Advantages Of SubVersion Over CVS When Working With XP Projects

Marcus Eliasson, D02 (d02me@efd.lth.se)

22nd February 2005

Abstract

Version control tools are today used in many softwareprojects. One popular software solution used for version control is the Concurrent Versions System (CVS). A second year course, given at Lund Institute of Technology, uses this tool when teaching the ExtremeProgramming methodology, along with java and Eclipse.

The aim of this paper is to investigate what advantages, if any, an XP team would have if CVS was replaced by SubVersion. The second part of this paper contains two HOW-TOs, one for setting up a SubVersion repoitory and one for using the Eclipse plugin Subclipse, used to integrate SubVersion version control.

Contents

1 Introduction

1.1 Background

Using Integrated Development Environments (IDEs), such as Eclipse[1] that supports the agile methods described in Extreme Programming [chro 03] (XP) makes learning the methodology less of a hassle. One of the more powerful tools in Eclipse, seen from an XP perspective, is automatic refactorings. This makes renaming classes, methods, extracting interfaces etc. very easy, which in turn makes the students less hesitant to refactor.

While using these tools in Eclipse are very easy, we loose some of this functionality if we do not have a versioning tool that can handle this type of refactorings, whitch means they won't be performed as frequent. While the Concurrent Versions System (CVS) is a very powerful tool, some of the refactorings that are used while working with XP, are not supported. Among these are (only a selection):

- *Renaming of classes*: When renaming a class, the file containing the program code also has to be renamed. Using a good naming strategy is essential for keeping the project manageble, and refactoring of classnames is a powerful tool.
- *Moving classes*: Yet other refactoring is moving classes from one package to another, where they might be more suitable.

Besides the above, there are other useful features that can be used together with Xp projects, as well as other projects. Some of these are listed in Section 2.2.

These refactorings are used in other methodologies as well, but not to the same extent as in more agile methods, as XP. This means that this report is not aplicable for XP projects using SVN, but also other projects.

1.2 Report structure

This report is structured as follows:

- Section 2: Identify differances between CVS and SVN.
- Section 3: Identify points where using SVN would be an asset to software development teams using the XP methodology.
- Section 4: A short introduction to Subclipse[2].
- Section 5: In this section, migration from an existing software solution, in this case CVS, to SVN is discussed. This section also contains a small HOW-TO for migrating, discussing both the serverside and the clientside.
- Section 6: A small group of XP students were asked if they, after a short introductin to SVN, thought they would benifit from using it during the project. Here is a short summery of this discussion.

[1] A popular opensource IDE with many useful features.

[2] A plugin that integrates SVN usage into Eclipse, much like the current integration of CVS.

2 Concurrent Versions System and SubVersion

2.1 History of SVN

The development of SVN started in June 2000. It's goal was to develop a versioning system that was similar to CVS, but which removed some of the obvious drawbacks of CVS. With funding from CollabNet and RedHat the SubVersion project switched from using CVS as versioning tool to its own SubVersion on 31st August 2001.

Today SVN is gaining supporters, many of them migrating to get away from the drawbacks of CVS.

2.2 Comparison of CVS and SVN

As mentioned above, SVN hasn't the goal to re-invent or revolutionize version control. Therefore the two products are alike, SVN has most features that CVS has, with some extra added to increase it's functionality. To list a few:

- *Metadata, renames/moves and directories* are versioned. This is, as stated on the SVN homepage, one of the most common complaints against CVS. Moving/renaming is discussed in Section 3.2.
- *Using Apache2 as server* by only adding two modules and specifying a repository location in the filesystem makes the server both easy to install, and repositories viewable directly through a normal webbrowser without additional tools. This feature is discussed in Section 5.2.1.
- *Optimizations* like branching in constant time and sending only diffs when updating and committing are some of the implemented features.
- *Commits depend on version.* This feature can be used in several ways, as discussed in Section 3.1.
- *Binary files* are supported. This means that binary files can be version controlled in an efficient was using SVN's internal binary diff methods.

After listing features which SVN has, aren't there any that CVS has and that SVN is missing? Well I am sure there is, but I haven't found any significant feature that is implemented in CVS that is not in SVN.

