Evaluation of the Perforce Source Code Management Tool used in Agile Software Development

Morgan Ekmefjord
Abstract

Evaluation of the Perforce Source Code Management Tool used in Agile Software Development

Morgan Ekmefjord

Focus in this report is how the Perforce source code management tools can be used in the Extreme Programming methodology and how the different features of Perforce work with the challenges of managing source code while working in an agile way with extreme programming. The study shows how the extreme programming methodology users can use Perforce for their daily operation for paradigms such as continuous integration and 10 minutes build. The bridging between agile methods such as extreme programming and source code management tools are not very clear and in this report some aspects of uniting the two is explained.
## Contents

1 Introduction .......................................................... 1
   1.1 Background ......................................................... 1
   1.2 Problem Definition ............................................... 3
   1.3 Disposition ......................................................... 4
      1.3.1 Theory ......................................................... 4
      1.3.2 Method ......................................................... 4
      1.3.3 Empirical study .............................................. 4
      1.3.4 Analysis & Conclusion ....................................... 4

2 Software Development Practices ................................. 5
   2.1 Extreme Programming ........................................... 5
      2.1.1 Overview ...................................................... 5
      2.1.2 Comparison .................................................... 6
      2.1.3 Architecture and Design through Increments ............... 7
         2.1.3.1 Test Driven Design ................................... 7
         2.1.3.2 Feature Driven Design ................................ 8
      2.1.4 Continuous Integration ..................................... 8
      2.1.5 10 Minute Build .............................................. 9
      2.1.6 Refactoring .................................................... 10
      2.1.7 Coding Standards .......................................... 11
      2.1.8 Collective Code Ownership .................................. 11
      2.1.9 Version Tracking ............................................ 12
   2.2 Source Code Management ....................................... 13
      2.2.1 Actions ........................................................ 14
      2.2.2 Concurrent Editing .......................................... 14
      2.2.3 Version Control .............................................. 14
      2.2.4 Project Documentation ...................................... 16
      2.2.5 Files in Source Control .................................... 17
      2.2.6 Versioning ..................................................... 17
         2.2.6.1 Branching .............................................. 18
# Perforce

3.1 Features ........................................ 21
3.2 Technical Specification ................................. 22
   3.2.1 Internationalization .............................. 22
   3.2.2 File support .................................. 22
3.3 Exclusive Locking and Concurrent Editing .................. 23
3.4 Usage Areas .................................. 23
3.5 Tool stakeholders .................................. 23
3.6 Different Branch Conventions ......................... 24
3.7 Product Version Tracking ......................... 24

# Method

4.1 Choice of method .................................. 26
4.2 Gathering of data .................................. 27
4.3 Selection of correspondents ............................ 27
4.4 Credibility of the study .............................. 27
4.5 Interview .................................. 28
   4.5.1 Before the interview ........................... 28
   4.5.2 During the interview ............................ 28
   4.5.3 After the interview .............................. 28
4.6 Possible Criticism Against The Method ................. 28

# Empirical Data

5.1 Extreme Programming ................................. 30
   5.1.1 Continuous Integration ......................... 30
   5.1.2 10 Minute Build ............................... 30
   5.1.3 Collective Code Ownership ...................... 31
   5.1.4 Architecture and Design through Increments ...... 31
5.2 SCM and Perforce ................................... 31
   5.2.1 General .................................. 31
   5.2.2 Users .................................. 32
   5.2.3 Functionality ................................ 32
5.3 Perforce applicability for Extreme Programming ........... 33
   5.3.1 General .................................. 33
   5.3.2 Communication ................................ 33
   5.3.3 Risk Management ............................... 34
   5.3.4 Features .................................. 34
   5.3.5 Files in Source Control ........................ 34
   5.3.6 Version Control ............................... 35
   5.3.7 Product Versioning ............................ 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Analysis</td>
<td>36</td>
</tr>
<tr>
<td>6.1</td>
<td>Extreme Programming</td>
<td>36</td>
</tr>
<tr>
<td>6.1.1</td>
<td>10 Minute Build</td>
<td>37</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Refactoring</td>
<td>37</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Architecture and Design through Increments</td>
<td>38</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Coding Standards</td>
<td>38</td>
</tr>
<tr>
<td>6.1.5</td>
<td>Collective Code Ownership</td>
<td>38</td>
</tr>
<tr>
<td>6.2</td>
<td>SCM and Perforce</td>
<td>39</td>
</tr>
<tr>
<td>6.2.1</td>
<td>General</td>
<td>39</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Users</td>
<td>39</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Functionality</td>
<td>40</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Version Tracking</td>
<td>40</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>43</td>
</tr>
</tbody>
</table>
# List of Figures

