Introducing Perforce Visual Client as version control tool in an XP environment

Axelsson, Oscar
dat11oax@student.lu.se

Olsson, Daniel
atf10dol@student.lu.se

March 5, 2015

Abstract

In a programming course at LTH, Subversion is used. For inexperienced developers, features like merging, branching or checking differences between revisions can be overwhelming in an advanced version control system. The aim is to investigate if there is a better alternative to SVN. The investigation showed that developers using Perforce rated their expertise higher in being able to make diffs, handling merge tracking and branching, compared to developers that used SVN. As of today, using version control in a command line environment is more popular than using a visual client, but with the uprising of tools like Perforce Visual Client, this may change.

1 Keywords

Version control tools, branching, diff, merging, Perforce, student project, coaching.

2 Introduction

In the PVG course, the well known version control tool Subversion is used. For someone unfamiliar to version control systems, one might be overwhelmed by all of its functionality. Is there a better software for people who are scared by terminal commands and complex version controlling? The authors have gained some previous experience with Perforce from another course at LTH. From that experience the conclusions conducted that it could be a good and easy introduction for students with no experience of version control. The study conducted research on how a team would adapt to the use of Perforce.

When we later in the study mention Perforce, we mean the Visual Perforce client, P4V.

3 Background

3.1 Perforce background

Perforce is a version control system developed by Perforce Software Inc.

Perforce is, according to the developers themselves, a fast, scalable, secure and reliable version management tool and it fits both large and small teams.[3]

The system manages control and documents different versions, named by revision numbers. Every new version is documented by a number, the author, which files it affect, what changes and a commit message where the developer could describe the change that was made.

The synchronization model used by Perforce is the long transaction model [4]. Simply put, it prevents a developer to submit a file which is not up to date in relation to the depot. However, Perforce does not support the strict long transaction model. There can be dependencies between files which would require a developer to have a certain configuration locally before submitting it to the depot. This is only prevented with the strict long transaction model.

The principles and terminology is quite similar to other version control tools. What is called often called **commit** is called **Submit**, **add** is **Mark For Add**, **remove** is **Mark for Delete** and **pull** is **Get Latest Revision**. When you are to use a file in Perforce, you have to **check out** the file. The default setting is that a file is locked when it is not checked out.

In order to ease the usage of Perforce, it can be set up in different ways and can work for many different purposes. A versioning server can be reached through the internet by many users, as seen to the left in Figure 1. It could also be set up with the Git Fusion tool, as seen in the middle, by enabling Git users to collaborate with Perforce. The last suggestions is by simply be in a sandbox mode where one can bring everything home without having an internet connection to the server while developing, as seen to the right.

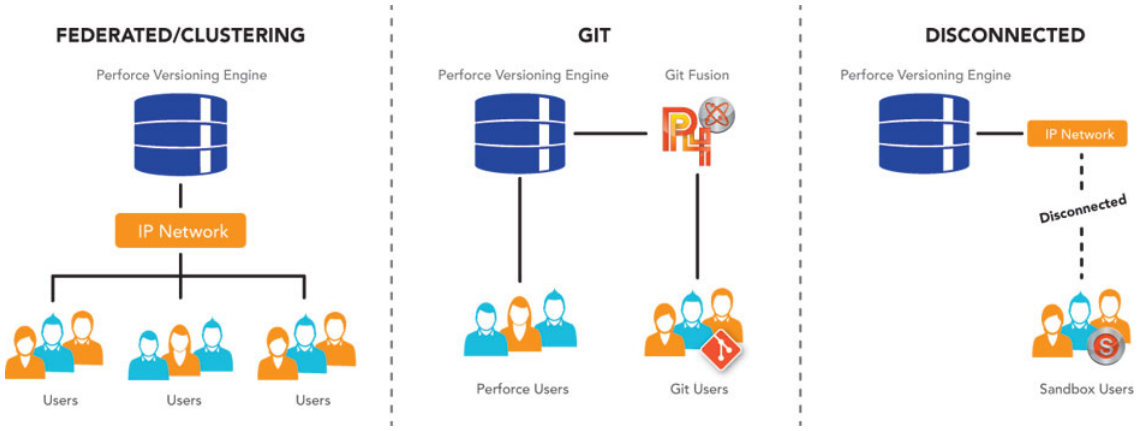

Figure 1: The figure shows three different ways of setting up a Perforce depot and how to use them. Source: Perforce Webpage[12]

Perforce can, as many other VCS, be run from the terminal. However, we have focused our in-depth study on the visual Perforce client, P4V, which is entirely run from a stand-alone program. This has been our chosen program to investigate.

