribution.

C. Copy the JDK to the appropriate installation directory.

The following screen listing assumes that you have sudo (or administrative rights) to create a directory in `/usr/java`.

```
$ chmod 700 jdk-1_5_0_19-linux-i586.bin
$ ./jdk-1_5_0_19-linux-i586.bin
$ sudo mv jdk1.5.0_19 /usr/java
$ ln -s /usr/java/jdk1.5.0_19 /usr/java/jdk
```

1.3.3. Step 3: Set the JAVA_HOME Environment Variable and PATH

Once the JDK is installed, you need to set an environment variable named `JAVA_HOME` which will point to the JDK directory.

If you have any issues setting an environment variable on your workstation, please ask the instructor to help with this configuration step. After successfully configuring your `JAVA_HOME` and `PATH`, running java -version should return the current version of the JDK.

```
$ java -version
java version "1.5.0_19"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_19-b02)
Java HotSpot(TM) Client VM (build 1.5.0_19-b02, mixed mode, sharing)
```

1.3.3.WIN. Setting JAVA_HOME and PATH on Windows

To set the JAVA_HOME and PATH environment variables in a Windows environment:

A. Go to Control Panel → System → Advanced → Environment Variables.

B. Add the environment variable `JAVA_HOME` to point to the JDK installation directory.

C. Add or modify the `PATH` variable to include `%JAVA_HOME%\bin`.

D. Run a new instance of cmd.exe.

E. Execute echo %JAVA_HOME% and execute java -version.

If the environment variable has been successfully set you should be able to run java from the command-line.

1.3.3.MAC. Setting JAVA_HOME and PATH on Mac OSX

To set the JAVA_HOME and PATH environment variables in a Mac OSX environment running bash:

A. Edit your ~/.bash_profile and add the following lines:

```
export JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Home
export PATH=${JAVA_HOME}/bin:${PATH}
```

B. Start a new instance of Terminal to load your new environment settings.

If the JAVA_HOME environment variable has been successfully set, running echo $JAVA_HOME should return the installation directory of the JDK.

1.3.3.LINUX. Setting JAVA_HOME and PATH on Linux

To set the JAVA_HOME and PATH environment variables in a Linux environment running bash:

A. Edit your ~/.bash_profile and add the following lines:

```
export JAVA_HOME=/usr/java/jdk
export PATH=${JAVA_HOME}/bin:${PATH}
```

B. Logout and login again to ensure that your JAVA_HOME and PATH variables have been configured.

If the JAVA_HOME environment variable has been successfully set, running echo $JAVA_HOME should return the installation directory of the JDK.

Lab 2. Maven Installation

2.1. Goal

This lab exercise is designed to walk you through the process of installing Maven on your own workstation. After completing this lab exercise you should be able to install Maven on any workstation and configure the appropriate environment variables. This lab ends with a brief survey of the contents of the Maven installation directory.

2.2. Agenda

- Download Maven
- Install Maven
- Set the `M2_HOME` Environment Variable
- Add Maven to your `PATH`
- Update your Global `settings.xml` file

2.3. Lab Exercises

In this exercise, you are going to download and install Maven on your local workstation. The process is as straightforward as downloading and unpacking an archive. Let's get started.

> Note
>
> The following exercises make reference to a web site and download URL for a Maven website. If you do not have access to the Internet, please use the files that were distributed with this training material and referenced in Table 3, “Training Software Prerequisites”. When your instructor handed out these training materials you should have received a CD or a USB Key containing these files or instructions for downloading them from a web site.

2.3.1. Step 1: Download Maven Distribution

To download the Maven Distribution:

A. Go to http://maven.apache.org. You should see the following web site:

Step 1: Download Maven Distribution

Figure 2.1. The Apache Maven Web Site

B. On the right-hand side of the web page shown in Figure 2.1, “The Apache Maven Web Site” you should see "Get Maven 2.2.1". Click on the first link in this section, the link titled "Maven 2.2.1" next to the folder icon. This will take you to a list of Maven distributions to download shown in Figure 2.2, “Maven Download Page”.

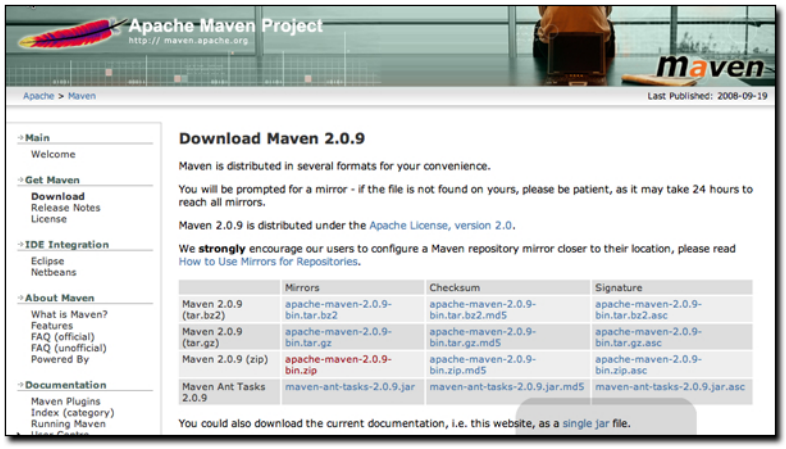

Figure 2.2. Maven Download Page

C. Click on the `apache-maven-2.2.1-bin.tar.gz` link in the "Mirrors" column of this table. You should then see the "Apache Download Mirrors" page shown in Figure 2.3, “Apache Download Mirrors Selection Page”.

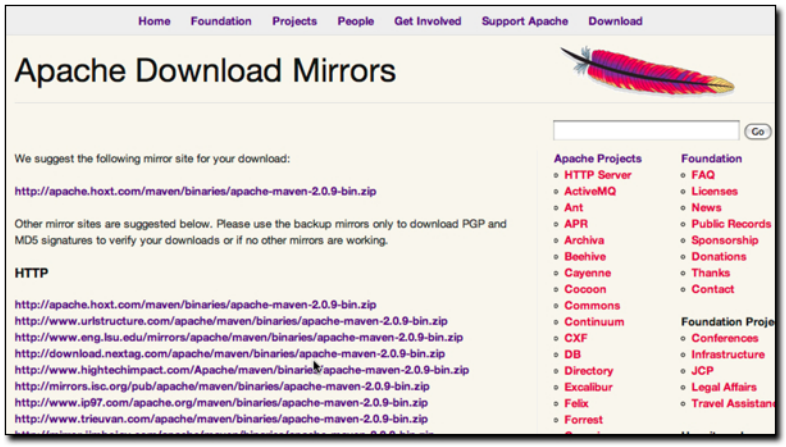

Figure 2.3. Apache Download Mirrors Selection Page

D. Click on the suggested download URL and download Maven to your local workstation.

   If you are running Windows, download the Maven distribution to `C:\`.

   If you running Mac OSX, Linux, or Unix, download the Maven archive to your home directory.

2.3.2. Step 2: Installing Maven

Once you've downloaded the Maven distribution:

A. Unpack the distribution on your local workstation.

   If you downloaded the archive to the directories suggested in the previous section you will end up with one of the following directories:

   Table 2.1. Installation Directories by Platform

Platform	Directory
Windows	`C:\Program Files\apache-maven-2.2.1`
Mac OSX	`~/apache-maven-2.2.1`
Linux/Unix	`~/apache-maven-2.2.1`

   Note

   For the purposes of this training material, you are instructed to install Maven in your home directory on a Linux or Unix machine. If you are installing Maven on a multi-user machine, you should install Maven to the appropriate directory. On a Mac OSX machine this could be `/Applications/` or `/usr/local/`, on a Linux machine this could

be /usr/local. This training exercise does not assume that you have the appropriate administrative privileges to install Maven in a shared location.

2.3.3. Step 3: Set the M2_HOME Environment Variable

Once Maven is installed, you need to set an environment variable named M2_HOME which will point to the Maven installation directory.

If you have any issues setting an environment variable on your workstation, please ask the instructor to help with this configuration step.

2.3.3.WIN. Setting M2_HOME in a Windows Environment

On a Windows machine:

A. Goto Control Panel → System → Advanced → Environment Variables

B. Add the environment variable M2_HOME to with a value of C:\Program Files\apache-maven-2.2.1

C. Go to the command-line in Windows, and execute echo %M2_HOME%.

If the environment variable has been successfully set, this should return the installation directory of Maven.

2.3.3.MAC_LINUX. Setting M2_HOME in a Mac OSX or Linux Environment

When running Mac OSX or Linux:

A. Edit your ~/.bash_profile and add the following line:

```
export M2_HOME=~/apache-maven-2.2.1
```

B. In Mac OSX, start a new instance of Terminal. In Linux, log out and log in again to reload your environment.

C. Run echo $M2_HOME at the command line.

If the M2_HOME environment variable has been successfully set, running echo $M2_HOME should return the installation directory of Maven.

2.3.4. Step 4: Add Maven to your PATH

With the M2_HOME environment variable set, the next step is to add Maven to your PATH. Once Maven is on your PATH, you will be able to call mvn from the command-line.

If you have any issues setting an environment variable on your workstation, please ask the instructor to help with this configuration step.

2.3.4.WIN. Setting PATH in a Windows Environment

On a Windows machine:

A. Goto Control Panel → System → Advanced → Environment Variables

B. Add `%M2_HOME%/bin` to the end of your `PATH` environment variable.

C. Start a new instance of cmd.exe to reload your environment.

2.3.4.MAC_LINUX. Setting PATH in a Mac OSX or Linux Environment

On a Mac OSX or a Unix/Linux machine:

A. Edit your `~/.bash_profile` and add the following line:

```
export PATH=${PATH}:${M2_HOME}/bin
```

B. In Mac OSX, start a new instance of Terminal. In Linux, log out and log in again to reload your environment.

2.3.5. Configuring Maven to work with a Proxy

If you need to go through a Proxy server to access the internet, you will need to tell Maven about it. To do this, create a file called `settings.xml` in the ".m2" directory in your user home directory. For example, for the "Training" user on a Windows XP machine, this directory would be in "C:\Documents and Settings\Training\.m2". An example of a simple configuration (where no authentication is required) is shown here:

```
<settings>
  <proxies>
    <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>proxy.acme.com</host>
      <port>3128</port>
    </proxy>
  </proxies>
</settings>
```

A more complete example is shown here:

```
<settings>
  <proxies>
    <proxy>
      <active>true</active>
      <protocol>http</protocol>
```

```
      <host>proxy.acme.com</host>
      <port>8080</port>
      <username>scott</username>
      <password>tiger</password>
      <nonProxyHosts>*.acme.com</nonProxyHosts>
    </proxy>
  </proxies>
</settings>
```

2.3.6. Step 5: Run Maven

With Maven installed and available on the `PATH`, you can now run Maven with mvn. Execute the following command, and you should see the following output:

```
$ mvn -v
Maven version: 2.2.1
Java version: 1.5.0_19
Default locale: en_US, platform encoding: MacRoman
OS name: "mac os x" version: "10.5.5" arch: "i386" family: "unix"
```

If you see this output you have successfully installed Maven, and you are ready to continue with your training. If you would like a brief survey of the files that were installed with Maven, continue reading the Extra Credit section.

2.4. Extra Credit

Once Maven is installed, take a look at the contents of the installation directory. The Maven directory should contain the following contents:

Table 2.2. Contents of the Maven Installation Directory

File/Directory	Description
`LICENSE.txt`	A copy of the Apache Software License version 2.0.
`README.txt`	Some notes about Maven for new users and pointers to support resources.
`/bin`	Shell scripts and batch files for Maven
`/boot`	Classworlds library used to bootstrap Maven
`/conf`	Global configuration settings for Maven
`/lib`	Core Maven library
`NOTICE.txt`	Notice required for Apache and Codehaus.

When you download Maven, you are downloading a very lightweight shell which contains only the core logic required to download and execute Maven plugins and to read Maven configuration files. Once you start to use Maven, you'll see it download artifacts and dependencies to your workstation.

These downloaded dependencies and plugins are going to be stored in what is called your Local Maven Repository.

Because you just downloaded and installed Maven, you won't have a Local Maven Repository on your system just yet. To create a Local Maven Repository, you'll have to run something that causes Maven to download some artifacts and plugins. To do this, run mvn help:system.

```
$ mvn help:system
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'help'.
[INFO] org.apache.maven.plugins: checking for updates from central
[INFO] org.codehaus.mojo: checking for updates from central
[INFO] artifact org.apache.maven.plugins:maven-help-plugin: checking
       for updates from central
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/\
             maven-help-plugin/2.1/maven-help-plugin-2.1.pom
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [help:system] (aggregator-style)
[INFO] ------------------------------------------------------------------------
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven-artifact/\
             2.0.6/maven-artifact-2.0.6.pom
1K downloaded
[INFO] [help:system]
[INFO]
================================================================================
========================== Platform Properties Details ==========================
================================================================================

================================================================================
System Properties
================================================================================

java.runtime.name=Java(TM) 2 Runtime Environment, Standard Edition
sun.boot.library.path=/System/Library/Frameworks/JavaVM.framework/Versions/\
                      1.5.0/Libraries
java.vm.version=1.5.0_16-133
...

================================================================================
Environment Variables
================================================================================

JRUBY_HOME=/usr/local/jruby
JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Home
HOME=/Users/tobrien
M2_HOME=/usr/local/maven
...

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 35 seconds
[INFO] Finished at: Sat Nov 29 10:45:21 PST 2008
[INFO] Final Memory: 5M/11M
[INFO] ------------------------------------------------------------------------
```

You just ran the system goal of the Help plugin. A goal is a unit of work in Maven, and a plugin is a collection of goals. The Help plugin's system goal, is a goal which simply prints out all of the system properties and environment variables. Don't worry if you don't understand exactly what

that means yet as you will be learning about plugins and goals later on in this workbook. We ran `help:system` to cause Maven to download a few artifacts and create a Local Repository. Let's take a look at the Local Repository and your own Maven settings directory.

Maven download artifacts and plugins to your Local Maven Repository in `~/.m2/repository`. Now that you are finished with this lab exercise, you might want to take some time to look at the contents of this Local Maven Repository to see what Maven downloaded.

2.5. Discussion

You should now be able to install Maven on your local workstation and configure the `M2_HOME` and `PATH` environment variables. You are now ready to continue learning about the various ways Maven can help you gain more control over your build process.

Lab 3. Installing Lab Solutions and Starting Points

3.1. Goal

In this brief lab exercise, you are going to copy the lab solutions for this workbook to your workstation.

3.2. The Lab Starting Points and Solutions

When a lab involves working on a Java application, you will find the starting point and the proposed solution for that lab in the 'labs' directory, which you will find in the Projects working directory. For example, for lab 6, the starting point can be found in C:\Solutions\labs\lab-06\start and the solution can be found in C:\Solutions\labs\lab-06\solutions.

When you are working on a particular lab from the starting point, you should work directly in the 'start' directory. Alternatively, you can copy the contents of the 'start' directory into another directory (e.g. C:\Projects) for each lab.

3.3. Agenda

- Download Lab Solutions and Starting Points
- Install Lab Solutions

3.4. Lab Exercises

3.4.1. Step 1: Download Lab Solutions

To download the Lab Solutions:

A. Open a web browser.

B. Type in the URL: http://sonatype-train.s3.amazonaws.com/labs.zip

C. Download this archive to your workstation

3.4.2. Step 2: Install Lab Solutions

Once you've downloaded the Lab Solutions archive:

Step 2: Install Lab Solutions

A. Unpack the Lab solutions in C:\Solutions.

If you are not working on a Windows platform, consult the following table for suggested location for the lab solutions:

Table 3.1. Installation Directories by Platform

Platform	Directory
Windows	`C:\Solutions`
Mac OSX	`~/solutions`
Linux/Unix	`~/solutions`

After unpacking the labs.zip archive in C:\Solutions, you should have a number of sub-directories, one for each lab involving work with source code:

```
$ ls
lab-03  lab-06  lab-07  lab-08  lab-09  lab-10  lab-11  lab-12  lab-13  lab-14  lab-15
```

Each lab directory has two sub-directories. The 'start' sub-directory contains a starting point for the lab. Import this project into Eclipse when you start the lab. The 'solution' directory contains the worked solution for this lab.

```
$ ls lab-03
solution        start
```

Lab 4. Creating a new Maven project

4.1. Goal

The aim of this lab exercise is to learn how to create a new Maven project from the command line, using the Maven Archetype plugin.

4.2. Agenda

- Create a new Maven project using mvn archetype:generate.
- Study the default directory structure
- Compile, test and package the project

4.3. Lab Exercises

In this exercise, you are going to create a very simple project. This project involves implementing a simple social networking web site, called Babble. We will be starting simple, and progressively adding features as we go. Let's get started.

4.3.1. Step 1: Create a project directory

The first thing to do is to create a directory called "projects".

A. Change directories to C:\: cd c:\

B. Create a new projects directory: mkdir Projects

> Note
>
> If you are working on a Mac OSX or Linux workstation, you should create your Projects directory under your home directory. For more information, see Section 1.2, “Projects Working Directory”.

4.3.2. Step 2: Creating a new Maven project

The first project we will create is the core module of our new killer social networking application, and will contain the domain model code. We will create this using the maven archetype plugin. Now before we create a new Maven project, we need a groupId and artifactId for our project. For this project:

- the groupId is com.sonatype.training,

- and the artifactId is babble-core.

To create this project:

A. Go to your projects directory and type mvn archetype:generate. Press Enter. This will display the following output.

```
C\:\\Projects> mvn archetype:generate
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [archetype:generate] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] Preparing archetype:generate
[INFO] No goals needed for project - skipping
[INFO] Setting property: classpath.resource.loader.class => 'org.codehaus.plexus.velo
[INFO] Setting property: velocimacro.messages.on => 'false'.
[INFO] Setting property: resource.loader => 'classpath'.
[INFO] Setting property: resource.manager.logwhenfound => 'false'.
[INFO] [archetype:generate]
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.arche
Choose archetype:
1: internal -> appfuse-basic-jsf (AppFuse archetype for creating a web application wi
2: internal -> appfuse-basic-spring (AppFuse archetype for creating a web application
3: internal -> appfuse-basic-struts (AppFuse archetype for creating a web application
4: internal -> appfuse-basic-tapestry (AppFuse archetype for creating a web applicati
5: internal -> appfuse-core (AppFuse archetype for creating a jar application with Hi
6: internal -> appfuse-modular-jsf (AppFuse archetype for creating a modular applicat
7: internal -> appfuse-modular-spring (AppFuse archetype for creating a modular appli
8: internal -> appfuse-modular-struts (AppFuse archetype for creating a modular appli
9: internal -> appfuse-modular-tapestry (AppFuse archetype for creating a modular app
10: internal -> maven-archetype-j2ee-simple (A simple J2EE Java application)
11: internal -> maven-archetype-marmalade-mojo (A Maven plugin development project us
12: internal -> maven-archetype-mojo (A Maven Java plugin development project)
13: internal -> maven-archetype-portlet (A simple portlet application)
14: internal -> maven-archetype-profiles ()
15: internal -> maven-archetype-quickstart ()
16: internal -> maven-archetype-site-simple (A simple site generation project)
17: internal -> maven-archetype-site (A more complex site project)
18: internal -> maven-archetype-webapp (A simple Java web application)
19: internal -> jini-service-archetype (Archetype for Jini service project creation)
20: internal -> softeu-archetype-seam (JSF+Facelets+Seam Archetype)
21: internal -> softeu-archetype-seam-simple (JSF+Facelets+Seam (no persistence) Arch
22: internal -> softeu-archetype-jsf (JSF+Facelets Archetype)
23: internal -> jpa-maven-archetype (JPA application)
24: internal -> spring-osgi-bundle-archetype (Spring-OSGi archetype)
25: internal -> confluence-plugin-archetype (Atlassian Confluence plugin archetype)
26: internal -> jira-plugin-archetype (Atlassian JIRA plugin archetype)
27: internal -> maven-archetype-har (Hibernate Archive)
28: internal -> maven-archetype-sar (JBoss Service Archive)
29: internal -> wicket-archetype-quickstart (A simple Apache Wicket project)
30: internal -> scala-archetype-simple (A simple scala project)
31: internal -> lift-archetype-blank (A blank/empty liftweb project)
32: internal -> lift-archetype-basic (The basic (liftweb) project)
33: internal -> cocoon-22-archetype-block-plain ([http://cocoon.apache.org/2.2/maven-
34: internal -> cocoon-22-archetype-block ([http://cocoon.apache.org/2.2/maven-plugin
35: internal -> cocoon-22-archetype-webapp ([http://cocoon.apache.org/2.2/maven-plugi
36: internal -> myfaces-archetype-helloworld (A simple archetype using MyFaces)
37: internal -> myfaces-archetype-helloworld-facelets (A simple archetype using MyFac
38: internal -> myfaces-archetype-trinidad (A simple archetype using Myfaces and Trin
39: internal -> myfaces-archetype-jsfcomponents (A simple archetype for create custom
```

```
40: internal -> gmaven-archetype-basic (Groovy basic archetype)
41: internal -> gmaven-archetype-mojo (Groovy mojo archetype)
Choose a number:  (1/2/3/4/5/6/7/8/9/10/11/12/13/14/15/16/17/18/19/20/21/22/23/24/25/
26/27/28/29/30/31/32/33/34/35/36/37/38/39/40/41) 15: :
```

B. Press ENTER to use the default archetype, archetype 15 which corresponds to "maven-archetype-quickstart". In this example, we only need a simple project structure, so the default option (maven-archetype-quickstart) will do fine.

C. When prompted for a groupId, enter com.sonatype.training

D. When prompted for an artifactId, enter babble-core

E. When prompted for a version, enter 1.0.0-SNAPSHOT

F. When prompted for a package, enter com.sonatype.training.babble.core

G. Press Enter or Y to confirm the configuration for the Maven Archetype generation. Once this configuration has been confirmed, the Maven Archetype plugin will generate the following output:

```
Define value for groupId: : com.sonatype.training
Define value for artifactId: : babble-core
Define value for version:  1.0-SNAPSHOT: : 1.0.0-SNAPSHOT
Define value for package:  com.sonatype.training: : com.sonatype.training.babble.core
Confirm properties configuration:
groupId: com.sonatype.training
artifactId: babble-core
version: 1.0.0-SNAPSHOT
package: com.sonatype.training.babble.core
 Y: : Y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating OldArchetype: maven-archetype-quicksta
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.sonatype.training
[INFO] Parameter: packageName, Value: com.sonatype.training.babble.core
[INFO] Parameter: basedir, Value: C\:\\Projects
[INFO] Parameter: package, Value: com.sonatype.training.babble.core
[INFO] Parameter: version, Value: 1.0.0-SNAPSHOT
[INFO] Parameter: artifactId, Value: babble-core
[INFO] ******************** End of debug info from resources from generated POM ****
[INFO] OldArchetype created in dir: C\:\\Projects\babble-core
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 minutes 52 seconds
[INFO] Finished at: Wed Apr 01 12:32:02 NZDT 2009
[INFO] Final Memory: 8M/15M
[INFO] ------------------------------------------------------------------------
```

H. Change directories to the newly created `babble-core/` directory: cd babble-core

4.3.3. Step 3: Compile your project

This project is structurally complete, and contains a sample class and test case. To compile the project from the `babble-core/` project directory:

A. Compile the application using mvn compile:

```
C\:\\Projects\babble-core> mvn compile
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building babble-core
[INFO]    task-segment: [compile]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[INFO] skip non existing resourceDirectory C\:\\Projects\babble-core\src\mai
[INFO] [compiler:compile]
[INFO] Compiling 1 source file to C\:\\Projects\babble-core\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1 second
[INFO] Finished at: Wed Apr 01 13:18:09 NZDT 2009
[INFO] Final Memory: 6M/15M
[INFO] ------------------------------------------------------------------------
```

B. Change directories to `target/classes`: cd target/classes

C. List the directory contents to see the compiled bytecode of the babble-core project

D. Go back to the `babble-core/` project directory

4.3.4. Step 4: Test your project

The default JUnit test case doesn't do very much, but you can still execute the application unit tests. To execute unit tests from the `babble-core/` project directory:

A. Run the test phase by running mvn test

```
C\:\\Projects\babble-core> mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building babble-core
[INFO]    task-segment: [test]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[INFO] skip non existing resourceDirectory C\:\\Projects\babble-core\src\mai
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] skip non existing resourceDirectory C\:\\Projects\babble-core\src\tes
[INFO] [compiler:testCompile]
[INFO] Compiling 1 source file to C\:\\Projects\babble-core\target\test-clas
[INFO] [surefire:test]
[INFO] Surefire report directory: C\:\\Projects\babble-core\target\surefire-

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.core.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.036 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Wed Apr 01 13:19:41 NZDT 2009
[INFO] Final Memory: 7M/15M
[INFO] ------------------------------------------------------------------------
```

B. Change directories to `target/test-classes`: cd target/test-classes

C. List the directory contents to see the compiled unit test bytecode of the babble-core project

D. Go back to the `babble-core/` project directory: cd ../..

E. Change directories to the `target/surefire-reports` directory: cd target/surefire-reports

F. Read the contents of the `com.sonatype.training.babble.core.AppTest.txt` file: more com.sonatype.training.babble.core.AppTest.xml

G. Go back to the `babble-core/` project directory: cd ../..

Maven compiles the unit tests and puts the bytecode for your project's unit tests in `target/test-classes`, it then executes the unit tests and creates test reports in the `target/surefire-reports` directory. Maven generates test results in two formats: text and XML.

4.4. Extra Credit

When you run mvn test, Maven will also run the lifecycle phases preceding the test phase. For example, Maven will also compile the application code if required. Did you notice the following lines in the output when you ran the tests?

```
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
```

To put this to the test, delete the target directory by executing mvn clean, and then run mvn test again. Does Maven compile the application classes this time?

4.5. Discussion

You should now be able to create a Maven project from the command line, and invoke the basic life cycle steps on your project. In the next lab, you will learn how to do the same thing in Eclipse.

Lab 5. m2eclipse Installation

5.1. Goal

This lab exercise is designed to walk you through the process of installing Eclipse and m2eclipse.

5.2. Agenda

- Download Eclipse
- Install Eclipse
- Start Eclipse
- Install m2eclipse Prerequisites
- Install m2eclipse
- Verify m2eclipse Installation

5.3. Lab Exercises

In these exercises, we are going to download and install Eclipse on your local workstation. The process is as straightforward as downloading and unpacking an archive. Let's get started.

> Note
>
> The following exercises make reference to a web site and download URL for the Eclipse website. If you are do not have access to the Internet, please use the files that were distributed with this training material. When your instructor handed out training materials you should have received a CD or USB key that contained all of the necessary software for this training or instructions for download the prerequisites.

5.3.1. Step 1: Download Eclipse

To download Eclipse:

A. Go to http://www.eclipse.org/downloads/. You should see the following web site:

Step 1: Download Eclipse

Figure 5.1. The Eclipse Downloads Page

B. Download the Eclipse IDE for Java EE Developers distribution. Clicking on this link, or the platform specific link for this distribution will show the following page.