3 Using SVN with XP

In this section I will investigate how, and with what benifits, SVN together with the XP methodology. Below I will discuss real/larger XP projects. Since I have no prior experiance of this, I will use assumptions as grounds for the arguments I make. Each feature will be divided into five sections:

- A brief explination of the feature.
- If/How it is supported by CVS.
- What benifits it has from the XP course perspective.

- A simple use case, both when using CVS and SVN.
- What benifits it would have if used in a real XP project.

3.1 One version per commit and modification lists

Having one version number per commit means that all files that are commited together are given the same version number. This version number is an unique positive integer and incremented by one for every commit made to the repository. When using Eclipse/Subclipse and when browsing the resource history of a file, the user can, when selecting a version, see what other files were changed during the commit of this version (See Figure 2). These features are related to *Moving/renaming files.*

3.1.1 Version numbers in CVS

In CVS, every commit that effects a file increments the version of the file.

3.1.2 Using SVN version numbers in the XP course

There are at least three different ways this feature can be used together with XP. One very tempting usage of the feature is to commit files on a story basis, effectivly tagging every story i.e the files modified as a result of the implementation, with a version number. This would be nice from a tracability point of view, but would violate the integrate continously practice if the stories are large. The second one is to split the story into smaller task, and commit one task at a time, thereby tagging every task with a version number.

The third and maybe the most effective is when doing larger refactorings, effecting a large number of files, the user gets a general view of which files were effected by the refactoring. This gives the students a good overview of, both the effects of refactorings, and how many larger refactorings have been made.

3.1.3 Larger XP projects

When using this feature in larger XP projects, it improves tracability and simplifies tracking alot. The tracker has information about what task were committed under which version numbers, and thereby has information about which features were implmented and/or why changes were made. The same approach can be used when bugfixing, for example Bug #144 has commit number 1667. This gives detailed information about why changes were made as well as a list of what files where modified during the commit of this version. Summarising what was done as commitcomments can often be hard, with this approach the commit comment only has to contain, for example, the bug id. If the same approach was used together with CVS it would result in, if multiple files were modified as a result of the fix, that the version numbers would have been spread out.

3.1.4 Scenarios

A scenario when CVS is used: When fixing the bug or when implementing a story/task, files One.java, Two.java and Three.java were modified. As a result, the files were commited under the following versions: 2.24, 1.13 and 5.2. How

would a tracker go about summerizing this information ? Either each commit has the bug/story/task number as commit comment, but there is still no obvious connection between the files. When looking at the resource history for One.java, you will be able to see that it was changed from version 2.24 to version 2.25 and why. But there will be nothing linking this to the changes made in Two.java and Three.java. Another solution would be to have the tracker write down the bug/story/task number, what files were modified and what versions the changes resulted in. Yet another solution might be to tag stories/bugfixes as versions under CVS.

A scenario when SVN is used: The same three files are modified. When browsing the resource history for One.java, we note that the files was changed from version 24 to version 78 as a result of the bugfix/story/task. By simply klicking the version number, the user can see what other files were changed during this commit, i.e what other files where modified as a result of fixing the bux/implementing the sotry/task. This gives a good picture of what was done and why, at a very low cost of the user.

In both scenarios above, bug/story/task can be substituted with refactoring.

3.2 Moving/renaming files

When using XP, developers are encouraged to refactor frequently. The refactorings can be on a small scale, maybe extracting a method, or they can be more global, effecting a large number of files. As meantioned above, in Section 3.1, using SVN improves tracability when doing global refactorings. A part of doing both large and smaller refactorings can be renaming and/or moving files. When using XP, or for that matter any methodology, developers should use a good naming strategy. Classes can start out doing one thing and as the project moves a long and code is added, they end up doing something else. When this occurs the classes have to be renamed, and therefore the files have to be renamed. Or maybe they end up as something that does not even belong in the same package anymore and have to be moved. Refactoring these kinds of problems keeps the project easy to maintain and new developers can easyly be integrated into the project.

When using SVN for version control these refactorings are no problem. Files can be renamed, moved and then new files can be created with the old ones names. In the next Section we will see that this is not the case for CVS.