3.2 Project Settings

Our study was performed during the course EDA260 "Software Development in Teams – Project". The course aims to give students practical experience with software development in teams and the method eXtreme Programming. The students are divided into team of 12-14 people. The team's goal is to develop a program that registers and sorts time for any kind of race. For our development the race is an Enduro race. The team gathers every Monday from 8-17 for the development phase, and on Wednesday for a two hour planning meeting. The course ends with a live competition where the software is tested by judges of a race.

4 Problem statement

What are the struggles of using Perforce and what functionality do the students find easy to us? Is it better than subversion and would it be appropriate for the PVG course to change their default tool? What things are better or worse with Perforce compared to subversion, git or any other version control tool? Seeing as branching is hardly used in the PVG projects, can Perforce with its everyday looking user interface being the comfort that the students need to want to try out branching? Merging is a common problem in the course, can Perforce help the team feel more secure with merging.

By the end of the project we hope that the team is satisfied with the result and product that they have produced. With the help of Perforce, the students have improved their expertise of understanding version control systems in general and feel comfortable using Perforce. We also hope that the group feels that Perforce has been an asset rather than a burden.

Hopefully, the graphical user interface that Perforce brings attracts the students more than scaring them away.

5 Method

Before the first lab, we set up Perforce in a way we found appropriate. We had to make a decision whether we wanted to host the repository "locally" on a server or on LTH or at a network attacked storage[6], such as assembla[7]. It was decided that we would use an LTH hosted server.

At the start of the PVG course, we introduced Perforce to the team and they received a short introduction/manual, see Appendix A, of Perforce that we had authored. In order for them to start, we supervised and helped them if they found anything difficult. We found it important to remember that people learn from mistakes and therefore, we decided to let the team make all mistakes they had to make in order to learn from them. We did document the mistakes and the team did document their experiences with the version control tool at the end of each session. It was beneficial for themselves as they were forced to go through and reflect on what they had learned. It was also helpful for us because we could analyze the result. In an earlier course, the team had taken a CVS lab and they gave us a brief analysis of their impressions and thoughts regarding CVS compared to Perforce.

As we wanted to spread knowledge and expertise within the group, we decided that switching partners was going to be done continuously. We hoped that the students would have learned a lot within a couple of weeks. This would result in the team being self going within a couple of weeks.

We discussed whether we would investigate any of Perforce's scripting possibilities but decided that we had enough on our plate. An article on Perforce scripting can be found at [8].

We compared our results within the group with the result of the rest of the teams that used subversion or git. We wanted to compare our branching strategy with other group's different branching strategies but decided that there was no time for it. Comparison regarding checking differences between files, branching and merging were the three main metrics we focused on. Other impressions were also asked about from a more broad perspective.

Since the team was using Eclipse, we discussed whether we would use an integrated Perforce tool, such as P4Eclipse [5], as opposed to the native Perforce client. Due to the fact that our whole in-depth study focused on the Perforce Visual Client, we felt that it would defeat the whole purpose of the in-depth study if we used the integrated Eclipse plug-in.

We wanted to force the team members to learn by themselves and spread that knowledge early, but at the same time, we wanted to be there to help them when they need help. The positive thing about us not knowing too much about Perforce was that we could have a discussion with the team members where everyone really felt that both the students and we, the coaches, were on the same level and could learn from each other.

At the first meeting of our team we made a short introduction of Perforce and distributed a manual of the program that describes how it is used. The manual that we had written assumed that the students had previous experience with version control and especially CVS. The team studied the manual and by

the first development session the team got its first hands-on experiences with Perforce. The first step was to set up Perforce and create a user. Also to connect to our server and retrieve the depot that consisted of a skeleton that the team had created and the coaches implemented. When that step was completed the team was up and running. At the end of the development session that day the team had time to reflect on their first day. The also reflected on how perforce was and what their struggles where. All this was collected via a survey conducted by the coaches.

At the third planning meeting a decision was made by the team to start using branching. We introduced branching for the team with some guidance from [9]. Patterns were discussed with inspiration from Brad Appleton's branching patterns [11]. A decision was taken by the team that it was necessary to use a simple branching model because problems were encountered when everyone was submitting to the same branch. The branching strategy was visualized with help of a version tree diagram, as seen in Figure 2. At the third development session the team started to use branching for the first time.