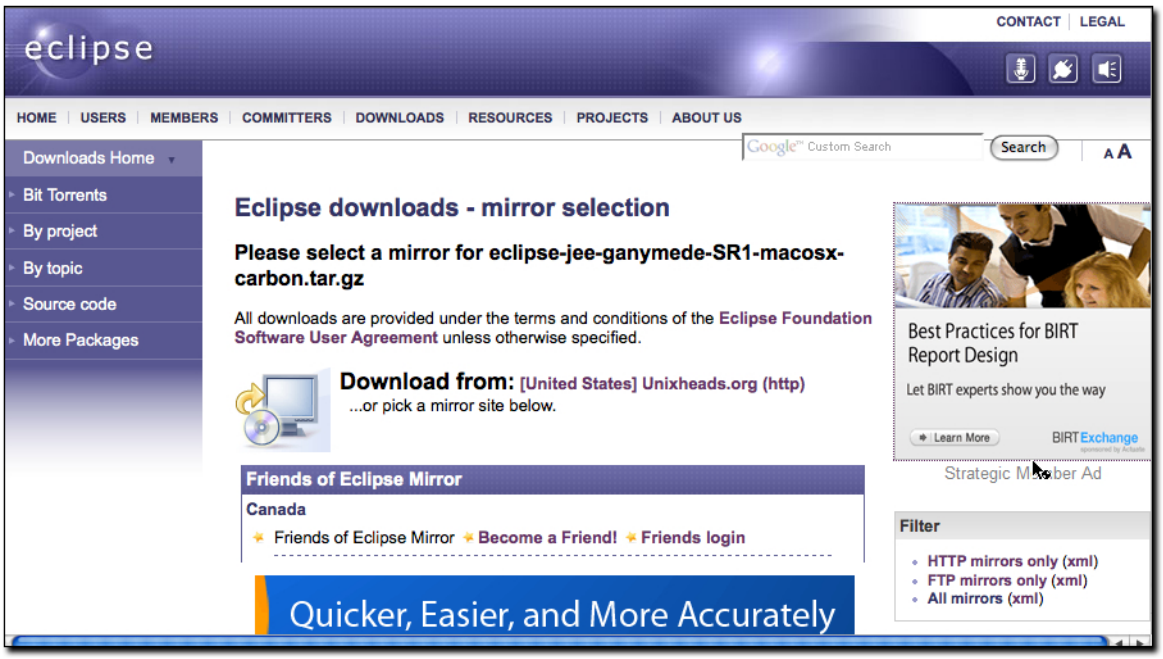

Figure 5.2. Eclipse Download Page: Mirror Selection

C. Click on the link next to "Download from:" and save the Eclipse distribution archive to your local workstation.

5.3.2. Step 2: Installing Eclipse

The following list shows the recommended installation paths for each platform:

Table 5.1. Installation Directories for Eclipse by Operating System

Operation System	Installation Directory
Windows	`C:\Program Files\eclipse`
Mac OSX	`/Applications/eclipse`
Unix/Linux	`/usr/local/eclipse`

If you are working on Windows Vista you may need administrator priviledges to install files into "C:\Program Files". If this is not possible, you can always install Eclipse in another directory, such as "C:\Tools\eclipse".

Once you've downloaded the Eclipse distribution:

A. Unpack the distribution on your local workstation.

B. Copy the eclipse directory to the suggested installation location for your platform.

i. On the Windows operating system, you should be able to double-click or right-click on the eclipse distribution archive and unpack the archive in a directory. You should then drag and drop the eclipse directory to "`C:\Program Files`".

ii. On a Mac, you should also double click on the downloaded archive and drag it to the `/Applications` folder. You can also unpack the archive using the command line as follows:

```
$ tar xvzf eclipse-jee-ganymede-SR1-macosx-carbon.tar.gz
$ sudo cp eclipse /Applications
```

5.3.3. Step 3: Adding the -vm Argument for m2eclipse

> Note
>
> This step is only relevant for students who are using a Windows operating system. If you are running on a Mac OSX or Linux environment, you can skip this step.

To use m2eclipse, you need to run Eclipse with the JDK. However, when running Eclipse on Windows, Eclipse may run by default with the JRE (Java Runtime Edition) version of Java. If this

happens, you might get an error similar to the following when you try to use Maven in Eclipse (while importing a project, for example):

```
6/25/07 1:15:44 PM CDT: ERROR mojo-execute : compiler:compile : Compilation failure
Unable to locate the Javac Compiler in:
C:\Program Files\Java\j2re1.4.2_14\..\lib\tools.jar
Please ensure you are using JDK 1.4 or above and
not a JRE (the com.sun.tools.javac.Main class is required).
In most cases you can change the location of your Java
installation by setting the JAVA_HOME environment variable.
```

To get around this, you need to ensure that Eclipse is running under a JDK. You can do this by using the -vm option to provide a path to your Java JDK. To do this:

A. Create a shortcut on the desktop (drag the eclipse icon onto the desktop while pressing ALT).

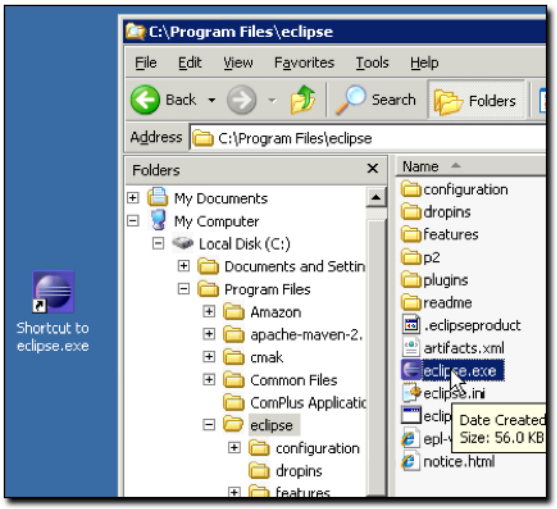

Figure 5.3. Creating a Shortcut for the eclipse.exe Executable

B. Modify the shortcut properties, right-click on the shortcut and select Properties.

C. Click on the Shortcut tab in the "Shortcut to eclipse.exe Properties" dialog.

D. Add the text: -vm "C:\Program Files\Java\jdk1.5.0_19\bin" to the Target. After adding this argument the text in Target should contain the following text (including the quotes):

```
"C:\Program Files\eclipse\eclipse.exe" -vm "C:\Program Files\Java\jdk1.5.0_19\bin"
```

Step 3: Adding the -vm Argument for m2eclipse

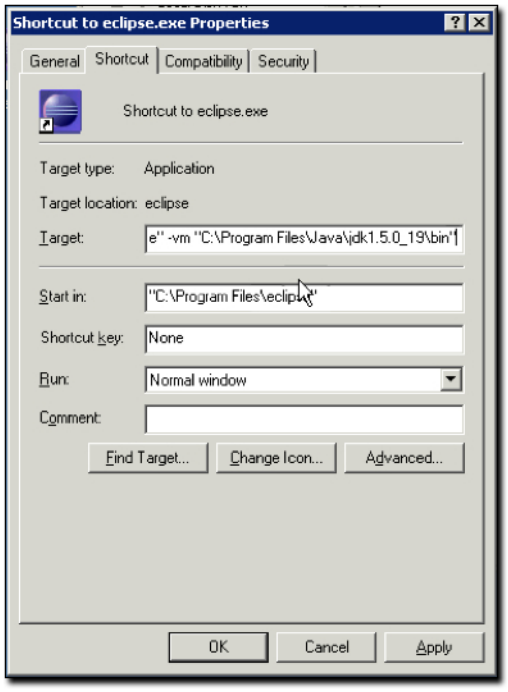

Figure 5.4. Adding the -vm argument to the Eclipse shortcut

E. You may also need to define a JDK in the Eclipse Preferences. Open the "Window / Preferences... / Java / Installed JREs" screen and add a reference to your JDK if it is not already present. You should also tick the JDK installation to make it the default JDK.

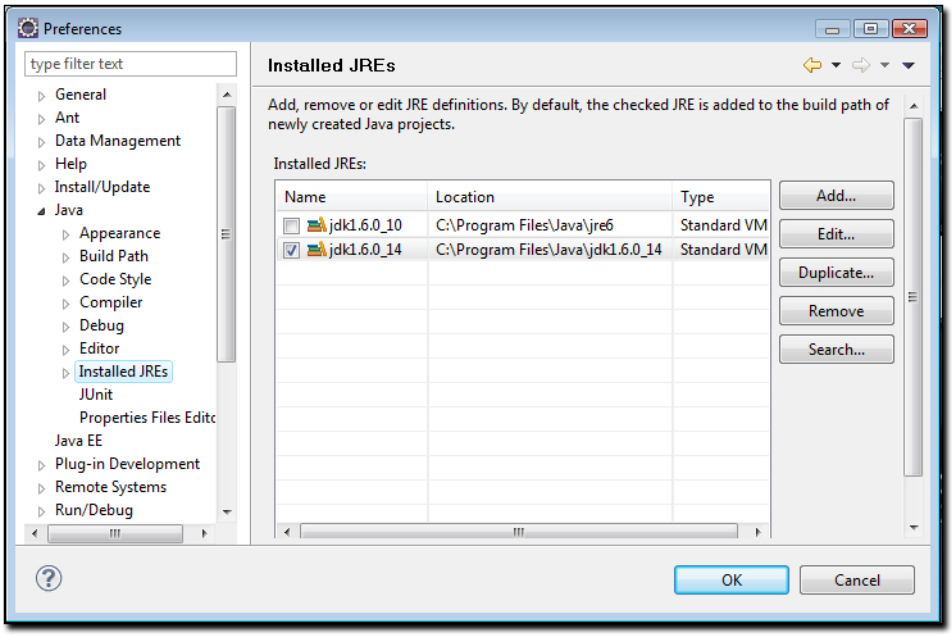

Figure 5.5. Defining a JDK in the Eclipse preferences

5.3.4. Step 4: Start Eclipse

To start eclipse:

A. Create a shortcut

B. Run the appropriate executable for your platform.

i. On Windows, double-click on the shortcut created in the previous step.

ii. On a Mac, you can run Eclipse by running the `eclipse` executable in the Eclipse installation directory using Finder or the from the command-line.

iii. On Linux, execute the `eclipse` executable from the command-line.

Once eclipse has been started, you should see the following screen while Eclipse is starting:

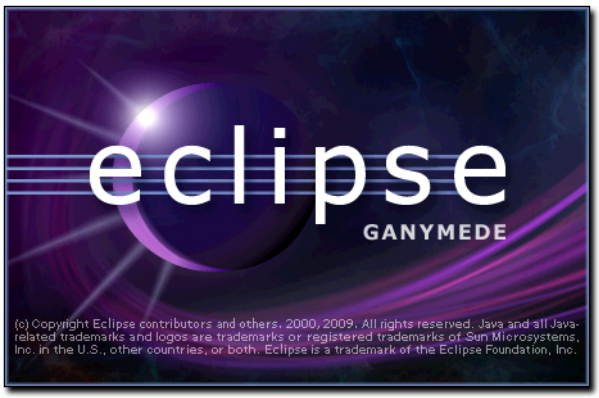

Figure 5.6. Eclipse Startup Screen

C. The first time you start Eclipse it will prompt you to create a new workspace. Create a new workspace in C:\workspace as shown in the following figure:

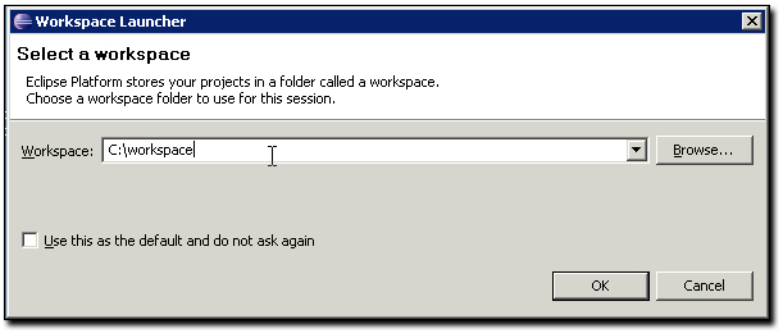

Figure 5.7. Creating a new Eclipse Workspace

D. Once Eclipse creates a new workspace, it will show you the Welcome tab. Click on the link to the Workbench on the right of this window.

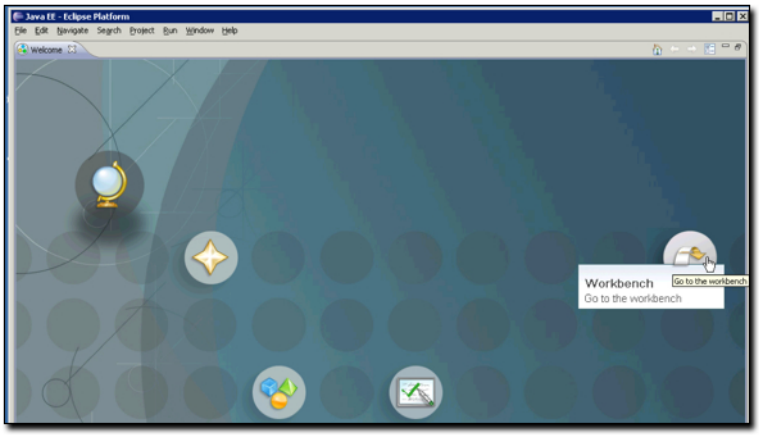

Figure 5.8. The Eclipse Welcome tab

> **Note**
>
> The Eclipse Welcome tab contains some helpful pointers to Eclipse resources for a new Eclipse user, to return to the Welcome tab at any time, select Welcome from the Help menu.

After starting Eclipse for the first time, you should see the Eclipse workbench:

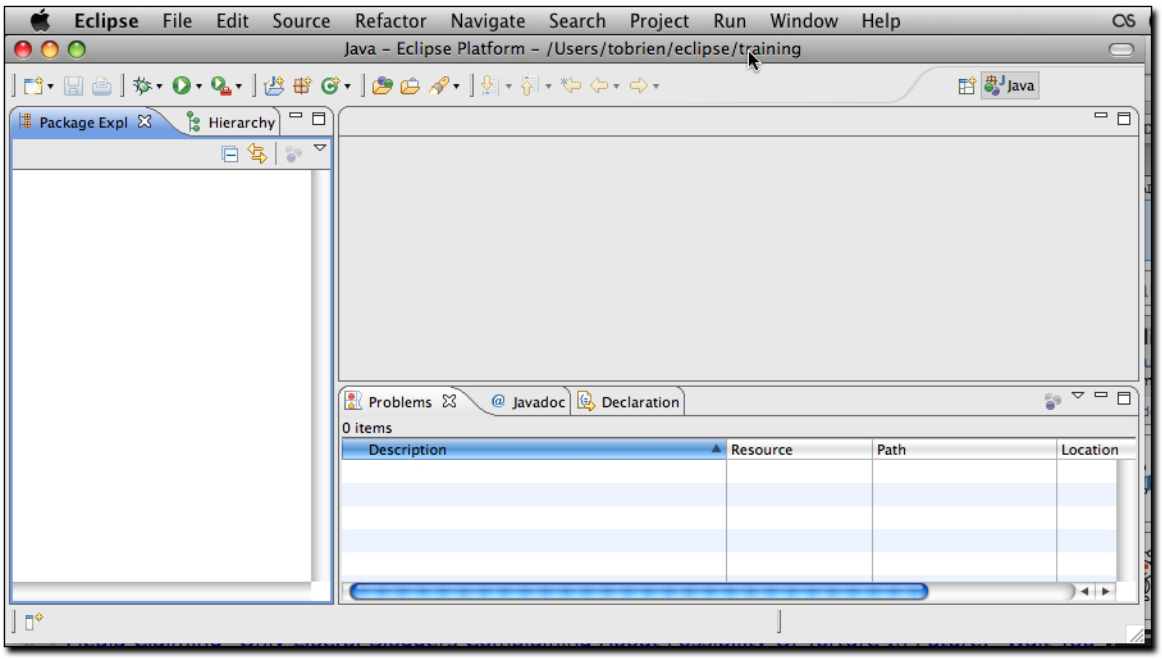

Figure 5.9. Eclipse Workbench After Installation

5.3.5. Step 4: Installing the m2eclipse Prerequisites

In this exercise, we're going to install one m2eclipse prerequisite, the Subversion plugin called Subclipse developed by the Tigris project. To do this:

A. Go to the Help menu and select Software Updates...

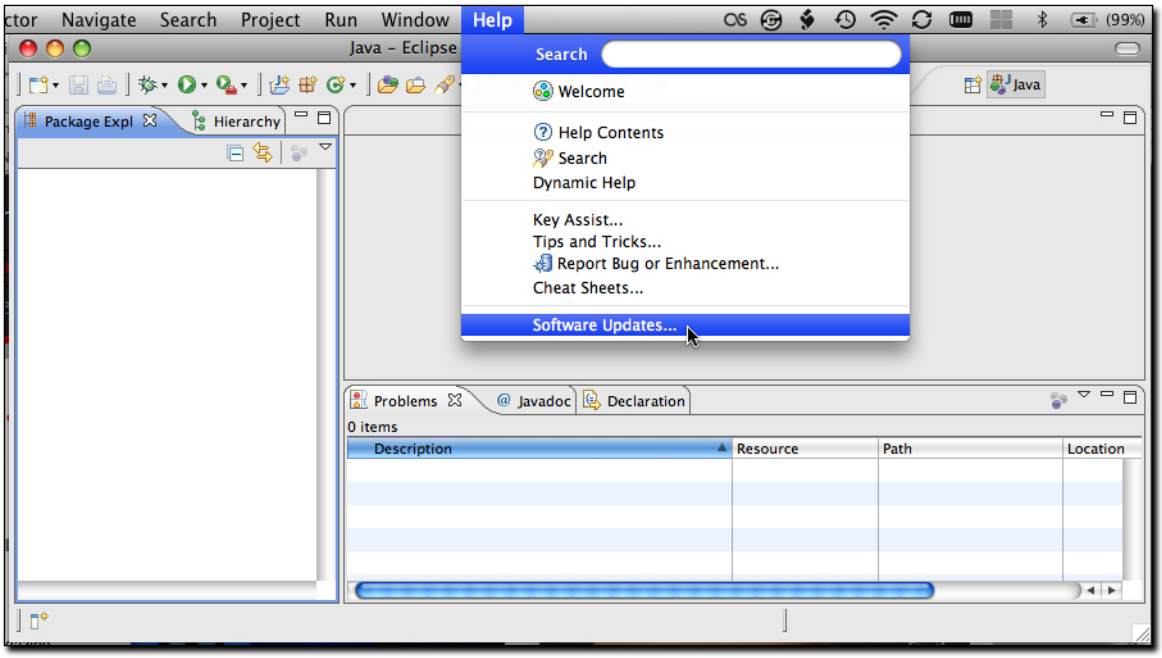

Figure 5.10. Select Software Updates from the Help Menu

B. Once you select this option, you will see the following dialog.

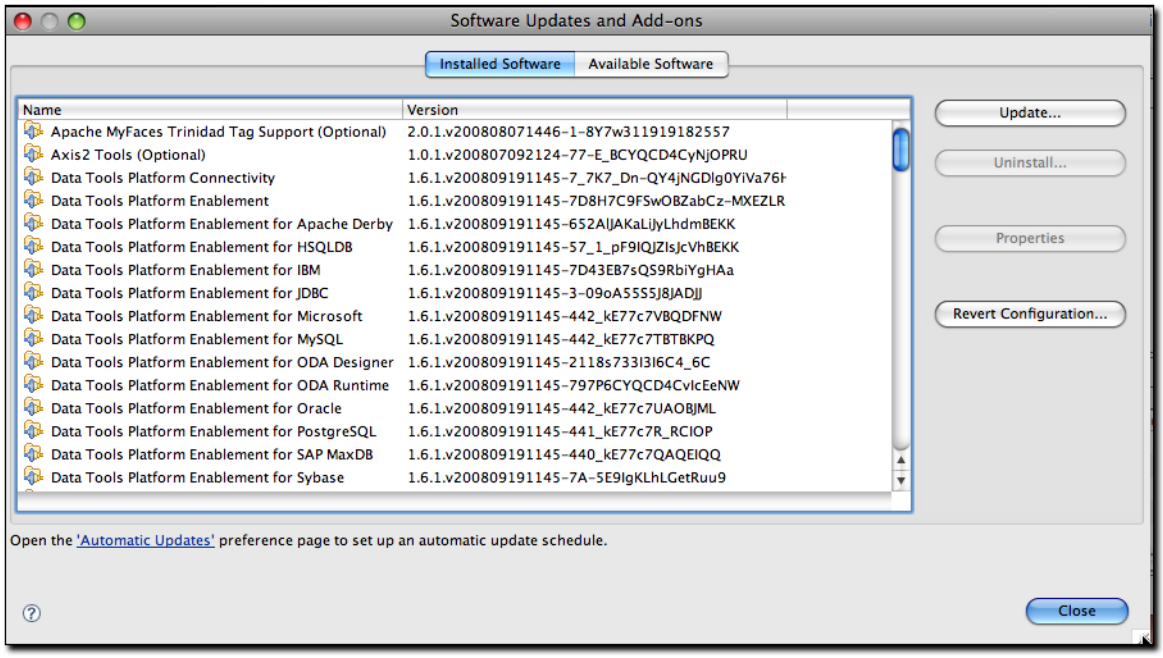

Figure 5.11. The Software Updates and Add-ons Dialog

This initial view shows you the components that are currently installed in your Eclipse installation. Because you downloaded the distribution for Java EE developers, you are going to have a distribution which already has components geared toward enterprise application development.

C. Click on Available Software and you will see a list of available update sites which Eclipse can check for new or updated plugins.

D. To add the Subclipse plugin's update site, click on Add Site, and add type in the URL for the Subclipse plugin's update site: http://subclipse.tigris.org/update_1.6.x

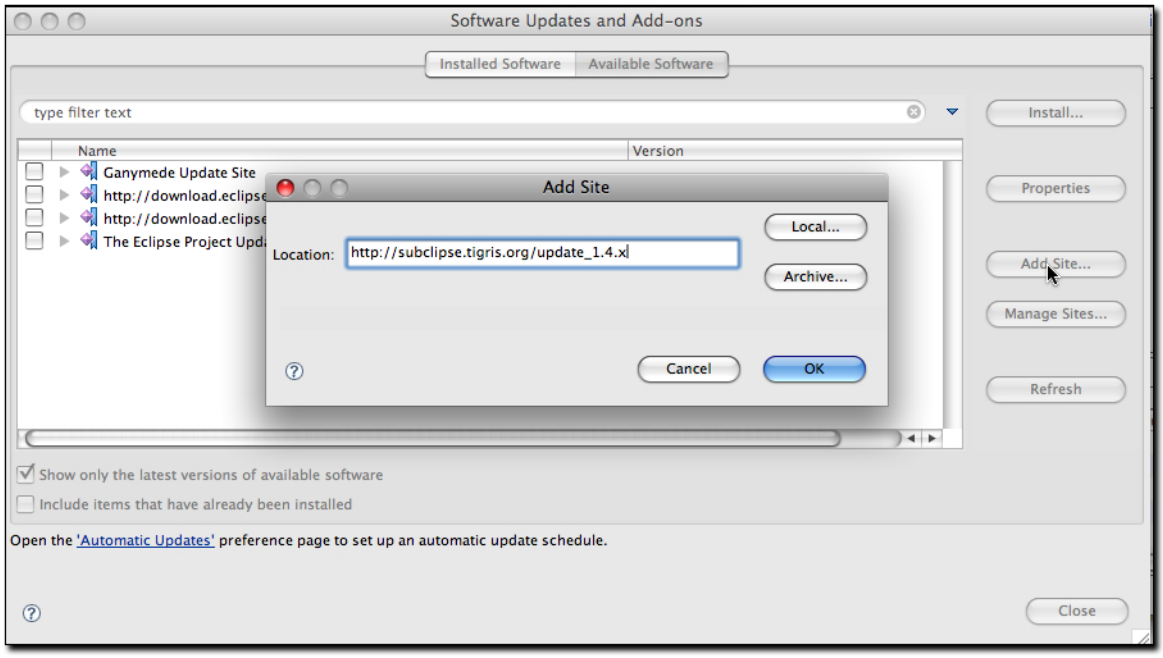

Figure 5.12. Adding the Subclipse Update Site

E. Click OK, and Eclipse will read the contents of the Subclipse Update site. A new update site should appear in the list of available update sites: http://subclipse.tigris.org/update_1.6.x.

F. Expand this site and select all of the components.

5.3.6. Installing the GEF framework

For m2eclipse to work in recent versions of Eclipse, you also need to upgrade the GEF libraries. To do this:

A. Go to the Help menu and select Software Updates... as shown in the previous exercise.

B. Add the GEF update site: click on "Add Site..." and add the GEF update site: http://download.eclipse.org/tools/gef/updates/release.

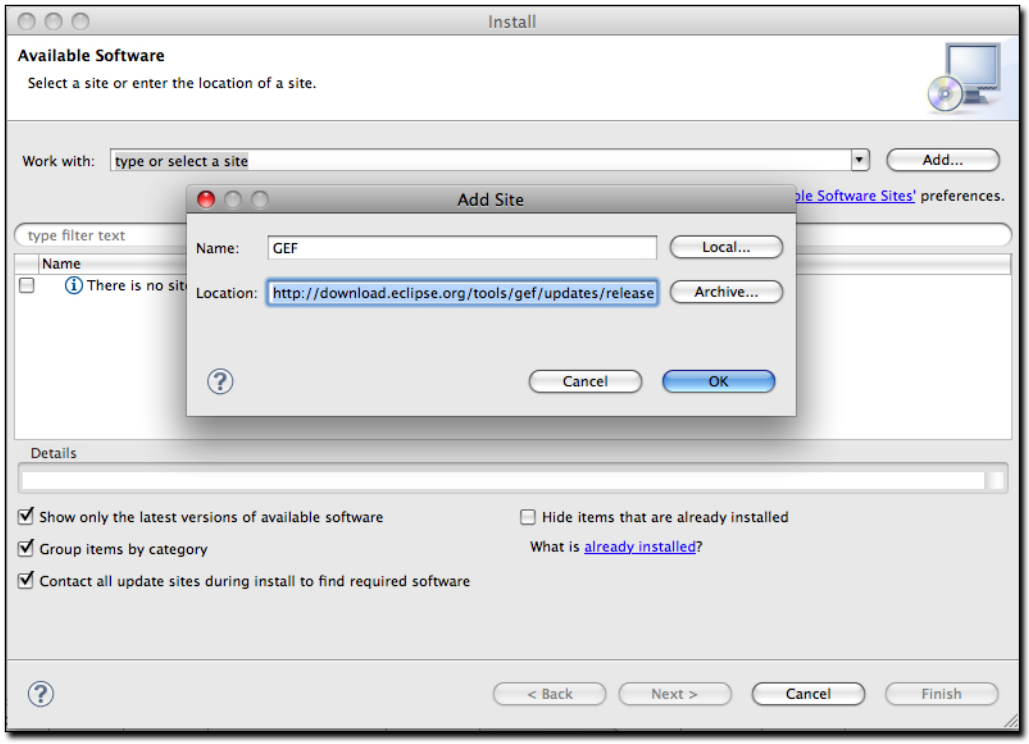

Figure 5.13. Adding the GEF Update Site

C. Eclipse will display a list of versions of the GEF framework - choose the one that matches your Eclipse installation (for example, GEF SDK 3.4.2 for Eclipse Ganymede or GEF SDK 3.5.0 for Eclipse Galileo).