3.2.1 Moving/renaming files with CVS

When using CVS, the scenario is treated as one file being deleted, and another one beeing added to the repository. This results is loss of history for the renamed/moved file. Since CVS uses filenames as a base for it's version control, a certain filename has a version and a history. After moving/renaming a file, it's history will be, as menshoned above, lost. But this will also have another unwanted effect. When commiting a new file, under the same path and with the same filename as the old moved/renamed file, the new file will "take over" the old file's version and history.

3.2.2 The XP course

During the XP course, the students are encouraged to refactor, even refactor mercilessly, but CVS doesn't support such a trivial refactoring as moving/renaming files. To what extent this refactoring is used is uncertain. Another concern is in what extent the students use the available trace infomation and if loss of it would have any negative effect at all.

3.2.3 Larger XP projects

In larger projects trace information is more imortant than for example the XP project in our course. Therefore the impact of lost history will be greater. This might result in these kinds of refactorings never being done or developers overdesigning problems to avoid having to do the refactorings.

3.2.4 Scenarios

The scenatios for moving and renaming files with CVS and SVN are clear, both benifits/drawback when using the two.

4 Subclipse and Eclipse

The graphical tool that is used together with Eclipse and SVN is called Subclipse. It adds functionality which is similar to that of the CVS interface in Eclipse. Except for it saying SVN instead of CVS and the repository locations starting with HTTP, the most noticable differance is that the submenu "Team" (Figure 1 contains very descriptive icons, making SVN features easier to find. This is a surprisingly nice feature.

Figure 1: The "Team" menu for a specific file in an SVN version controlled project

When comparing files and synchronizing the workspace with the repository, the graphical views are the same as when using CVS. Features that are available for the CVS interface, like annotations, commit/update, branch/merge , resource history, revert and synchronization are also available in the SVN interface.

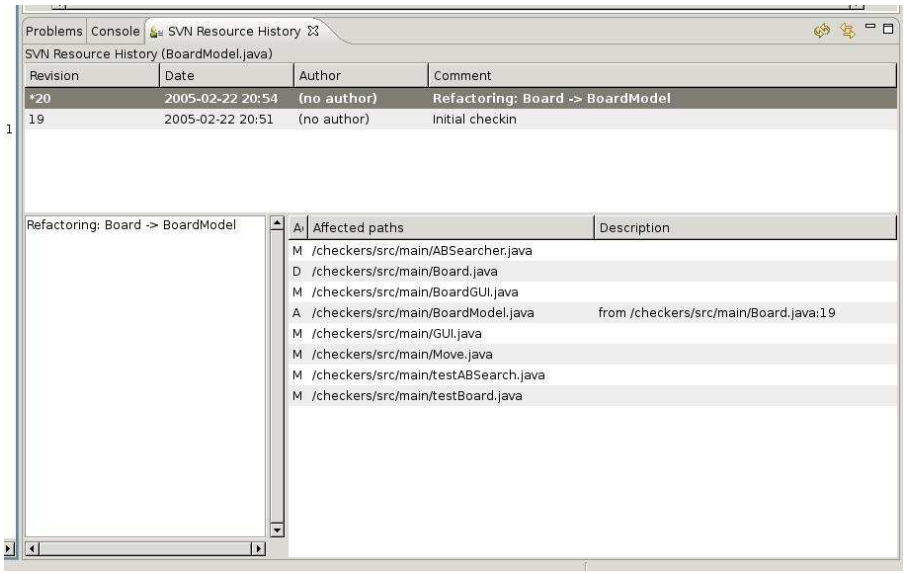

Figure 2: View of the resource history in Eclipse/Subclipse

In Figure 2 the list of changed files discussed in Section 3.1 can be seen. In this example the Board.java file was renamed to BoardModel.java. In the list we can see what other files was effected by this refactoring, as well as the commit comment.

5 Migrating from CVS to SVN

5.1 The cost of migrating

Like LTH, many workplaces already have a version control system in use, in this case CVS. When discussing migrating from CVS to SVN we have to take into account the cost as well as the benifits. There are two main costs to concider , the cost of actualy installing, configurating and maintaining the server side, as well as the cost for the users at the client side. These two are discussed below.

- *Server side*: Using tools to convert an existing CVS repository into an SVN repository is very easy. Both files and user accounts are converted. Maintaining the SVN repository, adding users and additional repositories, is roughly the same as for CVS. The same basic functions, as adding users and repositories, had to be performed. A more detailed description of the options for the repository are discussed in Section 5.2.1.