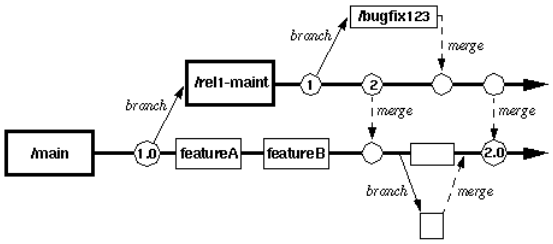

Figure 5: A sample version tree diagram for a project or system

Figure 2: The figure shows a version tree diagram which is used to describe a branching strategy. Source: Brad Appleton's Webpage[11]

At the fifth development session the team encountered the first big problem. With their own initiative they started using rollback. The problem was not solved during the session but at the sixth and final session the program was up and running again.

6 Results

6.1 Merging

Perforce offers a merging capabilities as shown in Figure 3

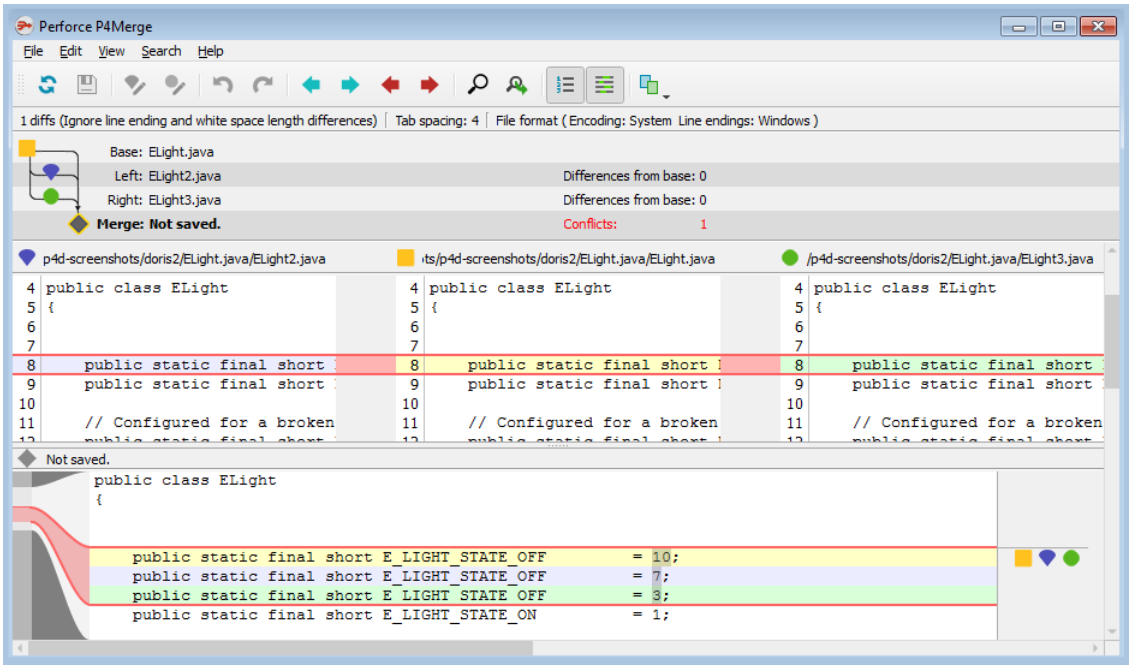

Figure 3: The figure shows the visual merging tool of Perforce. Source: Perforce Website Merge[13].

The evaluated knowledge in merging and merge tracking of the students is displayed in Figure 4.