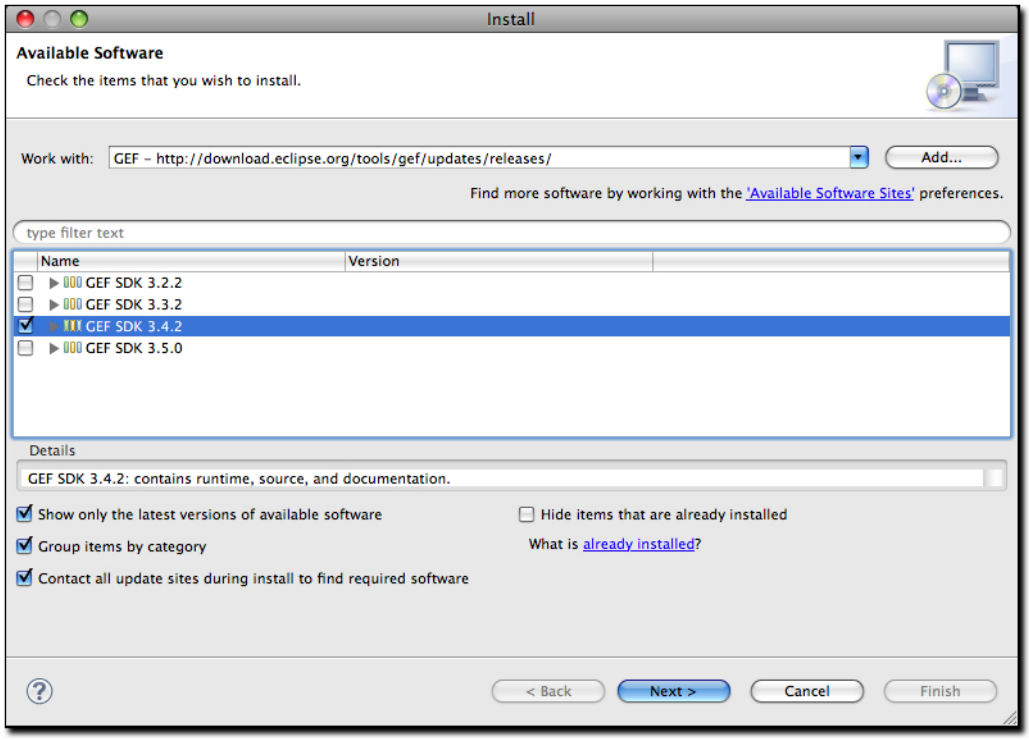

Figure 5.14. Choose the version of GEF that corresponds to your Eclipse installation

5.3.7. Step 5: Install m2eclipse

After adding the Update site for Subclipse and GEF, you should still have the Software Updates and Add-ons dialog open. To install the m2eclipse plugin:

A. Click Add Site... again and add the update site for m2eclipse: http://m2eclipse.sonatype.org/update/.

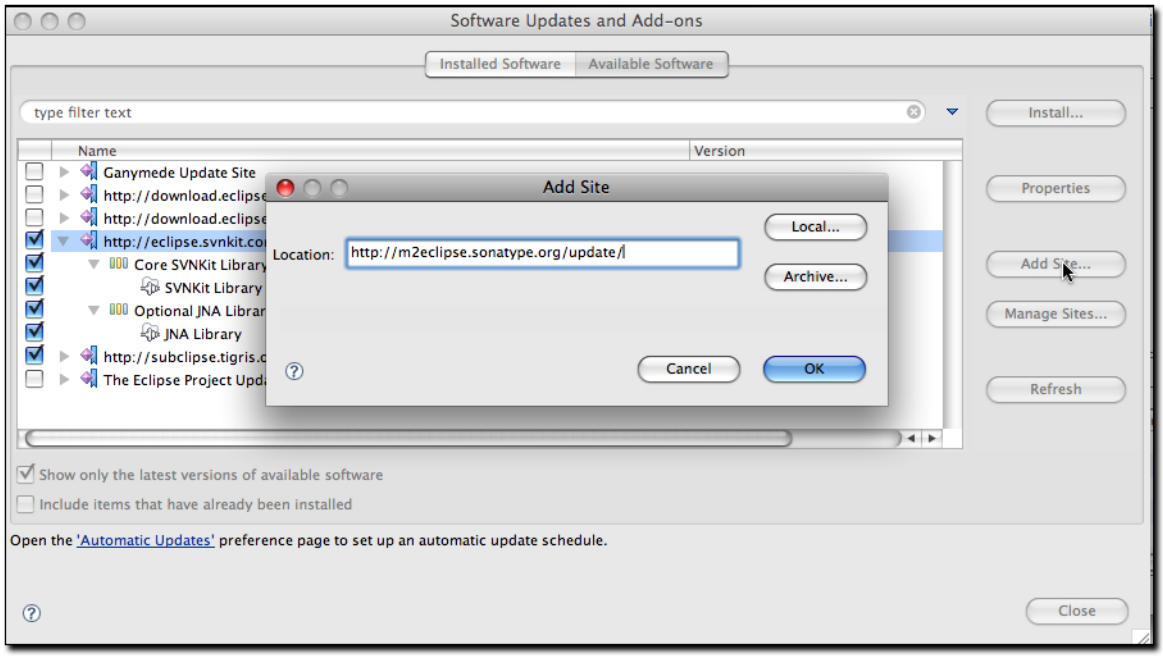

Figure 5.15. Adding the Update Site for m2eclipse

B. Click on OK, and then select all of the components under Maven Integration and all of the Maven Optional Components, with the exception of "Maven Integration for AJDT".

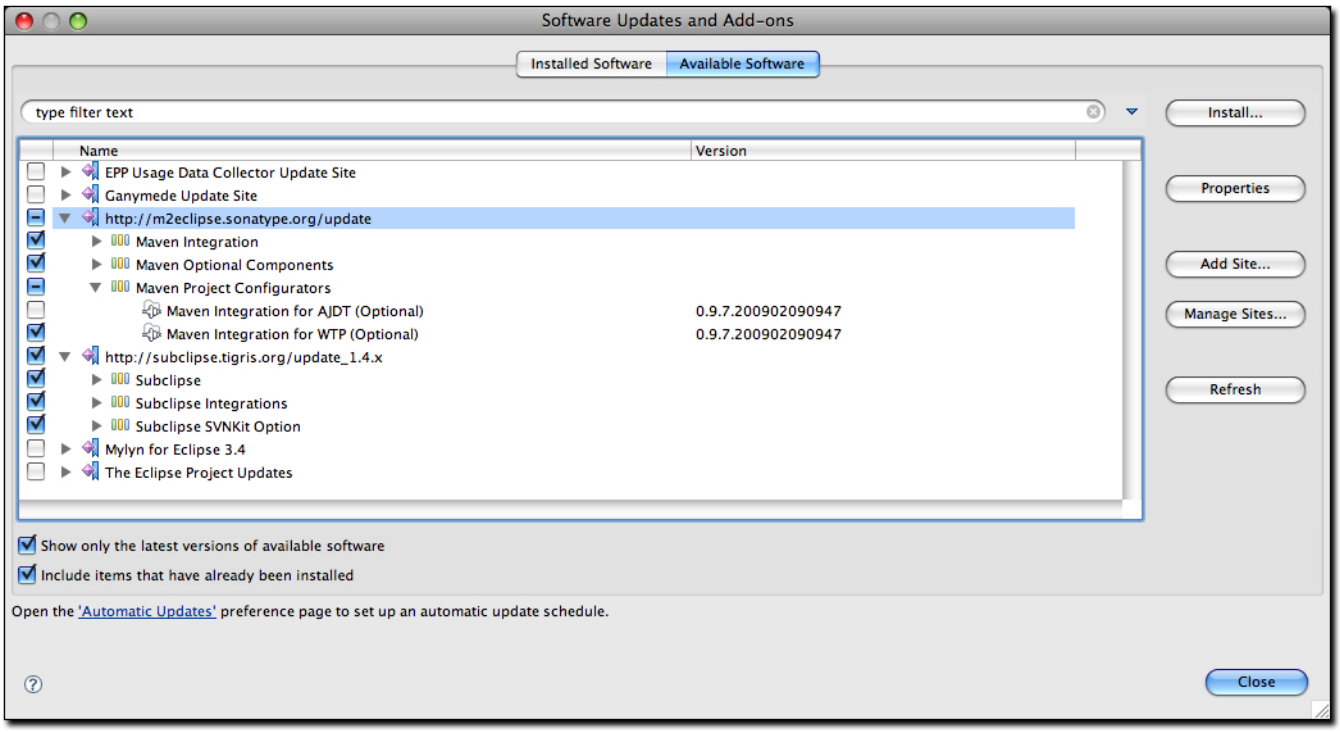

Figure 5.16. Selecting Components in m2eclipse

C. Click the Install button in the Software Updates and Add-ons dialog.

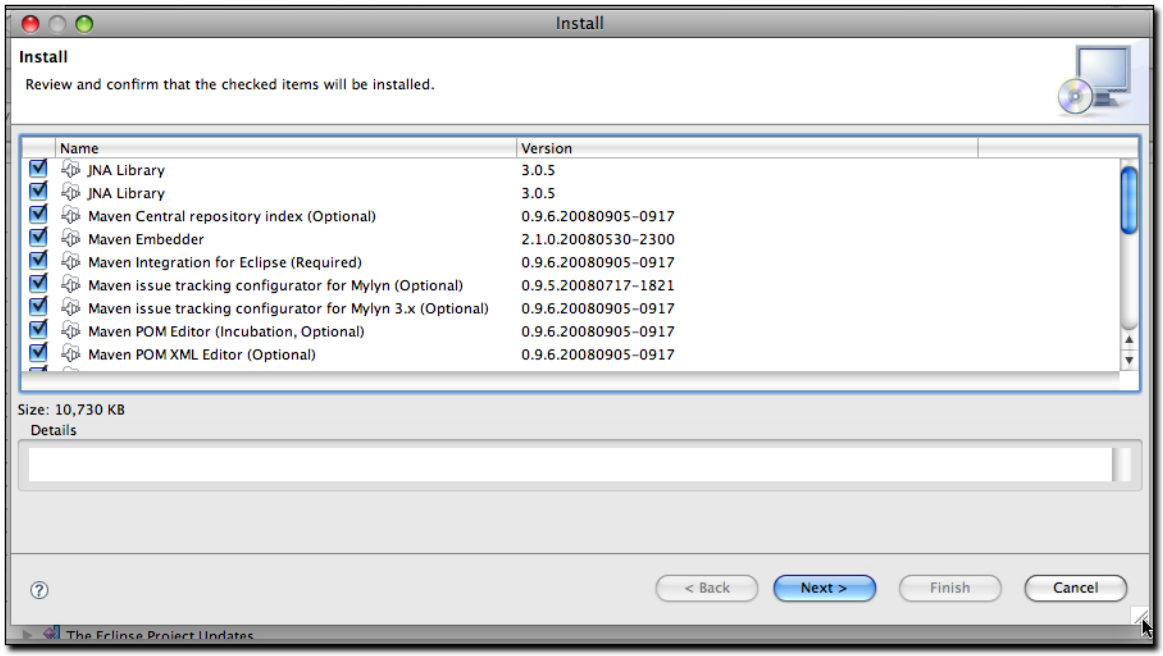

Figure 5.17. Review and Confirm Plugin Components to Install

Clicking the Install button brings up an Install dialog which lists all of the components selected for installation. If Eclipse finds any missing dependencies or problems which would prevent a successful installation of a plugin, these errors would show up in this dialog.

D. Click the Next button to continue installation of the m2eclipse plugin. Once you click Next, you will be asked to agree to the license agreements for the various components and plugins being downloaded.

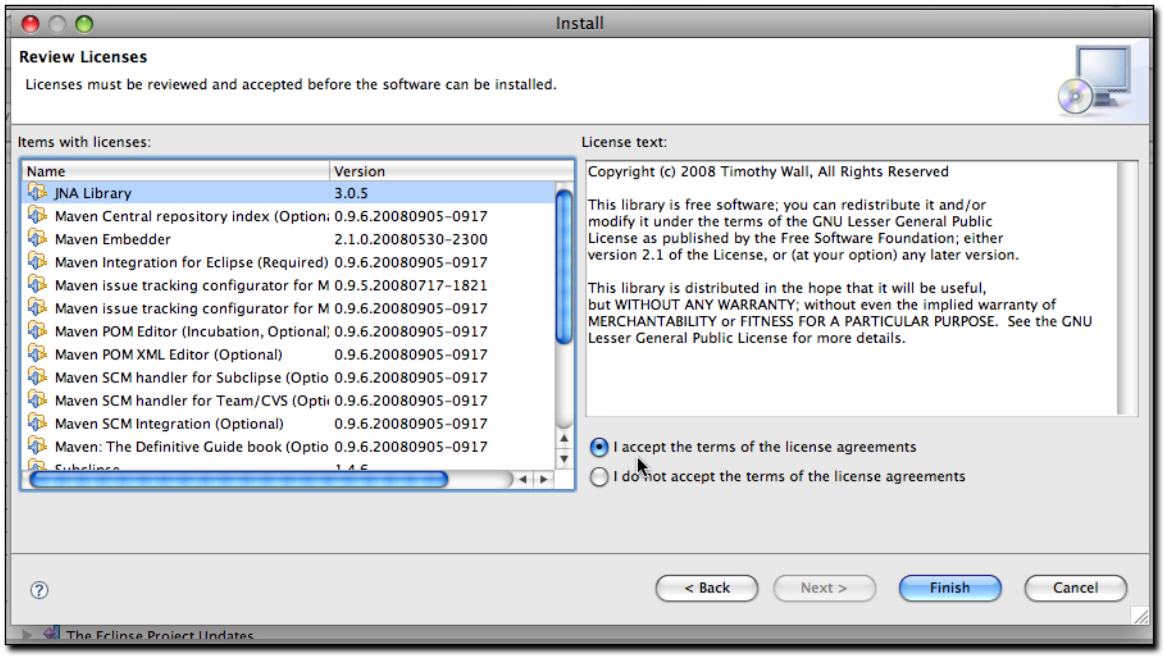

Figure 5.18. Accepting the License Agreements for Newly Installed Components

E. Click the Finish button and wait for Eclipse to download and install all of the selected Eclipse plugins and components. This process may take minutes to complete as Eclipse downloads components from the remote update sites. While Eclipse is installing m2eclipse and its prerequisites, you can watch the status of the installation in the following Install dialog.

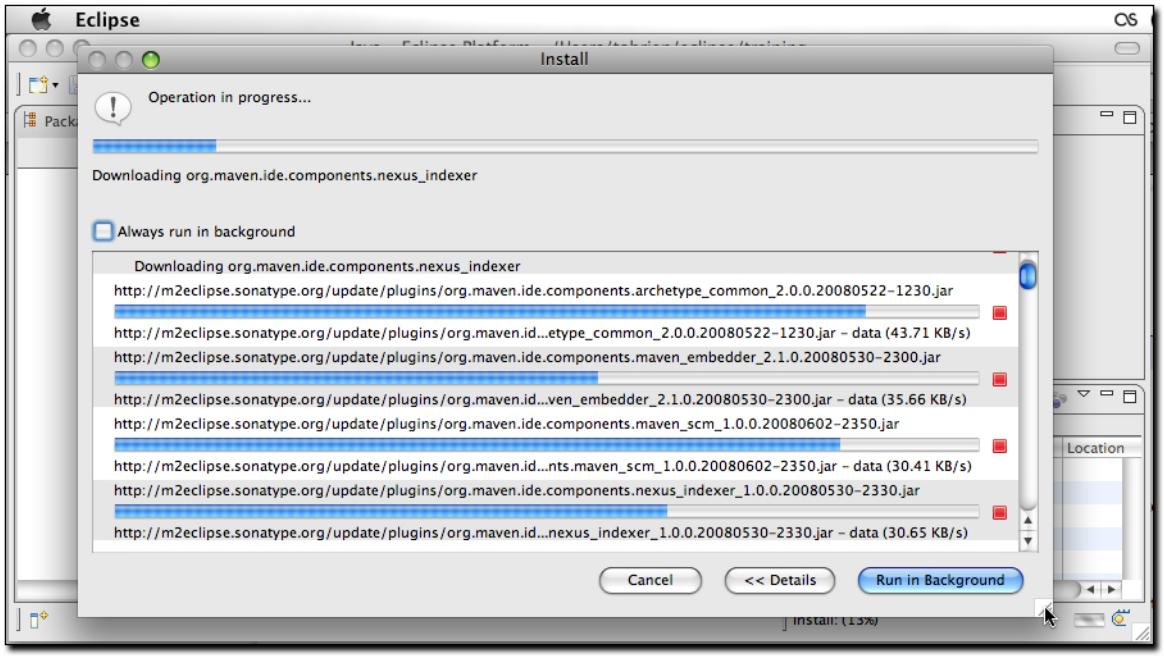

Figure 5.19. Installation Dialog Showing Status of Installation

F. Once the plugin has been installed, Eclipse will recommend that you restart the Eclipse IDE. Click Yes to restart.

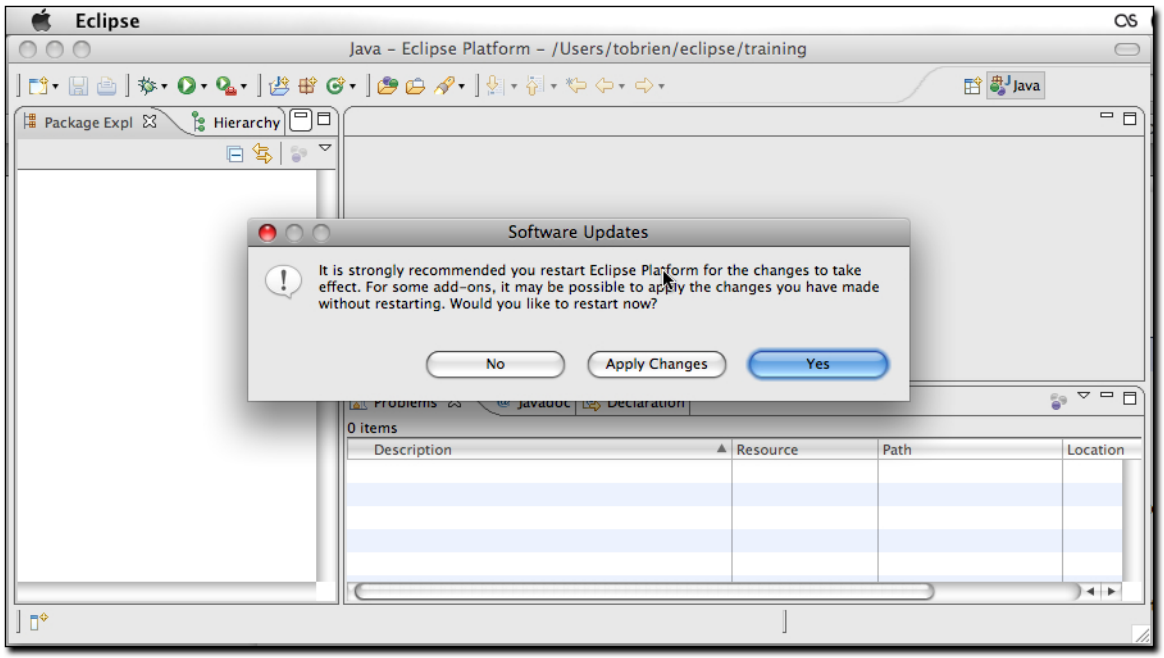

Figure 5.20. Installation Completion Dialog

5.3.8. Step 5: Verify the Installation of m2eclipse

Once the installation if complete, and Eclipse restarts, you should see the follow welcome page for Eclipse.

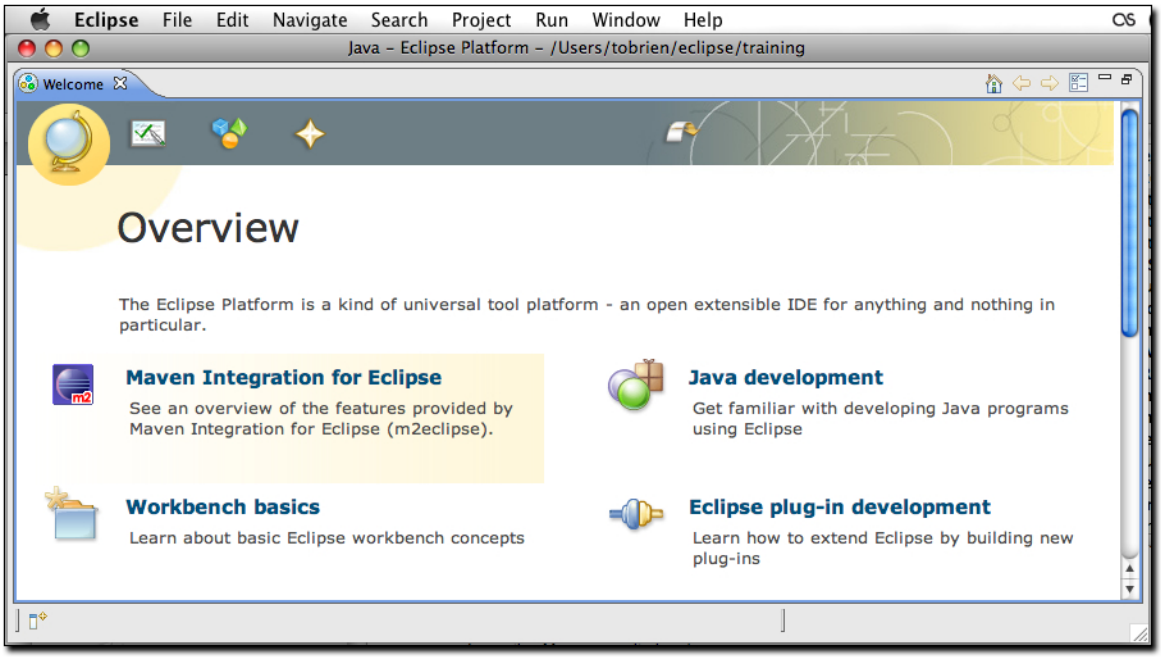

Figure 5.21. Welcome Page after Successful m2eclipse Installation

If you see Maven Integration for Eclipse, you have successfully installed m2eclipse on your installation of Eclipse.

5.3.9. Step 6: Point m2eclipse at an External Maven Installation

m2eclipse ships with an embedded instance of Maven called the Maven Embedder. Often organizations need to point to an external instance of Maven for consistency with command-line builds and automated builds. In this exercise, we're going to configure m2eclipse to point to the instance Maven we configured in a previous lab.

To see the preferences for m2eclipse from the Eclipse workbench:

A. Select Preferences... from the Eclipse menu.

B. Once you can see the preferences dialog, click on Maven in the list of components on the left-hand side of this dialog. You should see the following.

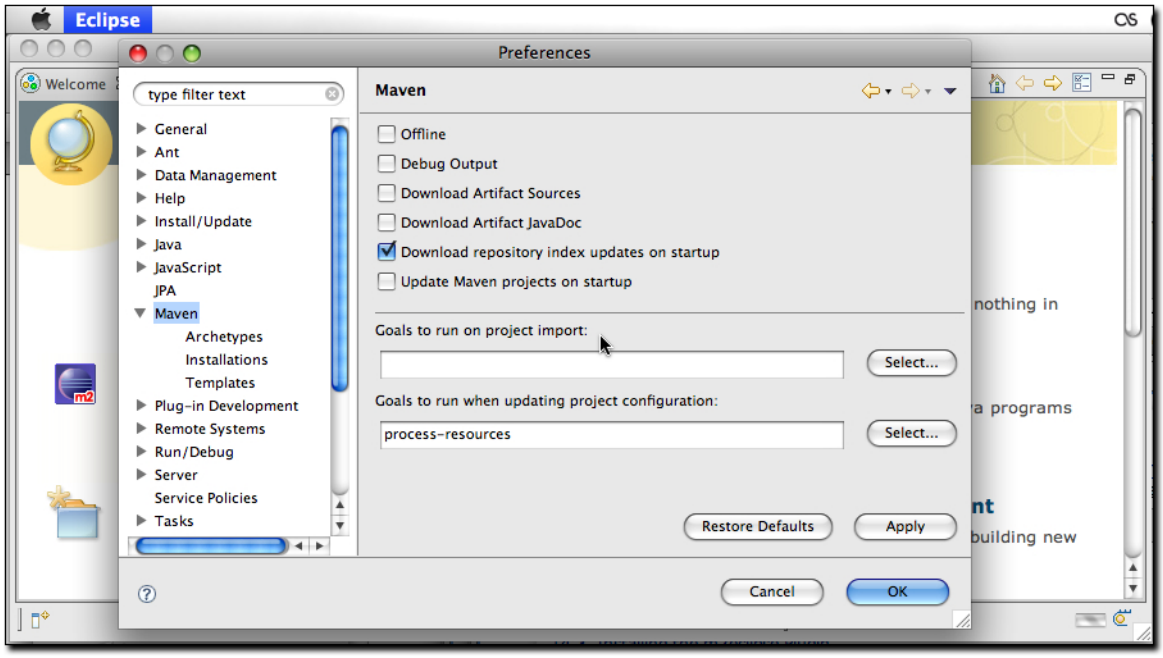

Figure 5.22. Maven Preferences in the Eclipse

On this preferences page you can control some of the global settings for m2eclipse. You can tell m2eclipse to only run Maven in offline mode or to download sources and Javadocs for dependencies. You can also control when m2eclipse downloads indexes from remote repositories.

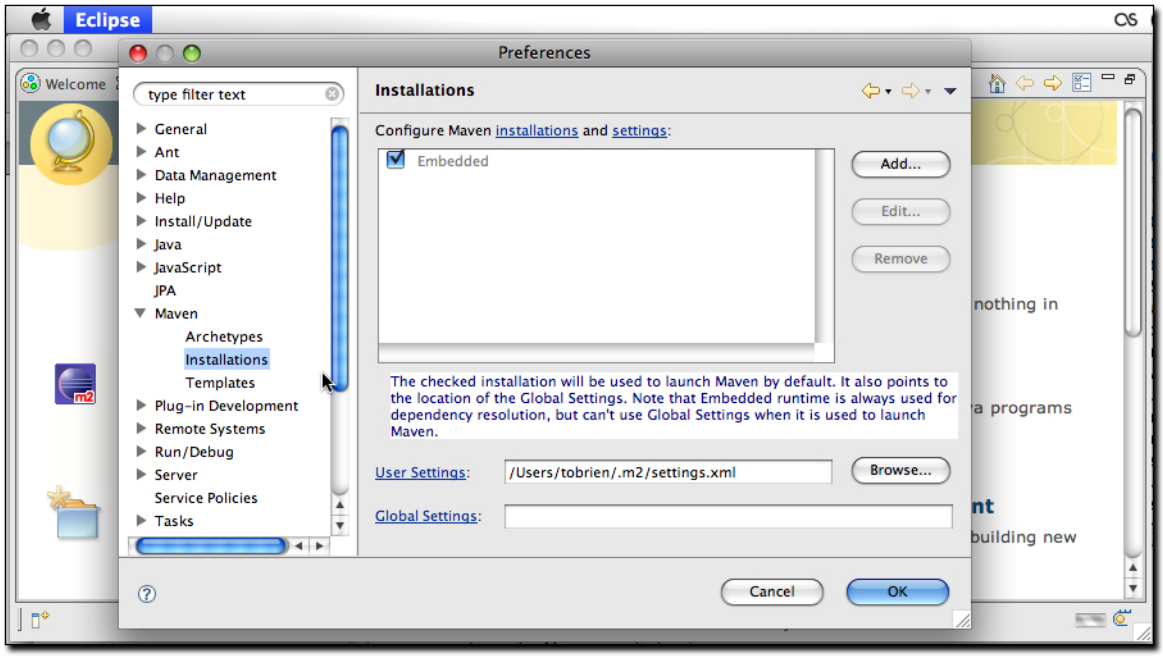

Figure 5.23. Maven Installations Preferences in m2eclipse

C. Click Add... to add a new instance of Maven to this list. Select the directory in which you installed Apache Maven and click OK. Make sure that the external instance is selected by clicking on the check-box.

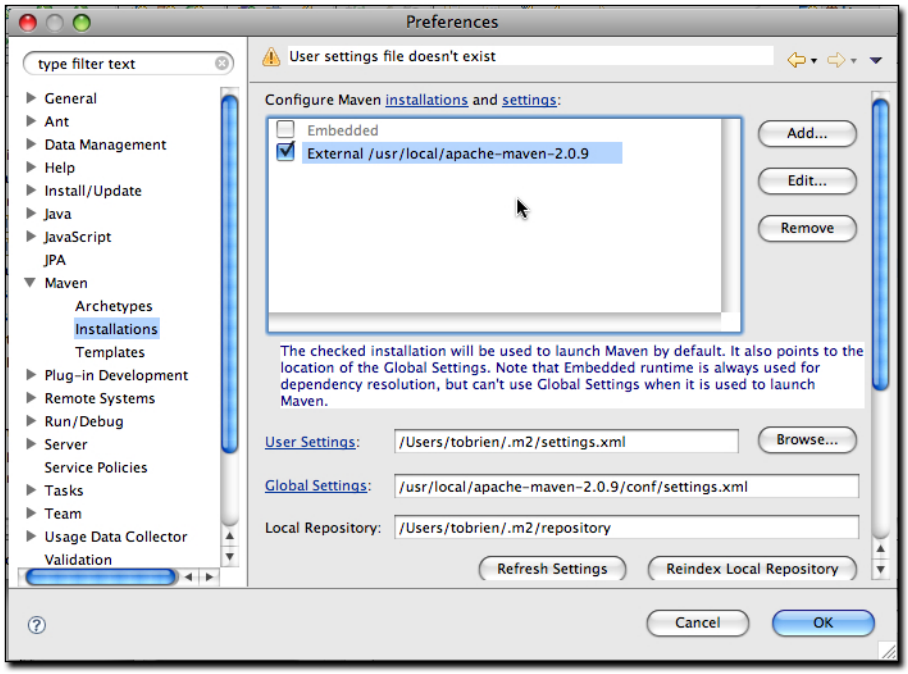

Figure 5.24. Select the External Instance of Maven

5.4. Extra Credit

Once m2eclipse has been successfully installed, you can start to explore the interface. The best reference for m2eclipse is the Developing with Eclipse and Maven[1] book - go to the Sonatype website and check it out. This book serves as the documentation for the m2eclipse plugin.