- *Client side*: There are at leat two scenarios here. The user has prior knowledge of version control, in this case CVS, and the user has no prior knowledge. Starting from scratch and learning SVN or CVS, the cost is the same. The two systems has the same basic functions, and the extra features implmented in SVN can be learn at a very low cost. The same goes for users that has prior knowledge.

5.2 Migration HOW-TO

The migration from CVS to SVN is easy, if the proper tools are used. Some easy steps are listed below.

5.2.1 Serverside

The repository

Tools are available for automatically converting an existing CVS repository to a SVN repository. One example is cvs2svn. It has a number of different types of conversion, that move different amounts of data over to the new SVN repository. They range from just moving the files in HEAD to the SVN repository and giving them version 1, without any history, to moving branches, verions and history along with all the files contained in the repository.

svnserv vs apache2 modules

When accessing the repository two different methods can be used. They are listed below together with som advantages of each approach.

- *svnserv* together with ssh used as a standalone daemon. The advantages of this approach is a smaller, easier to set up daemon with no additional software needed. Access to the repository is either controlled by existing system accounts (ssh) or by a private users file.
- *apache2* using SVN modules to access the repository through HTTP, using the WebDAV/DeltaV protocol. This sets up a slightly slower server, but it has a number of advantages. The repository can be viewed directly, without any additional tools. However, this is a very basic interface. To get added features 3rd party tools, like ViewCVS, can be used. Access to the repository is controlled by the apache2 server, which gives the admin a lot of different options. Here are some of them: HTTP(S) basic auth, X.509 certificates, LDAP, NTLM, or any other mechanism that is supported by the apache httpd server.

If users require both of the options, they can run in parallel and access the same repositories without locking the otherone out. If the SVN repositories is to be used together with Subclipse, the apache2 option has to be setup, since it is the only one supported.

5.2.2 Clientside

Subclipse

In order to use the Subclipse plugin for Eclipse on Linux/Unix systems, the SVN binaries also has to be installed. When using the plugin under Windows,

this is not needed.

As for installing Subclipse, it is just another plugin, and is easily added and updated by adding the pluginsite hosted by `subclipse.tigris.org`.

svnant

When using Ant together with CVS, there is a set of commands that can be used to control versioning. Commands for updating the local workspace and committing files are some of them. When using SVN, there is a similar set of commands that can be used. To use them svnant has to be installed at the client side. Roughly the same features can be used.

One of the ways this can be used in projects, and especially in the XP course, is automation of the release process. It is easy to add a command in the Ant script that ,for example, creates a branch with the release name and date.

6 XP student feedback

After a short introduction the students were asked if they felt that using SVN in the project would be an asset. The following points were made:

- Being able to version control binary files was a feature that the students wanted. For example version controlling JAR files.
- Being able to version control directories was also a feature that the students wanted. A couple of times empty directories had been checked in, but without success. Students had to put files in the directories to get them into the CVS.
- Even though they haden't done any refactorings that renamed/moved files, they liked the idea of being able to do it.

7 Conclusion

After investigating SVN, and it's new features, I found that it is a good alternative to CVS. The features that are added, which aide the XP methodology, can be used with success.

As for using it as a replacement to CVS during the XP course, I'm more sceptic. It does have many good features, and I see nothing negative with using it instead, since no features are lost. But if the features are worth the extra cost, even though it is low, is unsure.

Another concern is in what extent the students use the available trace infomation. Would improved tracability when moving files and refactoring be something the students could have use for, or would it simply be ignored. There is a possiblility that the project lifespan is too short for trace information to be used to any larger extent.

8 References

- Eclipse Project Homepage — `www.eclipse.org`
- SubVersion Project Homepage — `subversion.tigris.org`
- CVS Project Homepage — `www.cvshome.org`
- Subclipse Project Homepage — `subclipse.tigris.org`
- cvs2svn Project Homepage — `cvs2svn.tigris.org`
- CollabNet Homepage — `www.collab.net`
- Apache Project Homepage — `www.apache.org`
- chro03 — Extreme Programming Pocket Guide, chromatic, O'Reilly 2003 ISBN: 0-596-00485-0
- Version Control With SubVersion — `http://svnbook.red-bean.com/`