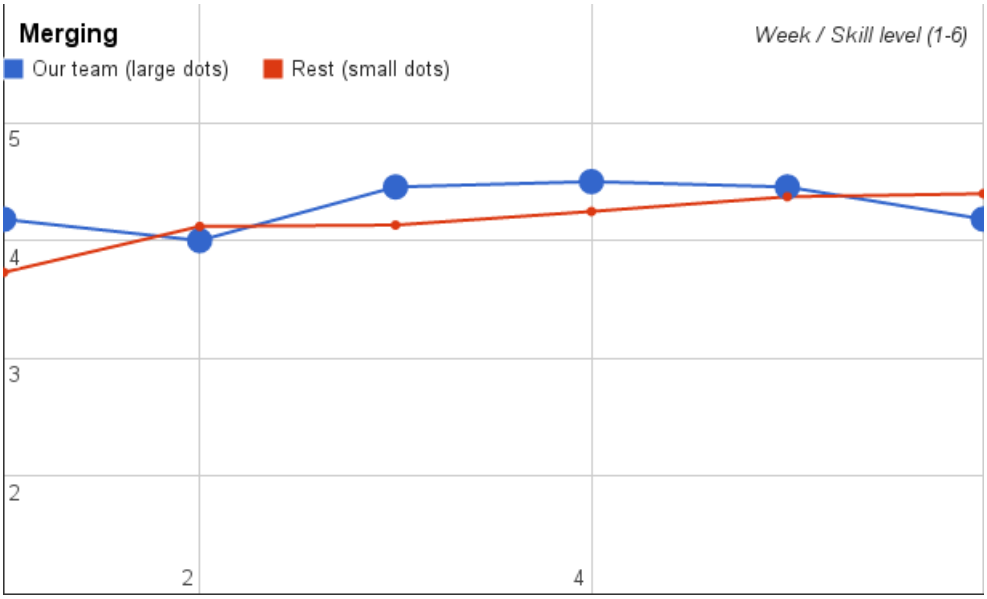

Figure 4: The figure shows the knowledge in merging and merge tracking of the students of our team compared to the rest of the teams.

6.2 Branching

We had a main branch called ”release”, a development branch called ”dev” and one individual branch for each functionality which were created as the teams wanted. It was highly appreciated by the team.

The evaluated knowledge in branching of the students is displayed in Figure 5.

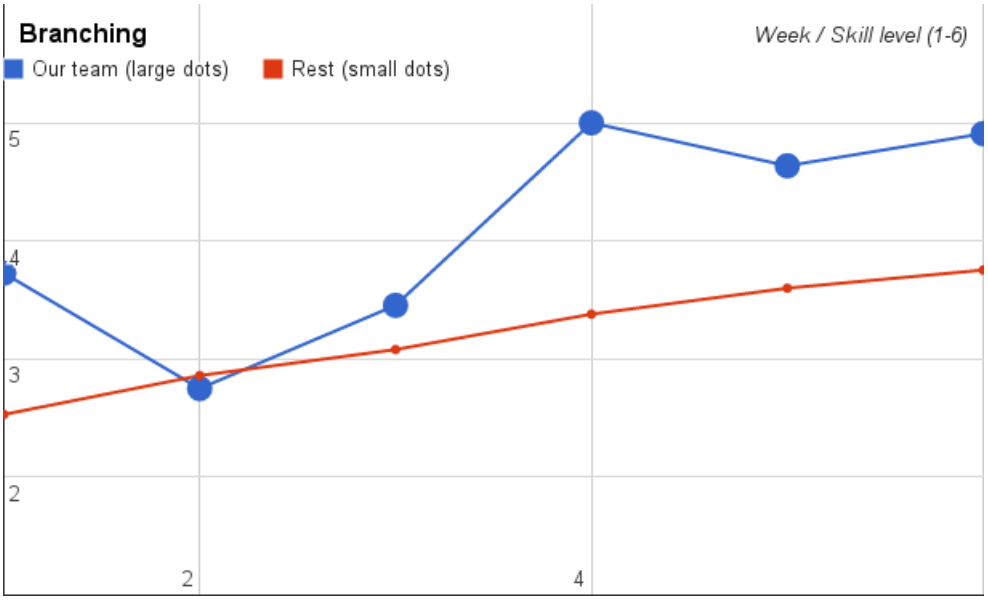

Figure 5: The figure shows the knowledge in branching of the students of our team compared to the rest of the teams.

6.3 Diff

The evaluated knowledge in being able to compare differences between files and different revisions of the same file by the students is displayed in Figure 6.

