**Software PM Lab – 5 Points**

***Subversion and the Subclipse plug-in***

**Introduction**

*"Version control is to programmers what the safety net is to a trapeze artist. Knowing the net is there to catch them if they fall, aerialists are free to fly. In the same way, version control enables you to take programming risks that you would never otherwise consider. If something goes wrong, you can always revert back to a known, good-working version of your code. You can experiment in a branch 1 off the main trunk without interfering with other team members. When bugs are discovered in an older version of a shipped product, you can easily check out that specific version to confirm, fix, and generate a patch for the bug. Without version control, you would have to be much more cautious, move more slowly, and generally be less productive.”*

*-Elliotte Harold, Adjunct Professor, Polytechnic University*

The main idea of a version control system is to contain the software project in a so called *repository*; from which developers can *check out* copies of a project and in this way create local *working copies*. These working copies can then be edited and eventually uploaded, or *committed*, back to the repository, as a new version, or *revision*, of the file that has been modified. If several developers simultaneously are editing working copies of the same file, the version control system must be able to deal with *conflicts* that might occur when the developers later try to commit their different working copies to the repository. To be sure that a developer always has the latest revision of a file he or she should *update* his copy from the repository frequently, and at least before commit.

The most commonly used version control system for the latest 10 years has been

*Concurrent Versions System*, or *CVS*. In contrast to most other version control systems

CVS uses a so called *non-locking repository*, which allows several developers to

simultaneously edit local working copies of a file. This is in contrast to a *locking*

*repository*, in which only one developer at a time can gain write access to a specific file.

Although it is a good system, CVS is starting to get aged and is, because of its limited

ability to handle file content, not suitable for all kinds of modern software development.

Because of this, in 2004, the first version of its successor *Subversion*, or *SVN*, arrived,

developed by the same people responsible for CVS.

SVN can be used in combination with Eclipse by using the plug-in *Subclipse*. In this way

you can perform version control in a window environment directly on the projects you are developing in Eclipse.

1The normal convention is to use three root folders for a software project: *branches*, *tags*, and *trunk*.

Branches are for experiments. Tags normally identify older, already released versions of the software.

However, most of the time, you'll want to work on the main branch, which is called trunk.

**Purpose**

The purpose of this exercise is to give you a fundamental understanding of how a version control system works in general, some of the things that are possible with SVN and how it is integrated in the Eclipse IDE.

**Tasks**

To demonstrate some problems that may occur using version control and the features of SVN, you will in this lab work with three different workspaces in Eclipse, of which two

simultaneously. To make life a bit easier we will in the later part of the exercise pretend

that we are monitoring two different developers, named Peter and Sally, who are working for the same company. This company has recently bought the project you created in Lab 1, the *HelloWorld*. However, the product manager of the company would like to make some changes to *HelloWorld* and has put both Peter and Sally to work on the project. Of course, since Peter and Sally do not exist you will have to carry out the tasks for them. When you have finished the tasks you should report to the lab assistant and give an oral explanation and a demonstration of what you have done and what you learned from it. Be prepared to answer questions about details in your code.

**Note: Using several workspaces can be a bit confusing. Therefore it is very important that you read what you should do, before you do it! In the worst case you will have to re-do the lab from start if you just miss a single instruction.**

**Task 1. Setting up a SVN repository and importing a project**

In this first task you will create a SVN repository and import the *HelloWorld* project to the repository. However, since this is something that developers usually don’t have to bother about, we will not pay it that much attention, but only go through the task.

1) First of all you must add Subversion to your account.

2) You can now create a SVN repository

3) Now, start Eclipse and use the workspace containing the *HelloWorld* project. To

import the project to the SVN repository right click on the *HelloWorld* project, and

select *Team->Share Project*. Select SVN and click *Next*. Enter the location url for

the repository:

*file:///dir/repository*

and click *Next* again. Select *Use specified folder name* and type *HelloWorld/trunk*

in the text field and click *Next*. Use the suggested commit comment: *Initial import*

and click *Finish.* DO NOT check the box *Save author name*, but click *OK* in the

pop-up window2. Choose *Select All* to commit all files to SVN version control in

the appearing window and click *OK*. If you have performed this correctly the last

line in the *Console* window should read *Committed revision 3*.

4) Close Eclipse

**Task 2. Checking out working copies to two different workspaces**

In this task we will set up two new workspaces, one for Peter and one for Sally, and check out one working copy from the SVN repository to each of the workspaces. You will from now on have two Eclipse sessions running at the same time. Since this can be somewhat confusing, a good idea to use the *Workspace Switcher* in the lower right corner of the *JavaGnome* desktop and use a separate desktop for each Eclipse session.

5) Start Eclipse again but create a new workspace called *wsp\_peter*.