[1] http://www.sonatype.com/book

Lab 6. Creating a new Maven project in Eclipse

6.1. Goal

The aim of this lab exercise is to learn how to create a new Maven project from within Eclipse, using the m2eclipse plugin

6.2. Agenda

- Create a new Maven project from within Eclipse
- Study the default directory structure as seen in Eclipse
- Compile, test and package the project

6.3. Lab Exercises

In this exercise, you are going to create another module for Babble, the simple social networking web site used throughout this workbook.

6.3.1. Step 1: Start Eclipse

The first thing to do is to start Eclipse.

6.3.2. Step 2: Create a new Maven project in Eclipse

The next module of our new killer social networking application will contain a service layer. This will expose our domain code and business logic to web applications, web services, iPhone applications and the like. Like the first module, we will create this one using the maven archetype plugin. Only this time, we will be using the m2eclipse plugin in Eclipse to do the work.

As before, we need a groupId and artifactId for our project.

- groupId: com.sonatype.training
- artifactId: babble-services

A. Select “File -> New -> Project” in Eclipse, and then choose “Maven # Maven Project” in the "Select a Wizard" screen shown below.

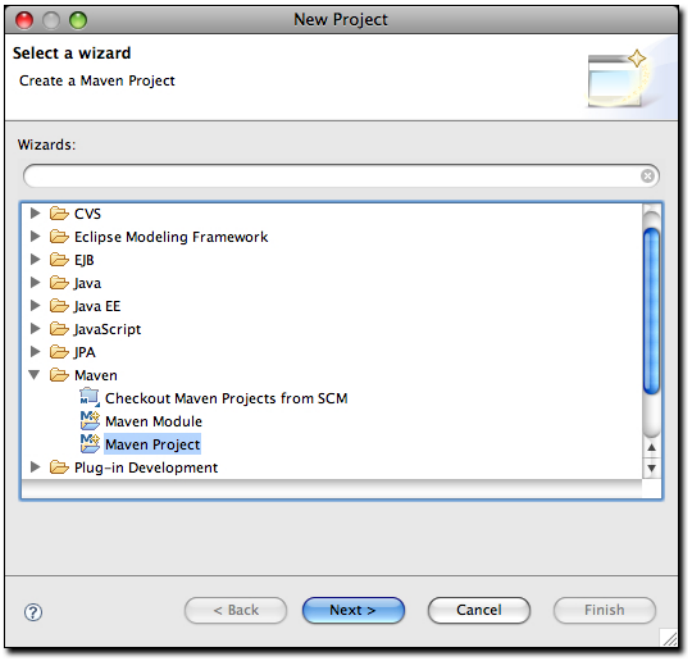

Figure 6.1. Creating a new project in m2eclipse

B. Click Next, uncheck “Use default workspace location”, and choose your `C\:\\Projects` directory as the location instead. Click "Next".

C. You will now need to choose an archetype for your project. Select the “Internal” archetype catalog and pick the “maven-archetype-quickstart” archetype.

Step 2: Create a new Maven project in Eclipse

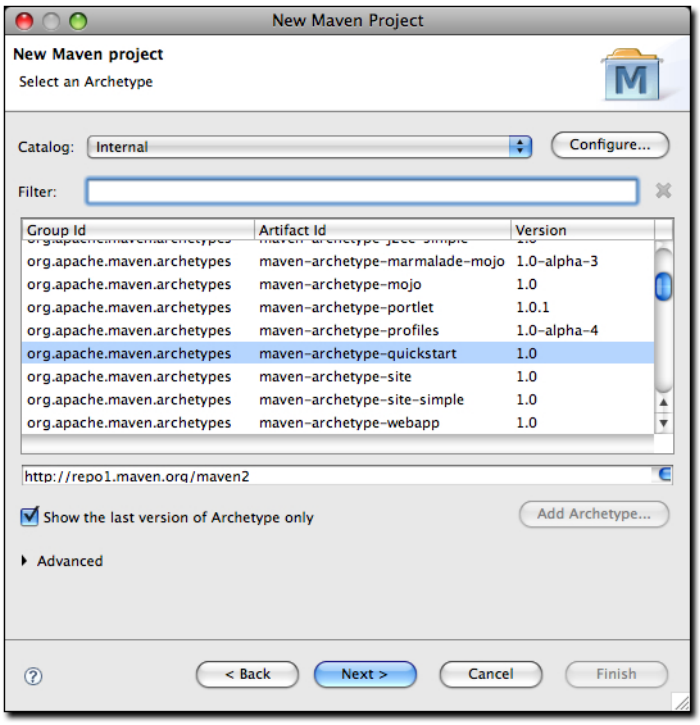

Figure 6.2. Choosing an archetype

D. As in the previous lab, you need to provide coordinates for your Maven project. Enter the values indicated below:

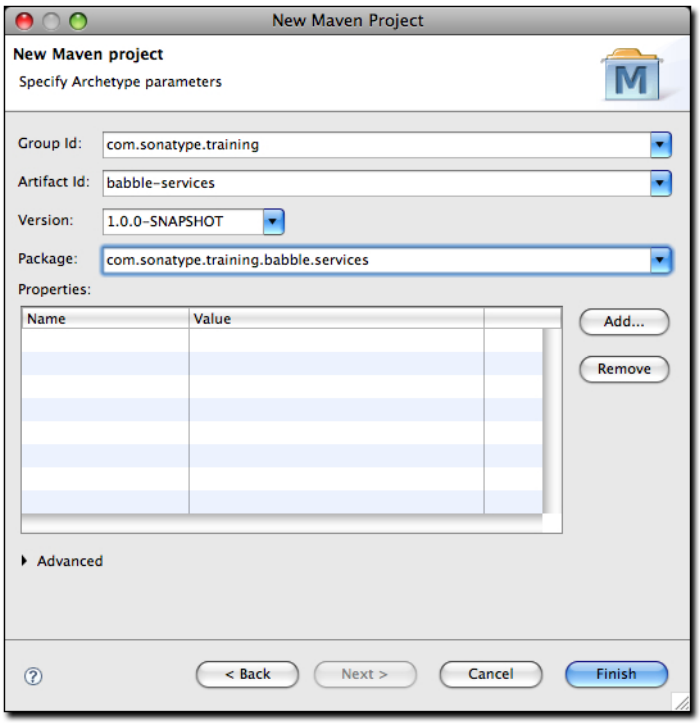

Figure 6.3. Specifying the coordinates

E. Click Finish, and Eclipse will create a new Maven project for you. In this project you can see that the Java source files are stored in src/main/java and the unit tests are stored in src/test/java. Both of these directories are configured as Source Paths in the Eclipse project. There is a new Classpath container in Eclipse called Maven Dependencies which lists a single dependency JUnit 3.8.1.

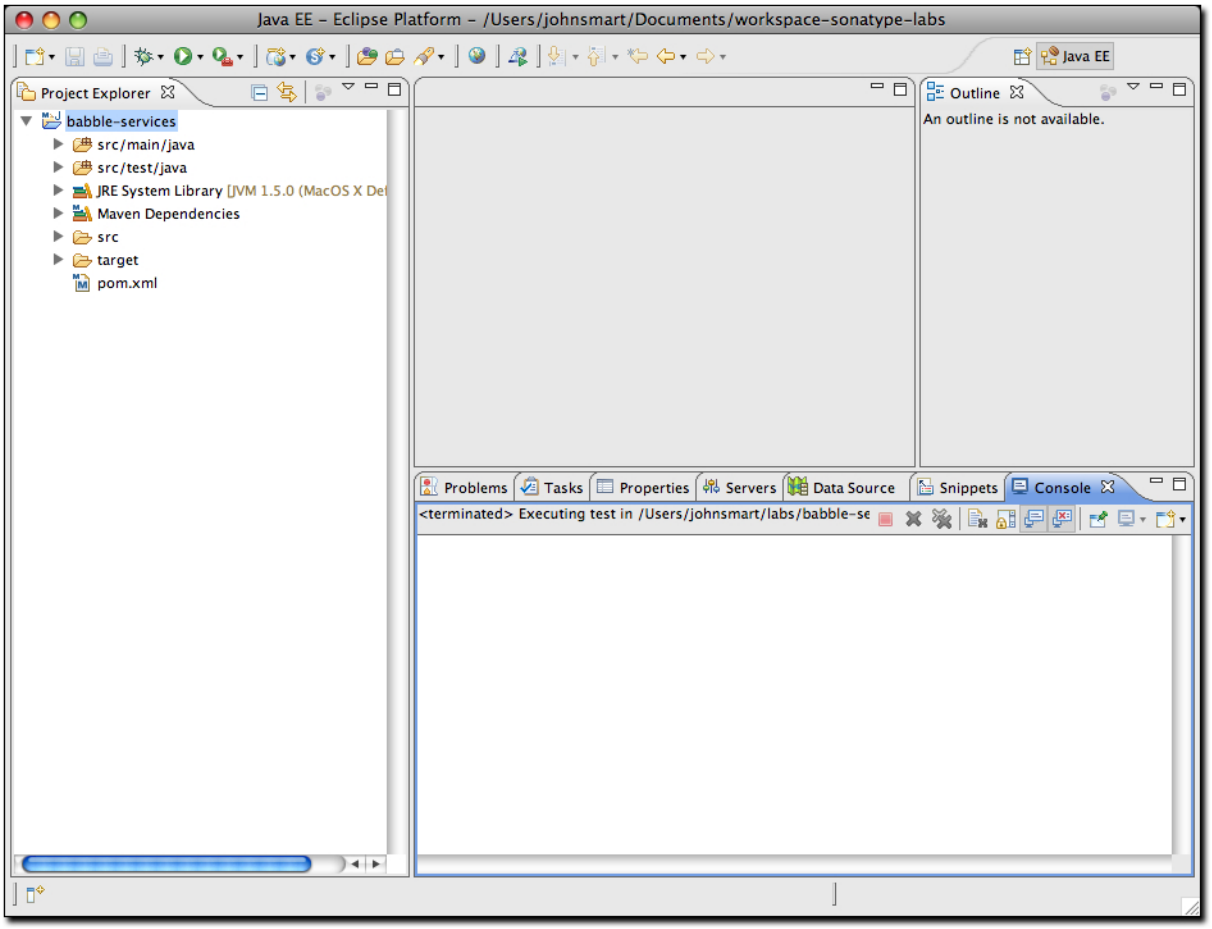

Figure 6.4. The new Maven project in Eclipse

F. To see how the Maven Archetype plugin and m2eclipse used the identifiers supplied in this Lab, double-click on the pom.xml to bring up the Graphical POM Editor. the Overview tab of the POM Editor for this project's pom.xml. In Figure 6.5, “The new project's pom file in Eclipse's graphical editor”, you can see that the Maven Archetype plugin generated a pom.xml using the identifier you supplied in this lab exercise.

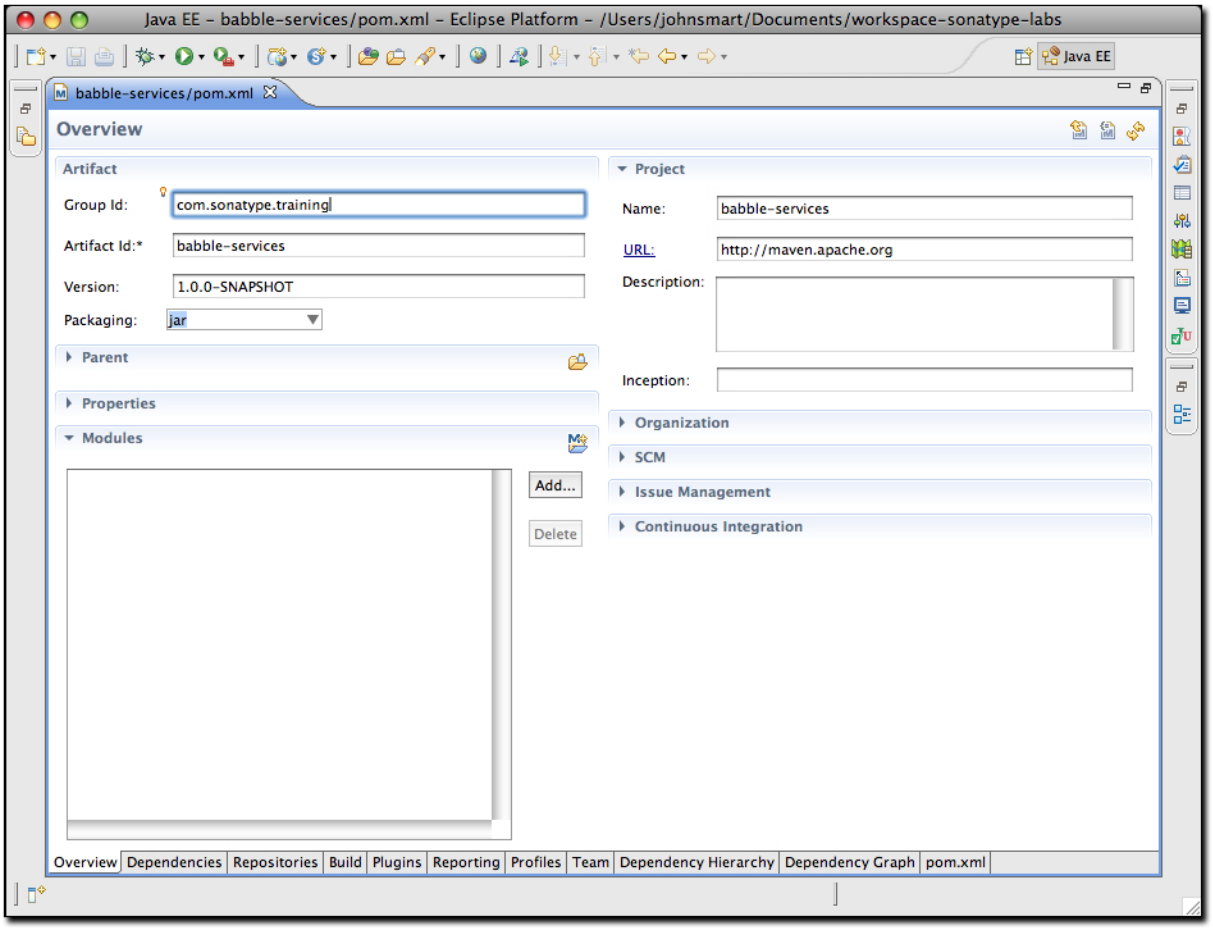

Figure 6.5. The new project's pom file in Eclipse's graphical editor

G. The pom.xml also contains a dependency on JUnit 3.8.1. To see this dependency listed in the Graphical POM Editor, click on the Dependencies tab at the bottom of the POM Editor window. Clicking on this tab will bring up a dialog displaying a list of the project's dependencies.

6.3.3. Step 3: Run Maven on the Project

You can build this project using Maven directly from within Eclipse. To build the project:

A. Right click on the project (or on the pom.xml), and select Run As... # Maven test. This will have the same effect as running mvn test from the babble-services project directory.

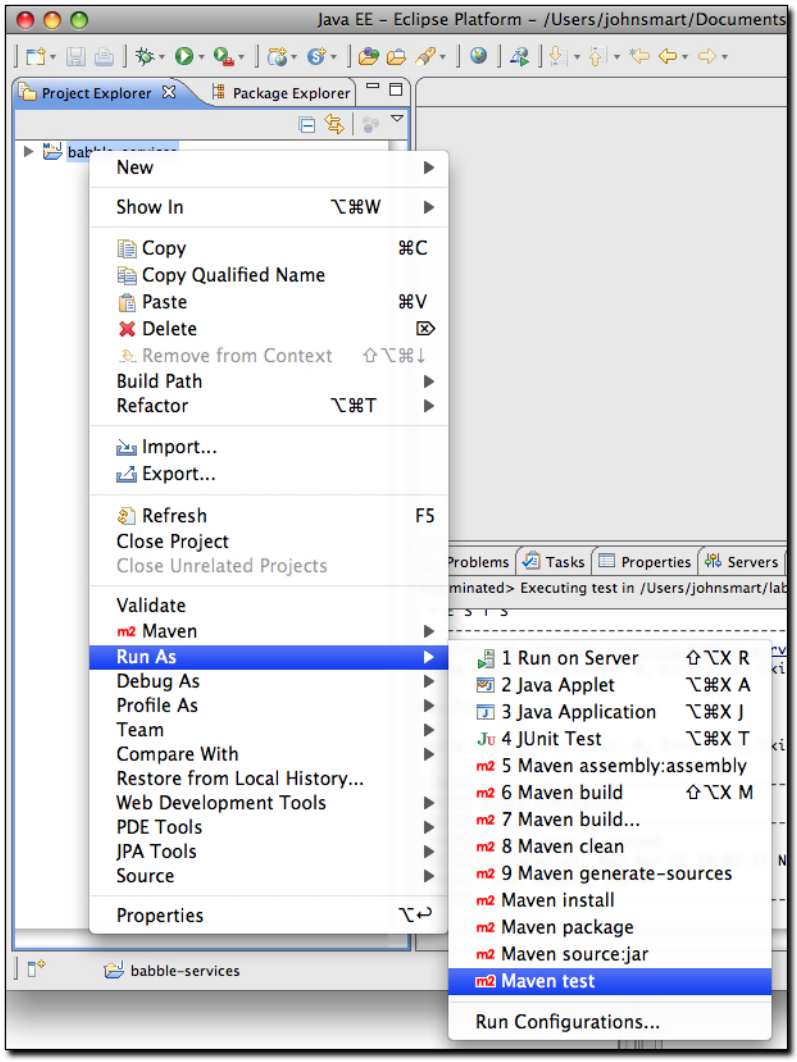

Figure 6.6. Running Maven commands from within Eclipse

B. Once the unit tests are run, maximize the Console tab in the lower right-hand of the Eclipse window by double clicking on the Console tab. In this tab, you will see the output of mvn test:

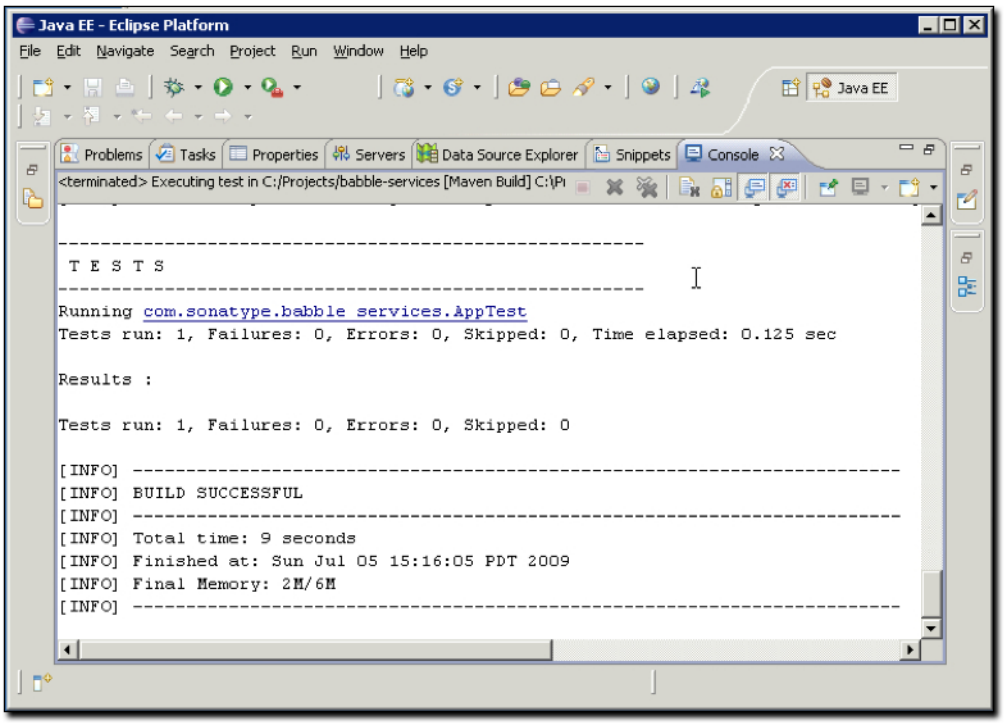

Figure 6.7. Output from Running Maven test

6.3.4. Step 4: Configure a Maven Run Configuration

In this step, you will see how to configure a Maven Run Configuration, this will allow you to pass options and flag that will modify a Maven execution:

A. Right click on the project (or on the pom.xml), and select Run As... # Run Configurations....

B. In the Run Configurations dialog, select Maven Build, and click on New Launch Configuration as shown in the following figure:

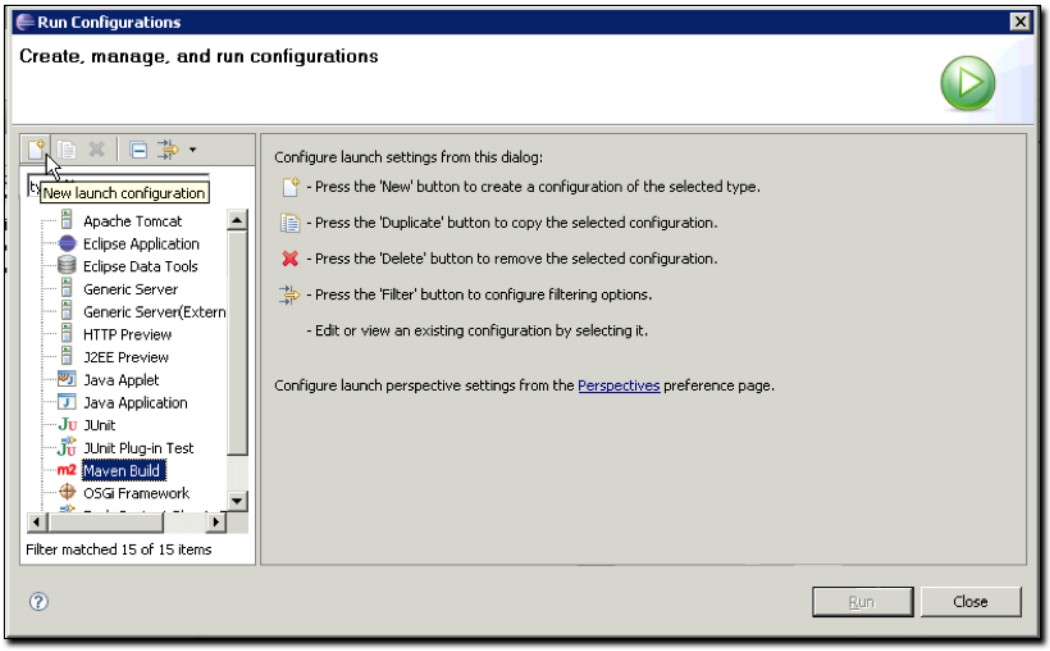

Figure 6.8. Creating a New Maven Build Run Configuration

C. In the new run configuration:

a. Change the name to "Test Configuration"

b. Click on Browse Workspace and select the babble-services project.

c. Click on the Select... button next to the Goals field and select "install" under "Build Lifecycle Phases"

d. Make sure that "Debug Output" is checked

e. Click on Apply to save the new configuration

f. Click on Run to run the Test Configuration.

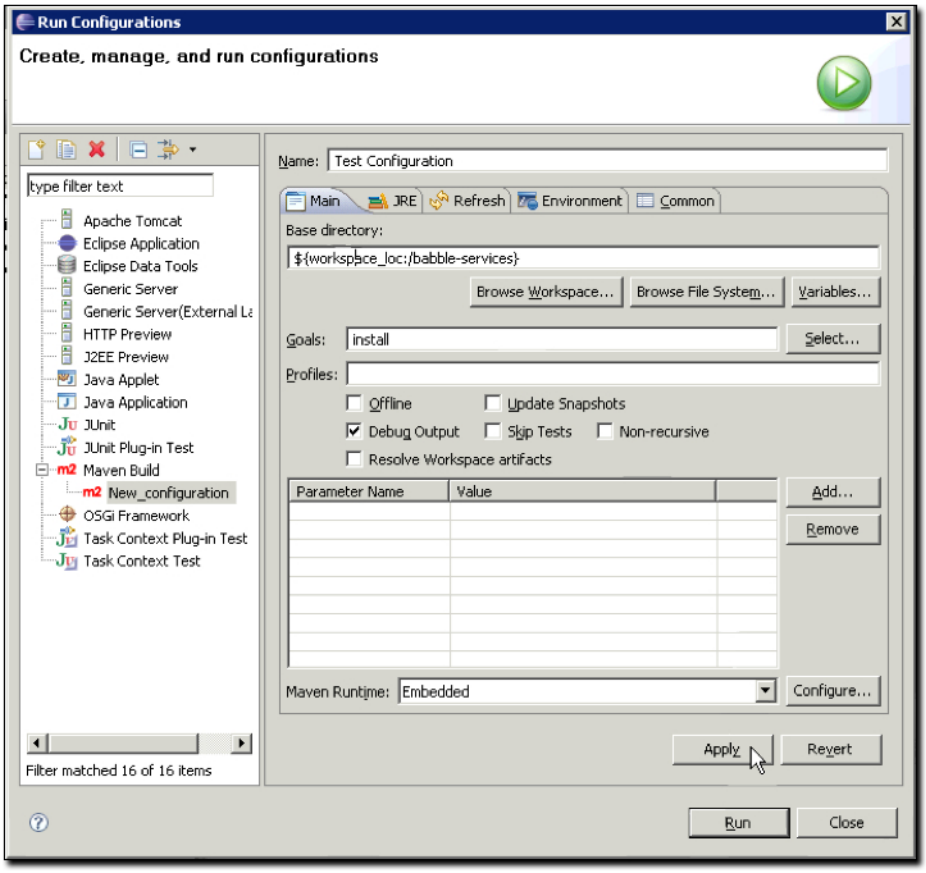

Figure 6.9. A Custom Maven Run Configuration

D. Once the unit tests are run, maximize the Console tab in the lower right-hand of the Eclipse window by double clicking on the Console tab. In this tab, you will see the output of mvn -X install with the debug output printed in the console:

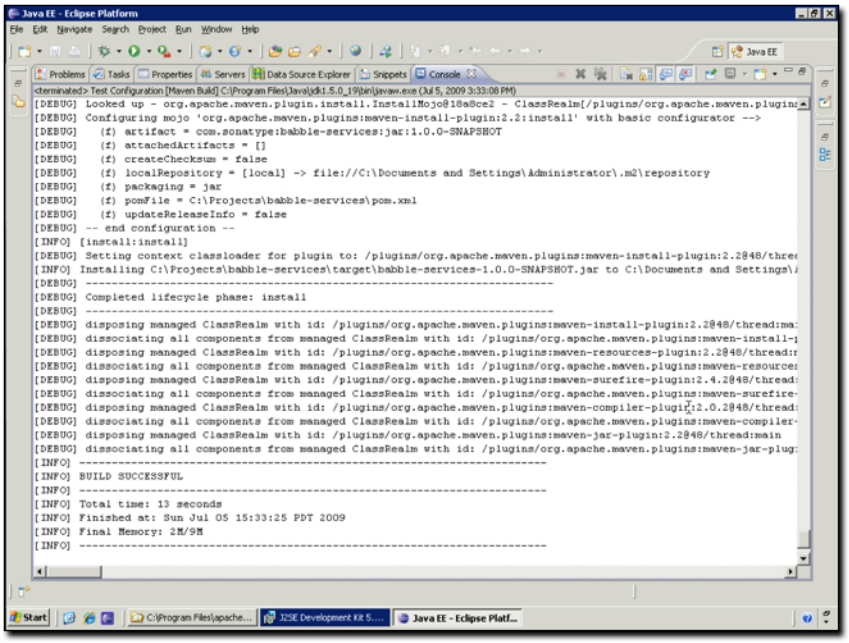

Figure 6.10. Output from Running the Test Configuration

6.3.5. Import an existing Maven project into Eclipse

You can also import existing Maven projects into your Eclipse workspace using m2eclipse. We will now try this out on the babble-core project. To do this:

A. Select “Import” and choose “General # Maven Projects”

B. Click Browse.. and choose the babble-core project directory as the root directory (see Figure 6.6, “Running Maven commands from within Eclipse”).