2.1  Overview of the Extreme Programming Values  
2.2  Illustrates a possible incremental change to a class  
2.3  Designing both the class and the test class for stressing functionality  
2.4  Illustrates the different types of ownership and possible scenarios  
2.5  Example of file lifetime in a repository  
2.6  Locking Mode  
2.7  Concurrent Mode  
2.8  1. illustrates a branch. 2. Illustrates a branch merging, note that the source have to merge in the branch before the branch is accepted back into the trunk  
3.1  Shows the visual revision graph of file branch and merge history
Chapter 1

Introduction

1.1 Background

In the beginning source code was maintained manually by company employees. While they were developing in parallel on different machines they had to try and manually transfer files on disks or over the network. This led to communication overhead and much trouble because no control routine or protocol on this procedure was established. When a file was modified locally on a developer workstation no one else would know about the change and thus could go unnoticed for weeks. Sometimes developers would not even notice change until projects where abandoned (Shore & Warden, 2008, p. 169). No one in management was able to track their programmers contribution as soon as the number of programmers outnumbered management staff. If a very specific machine broke down that contained vital parts of the source code the whole project would be in danger and weeks of work could be lost. This called for urgent measures in establishing proper procedures and tools for the support of a unified code base able to store files in a safe unified place. Furthermore such tools ultimately provide tracking of file changes through version control. Version control provides project managers with better abilities for managing and monitoring important digital assets.

Today more or less every software company makes use of source code management tools for their code and other related digital assets. The tools provide versatility that can help in numerous areas in a software company daily business.

From the methodology perspective the forefather of software development processes have came from the widespread industry standards of executing projects dubbed "the waterfall model". This model is based on the solid understanding of methodically first collecting requirements, constructing a design plan, executing the design plan and implementing the code and last but not least test software functionality and then deliver to the customer (Rerych, 2002). This process
of development however have come to show that it lacks significant understanding for the way of developing software and its inability to adapt to the various different project needs that might raise during software scoping and developing (McConnel, 2004).

Companies have over the years had significant profit losses in software development due to exceeded expenses and scrapped projects, mainly due to faulty or insufficiently adapted development methodologies. The lack of proper management and methodologies has called for a number of different new methods for the purpose of developing software. Over the last few years they have gained momentum due to their very adaptive and flexible nature in areas such as management and resource tracking, source code management. Many of these methodologies are so called "agile" development practices. One of the most popular agile methods is the extreme programming method examined in this thesis (Evans Data Corporation, 2007). New methodologies have shortcomings as well and criticism has been aimed at agile software development methods for the lack of proper control of resource and manpower and the lack of long term planning of software development (Wikipedia, 2008).

Extreme programming literature such as (Goodliffe) state that working with source code management tools should be done with very much care and tightly bound to the every day operation of programmers duties. Checking in code frequently to reduce the risk followed by having code outside of version control. In (Shore & Warden, 2008, p. 172) they state that extreme programming has strict requirements when it comes to maintaining a functioning code base. This put many new requirements on the development process and source code management system in operation to cover the needs of daily operation.

Extreme programming has defined paradigms for collaboration on code, code review, writing, testing and for integration of code into releases. These according to (Shore & Warden, 2008, p. 170) be viewed as tasks that programmers have to run concurrently without the need for sharing resources. Source code systems therefore have to be able to maintain this functionality based on the demand of full-scale concurrency and availability.

What Perforce can offer for the software business is a product complete with tools for most of the developer and staff needs regarding file management, product versioning and code safekeeping. The Perforce suite consist of sub products that cover the various areas related to file and version control management. File tools are available to work with the repository and with local files. Bug tracking tools are available or basic management of bugs and relating them to the version control history. Report generation and web access are available for thin-client usage or business user access. Perforce also have server side features that aid the software development such as platform independence and repository caching for fast
replication. Interesting with Perforce is that a lot of companies have chosen to use Perforce as their tool and many of these companies also make use of agile methods such as extreme programming. Perforce therefore might be useful for companies in some aspect but the internal workings of software companies are often blurred or uncharted while Perforce features and usability is clearly defined.

Source code management tools have become widely adopted and agile software development are taking more and more ground, working with source code tools specifically aiding in agile programming methods such as extreme programming and its applicable paradigms are still not fully charted.