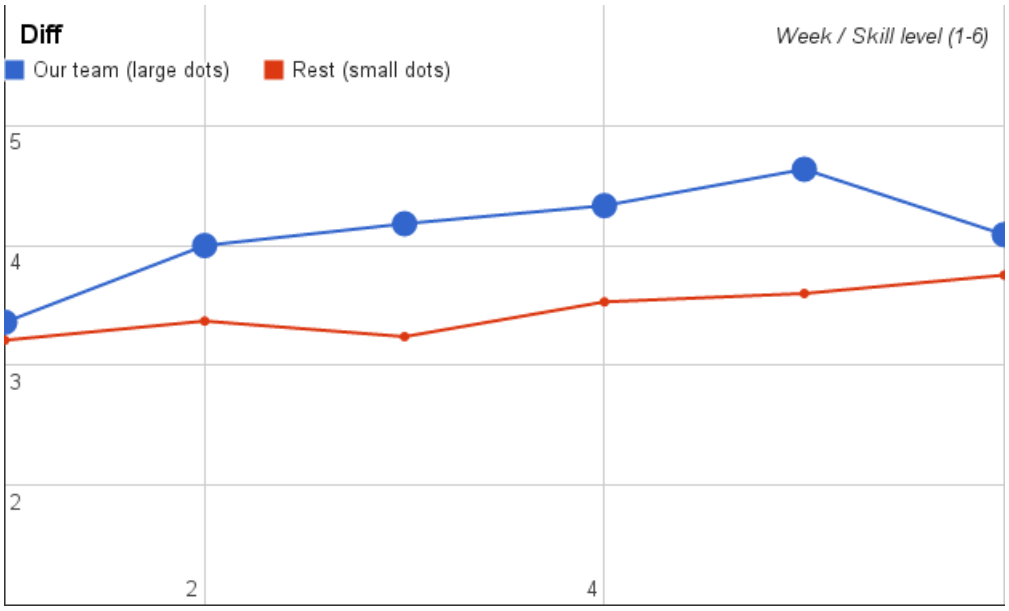

Figure 6: The figure shows the knowledge in being able to compare differences between files and different revisions of the same file by the students of our team compared to the rest of the teams.

7 Discussion

At the first development session Perforce was setup by the team. We had before the session estimated that it would take up to several hours to set up and get the team running because of the inexperience by the team. But to our surprise the whole team was up and running within the first half hour. The team thought that the manual and the set-up instructions which was given to them before the session was to very good use, giving them the support they needed. The development continued and by the end of the day we had over 100 submits from the team. Some of the team got expressions from the merge tool but not all. They who tried it had a pleasant experience with it. The merge tool in Perforce differs a lot from the one the team have experienced in CVS. In Perforce all is visual for the developer to see. Your code is compared with the two latest versions i the depot, from that the developer can select what lines of code should be merged in and not. The team found this to be a great asset and felt no fear or anxiety to be faced with merge conflicts. One of the problems we discovered with Perforce from the first development session was that when a user created a Perforce account it was automatically looked to that host/computer where the account was created. This was later solved by changing the settings for the users.

In week two, there were problems with red code in the depot and tests were failing. The problem with Perforce not having support for strict long transactions was solved by simply telling the students to retrieve the whole depot when getting the latest revision and when they were submitting, they were told to submit all of their files and not just a single one. The problem was

thought to be a trivial one but turned out to cause quite the mess in the depot. If Perforce had support for strict long transactions, we would believe it would make it more secure to use for inexperienced developers.

At the third development session branching was introduced and the team adapted to it very well. They saw the benefits with it early and became more confident and felt more secure with there development. But at the fifth development session a larger problem occurred and the team decided that they needed to rollback the program. This created some issues for the team because they where inexperienced by it and we coaches had not given them enough introduction to it. But by the beginning of the sixth development session the team had through a spike gotten better knowledge with it and could then localize the issue. The team also found it easier to localize the problem thanks to P4V:s diff command, which presents it in a easy way, more about this is taken up in the diff session.

The program was finally released and the team was satisfied by the result. By the end of it all the team had a pleasant experience with Perforce. They had some troubles with rollback and branches but that can be the lack of experience and the approach of trial and error instead of gathering knowledge before trying. The team have from this project gained a great experience with version control and we think that Perforce is a great tool for an inexperienced developer. Thanks to the visual aspect of both merging and branching the developer can easily see what has happened.

The reader has to keep in mind that in a study with this few people, the results can show something that could be interpreted wrong and therefore, the conclusion might be misleading.

7.1 Merging

The results tend to point towards the fact that Perforce and git/SVN perform typically the same when it comes to merging different versions of files. We got a lot of good critique regarding merging in Perforce which is not shown in the diagram. It could be that our team has easier to get started with merging on Perforce thanks to the visual client, but when the other teams learn to handle merging on SVN they feel just as comfortable as our team with Perforce.

There were a few people in the team that expressed their love for the merging capabilities of Perforce and thought it was genius. They found it very easy to use because of the interface P4V presents.