C. Check the babble-core pom and click on Finish. Eclipse will now import this project into your workspace.

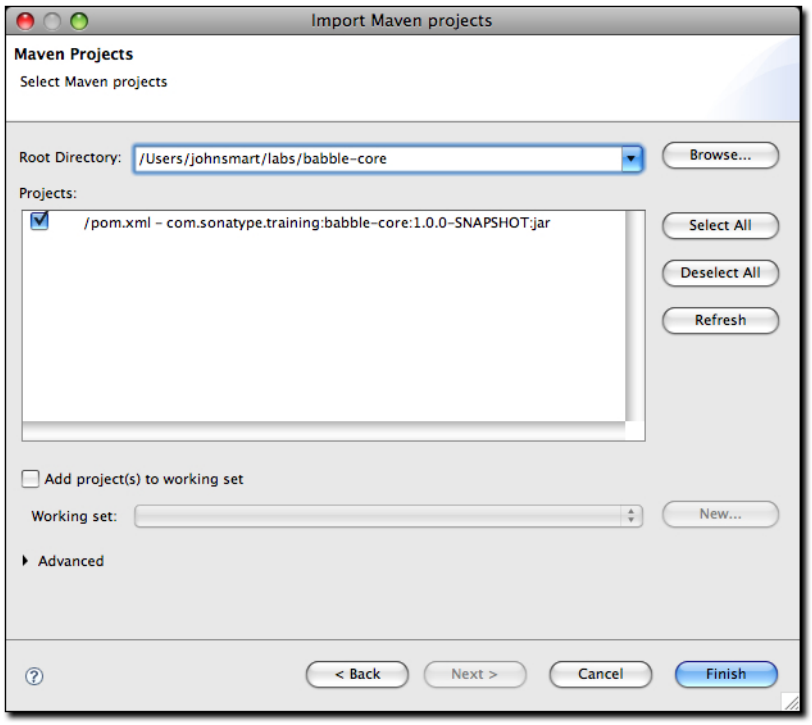

Figure 6.11. Importing an existing Maven project into Eclipse

6.4. Discussion

You have successfully created a new project using the m2eclipse New Maven project wizard. Using the techniques in this lab, you now have access to hundreds of Maven archetypes currently available in the Central Maven repository.

Lab 7. Adding Project Information

7.1. Goal

The aim of this lab is to learn how to add human-readable details to your pom.xml file.

7.2. Agenda

- Add some SCM configuration details into the POM file
- Add some issue management details
- Add some team members
- Generate the Maven web site and see how these details are displayed

7.3. Lab Exercises

In this exercise, you are going to add some details about the babble-core project into the pom.xml file. These details are primarily for human consumption, so the values we enter don't have to be real ones for the moment.

7.3.1. Step 1: Add SCM details

To add SCM details:

A. Go to the babble-core project and open up the POM file. We will be working with the raw XML format, so click on the "pom.xml" tab (see Figure 7.1, “Editing the babble-core pom file.”).

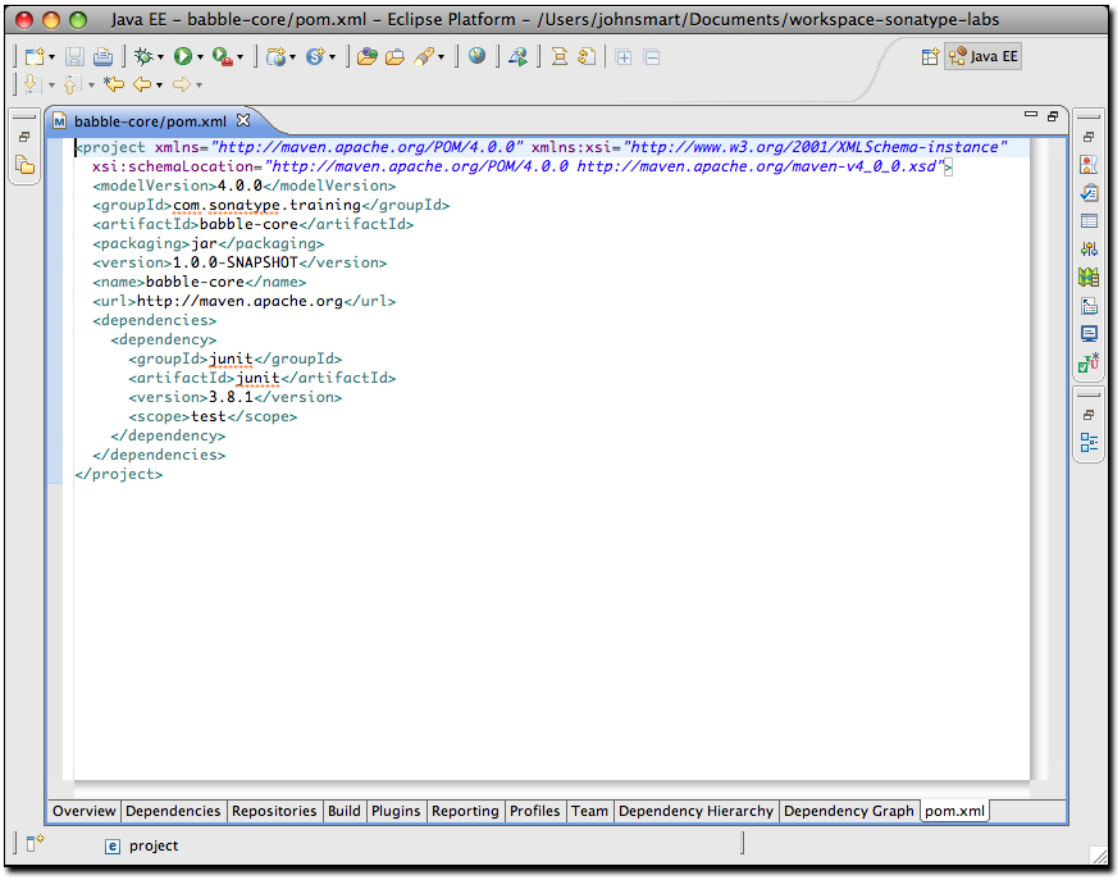

Figure 7.1. Editing the babble-core pom file.

B. Add some details about the SCM used for the project. Later on, you will use your very own git repository, but for now, we will use a Subversion one. Add the following block just before the dependencies:

```
<scm>
 <connection>
  scm:svn:https://wakaleo.devguard.com/svn/sonatype-maven-training/babble-core/trunk
 </connection>
 <developerConnection>
  scm:svn:https://wakaleo.devguard.com/svn/sonatype-maven-training/babble-core/trunk
 </developerConnection>
 <url>https://wakaleo.devguard.com/trac/sonatype-maven-training/</url>
</scm>
```

C. Make sure there are no syntax errors (red bars in the margin), and save your changes.

7.3.2. Step 2: Adding issue tracking

To add issue tracking details:

A. Go to the babble-core project and open up the POM file. We will be working with the raw XML format, so click on the "pom.xml" tab (see Figure 7.1, “Editing the babble-core pom file.”).

B. Add issue management details and an inception year. For this lab, you can use the following values just after the SCM details:

```
<issueManagement>
  <system>trac</system>
  <url>https://wakaleo.devguard.com/trac/sonatype-maven-training</url>
</issueManagement>
<inceptionYear>2009</inceptionYear>
```

C. Make sure there are no syntax errors (red bars in the margin), and save your changes.

7.3.3. Step 3: Add some team members

To add team members to your project's POM:

A. Go to the babble-core project and open up the POM file. We will be working with the raw XML format, so click on the "pom.xml" tab (see Figure 7.1, “Editing the babble-core pom file.”).

B. Now, add some details about the project team. You can add as many team members as you want. The format looks like this:

```
<developers>
  <developer>
    <id>john</id>
    <name>John Smart</name>
    <email>john.smart@wakaleo.com</email>
    <roles>
      <role>Developer</role>
    </roles>
    <organization>Wakaleo Consulting</organization>
    <timezone>+12</timezone>
  </developer>
  ...
</developers>
```

C. Make sure there are no syntax errors (red bars in the margin), and save your changes.

7.3.4. Step 4: Build the Maven site

To build a site for your project:

A. Open a command line window in the babble-core project directory.

B. Run “mvn site”. This may take some time the first time you run it, as it requires a lot of dependencies that we haven't used before.

```
C:\Projects\babble-core> mvn site
...
[INFO] Generating "Continuous Integration" report.
[INFO] Generating "Dependencies" report.
[INFO] Generating "Issue Tracking" report.
[INFO] Generating "Project License" report.
[INFO] Generating "Mailing Lists" report.
[INFO] Generating "About" report.
[INFO] Generating "Project Summary" report.
[INFO] Generating "Source Repository" report.
[INFO] Generating "Project Team" report.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4 minutes 27 seconds
[INFO] Finished at: Tue Jun 10 23:19:28 NZST 2008
[INFO] Final Memory: 14M/25M
[INFO] ------------------------------------------------------------------------
```

C. Open the target/side/index.html page and view the results in your web browser. Look at the "Soure Repository", "Issue Management" and "Project Team" sections to see how the information you entered has been rendered.

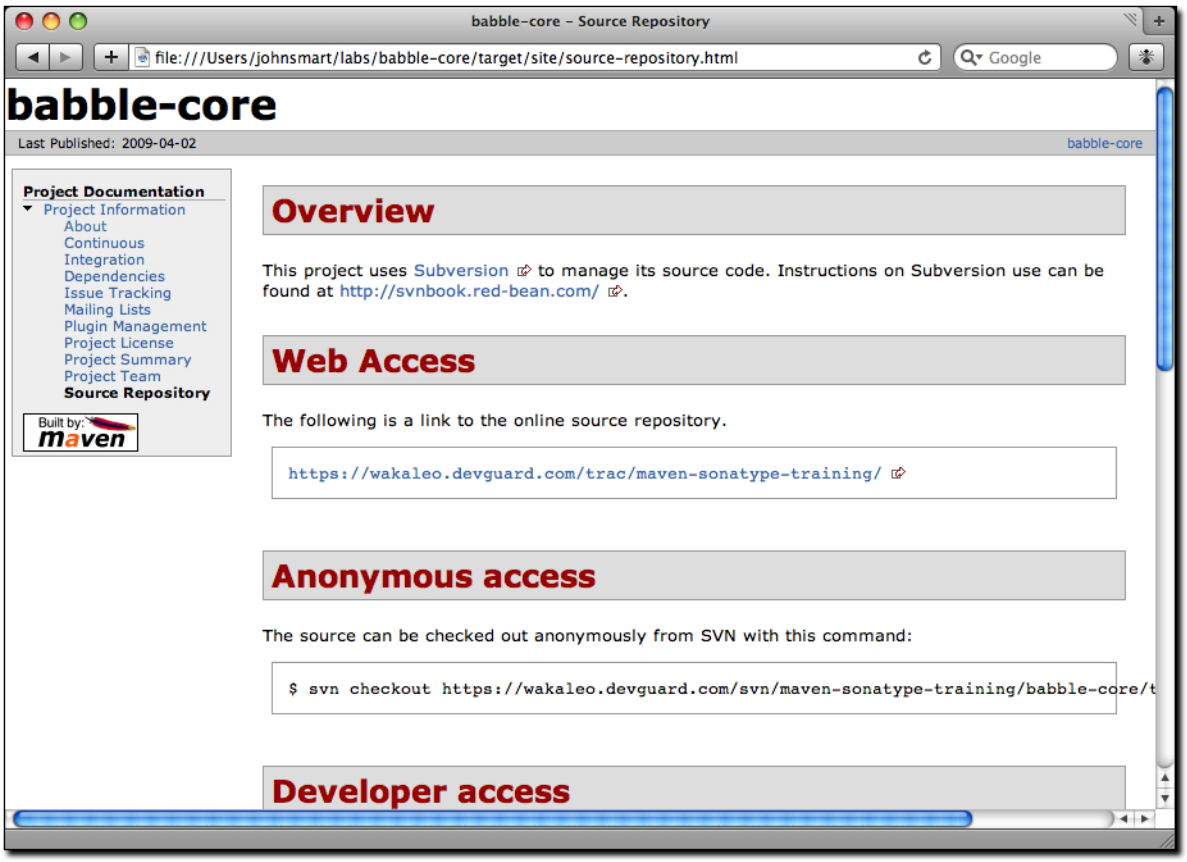

Figure 7.2. The generated Maven web site

7.4. Discussion

You should now have an understanding of how Maven uses the information in the POM file to generate a web site containing useful technical information about the project.

Lab 8. Working with dependencies

8.1. Goal

The aim of this lab is to learn how add dependencies to your project, and how to work with Maven dependencies from with Eclipse.

8.2. Agenda

- Add some dependencies to our project by hand
- Update an existing dependency
- Add a dependency using the m2eclipse plugin
- See how dependencies are viewed in Eclipse

8.3. Lab Exercises

In this exercise, you are going to add some dependencies to the babble-core project, and then see how we can manage them in Eclipse. We will start off with a broken project - one with source code that is missing some dependencies and proper configuration in the pom file. Import the starting point for this lab into Eclipse from the labs directory (from the labs/lab-08/start/babble/babble-core directory). You will first need to delete your existing babble-core project from your Eclipse workspace.

This will add some source code to your babble-core project, containing some simple domain classes and unit tests.

8.3.1. Step 1: Add new dependencies

In this step we will add new dependencies to the babble-core project. There is already a dependency to JUnit 3.8.1. We will add dependencies on spring-core, spring-beans, joda-time, and log4j. To add new dependencies:

A. Go to the babble-core project and open up the POM file. For this exercise, we will be working with the raw XML format, so click on the "pom.xml" tab (see Figure 7.1, “Editing the babble-core pom file.”).

B. Now add the dependencies, as shown here:

```
<dependencies>
```

```
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>2.5.6</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>2.5.6</version>
    </dependency>
    <dependency>
      <groupId>joda-time</groupId>
      <artifactId>joda-time</artifactId>
      <version>1.6</version>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.12</version>
    </dependency>
  </dependencies>
```

C. Make sure there are no syntax errors (red bars in the margin), and save your changes.

D. Run the Maven install phase on the project, by right-clicking on the babble-core project, selecting Run As... -> Maven Install

E. Examine the console output to see if Maven successfully downloaded the newly added dependencies.

8.3.2. Step 2: Update a Dependency

This code relies on annotations, JUnit 4.7 and the Hamcrest library, so initially it won't compile. You will first need to modify your babble-core pom.xml file to add support for Java 5 features. To do this:

A. Go to the babble-core project and open up the POM file. For this exercise, we will be working with the raw XML format, so click on the "pom.xml" tab (see Figure 7.1, “Editing the babble-core pom file.”).

B. Add the following <build> section to your pom.xml file, just after the <dependencies> section:

```
    <build>
       <plugins>
          <plugin>
             <groupId>org.apache.maven.plugins</groupId>
```

```
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
              <source>1.5</source>
              <target>1.5</target>
            </configuration>
          </plugin>
        </plugins>
      </build>
```

C. Go to the JUnit dependency and update the version number to 4.7, using the auto-completion as shown in Figure 8.1, “Using code completion to update a dependency”. The default keystroke to activate autocompletion is CTRL-SPACE.

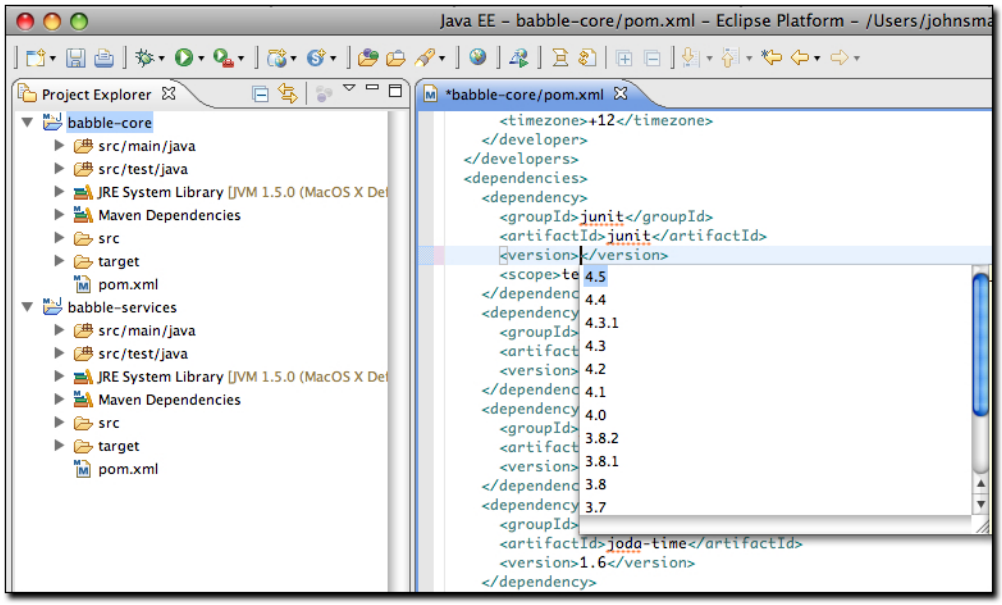

Figure 8.1. Using code completion to update a dependency

D. Once you have updated the version, save your changes.

8.3.3. Step 3: Add a new dependency

Previously, we added dependencies directly to the pom file by hand. We will now see how to use the m2eclipse plugin to add a new dependency provides powerful auto-completion features within the pom editor. We will be adding a dependency on the Hamcrest libraries to our project.

A. First, use the contextual menu and choose "Maven -> Add Dependency", as shown in Figure 8.2, “Using code completion to update a dependency”.

Step 3: Add a new dependency

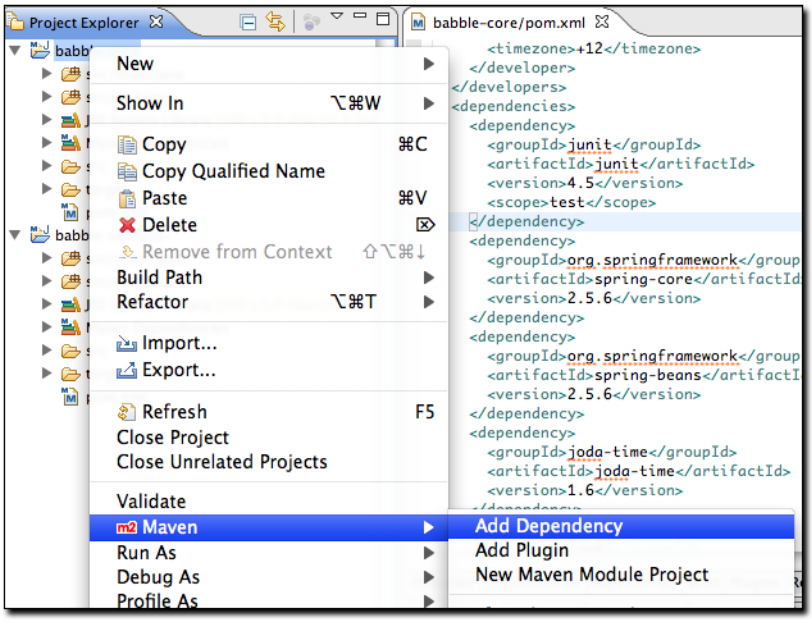

Figure 8.2. Using code completion to update a dependency

B. This will open a window in which you can search for the dependency you need, and add it directly to the pom. Enter "hamcrest", and select the "hamcrest-all-1.1.jar" (see Figure 8.3, “Selecting a new dependency to add”).

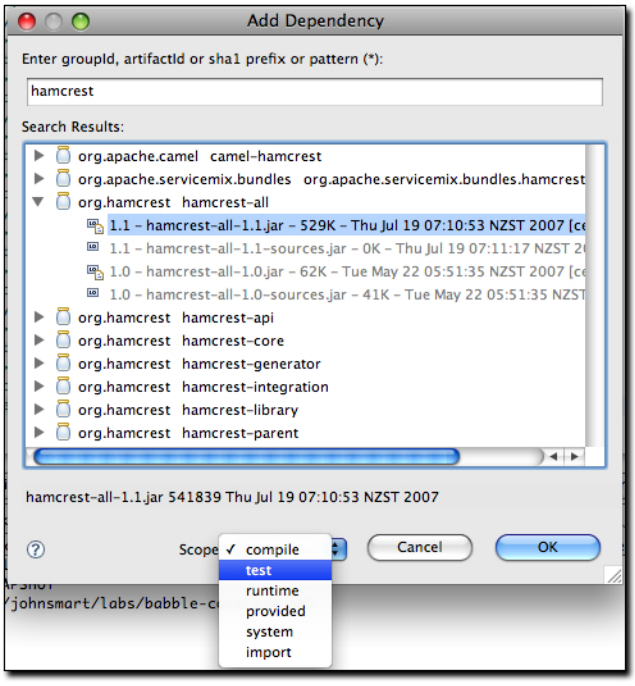

Figure 8.3. Selecting a new dependency to add

C. Click on OK. Eclipse will then add this dependency to your pom file.

D. Check that the new dependency has been added into your pom.xml file.

E. Now, save your changes, and make sure there are no remaining syntax or compiler errors in the project.

8.3.4. Step 4: View the dependency hierarchy and graph

We are now going to see how m2eclipse lets you visualize dependencies.

A. Go to the Dependency Hierarchy and see how the dependencies are represented.

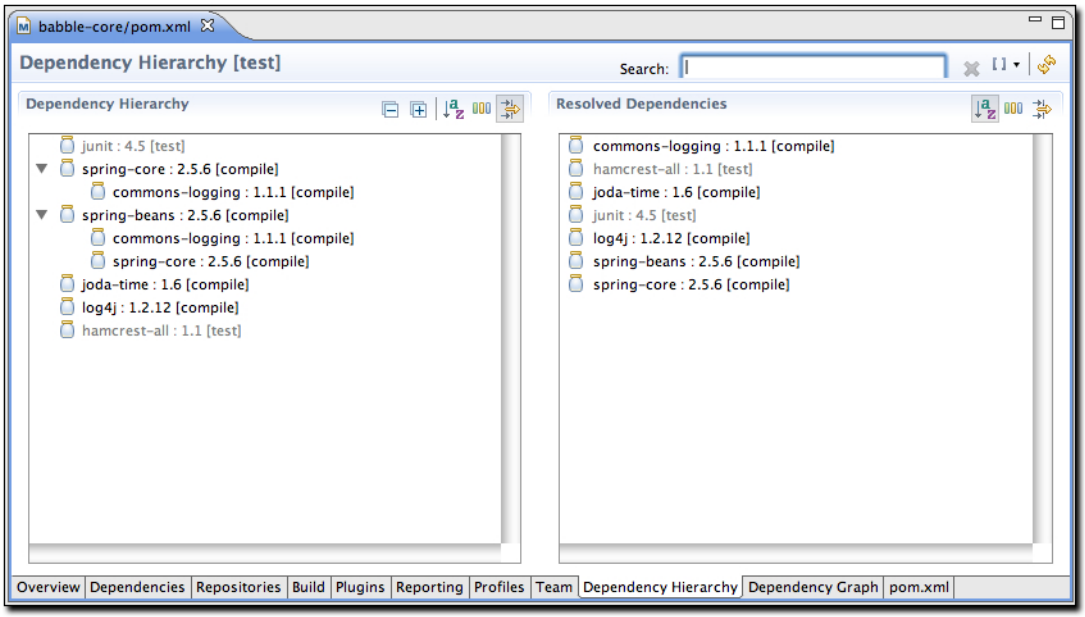

Figure 8.4. The dependency tree

B. Now switch to the dependency graph tab. Look at how the dependencies are represented here.

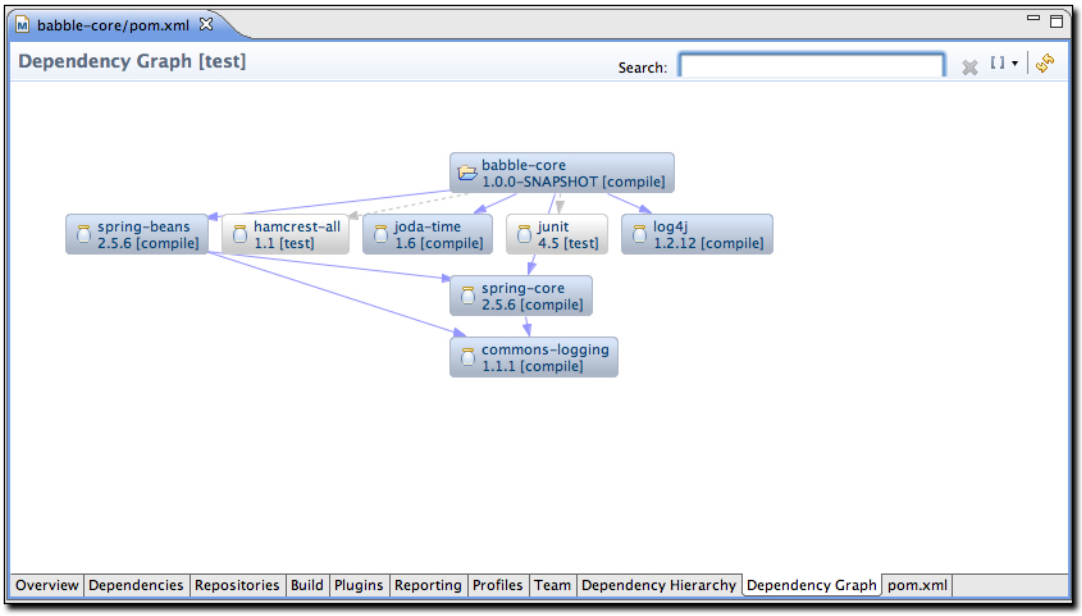

Figure 8.5. The dependency graph

8.3.5. Step 5: Handling dependency conflicts

We are now going to introduce an artifical dependency conflict. Suppose our application uses the Commons Logging library.

A. Go to the babble-core project and open up the POM file. For this exercise, we will be working with the raw XML format, so click on the "pom.xml" tab (see Figure 7.1, “Editing the babble-core pom file.”).

B. Add a new dependency to commons-logging version 1.1, as shown here:

```
<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging</artifactId>
  <version>1.1</version>
</dependency>
```

C. Now open the Dependency Graph tab. Observe how the red lines indicate a dependency conflict. The spring-core and spring-beans libraries need commons-logging 1.1.1, but our declared dependency on version 1.1 overrides this.

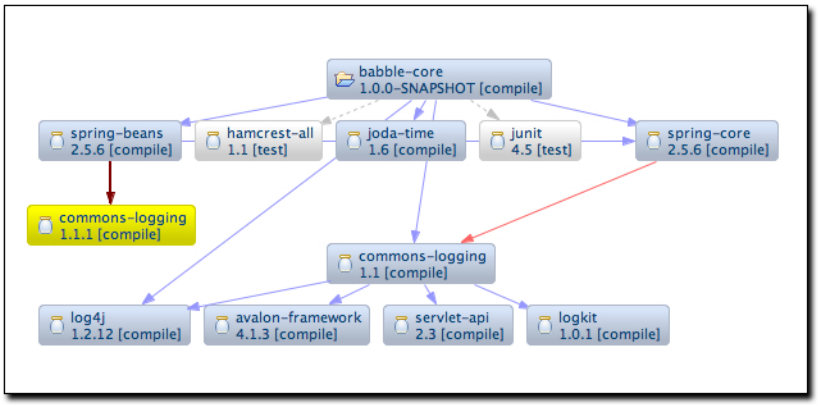

Figure 8.6. Displaying a dependency conflict

D. Open the Dependency Hierarchy tab to see how version 1.1 is listed in the resolved dependencies.

E. To resolve this conflict, change your dependency to commons-logging version 1.1.1. How does this affect the dependency hiercarchy and graph views?

8.3.6. Step 6: Add an internal dependency

We are now going to add some code to the other babble module, babble-services.

A. Delete your existing babble-services project from your Eclipse workspace and import the starting point project from the labs/lab-09/start/babble/babble-services directory.

B. Go to the babble-services project and open up the POM file. For this exercise, we will be working with the raw XML format, so click on the "pom.xml" tab (see Figure 7.1, “Editing the babble-core pom file.”).