6) To check out Peter’s working copy of *HelloWorld*, right click in the *Package*

*Explorer* and select *New->Project*. Extend *SVN* and select *Checkout Projects from*

*SVN* and click *Next*.

7) Select *Create a new repository location,* click *Next.* (Note: Apparently there is a

bug in Eclipse here. If it is not possible to click *Next*, first click on the other option

2 If you do not get a pop-up window asking for this, it will not be possible to specifically change the author

name when committing files. Instead, you must make sure to include the name of the fictive author (i.e.,

Peter or Sally) in the comment itself.

*Use existing repository location* and then click on *Create a new repository location*

again. You should now be able to click *Next*.)

8) Type in the url for the *HelloWorld* project contained in the SVN repository:

[*file:///dir/repository/HelloWorld*](file:///C:\dir\repository\HelloWorld)end click *Next*.

9) Select the folder called *trunk* and click *Next*.

10) Since the *HelloWorld* project contains a .project file your only option is to check

out the project directly in the workspace. Click *Next* and then *Finish*.

11) Extend the *HelloWorld* project in the *Package Explorer*. What revision of the file

*HelloWorldFrame.java* do you have?

12) Now, do Task 2 again but check out a working copy for Sally. Remember that you

should not close the current Eclipse session! Use the Workspace Switcher on the

desktop!

**Task 3. Modify-Commit-Conflict that can be automatically resolved**

Both Peter and Sally will now begin their modifications of *HelloWorld*. Their first

assignment is to add some comments in the file *HelloWorldFrame.java*. Although, it is the only file in the project, the CEO still wants to clarify that it is the main file.

13) Sally starts by adding a comment at the top of the file *HelloWorldFrame.java*,

clarifying that this is the main file;

*//This is the main file.*

Be sure that you are in Sally’s workspace, add this comment and save the file!

14) Sally then checks if there has been a new revision to the repository and performs an

update. To do this select *Team->Update*. Has anything happened since the latest revision?

15) Switch to Peter’s workspace.

16) Peter, who is lagging behind somewhat, adds a comment right above the *main*

method;

*//This is the main method.*

Add this to his working copy of the file and save it. Peter also checks for new

revisions. Do this! Has anything happened? Why/why not?

17) Switch to Sally’s workspace.

18) Sally now feels pretty satisfied and decides to commit her working copy to the

repository, do this for her! Remember to add a commit comment and be sure to

write *Sally* in the *Author Name* text field. What revision is Sally working with after

the commit?

19) Switch to Peter’s workspace.

20) Peter also feels it is time to make a commit. Therefore commit the file to the

repository. Remember to enter a commit comment. What happened? Help Peter

solve the problem and commit again!

21)Would it have been possible to solve the problem just as easily if Peter had added a

comment at the top of the file as well?

22)Which is the latest version of the file after Peter successfully has committed the

file?

**Task 4. Modify-Commit-Conflict that must be manually resolved**

The product manager has also asked Peter to change the color of the “Hello World”-

message from blue to green when pressing the button. However, due to a misunderstanding Sally thought she was assigned to do this, she also thought the color should be red instead of blue.

23) Switch to Sally’s workspace.

24)Make an update for Sally so that she has the latest revision. Locate the section in

the file containing the information of the color, and enter the code for red. Save the

file and run the application to make sure the text turns red instead of blue. Then make a commit with a suitable commit comment.

25) Switch to Peter’s workspace.

26)Without first making an update, help Peter change the color to green. Save the file

and test the application! When you are done, try to commit the file. If it does not work, you must update it! What happens? Can this problem be solved like before?

Why/why not?

27) Peter has now noticed that he is not the only one working on the file. Help him to

detect how the history of the file has changed by selecting *Team->Show in*

*Resource History*.

28) After confirming Sally’s mistake from the *Resource History*, help Peter fix the

problem, by deleting code appropriate parts of the code. Make sure the “Hello

World”-message now turns green. Save the file and confirm the manual change by

selecting *Team->Mark Resolved*. Finally, commit the file.

**Task 5. Roll-back to an earlier revision**

The CEO of the company, who has been under a lot of stress lately, suddenly realizes that it is probably best to go with the blue color after all. The product manager is notified about this and asks Sally to fix it.

29) Switch to Sally’s workspace and make sure she has the latest version.

30) Sally, being a practical developer, thinks the best way to fix this is to make a so

called *roll-back* to an earlier revision instead of changing the code. Help her with

this by using *Compare With->Revision*. Right-click on a suitable revision and select

*Get contents* and save. Finally, make a new commit so that the latest revision of the

application is the one with the blue “Hello World”-message.

31)Which is the final revision of the file?

References and resources:

*Version control with Subversion*

http://svnbook.red-bean.com/

*Tigris.org, Open Source, Subversion*

http://subversion.tigris.org/

*Introducing Subversion*

http://www-128.ibm.com/developerworks/java/library/j-subversion/index.html