7.2 Branching

As mentioned before, it is impossible to say anything for certain with too few people answering these questions. Although, if one is to take anything from this graph, it could be that Perforce, after at least two weeks or three, seemed to give somewhat better overview of branching compared to git/SVN. This difference could also be that we in the third development session introduced branching and the team then got a better understanding of it compared to teams that had not tried it.

An important thing to keep in mind is the following quote from one of the students in our team: "It would be an extreme hassle to execute this whole project without branching.".

The principles of branching were embraced immediately and the team quickly spoke in terms of branching which tells us that it was highly needed for the team. The team felt more confident in their development and felt they had more control and stability of the project. They also felt less stress before a release because the know that their release branch was always working and a release could easily be produced within minutes.

7.3 Diff

The results tell us that checking differences in files was somewhat at the same difficulty in Perforce compared to git/SVN. It is worthy to mention that we in our team has presented and introduced diffing as a function which other teams may not have done.

7.4 Limitations

In a study conducted on few people, one has to realize that the results may vary a lot. The study was performed on developers where most of them have only a small amount of experience using a version control system. Our results and assessments do not necessarily correspond with people who have previous experience using a version control system.

We chose to not approach the Perforce plug-in for Eclipse, P4Eclipse, because we felt that it would defeat the purpose of the whole study which focuses on Perforce Visual Client. Also, we did not dive deep in the scripting possibilities of Perforce because we did not feel like it was relevant for the project. The in-depth study focused on the graphical aspects of a version control system and therefore, a decision was made to ignore testing Perforce functionality via terminal commands.

7.5 Possible weaknesses

There are always drawbacks in a realistic solution. What are the drawbacks of using Perforce in the EDA260 course? We discussed earlier on that a student in second year may feel uncomfortable using version control in the command line, we know from experience that some of us did. If the students only learn Perforce Visual Client, will the missing expertise in using SVN/Git in a terminal window be crucial to have when one has to work at a company? We believe that learning the principles of using a version control tool is more important than learning how to use it via a command line. The students thereby get a good experience from version control and can easier see the benefits of it.

8 Conclusion

The results tell us that the usage of Perforce has made it easier for the developers to maintain their code. This may vary from different groups and the small sample of students may skew the results.

Perforce repeatedly shows us that the visual aspects of the client gives the user a very flat learning curve. It does not have the same effect as many other terminal based tools where the user feels frightened because of a program's potential complexity. This does not mean that the program is the perfect selection

in every situation, but with an inexperienced team of version control, Perforce is a great alternative.

Our expectations were exceeded. We did not think that we would see such ease when the students worked with a completely new version control system.

All in all, Perforce is an intuitive and easy to use tool for version control. We recommend it highly.

References

[1] *"Version Control"* http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1576667 Retrieved 2014-12-21

[2] Laura Wingerd, *"Practical Perforce"*, 2005 http://books.google.se/books?hl=en&lr=&id=mlm61wb2v3MC&oi=fnd&pg=PT7&dq=perforce&ots=IYr5yQPoPy&sig=L7YQsBmV6Agjsw95XXZon1b3NFU&redir_esc=y#v=onepage&q=perforce&f=false

[3] Perforce Platform — Perforce http://www.perforce.com/platform#advantages

[4] Peter H. Feiler, *Configuration management models in commercial environments*, March 1991, Pittsburgh USA.

[5] *"P4Eclipse"* http://www.perforce.com/resources/presentations/developers-corner/developer-takes-new-visual-tools-p4eclipse-v20101

[6] Richard Geiger, *"Perforce with Network Appliance Storage"*, Perforce User Conference 2001 http://www.perforce.com/sites/default/files/network-application-storage-geiger.pdf

[7] *"Assembla"* https://www.assembla.com/repositories/perforce

[8] J. Bowles, S. Vance, *"Scripting with Perforce"* 2005 http://www.perforce.com/sites/default/files/scripting-with-perforce-paper_0.pdf

[9] R. Cowsam, *"Introduction to Branching in Perforce"* http://www.vaccaperna.co.uk/scm/branching.html

[10] Y. Zhang, R. Chang, *"Branch Management and Atomic Merge in a Continuous Integration Environment"* http://www.perforce.com/sites/default/files/emc-white-paper.pdf

[11] Brad Appleton Streamed Lines: Branching Patterns for Parallel Software Development http://www.bradapp.com/acme/branching/

[12] Perforce Webpage http://www.perforce.com/sites/default/files/git-p4-sandbox.jpg

[13] Merging http://www.perforce.com/sites/default/files/perforce-p4-merge.png

9 Appendix A

9.1 Quick guide to Perforce

This quick guide serves as an introduction to the most commonly used functions within Perforce. It is assumed that you have previous knowledge about CVS and that you already have set up your repository and workspace.