C. Add Java 5 support to your babble-services pom file:

```
<build>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-compiler-plugin</artifactId>
         <configuration>
           <source>1.5</source>
           <target>1.5</target>
         </configuration>
       </plugin>
   </plugins>
</build>
```

D. Next, you need to add a dependency to the babble-core. First, you need to ensure that the babble-core artifact is available to the other projects on your workstation.

i. To do this, you need to run mvn install so that babble-core is available in your local repository.

ii. Go to the babble-core directory and run mvn install, as shown here:

```
$ mvn install
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building babble-core
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered resources, i.e
[INFO] Copying 0 resource
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered resources, i.e
[INFO] skip non existing resourceDirectory /Users/johnsmart/labs/babble/babble-core/s
[INFO] [compiler:testCompile]
[INFO] Compiling 2 source files to /Users/johnsmart/labs/babble/babble-core/target/te
[INFO] [surefire:test]
[INFO] Surefire report directory: /Users/johnsmart/labs/babble/babble-core/target/sur

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.06 sec
Running com.sonatype.training.babble.core.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.006 sec

Results :

Tests run: 10, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar]
[INFO] Building jar: /Users/johnsmart/labs/babble/babble-core/target/babble-core-1.0.
[INFO] [install:install]
[INFO] Installing /Users/johnsmart/labs/babble/babble-core/target/babble-core-1.0.0-S
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 seconds
[INFO] Finished at: Fri Apr 03 11:39:11 NZDT 2009
[INFO] Final Memory: 10M/21M
[INFO] ------------------------------------------------------------------------
```

E. Go to the babble-services project and open up the POM file. For this exercise, we will be working with the raw XML format, so click on the "pom.xml" tab (see Figure 7.1, “Editing the babble-core pom file.”).

F. Now, add the following dependencies to your babble-services project:

```
      <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>4.7</version>
          <scope>test</scope>
      </dependency>
```

```
        <dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-all</artifactId>
            <version>1.1</version>
        </dependency>
        <dependency>
            <groupId>com.sonatype.training</groupId>
            <artifactId>babble-core</artifactId>
            <version>1.0.0-SNAPSHOT</version>
        </dependency>
```

G. Now compile and test the babble-services module to ensure that it works correctly. Go to the babble-services directory and run mvn verify:

```
$  mvn verify
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building babble-services
[INFO]    task-segment: [verify]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered resources, i.e
[INFO] skip non existing resourceDirectory /Users/johnsmart/labs/babble/babble-servic
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered resources, i.e
[INFO] skip non existing resourceDirectory /Users/johnsmart/labs/babble/babble-servic
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory: /Users/johnsmart/labs/babble/babble-services/target

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.services.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.023 sec
Running com.sonatype.training.babble.services.BabbleManagerTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.015 sec

Results :

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar]
[INFO] Building jar: /Users/johnsmart/labs/babble/babble-services/target/babble-servi
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Fri Apr 03 11:53:31 NZDT 2009
[INFO] Final Memory: 9M/17M
[INFO] ------------------------
```

8.4. Discussion

You have now seen how you manage both internal and external dependencies in Maven and in Eclipse.

Lab 9. Multi-module projects

9.1. Goal

The aim of this lab exercise is to create a multi-module project, and understand how inheritence and aggregation work in Maven projects.

9.2. Agenda

- Create a new aggregator/parent project to contain the babble-core and babble-services modules
- Make these two projects inherit from our new aggregator/parent project
- Compile, test and package the modules from the aggregator/parent project

9.3. Lab Exercises

In this exercise, you are going to turn the babble-core and babble-services applications into module in a multi-module project. We need to restructure the directory structure to reflect this new architecture (see Figure 9.1, “The Babble project as a multi-module project”).

9.3.1. Step 1: Adjust your project directory structure

First of all, you need to restructure your project directories to match the new architecture. We need a directory structure like the one in Figure 9.1, “The Babble project as a multi-module project”.

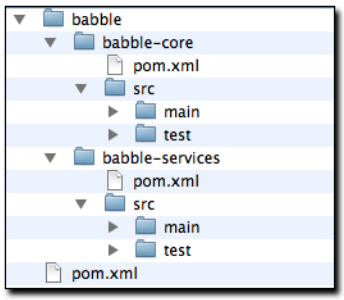

Figure 9.1. The Babble project as a multi-module project

This directory structure has been prepared for you in the labs/lab-09/start directory. Delete your existing babble-services and babble-core projects from your Eclipse workspace and import the starting point project from the labs/lab-09/start/babble directory.

9.3.2. Step 2: Creating a new aggregator pom file

Now create a pom.xml file in the babble directory.

A. Open Notepad (or another Text editor)

B. Enter the following text into your text editor:

```
<?xml version="1.0" encoding="UTF-8"?>
<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/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.sonatype.training</groupId>
    <artifactId>babble</artifactId>
    <packaging>pom</packaging>
    <version>1.0.0-SNAPSHOT</version>
    <name>babble</name>
    <description>Babble - the art of meaningless utterences</description>
    <modules>
        <module>babble-core</module>
        <module>babble-services</module>
    </modules>
</project>
```

C. Save this file as `babble\pom.xml` in your labs/lab-09/start directory. This file is your new aggregator POM which will create a multi-module project including both babble-core and babble-services.

D. From the babble directory, run mvn test from the command line.

Now, when you compile and test the project from the root directory, Maven will compile and test each of the modules:

```
C:...\babble> mvn test
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   babble-core
[INFO]   babble-services
[INFO]   babble
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] babble-core ....................................... SUCCESS [1.662s]
[INFO] babble-services ................................... SUCCESS [0.386s]
[INFO] babble ............................................ SUCCESS [0.001s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

9.3.3. Step 3: Project inheritance

Now, we will see how we can define the babble aggregator project to be a parent as well, and see how this can simplify our pom files.

Step 3: Project inheritance

A. Open up Notepad (or another text editor)

B. Load babble\babble-core\pom.xml into the editor.

C. Modify the babble-core pom file so that it is a child of the babble project. Modify the the babble-core pom.xml file to look like this:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>babble</artifactId>
    <groupId>com.sonatype.training</groupId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>
  <artifactId>babble-core</artifactId>
  <packaging>jar</packaging>
  <version>1.0.0-SNAPSHOT</version>
  <name>babble-core</name>
  ...
```

You have now made the babble-core module a child of the babble project we created earlier.

D. Compile the project to make sure it still works by running mvn compile in `C:\Projects\babble`.

E. Load `babble\babble-services\pom.xml` into the editor.

F. Modify the babble-services pom file so that it is a child of the babble project. Modify the the babble-services pom.xml file to look like this:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>babble</artifactId>
    <groupId>com.sonatype.training</groupId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>
  <artifactId>babble-services</artifactId>
  <name>babble-services</name>
  <version>1.0.0-SNAPSHOT</version>
```

You have now made the babble-services module a child of the babble project we created earlier.

G. Compile the project to make sure it still works by running mvn compile in the `babble` directory.

Now we are going to refactor the pom files. In the babble-core and babble-services pom files, there are redundent dependencies and compiler configurations. Refactor the common dependencies and plugin configurations into the parent pom.xml file. Then recompile the entire project to ensure that everything still works.

9.4. Extra Credit

The compiler configuration could also be placed in a <pluginManagement> block in the parent pom.xml file. Try using the <pluginManagement> for this, rather than the <plugins> block.

9.5. Discussion

You should now be familiar with Maven inheritence and aggregation, and how to use these concepts to structure your projects.

Lab 10. Working with Properties and Filters

10.1. Goal

The aim of this lab is to learn how to use filtering in your resources.

10.2. Agenda

- Activate filtering on a log4j configuration file
- Add a property to the pom file to parameterize the log4j logging.
- Control this parameter from the command line

10.3. Lab Exercises

First, delete your existing babble, babble-services and babble-core projects from your Eclipse workspace and import the starting point project from the labs/lab-10/start/babble directory.

In this exercise, we are going to control the level of information displayed in the babble-core application log messages using properties and filters. The babble-core application uses log4j to handle log messages. The quantity of information displayed is managed in the `src/test/resources/log4j.properties` file, shown here:

```
log4j.rootLogger=debug, stdout, R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=babble.log

log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

Currently, you need to modify the `log4j.rootLogger` property to modify the quantity of log messages displayed. For example, if you leave this value as shown above, and run the unit tests, a large number of messages will be displayed.

```
$ mvn test
...
-------------------------------------------------------
 T E S T S
```

```
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: jake
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Dude!
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Dude!
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.122 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

However, replace the first line of the `src/test/resources/log4j.properties` file with the following:

```
log4j.rootLogger=info, stdout, R
```

Now, when you run the unit tests, only a small number of messages will be displayed:

```
$ mvn test
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.114 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

In this exercise, we will use properties to control the quantity of information displayed.

10.3.1. Step 1: Insert a property into the log4j properties file

The first thing to do is to introduce a variable into the log4j.properties file:

A. Open `babble\babble-core\src\test\resources\log4j.properties` in Notepad (or another Text Editor)

B. Replace the first line of the `babble\babble-core\src\test\resources\log4j.properties` file with the following:

```
log4j.rootLogger=${log4j.level}, stdout, R
```

10.3.2. Step 2: Activate filtering

We also need to activate filtering on files in the test resource directory:

A. Open `babble\babble-core\pom.xml` in Notepad (or another Text Editor)

B. Adding the following XML to the babble-core pom.xml file, if there is already a <build> element, add the <testResources> element to the existing <build> element:

```
  <build>
    <testResources>
      <testResource>
        <directory>src/test/resources</directory>
        <filtering>true</filtering>
      </testResource>
    </testResources>
  </build>
```

10.3.3. Step 3: Define a property

Now we need to define a property in our `pom.xml` file.

A. Open `babble\babble-core\pom.xml` in Notepad (or another Text Editor)

B. You can do this by adding the following code at the end of the pom.xml file:

```
  ...
  <properties>
    <log4j.level>info</log4j.level>
  </properties>
</project>
```

This will define a property called log4j.level, with a value of "info".

C. Now run the unit tests from `babble\babble-core`. You should only get a small quanity of log messages:

```
$ mvn test
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
```

```
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.114 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

10.3.4. Step 4: Setting a property value on the command line

You can override the property values on the command line, using the -D option. Try using the -D command line parameter to set the logging level to different values (debug, info, warn and error), and compare the results:

```
$ mvn test -Dlog4j.level=warn
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.109 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

10.3.5. Step 5: Selectively filtering files

Now we will apply filtering to a subset of our resource files. For the sake of argument, we will say that we only want to activate filtering for property files, and not for XML files. This involves two steps: activating filtering for the files we want to filter, and disabling filtering for the others. First of all, add an <includes> block to ensure that only properties files will be filtered:

```
  <build>
    <testResources>
      <testResource>
        <directory>src/test/resources</directory>
        <filtering>true</filtering>
        <includes>
          <include>**/*.properties</include>
        </includes>
      </testResource>
    </testResources>
  </build>
```

Now run "mvn clean test". How many files are copied into the target/test-classes directory?

```
$ mvn clean test...

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
```

```
[INFO] ------------------------------------------------------------------------
```

Now add an <excludes> block to ensure that the other resource files are copied, but not filtered:

```
  <build>
    <testResources>
      <testResource>
        <directory>src/test/resources</directory>
        <filtering>true</filtering>
        <includes>
          <include>**/*.properties</include>
        </includes>
      </testResource>
      <testResource>
        <directory>src/test/resources</directory>
        <filtering>false</filtering>
        <excludes>
          <exclude>**/*.properties</exclude>
        </excludes>
      </testResource>
    </testResources>
  </build>
```

Now run "mvn clean test" again. How many files are copied into the target/test-classes directory this time?

10.4. Discussion

You should now have an understanding of how properties can be used in a Maven project to parameterize the build process.

Lab 11. Working with Profiles

11.1. Goal

The aim of this lab is to learn how build profiles can be used to parameterize your build process.

11.2. Agenda

- Define a build profile
- Use this build profile to control the log4j logging configuration
- Define multiple build profiles

11.3. Lab Exercises

In this exercise, we are going to build on the previous example, and set up build profiles to control the log4j logging level. You can either continue with the project you were using in the previous lab, or use the lab-11 starting point. If you want to start with a clean project, delete your existing babble, babble-services and babble-core projects from your Eclipse workspace and import the starting point project from the labs/lab-11/start/babble directory.

11.3.1. Step 1: Create a new build profile

The first thing to do is to define a development profile:

A. Open `babble\babble-core\pom.xml` in Notepad (or another Text Editor)

B. Add a <properties> and <profiles> section to the pom.xml file as shown below:

```
  <properties>
    <log4j.level>info</log4j.level>
  </properties>
  <profiles>
    <!-- Development environment -->
    <profile>
      <id>development</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <log4j.level>DEBUG</log4j.level>
      </properties>
    </profile>
  </profiles>
</project>
```

11.3.2. Step 2: Run the tests using this profile

Now run the tests using mvn test.

A. Run mvn test from `babble\babble-core`.

Note that the development profile added in the previous step is activated by default, so the property value it defines will override the default value defined in the initial <properties> block:

```
C:\Projects\babble\babble-core> mvn test

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: jake
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Dude!
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Dude!
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.125 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

11.3.3. Step 3: Defining multiple profiles

Next let's add a second profile:

A. Open `babble\babble-core\pom.xml` in Notepad (or another Text Editor)

B. Remove the activeByDefault block from the development profile, and add a second profile for a production environment, which only displays warnings:

```
  ...
  <properties>
    <log4j.level>info</log4j.level>
  </properties>
  <profiles>
    <!-- Development environment -->
```

```
    <profile>
      <id>development</id>
      <properties>
        <log4j.level>DEBUG</log4j.level>
      </properties>
    </profile>
    <profile>
      <id>production</id>
      <properties>
        <log4j.level>WARN</log4j.level>
      </properties>
    </profile>
  </profiles>
</project>
```

C. Run the tests with no profile active. Note that the "default" property value is now being used:

```
C:\Projects\babble\babble-core> mvn test
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.114 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

D. Now run the tests with each of the profiles, using the -P option to activate a profile, e.g.

```
C:\Projects\babble\babble-core> mvn test -Pdevelopment
...

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: jake
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Dude!
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Dude!
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.122 sec

Results :
```

```
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

11.3.4. Defining a default profile

Configure the development profile to be the default profile,

```
  ...
  <profiles>
    <!-- Development environment -->
    <profile>
      <id>development</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <log4j.level>DEBUG</log4j.level>
      </properties>
    </profile>
    ...
```

Now run the tests again. You should get something like this:

```
C:\Projects\babble\babble-core> mvn test
...

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: jake
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Dude!
DEBUG [main] (Babbler.java:37) - Creating new babbler: joe
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Yo!
DEBUG [main] (Babbler.java:73) - Babble uttered for joe :Dude!
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.122 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

How has this affected the default property value defined in the properties?

Now use the -P option to deactivate the 'development' profile:

```
$ mvn test -P -development
```

You should get something like this:

```
C:\Projects\babble\babble-core> mvn test
...

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.sonatype.training.babble.domain.BabblerTest
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
 INFO [main] (Babbler.java:34) - Attempt to create a babbler with an empty name
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.201 sec
Running com.sonatype.training.babble.domain.BabbleTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.066 sec

Results :

Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
```

11.3.5. Using the Help plugin to identify active profiles

The maven-help-plugin plugin is useful to determine which profiles are active in a given environment. Try running it to see which profiles are active with the various command-line options we have used in this lab, e.g:

```
C:\Projects\babble\babble-core> mvn help:active-profiles
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building babble-core
[INFO]    task-segment: [help:active-profiles] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [help:active-profiles {execution: default-cli}]
[INFO]
Active Profiles for Project 'com.sonatype.training:babble-core:jar:1.0.0-SNAPSHOT':

The following profiles are active:

 - development (source: pom)



[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Thu Jan 14 20:39:37 NZDT 2010
[INFO] Final Memory: 10M/80M
[INFO] ------------------------------------------------------------------------
```

11.4. Discussion

You should now have a further understanding of how profiles and properties can be used together to parameterize the build process.

Lab 12. Adding Reporting

12.1. Goal

The aim of this lab is to learn how to add code quality metricsreporting to your project

12.2. Agenda

- Add unit test reports and javadoc
- Add Checkstyle, PMD and FindBugs code quality metrics reporting to your project

12.3. Lab Exercises

In this exercise, we are going to add a variety of reports to the babble-core project. Delete your existing babble, babble-services and babble-core projects from your Eclipse workspace and import the starting point project from the labs/lab-12/start/babble directory.

12.3.1. Step 1: Adding unit test reports

The first report we will add is the unit test results. These are generated using the maven-surefire-report plugin. To add the unit test report:

A. Open `babble\babble-core\pom.xml` in Notepad (or another Text Editor)

B. Add a reporting section as shown here:

```
  ...
  <reporting>
    <plugins>
      <plugin>
        <artifactId>maven-surefire-report-plugin</artifactId>
        <version>2.4.3</version>
      </plugin>
    </plugins>
  </reporting>
</project>
```

C. Generate the Maven site, run mvn site from `babble\babble-core`.

D. View the results in your web browser by opening `babble\babble-core\target\site\index.html`.

12.3.2. Step 2: Adding javadoc and HTML source code

The next report we will add is Javadoc.

A. Open `babble\babble-core\pom.xml` in Notepad (or another Text Editor)

B. Add the following plugins to the reporting section:

```
  ...
  <reporting>
    <plugins>
      <plugin>
        <artifactId>maven-surefire-report-plugin</artifactId>
        <version>2.4.3</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jxr-plugin</artifactId>
        <version>2.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>2.5</version>
      </plugin>
    </plugins>
  </reporting>
</project>
```

C. Generate the Maven site, run mvn site from `babble\babble-core`.

D. View the results in your web browser by opening `babble\babble-core\target\site\index.html`.

12.3.3. Step 4: Add Checkstyle, PMD and FindBugs reports

Next, we will add Checkstyle and PMD and FindBugs reports.

A. Open `babble\babble-core\pom.xml` in Notepad (or another Text Editor)

B. Add the following plugins to the reporting section:

```
      <plugin>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <version>2.2</version>
        <configuration>
          <configLocation>config/maven_checks.xml</configLocation>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-pmd-plugin</artifactId>
        <version>2.4</version>
        <configuration>
          <targetJdk>1.5</targetJdk>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>findbugs-maven-plugin</artifactId>
        <version>1.2</version>
      </plugin>
```

C. Generate the Maven site, run mvn site from `babble\babble-core`.

D. View the results in your web browser by opening `babble\babble-core\target\site\index.html`.

12.3.4. Step 4: Add Code Coverage reports

Next, we will add Cobertura code coverage reposrts to your application.

A. Open `babble\babble-core\pom.xml` in Notepad (or another Text Editor)

B. Add the following plugins to the reporting section:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>cobertura-maven-plugin</artifactId>
</plugin>
```

C. Generate the Maven site, run mvn site from `babble\babble-core`.

D. View the results in your web browser by opening `babble\babble-core\target\site\index.html`.

12.4. Discussion

You should now have an understanding of how to generate reports in the Maven site.

Lab 13. Working with Jetty

13.1. Goal

The aim of this lab is to learn how to run a Jetty server using Maven.

13.2. Agenda

- Add a web module to our project
- Run this web module using Jetty
- Modify other modules and observe how changes are taken into account

13.3. Lab Exercises

In this exercise, we are going to add a web application to our project, and run it with Jetty.

13.3.1. Step 1: Prepare your web application

The lab starting point will provide you with a simple web application that you will need to complete. Delete your existing babble, babble-services and babble-core projects from your Eclipse workspace and import the starting point project from the labs/lab-13/start/babble directory.

This project won't compile initially - you will need to add a dependency in the web application's pom.xml file to the babble-services module, as shown here:

```
<dependencies>
  <dependency>
    <groupId>com.sonatype.training</groupId>
    <artifactId>babble-services</artifactId>
    <version>1.0.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

The project should compile correctly now. In some versions of M2Eclipse, Eclipse may have trouble finding the Babble and Babbler classes. In this case, you should disable the "Resolve dependencies from Workspace" option in the project properties:

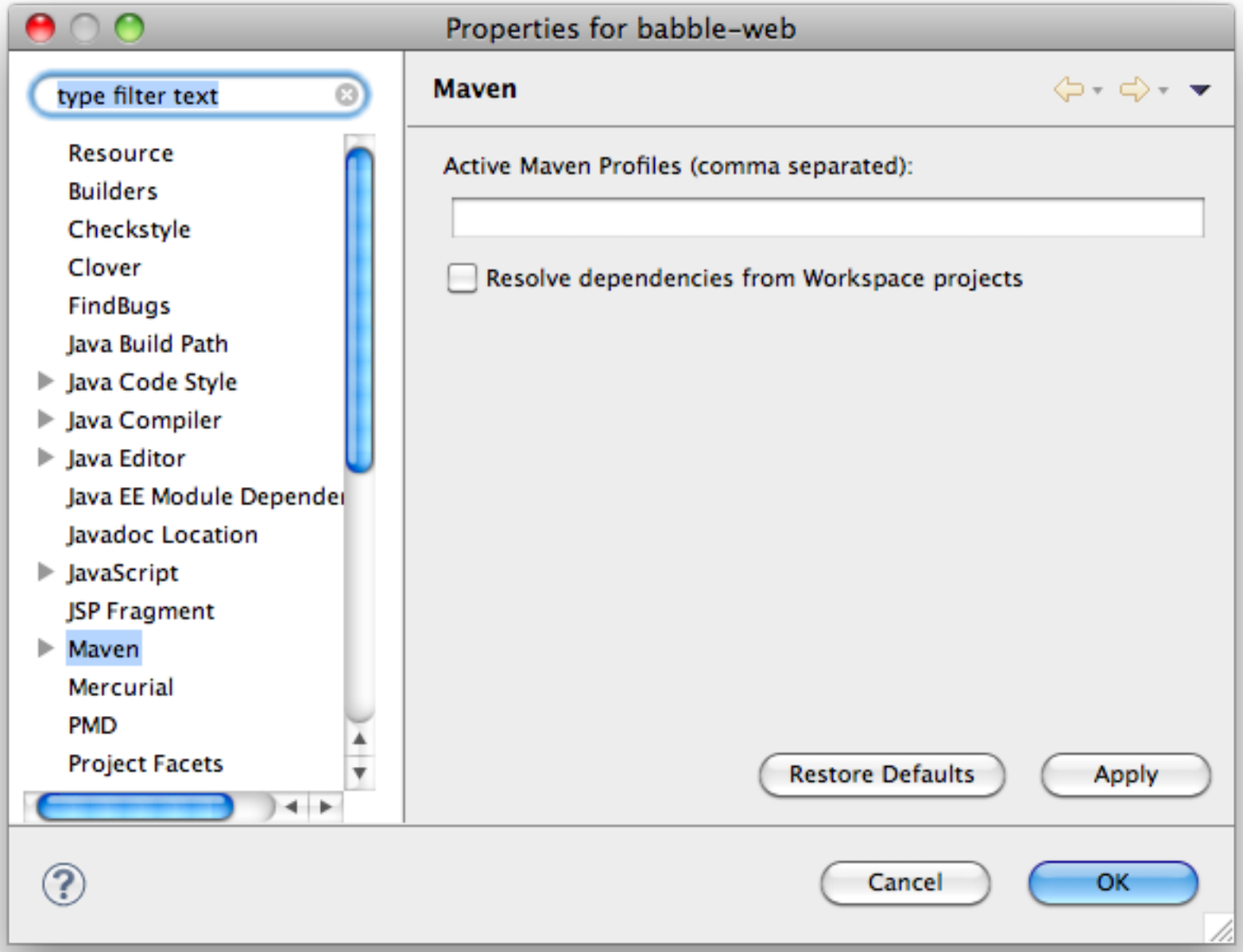

Figure 13.1. In some versions of Eclipse, you may need to turn off the "Resolve dependencies from Workspace' option

13.3.2. Step 2: Configure the Jetty Plugin

Now you need to configure Jetty. This is easy:

A. Add the following to the <build> section of the babble-web pom.xml file:

```
  <build>
    ...
    <plugins>
      <plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>maven-jetty-plugin</artifactId>
        <version>6.1.15</version>
        <configuration>
          <scanIntervalSeconds>5</scanIntervalSeconds>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

13.3.3. Step 3: Configure Resource Filtering

The `babble-web` project contains a properties file that is filtered during the build process. To configure resource filtering on the `src/main/resources` directory:

A. Add the following `resources` element as a child of the `build` element after the `plugins` element added in the previous step:

```
<resources>
  <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
  </resource>
</resources>
```

> Note
>
> If you are confused about where to put the `resources` element, take a quick look at the `pom.xml` in the lab solution's `babble-web` project.

13.3.4. Step 4: Build the Entire Multimodule Project

When you are developing a complex, multimodule project it is often a good idea to perform a clean install after you have changed any POMs.

A. Run mvn clean install from `C:\Projects\babble\`:

```
$ mvn clean install
[INFO] Scanning for projects...
```

13.3.5. Step 5: Start the Jetty Servlet Container

Start the Jetty Servlet Container.

A. Start up the Jetty web server from the command line by typing mvn jetty:run from `C:\Projects\babble\babble-web`:

```
$ mvn jetty:run
```

```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building babble web application
[INFO]    task-segment: [jetty:run]
[INFO] ------------------------------------------------------------------------
[INFO] Preparing jetty:run
...
[INFO] [jetty:run]
[INFO] Configuring Jetty for project: babble web application
[INFO] Webapp source directory = /Users/johnsmart/Projects/sonatype/sonatype-training
[INFO] Reload Mechanic: automatic
[INFO] web.xml file = /Users/johnsmart/Projects/sonatype/sonatype-training/lab-soluti
[INFO] Classes = /Users/johnsmart/Projects/sonatype/sonatype-training/lab-solutions/b
2009-04-03 22:47:17.356::INFO:  Logging to STDERR via org.mortbay.log.StdErrLog
[INFO] Context path = /babble-web
[INFO] Tmp directory =  determined at runtime
[INFO] Web defaults = org/mortbay/jetty/webapp/webdefault.xml
[INFO] Web overrides =  none
[INFO] Webapp directory = /Users/johnsmart/Projects/sonatype/sonatype-training/lab-so
[INFO] Starting jetty 6.1.15 ...JJ
2009-04-03 22:47:17.426::INFO:  jetty-6.1.15
2009-04-03 22:47:17.517::INFO:  No Transaction manager found - if your webapp require
2009-04-03 22:47:17.699::INFO:  Started SelectChannelConnector@0.0.0.0:8080
[INFO] Started Jetty Server
[INFO] Starting scanner at interval of 5 seconds.
```

B. Verify that Jetty is running on port 8080 by loading http://localhost:8080/babble-web/ in a web browser. Go take a look at the next killer app in social networking!

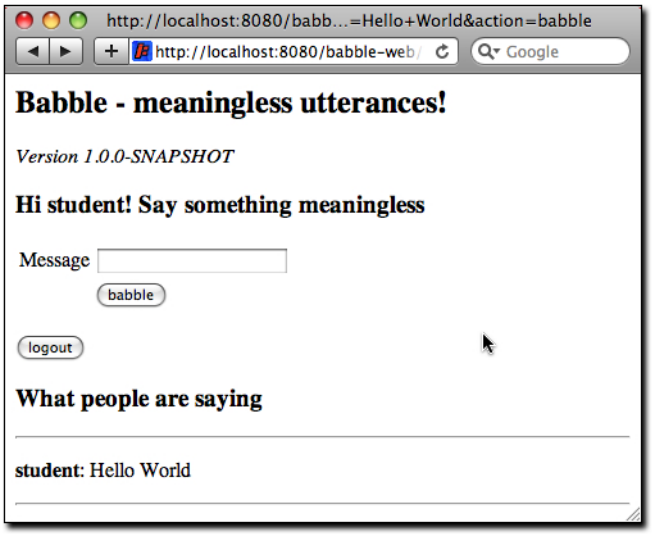

Figure 13.2. The babble web application in action

13.3.6. Step 6: Modifying the web application

Once Jetty is running, it will detect changes to the source code immediately. To test this idea, make some (harmless) modifications in the index.jsp file and refresh the screen. You should see the results immediately.

13.3.7. Step 6: Adding the web application to the parent container project

This project should be built along with the other modules. Add the 'babble-web' module to the 'modules' section in the parent project, as shown here:

```
  <modules>
    <module>babble-core</module>
    <module>babble-services</module>
    <module>babble-web</module>
  </modules>
```

Now rebuild the entire project from the babble directory. Maven should now build all three modules, as shown here:

```
$ mvn clean install
[INFO] Scanning for projects...
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] babble ..................................... SUCCESS [2.543s]
[INFO] babble-core ................................ SUCCESS [3.641s]
[INFO] babble-services ............................ SUCCESS [1.220s]
[INFO] babble web application ..................... SUCCESS [1.513s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 10 seconds
[INFO] Finished at: Fri Jan 15 08:47:47 NZDT 2010
[INFO] Final Memory: 35M/80M
[INFO] ------------------------------------------------------------------------
```

13.4. Extra credit

Jetty will also detect changes in dependencies. For example if you modify (and install) the babble-services module, Jetty will detect the change and restart the application with the new dependency. To try this out, modify the babble-services module so that it returns the list of babbles in chronological order rather than reverse chronological order as is the case in the lab solutions. Once you make the modification, you should only need to run a "mvn install" in the babble-services project for the new code to be used in the web application.

13.5. Discussion

You should now have an idea of how you can use Jetty to write and modify code with a faster turn-around time.

Lab 14. Working with Nexus

14.1. Goal

The aim of this lab is to become familiar with the Nexus Repository server.

14.2. Agenda

- Install Nexus
- Run this web module using Jetty
- Modify other modules and observe how changes are taken into account

14.3. Lab Exercises

In this exercise, we are going to install Nexus and deploy our project onto the local Nexus repository. You can either continue with the project you were using in the previous lab, or use the lab-14 starting point. If you want to start with a clean project, delete your existing babble, babble-services and babble-core projects from your Eclipse workspace and import the starting point project from the labs/lab-14/start/babble directory.

14.3.1. Step 1: Install Nexus

To install Nexus:

A. Unzip the Nexus bundle onto your hard drive. You can obtain the latest version of Nexus from the Sonatype web site (http://nexus.sonatype.com) or use the version provided in the lab resources (see Figure 14.1, “You can download Nexus from the Sonatype web site”).

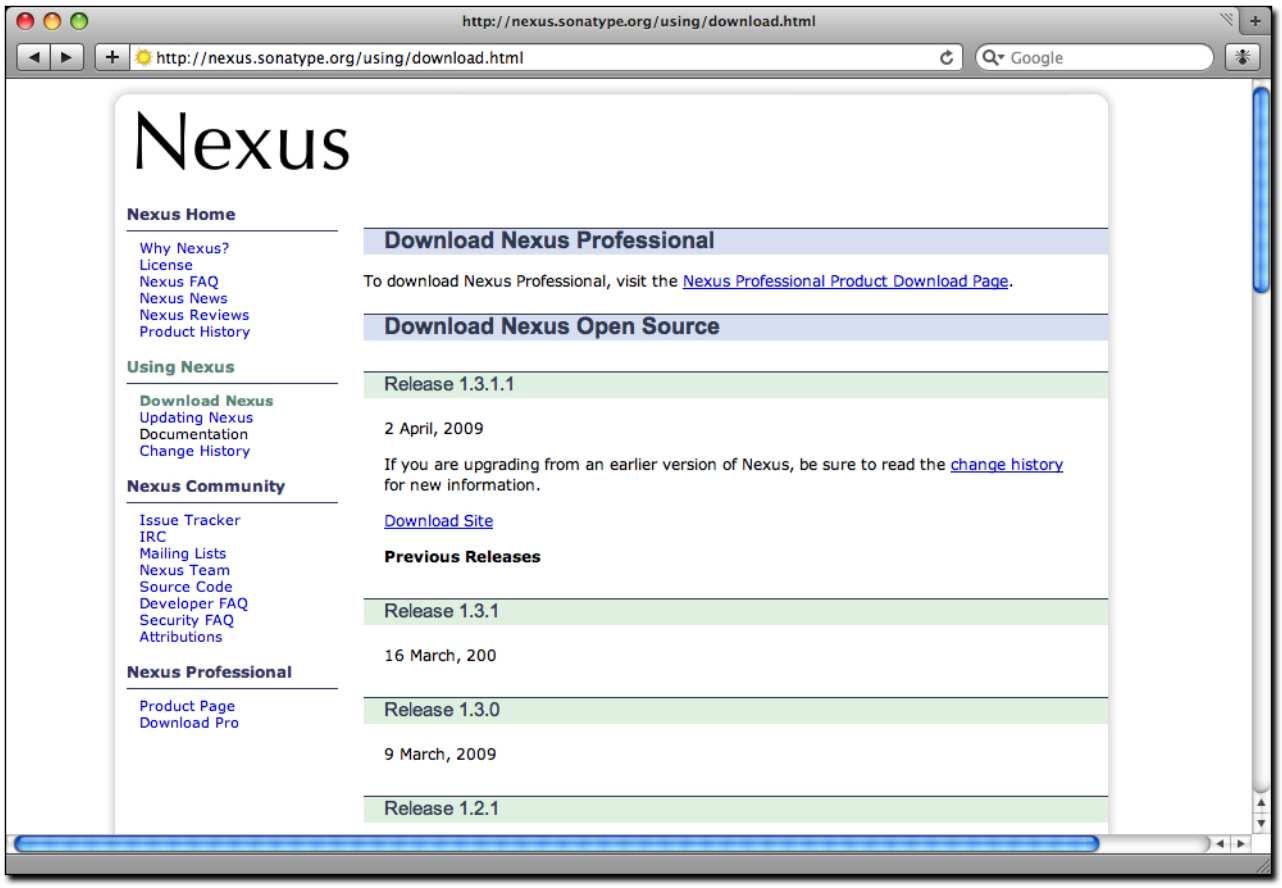

Figure 14.1. You can download Nexus from the Sonatype web site

B. Unzip Nexus in an appropriate directory on your hard drive (e.g. C:\Nexus on Windows or /usr/local/nexus on Linux or MacOS) and run the Nexus script in the directory corresponding to your O/S. For example, on Windows, you would do the following:

```
C:\nexus>nexus-webapp-1.3.1.1\bin\jsw\windows-x86-32\Nexus.bat
wrapper  | --> Wrapper Started as Console
wrapper  | Launching a JVM...
...
jvm 1    | 2009-04-05 19:22:42 INFO  - o.s.n.Nexus:default          -
jvm 1    | ------------------------------------------------
jvm 1    |
jvm 1    | Initializing Nexus (OSS), Version 1.3.1.1
jvm 1    |
jvm 1    | ------------------------------------------------
...
jvm 1    | 2009-04-05 19:22:44 INFO  - org.mortbay.log              - Started S
electChannelConnector@0.0.0.0:8081
```

Or on Linux or MacOS machine:

```
$ sudo /usr/local/nexus/nexus-webapp-1.3.1.1/bin/jsw/macosx-universal-32/nexus start
Password:
```

```
Starting Sonatype Nexus Repository Manager...
Started Sonatype Nexus Repository Manager.
```

C. Nexus should now be running on port 8081. Open a browser and go to http://localhost:8081/nexus to verify.

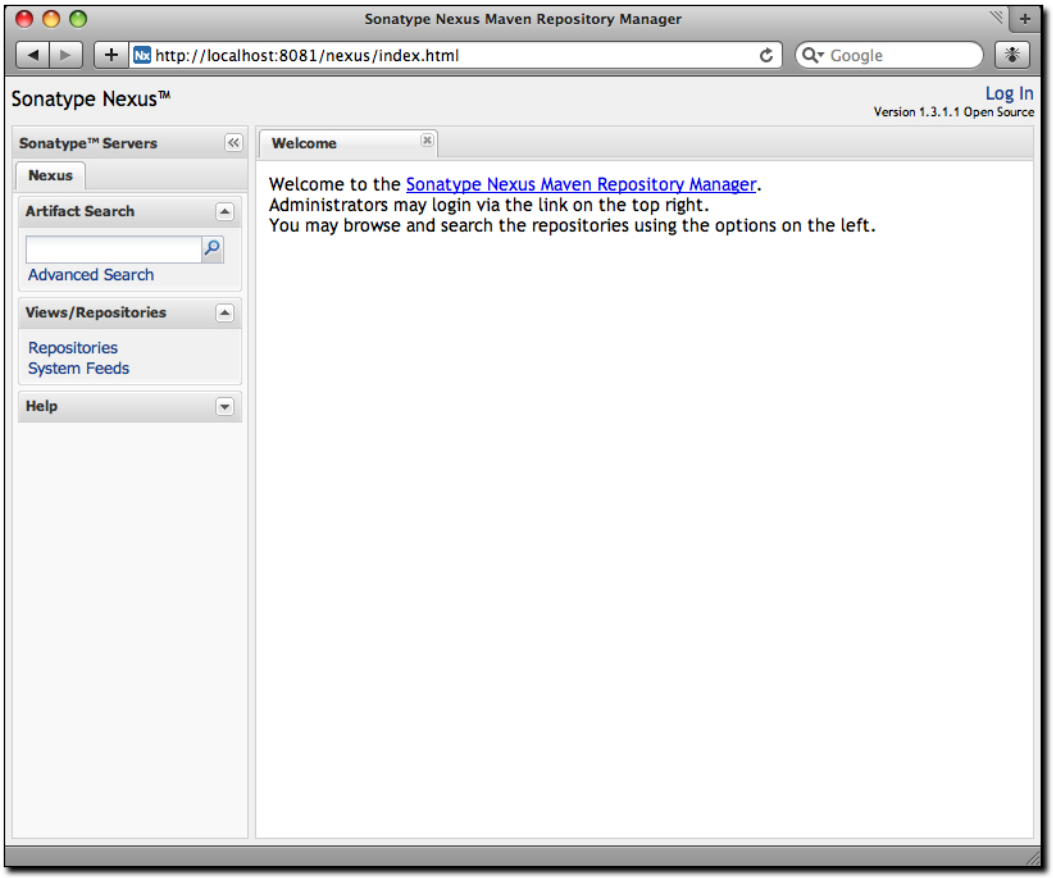

Figure 14.2. The Nexus home page

14.3.2. Step 2: Configure your settings.xml to use Nexus

Next, you need to tell Maven to use Nexus as a proxy. To do this:

A. Add the following to your ~/.m2/settings.xml file:

```
<settings>
    ...
    <mirrors>
        <mirror>
            <id>Nexus</id>
            <name>Nexus Public Mirror</name>
            <url>http://localhost:8081/nexus/content/groups/public</url>
            <mirrorOf>*</mirrorOf>
        </mirror>
    </mirrors>
```

```
</settings>
```

B. Delete or rename your local repository directory (~/.m2/repository), from your file explorer, or as shown here:

```
C:> rename %USERPROFILE%\.m2\repository repository.bak
```

or, on Linux or Mac:

```
$ mv ~/.m2/repository/ ~/.m2/repository.bak
```

C. Now rebuild the babble project. Look at where it obtains the JAR files - can you see the reference to your local Nexus repository?

```
C:\Projects\babble> mvn install
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   babble
[INFO]   babble-core
[INFO]   babble-services
[INFO] ------------------------------------------------------------------------
[INFO] Building babble
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
Downloading: http://localhost:8081/nexus/content/groups/public/org/apache/maven/plugi
11K downloaded  (maven-site-plugin-2.0-beta-7.pom)
Downloading: http://localhost:8081/nexus/content/groups/public/org/apache/maven/plugi
10K downloaded  (maven-plugins-11.pom)
Downloading: http://localhost:8081/nexus/content/groups/public/org/apache/maven/maven
23K downloaded  (maven-parent-8.pom)
...
```

14.3.3. Step 3: Deploying to Nexus

The next step is to deploy the babble artifacts to your Nexus repository. To do this:

A. Open `babble\pom.xml` in Notepad (or another Text Editor)

B. Add the following to the parent pom.xml file in your Babble project. Note that you need to configure a repository both for releases and for snapshots.

```
<project>
   <distributionManagement>
        <repository>
            <id>releases</id>
            <name>Internal Releases</name>
            <url>http://localhost:8081/nexus/content/repositories/releases/</url>
        </repository>
        <snapshotRepository>
            <id>snapshots</id>
            <name>Internal Releases</name>
            <url>http://localhost:8081/nexus/content/repositories/snapshots/</url>
        </snapshotRepository>
    </distributionManagement>
    ...
</project>
```

C. You also need to add a username and password for a user authorized to publish artifacts onto the repository. Use the default login/password of admin/admin123, and add the following server definition in your settings.xml file:

```
<settings>
   ...
   <servers>
     ...
     <server>
       <id>releases</id>
       <username>admin</username>
       <password>admin123</password>
     </server>
     <server>
       <id>snapshots</id>
       <username>admin</username>
       <password>admin123</password>
     </server>
   </servers>
   ...
</settings>
```

D. Now you are ready to deploy the babble project to your server. Run mvn deploy from the parent project directory:

```
$ mvn deploy
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   babble
[INFO]   babble-core
[INFO]   babble-services
[INFO]   babble web application
[INFO] ------------------------------------------------------------------------
[INFO] Building babble
[INFO]    task-segment: [deploy]
[INFO] ------------------------------------------------------------------------
...
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/com/sonat
[INFO] Retrieving previous metadata from local-snapshots
[INFO] repository metadata for: 'snapshot com.sonatype.training:babble-web:1.0.
[INFO] Uploading repository metadata for: 'snapshot com.sonatype.training:babbl
[INFO] Retrieving previous metadata from local-snapshots
[INFO] repository metadata for: 'artifact com.sonatype.training:babble-web' cou
[INFO] Uploading repository metadata for: 'artifact com.sonatype.training:babbl
[INFO] Uploading project information for babble-web 1.0.0-20090406.040739-1
[INFO]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] babble ..................................................... SUCCESS [4.445s]
[INFO] babble-core ................................................ SUCCESS [2.734s]
[INFO] babble-services ............................................ SUCCESS [0.851s]
[INFO] babble web application ..................................... SUCCESS [2.274s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

E. Now browse the Nexus repository to check that the babble artifacts have been redeployed.

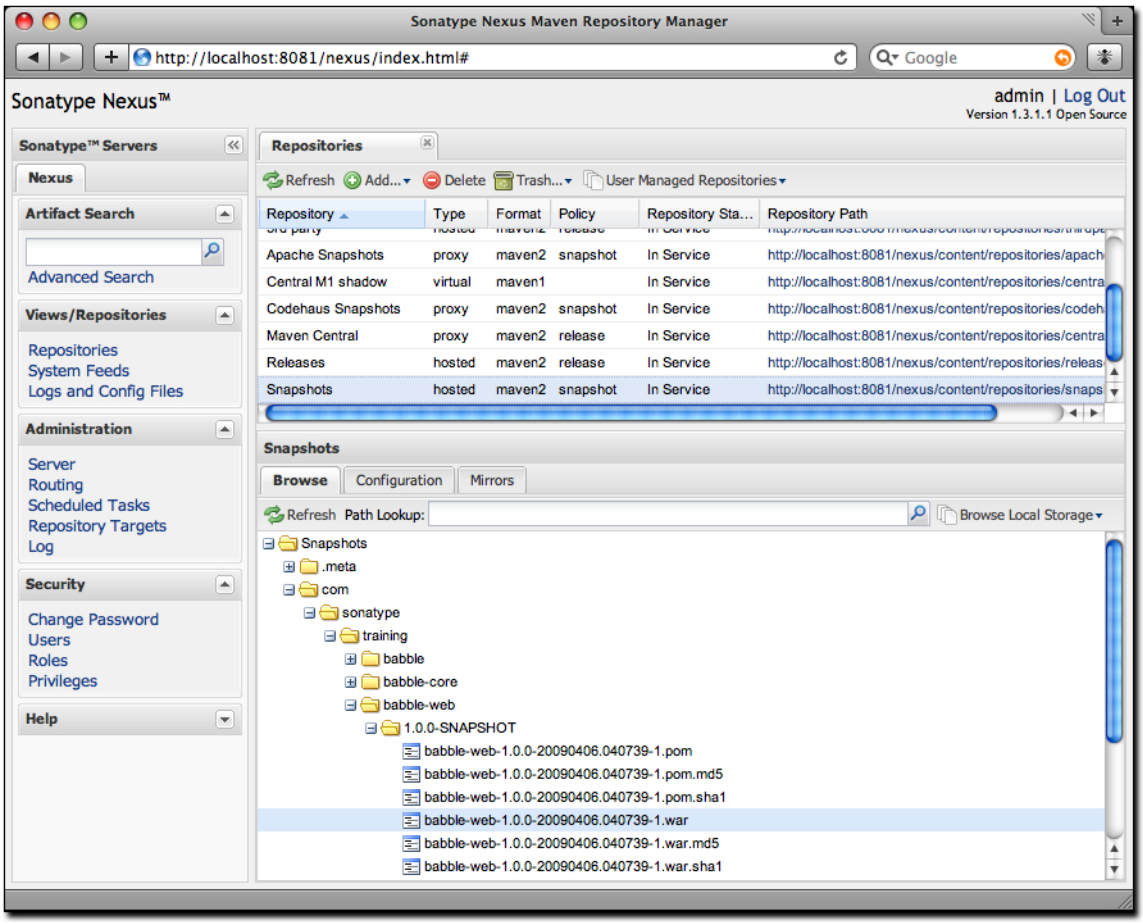

Figure 14.3. Babble artifacts deployed to Nexus

14.3.4. Extra credit

You would not normally deploy using the Nexus admin account. Try creating a new Nexus user with the 'deployment' role and configuring your settings.xml file to use this user to deploy your artifacts.

Lab 15. Automating Releases

15.1. Goal

The aim of this lab is to learn how to automate the release process with the maven-release-plugin.

15.2. Agenda

- Install Subversion
- Add the project to Subversion
- Build and deploy new releases of this application onto our Nexus server.

15.3. Lab Exercises

In this exercise, we are going to use the Nexus server we installed in the previous lab along with a local Subversion repository to manage our deployments. The lab solutions come with a pre-bundled Subversion repository that you can use for this lab. Delete your existing babble, babble-services and babble-core projects from your Eclipse workspace and import the starting point project from the labs/lab-15/start/babble directory.

15.3.1. Step 1: Starting a local Subversion server

Next, you need to set up your own personal Subversion server. We will use this server for the next two labs.

A. Change directories to your projects directory: cd c:\solutions\labs\lab-15\start

B. Now, start the server and leave it running in the background.

```
C:\..\lab-15\start>svnserve -d -r svn-repos
```

You should now have a Subversion server running on your machine.

The lab-15/start/babble directory is a checked-out working copy of the babble project stored in this local repository - you don't need to check out any code from Subversion. You are now ready to try out the automated release process.

15.3.2. Configure the <scm> section

The first thing you need to do is to configure the <scm> section in the babble/pom.xml file. Add the follow lines to this file:

```
        <scm>
            <connection>scm:svn:svn://localhost/babble</connection>
            <developerConnection>scm:svn:svn://localhost/babble</developerConnection>
        </scm>
```

Now commit these changes to Subversion, either from the command line or from Eclipse. From the command line, just run "svn commit" from the babble directory:

```
C:\..\lab-15\start>svn commit -m "Updating scm info"
```

15.3.3. Step 5: Preparing the release

The next step is to execute mvn release:prepare. This will prompt you for new version numbers and tag directories. Leave the default values, and click enter each time.

```
$ mvn release:prepare
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   babble
[INFO]   babble-core
[INFO]   babble-services
[INFO]   babble web application
[INFO] ------------------------------------------------------------------------
[INFO] Building babble
[INFO]    task-segment: [release:prepare] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [release:prepare]
[INFO] Resuming release from phase 'scm-check-modifications'
[INFO] Verifying that there are no local modifications...
[INFO] Executing: /bin/sh -c cd /Users/johnsmart/babble-work && svn --non-interactive
[INFO] Working directory: /Users/johnsmart/babble-work
[INFO] Checking dependencies and plugins for snapshots ...
What is the release version for "babble"? (com.sonatype.training:babble) 1.0.1: :
...
[INFO] Executing: /bin/sh -c cd /Users/johnsmart/babble-work && svn --non-interactive
[INFO] Working directory: /Users/johnsmart/babble-work
[INFO] Release preparation complete.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

This will test and verify that the application is sound, bump up version numbers, and create a new tag in Subversion for the new release.

15.3.4. Step 6: Perform the release

Now, run mvn release:perform to deploy the new releases to your Nexus server:

```
$ mvn release:perform
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   babble
[INFO]   babble-core
[INFO]   babble-services
[INFO]   babble web application
[INFO] ------------------------------------------------------------------------
[INFO] Building babble
[INFO]    task-segment: [release:perform] (aggregator-style)
```

```
[INFO] ------------------------------------------------------------------------
...
[INFO] Executing: /bin/sh -c cd /Users/johnsmart/babble-work && svn --non-interactive
[INFO] Working directory: /Users/johnsmart/babble-work
[INFO] Release preparation complete.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

This will check out and build the tagged version of the application, and upload the artifacts to Nexus.

15.3.5. Check that the artifacts have been deployed

Now browse through the Releases repository on your Nexus server to ensure that the artifacts have indeed been deployed.

Figure 15.1. The deployed artifacts in Nexus

15.4. Discussion

You have now successfully automated the Maven deployment process. In the next lab, we will take automation a step further with Continuous Integration.

Lab 16. Continuous Integration with Hudson and Maven

16.1. Goal

The aim of this lab is to learn how to set up a Continuous Integration environment with Maven and Hudson.

16.2. Agenda

- Install Hudson
- Add the babble project to Hudson
- Modify the source code and see how the changes trigger off builds.

16.3. Lab Exercises

In this exercise, we are going to install Hudson and use it to set up a Continuous Integration environment.

16.3.1. Step 1: Install Hudson

Before starting, we need to install Hudson. Download the latest version from the Hudson web site (`https://hudson.dev.java.net`), or use the version in the lab resources.

16.3.2. Step 2: Start Hudson

We are going to be running Hudson from the command line.

```
C:/Users/John>java -jar hudson.war
Running from: /Users/johnsmart/hudson.war
[Winstone 2009/04/07 21:42:57] - Beginning extraction from war file
[Winstone 2009/04/07 21:43:01] - No webapp classes folder found - /Users/johnsmart/.hudson/war/WEB-INF/classes
hudson home directory: /Users/johnsmart/.hudson
[Winstone 2009/04/07 21:43:04] - HTTP Listener started: port=8080
[Winstone 2009/04/07 21:43:04] - AJP13 Listener started: port=8009
[Winstone 2009/04/07 21:43:04] - Winstone Servlet Engine v0.9.10 running: controlPort=disabled
Apr 7, 2009 9:43:05 PM hudson.ClassicPluginStrategy createPluginWrapper
INFO: Loading plugin: /Users/johnsmart/.hudson/plugins/maven-plugin.hpi
Apr 7, 2009 9:43:05 PM hudson.ClassicPluginStrategy explode
INFO: Extracting /Users/johnsmart/.hudson/plugins/maven-plugin.hpi
Apr 7, 2009 9:43:05 PM hudson.model.Hudson load
INFO: Took 15 ms to load
Apr 7, 2009 9:43:05 PM hudson.TcpSlaveAgentListener <init>
```

```
INFO: JNLP slave agent listener started on TCP port 54052
```

Open a web browser to http://localhost:8080 to make sure the server is running correctly.

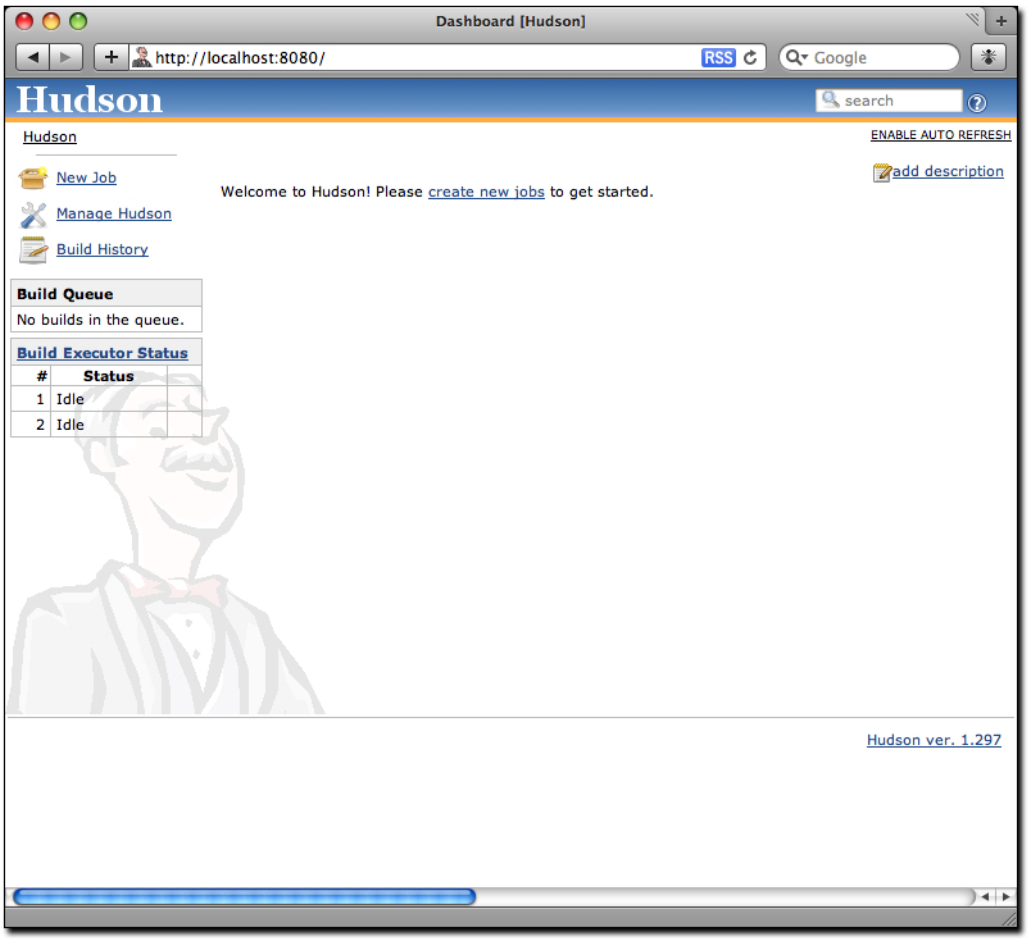

Figure 16.1. The Hudson home page

16.3.3. Step 3: Configuring Hudson

You now need to configure your Hudson installation. This involves telling it where your JDK and Maven installations are installed. Click on "Manage Hudson" and then on "Configure System". Add a JDK installation and a Maven installation corresponding to your environment. For Windows, for example this might be something like `C:\apache-maven-2.1.0` for Maven and `C:\Program Files\Java\jdk1.6.0_10` for Java.