9.1.1 Commit

To be able to edit a file you will need to Check out the file. Afterwards when you are finished with your updates you simply Submit in order push your edits to the repository. This may be done on both single and multiple files as well as directories.

9.1.2 Update

When you want to copy the files from the repository to your workspace, you use the command Get Latest Revision. This could be done by clicking the Depot tab and select the folder or file you want to update. There is a possibility to copy older versions to the workspace than the current. To do this the Get Revision... command is used with the corresponding revision of the file.

9.1.3 Add

To add files to the repository you have to perform two tasks, first the file needs to be marked for add (can be found in the context menu), which places the file in a changelist. When the file is marked for add, the file icon is marked by a red plus sign. The then changelist needs to be submitted to the repository, which adds the file to the repository.

9.1.4 Status

The status command in Perforce is compared to CVS a graphical display. The status of the file can be seen on the file with different icons. When you hover over the file a describing dialog appears, describing the status of the file. If the file e.g. needs merge or an update.

9.1.5 Log

The log of a file can be viewed in History where you can see all the Changelists that were created when submitting changes. You can view the submit comments and the version number of the changes. The changelists can also be compared with the diff command.

9.1.6 Diff

The diff function within perforce is a versatile tool which lets you check differences between files within your workspace and the repository. To diff you run the Diff against have revision command and this displays the differences with a P4V diff tool. You can manually set which revisions of the file you want to compare on as well as check the diff on a file between branches.

9.1.7 Branches

Perforce offers a wide support for branching, including the ability to perform the composite model in some sense with the use of labels. Branching release versions will allow you to maintain the previous release with bug fixes and updates while continue development on the next release without the need to mix code. In the context menu you select Branch and then you choose either source files, directories or a label (which is connected to different files or directories) and then select a target path. The files will then be copied into the new target path starting on revision 1 (corresponds to the selected revision as stated when branching).

9.1.8 Support for merge tracking

When handling merges and tracking of them, Perforce offers a large variety of tools. They offer a great overview of the different revision and merges of the files. Revision graphs displaying a visual representation of the revision history of the file can be generated in Perforce. It also displays the individual commit messages for each commit along with the time, author and revision number. The changelist of the commit is also displayable for the different revisions.

9.1.9 Useful links when learning more about Perforce

http://www.perforce.com/perforce/r14.1/manuals/intro/intro.pdf
http://www.perforce.com/perforce/r14.1/manuals/p4v-gs/p4v-gs.pdf

9.2 User manual

9.2.1 Hur hämtar jag Perforce till min hemkatalog?

Skriv dessa tre rader i terminalen.
cp -R /usr/local/cs/EDAN10/Perforce ~
cd ~
chmod -R 755 Perforce

9.2.2 Hur startar jag Perforce?

cd ~
./Perforce/bin/p4v&

9.2.3 Hur skapar jag en användare?

I **Server** skriver du in: vm57.cs.lth.se:1666
Till höger om **User** trycker du på **New** och fyller i uppgifterna.

9.2.4 Hur loggar jag in?

Server: **vm57.cs.lth.se:1666**
User: Ditt användarnamn
Tryck **OK!**

9.2.5 Ok, jag är inloggad. Now what?

Tryck på **Connection** i menyn. Välj **New Workspace.**
Tryck på **Browse** till höger om **Workspace root:** och välj en mapp som du vill använda som din lokala kopia av **depot** (dvs ditt **workspace**).
Tryck på **OK.** Välj **Depot**-fliken om den inte är vald och tryck på **Get Latest Revision**-knappen långt upp till vänster för att hämta det som finns på **depot.**
Öppna Eclipse och tryck på **File, Import, General, Existing Project into Workspace**. Välj samma root directory som **Workspace root** och importera projektet Enduro.

9.2.6 Hjäääälp hjäääälp, det fungerar inte!!

Skicka ett mail till **atf10dol@student.lu.se** eller **dat11oax@student.lu.se** så kommer vi inom 5 minuter. Garanterat.