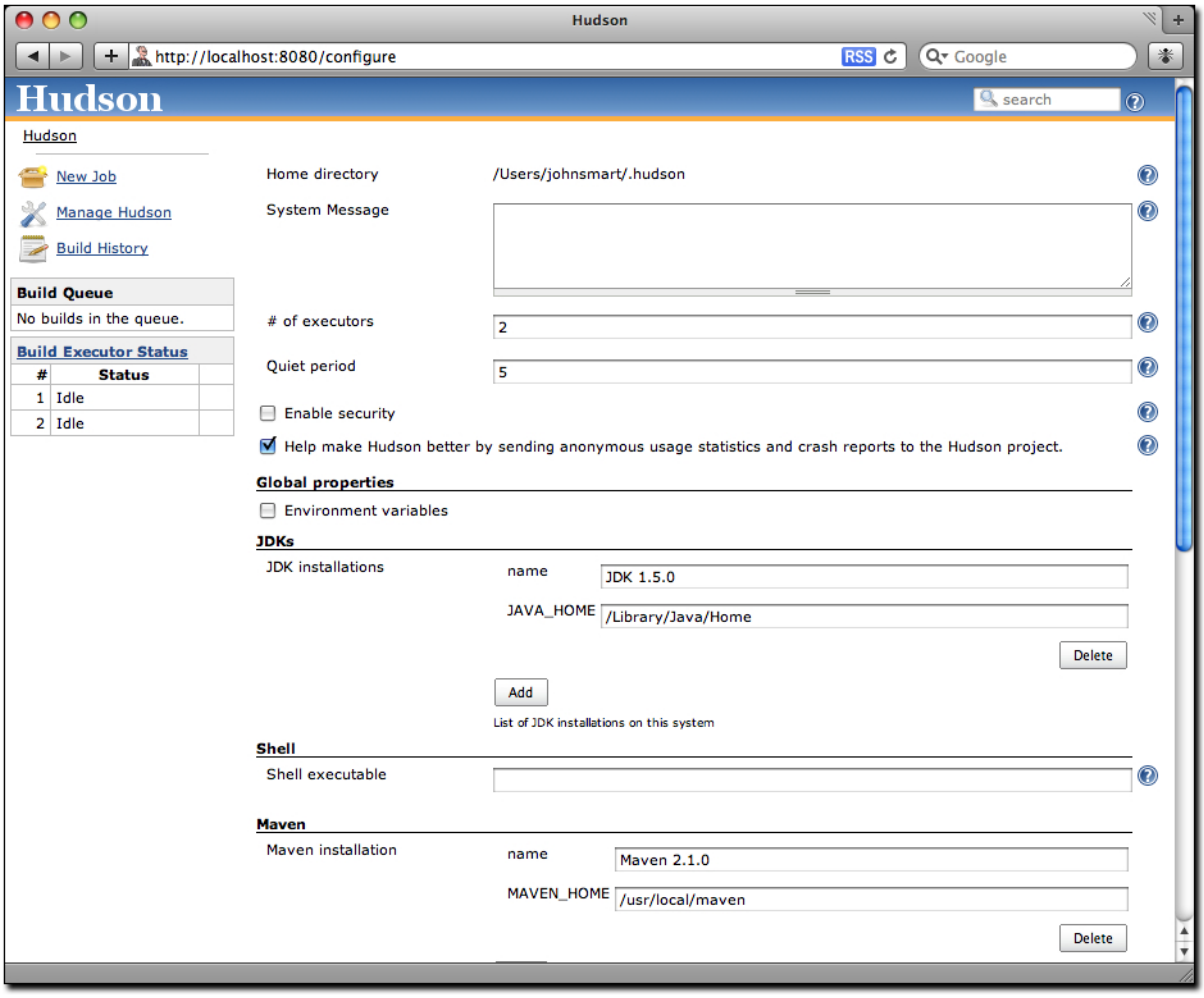

Figure 16.2. Configuring Hudson

You should also configure the email server. If you have an enterprise SMTP server handy, this is probably the way to go. Otherwise, if you have a Gmail account, you can use that (see

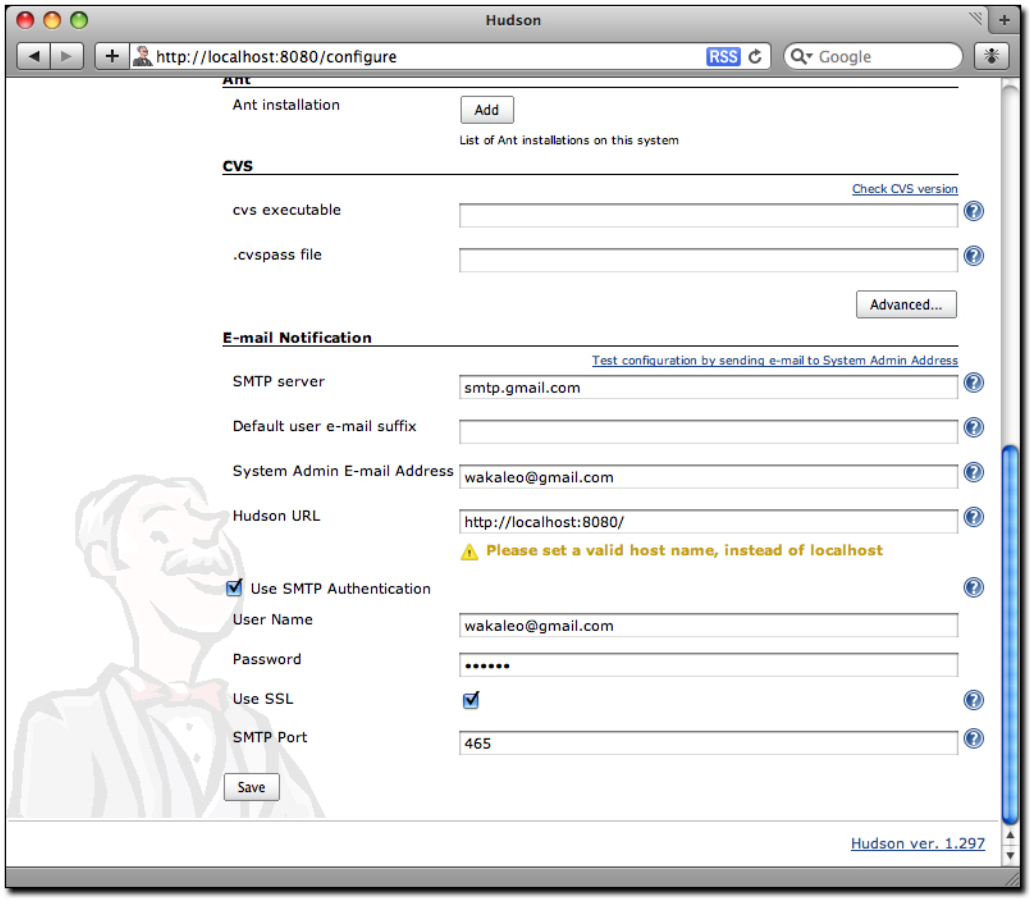

Figure 16.3. Configuring Email in Hudson

Hudson should now be ready for use.

16.3.4. Step 4: Adding a build job

Now you can add a new build job to Hudson. Click on "New Job", and choose a Maven build job called "babble"

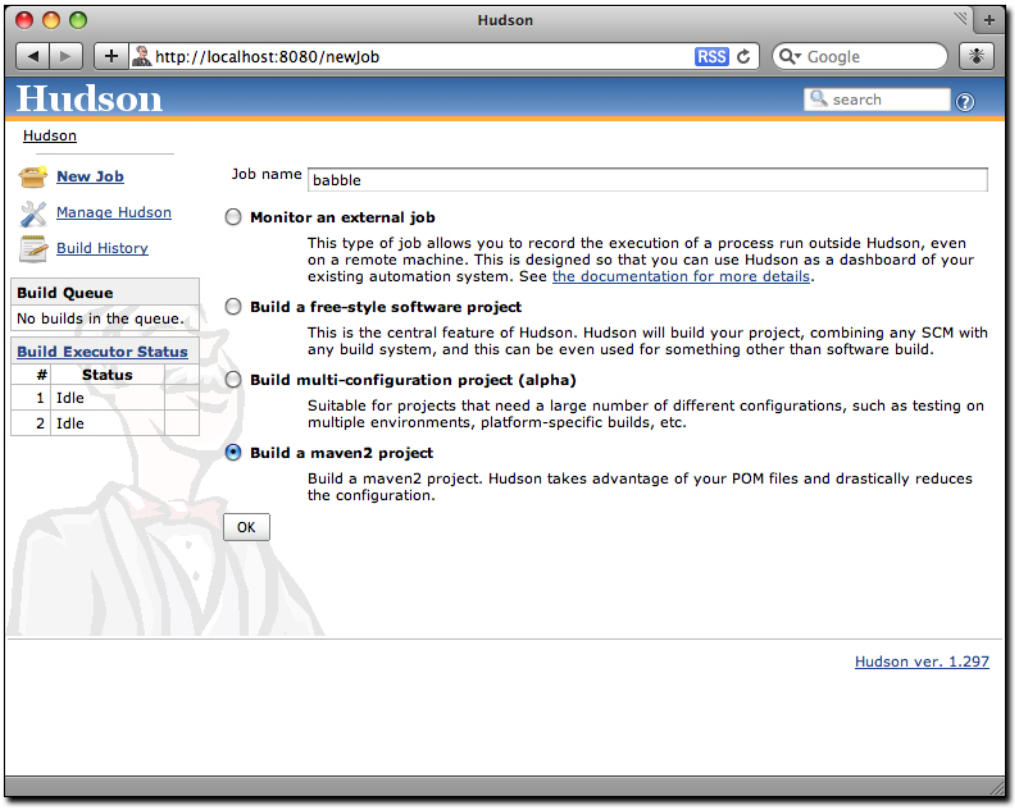

Figure 16.4. Adding a new Maven build job

Next, you need to set up this job. We will do a fairly minimal set up at this stage. You need to choose Subversion as your SCM, and provide the correct URL (svn://localhost/babble/trunk).

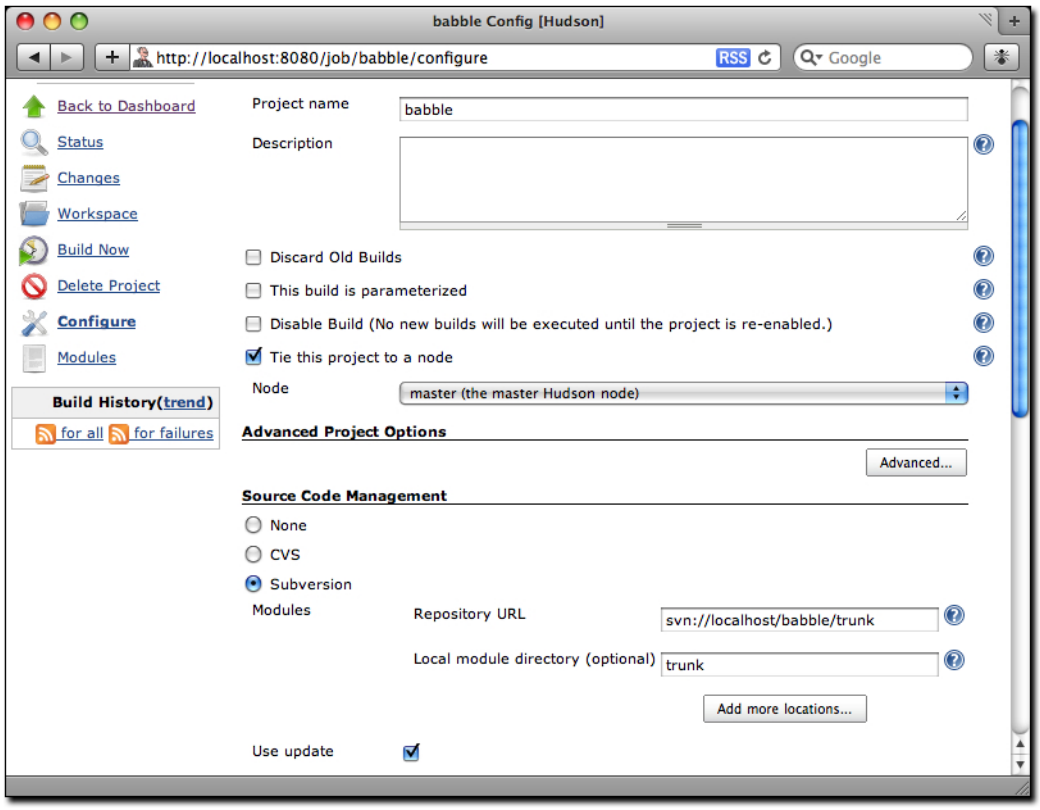

Figure 16.5. Setting up the SCM configuration

Finally, you need to configure the triggers and notification strategies. Configure Subversion to be polled every 5 minutes, or even (for less waiting) every one minute. You also need to specify what goals you want executed. In this case, "clean install" will do fine. Finally, you may also want to add your email address as a recipient, if possible.

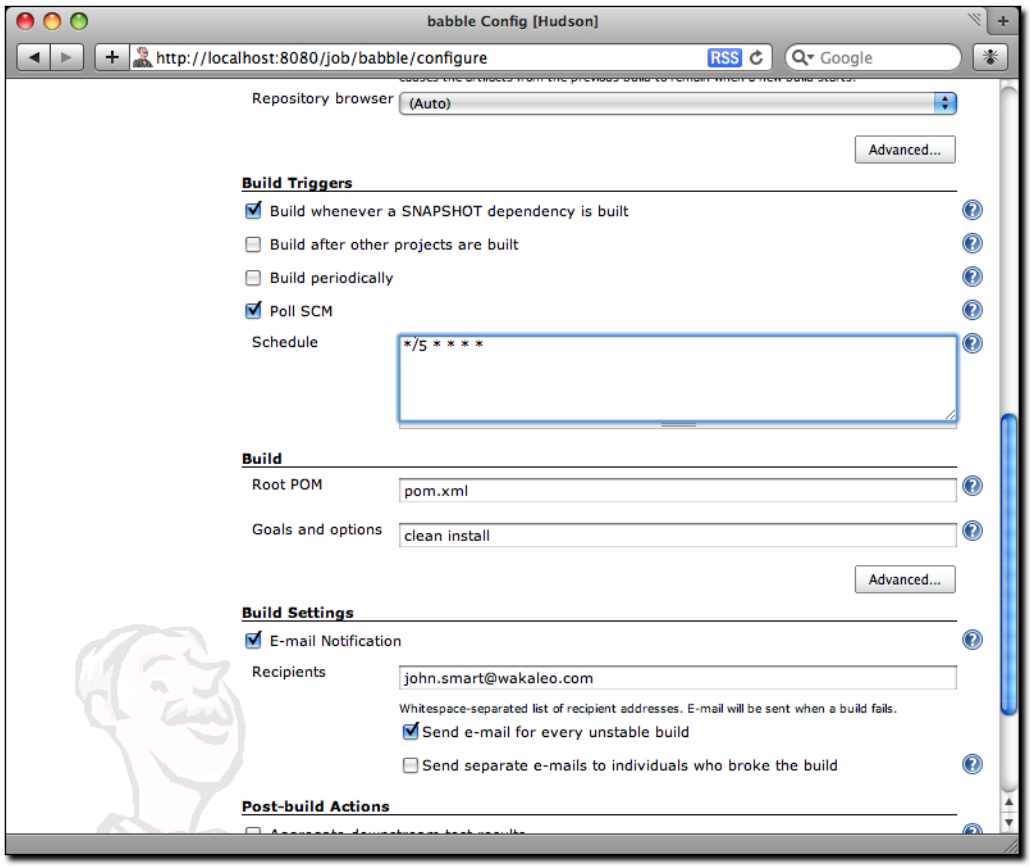

Figure 16.6. What triggers the build

> Note
>
> The cron job in the previous figure is "*/5 * * * *", make sure that there are spaces between the * characters.

Save this configuration. Now, you are ready to kick off a build.

16.3.5. Step 5: Running a build manually

To make sure everything is working correctly, click on "Build Now". Does the build run successfully? Navigate through the build result screens to see the sort of information displayed.

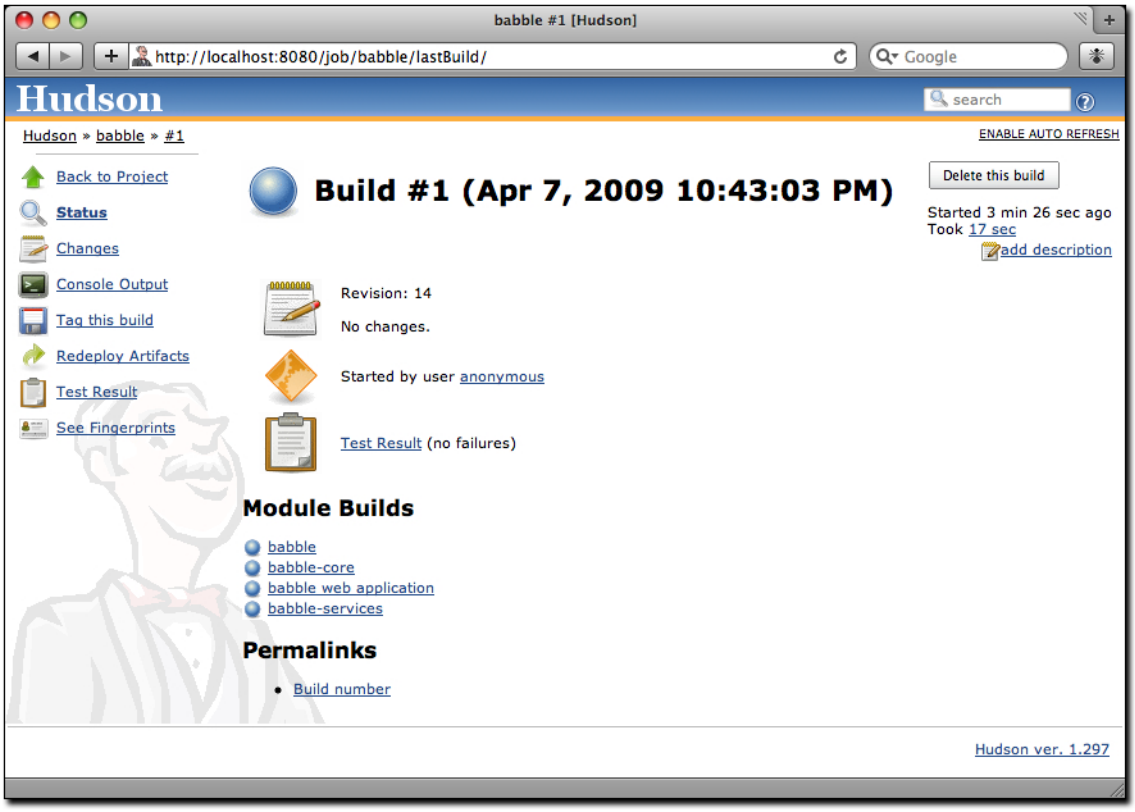

Figure 16.7. Build results

16.3.6. Step 6: Triggering a build

Now make some trivial change to the source code and commit your changes to your Subversion repository. Click on "ENABLE AUTO REFRESH" to be able to see when the build kicks in. The "ENABLE AUTO REFRESH" is a link that toggles to "DISABLE AUTO REFRESH" once it is clicked, you can see this option in the upper right-hand corner of the Hudson interface under the search box. After a short delay, you should see a new build start

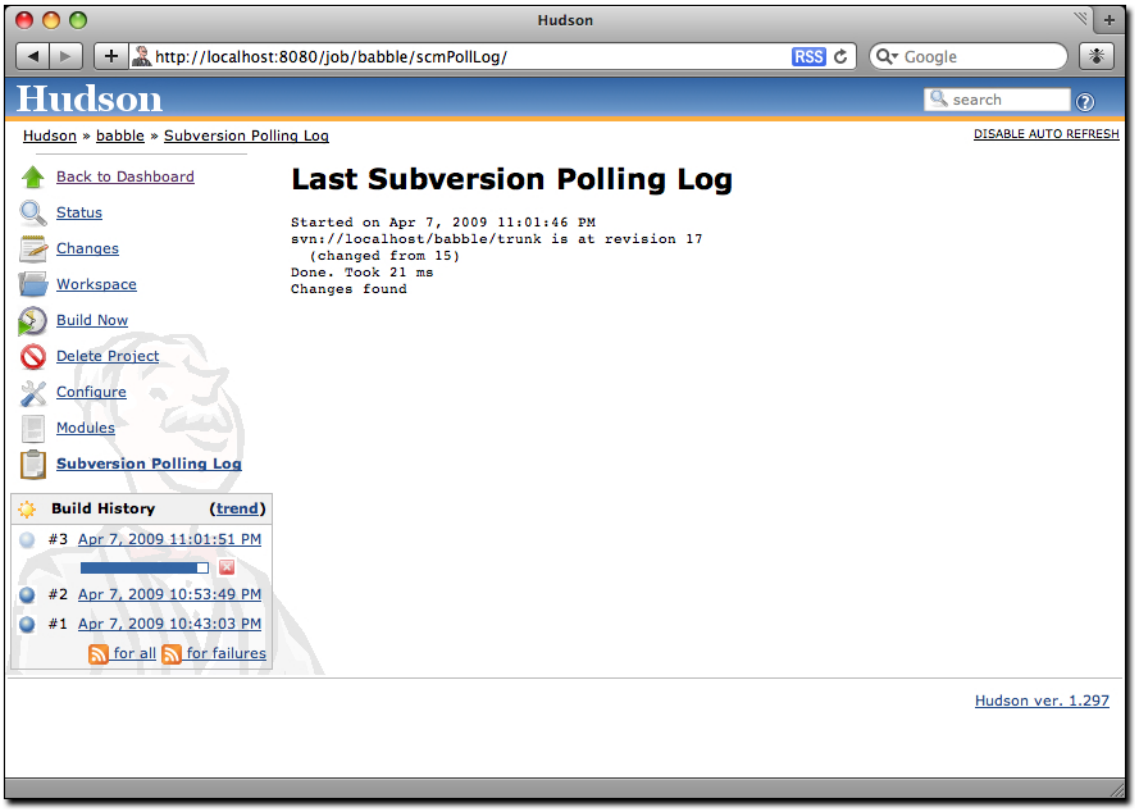

Figure 16.8. A pending build

16.3.7. Step 7: Build failures

Now introduce a deliberate error in one of the application unit tests, and commit your changes. Look at how Hudson displays the failed build, and the failing unit test. Also, look at how this test failure appears in the test result trend on the build job home page.

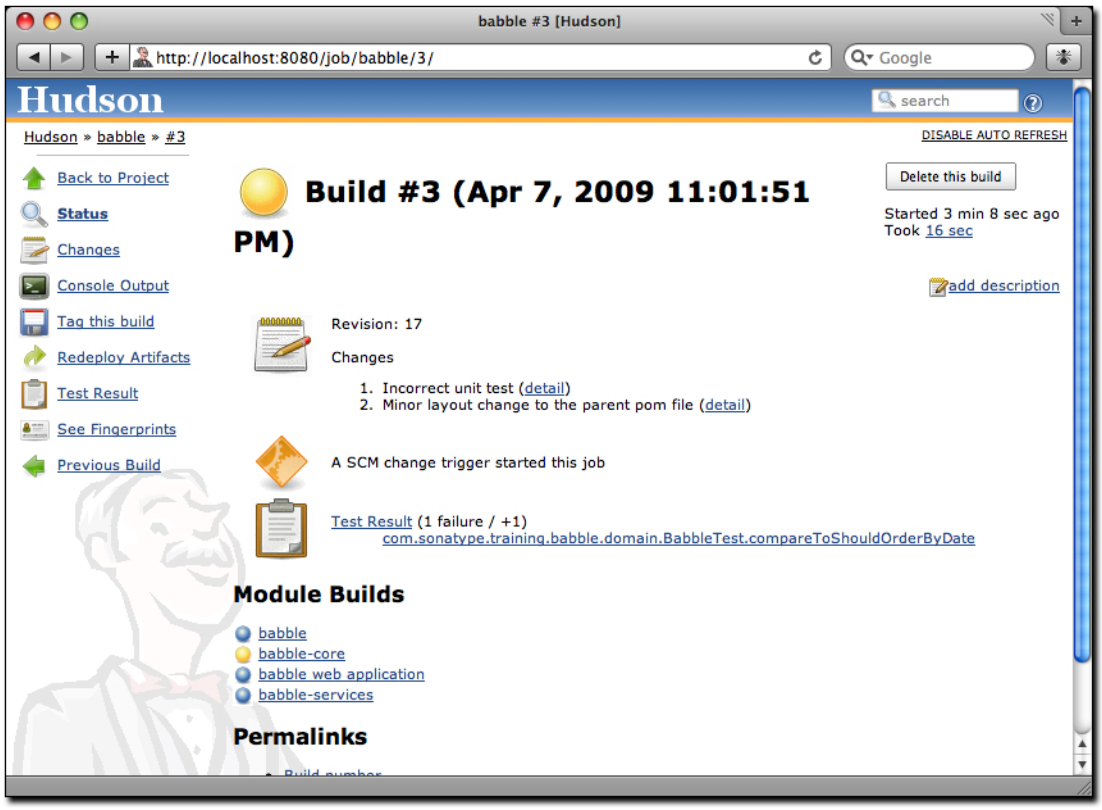

Figure 16.9. A failed build

16.3.8. Step 8: Build metrics

Hudson provides good support for code quality metrics and other similar graphs. We will now install a plugin that will display Checkstyle, PMD and Findbugs code quality metrics over time.

First, you need to install the Violations plugin. Go to "Manage Hudson", and click on "Manage Plugins". Select the "Violations Plugin" in the "Available Plugins" list and click OK. When the instalation is done, restart Hudson.

Next, we will create a build metrics job. Click on "New Job". Call the job "babble-reports". Choose "copy existing job" and enter "babble". This will create a carbon-copy of our initial project. Which is good, as there is little to change.

We will change the Build Triggers, so that the build is only triggered after the babble build job has succeeded. We will also change the build goals, from "clean install" to "clean site". This will generate the raw code quality metrics data Hudon needs for its reporting. Finally, we tick the "violations" checkbox, to activate Checkstyle, CPD, Findbugs and PMD reporting. Now save the build job and trigger a build manually.

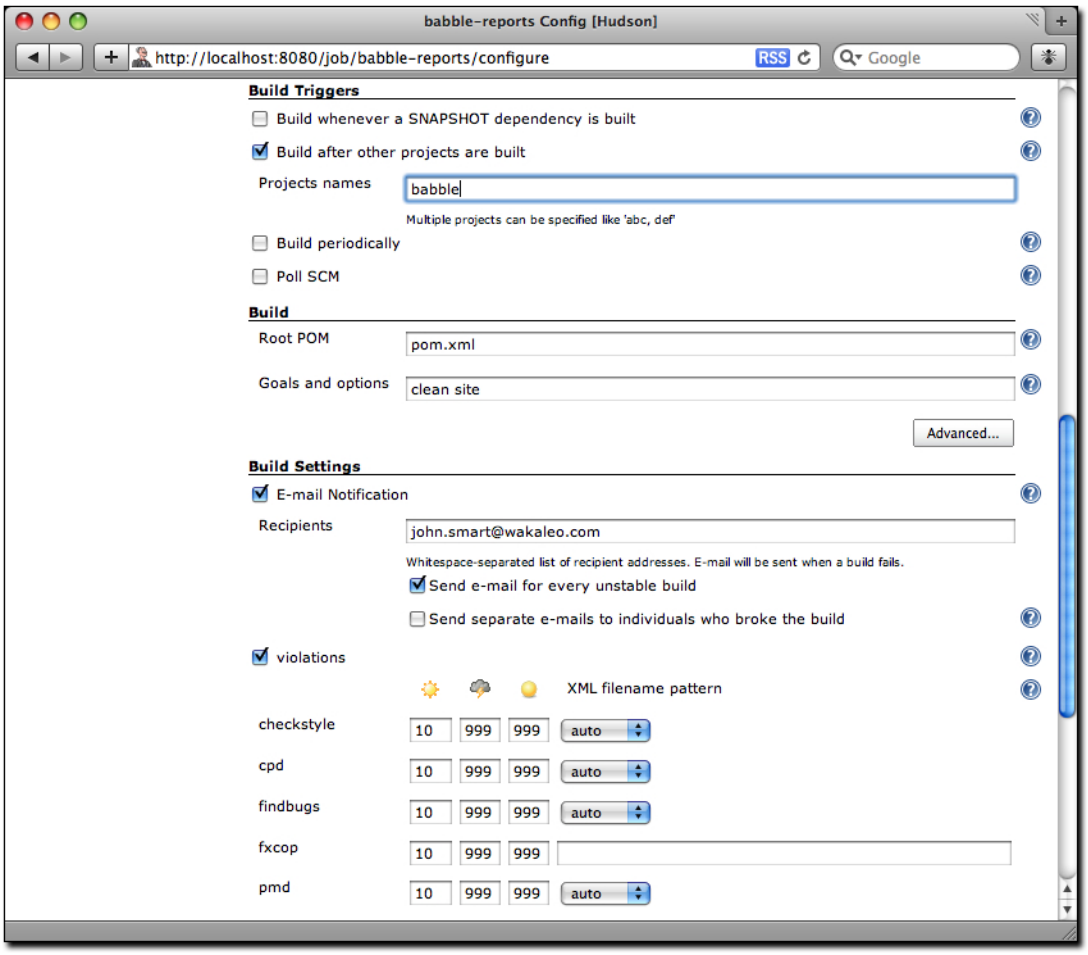

Figure 16.10. Configuring metrics in Hudson

16.4. Extra Credit

1) This application code base is small, so the number of PMD issues are limited. Try to introduce a few to generate some more significant statistics and graphs.

2) Because of an issue with multi-module projects, checkstyle reporting will not appear at the top-level project. You need to set up reporting build jobs for each module to see the checkstyle violations. See if you can do this.

16.5. Discussion

You have now successfully automated the Maven deployment process. In the next lab, we will take automation a step further with Continuous Integration.