This thesis will be produced and interviews will be carried out at the Shanghai offices of Autodesk Ltd. Autodesk Ltd is a software company originating from USA and their field of the software industry is mainly targeted towards providing various CAD/CAM oriented software. Autodesk Ltd have hundreds of thousands of licenses of their products sold and are active in several different business segments of construction and design such as mechanics, architecture, design, construction, plumbing, plant and piping and so on. Recently Autodesk Ltd also acquired the two top Animation and Rendering applications of the market making them business leading in these areas as well. The Autodesk branch that will be involved in this thesis performs development of both desktop and web applications. At Autodesk employees will be interviewed about their relation to source code management, extreme programming and general development paradigms and the connections between developing with extreme programming and version control related tasks with Perforce. Autodesk Ltd has licenses for several different source code management systems except Perforce and I see this as an important aspect when interviewing employees that they have experience from multiple products.

1.2 Problem Definition

Every day programmers and other stakeholders that operate in the company environment need to use tools to accomplish certain tasks. How does Perforce as an example of source code management tools aid in daily operation of software companies supporting adaptation of agile software development. What are the best practices of the tools and paradigms? What are the short and long term benefits and losses of the applied processes and tools? How are stakeholders affected and the level of risk affected? How can management gather measurable values from the development process? The task will hence be to evaluate how Perforce source code management tools can contribute to the process of developing software adopting extreme programming paradigms.
1.3 Disposition

1.3.1 Theory

This thesis begins with formulating the underlying theory about the source code management tools general purpose and then involves the Perforce tool implementation of these features. For the purpose of gaining an overview of the source code management area in general and about the Perforce tool in particular. Furthermore in the theoretical framework the agile software development practices exhibited by extreme programming are examined.

1.3.2 Method

This section explains and motivates the selection of a qualitative study. This section also describes and motivates the selection of correspondents and their applicability for this study. Furthermore it describes the methodology used while conducting the interviews and explains the circumstances of the study. Awareness of possible limits and shortcomings with the study is pointed out in the last section in the chapter.

1.3.3 Empirical study

The empirical study will be structured similarly as the theoretical part to show the correlations with the underlying theory and map the connection between the theoretical frame of reference and the collected empirical data. The analysis will focus on identifying and evaluation of the Perforce source code management tools and how they contribute to the development process of software based on the theoretical framework and the empirical study.

1.3.4 Analysis & Conclusion

The thesis ends with a finishing discussion and draw conclusions from the results that the study derives.
Chapter 2

Software Development Practices

2.1 Extreme Programming

2.1.1 Overview

Extreme programming was originally created by the programmer Kent Beck while working on a project to build a new payroll system for the Chrysler Company and today it is one of the most adopted agile software development paradigms available and it is used all over the globe and in numerous different contexts and project settings. Its benefits are the short development life-cycles and the ability to adapt to changed requirements throughout the whole process of development. Extreme programming paradigms concentrate on the most essential parts of software development and then focus to great extent on making a very good job in these areas while leaving other less important aspects of software development unused. In Extreme Programming the focus of essential parts of the development processes are the processes of coding, testing, designing and listening. The processes are somewhat loosely defined as to let XP projects define their own definitions of core processes in the software development life cycle. These processes are later defined in the context of different XP-values. Teams making use of extreme programming is working very close to each other and makes use of extreme programming specific paradigms such as programming in pairs, code reviews, continuous integration with tokens. In extreme programming the focus is on improving software development through having common values and actions. Important values are such as feedback, simplicity, communication, courage and respect (Beck & Andres, 2004). These values are to be emphasized throughout all the activities of extreme programming. In extreme programming the focus is not on the actions and activities performed in the project but rather to make sure that the values are maintained.
2.1.2 Comparison

Many of the features of Agile programming are similar but as extreme programming have the approach of emphasizing values rather than practices the formulation is differently although many of the strengths and weaknesses of these methodologies are similar. Most outstanding activity in extreme programming compared to other agile development methods is the use of pair programming (Beck & Andres, 2004). Even though extreme programming defines values they also have defined activities when working with projects developing software. There are recommended practices practically every different aspect of software development and every step in the software development life cycle. Stretching from project management and planning to software quality assurance and finally delivering to customer.

According to Shore & Warden (2008, pp. 177,183) important practices applicable in the field of extreme programming from the perspective of source code management systems and source configuration management is use of so called continuous integration and 10 minute builds. Furthermore the authors (2008, p. 191) mean that following XP practice that code should be stored in a single code base where everyone in the team shares responsibilities.

User stories are used for describing the different tasks at hand. User stories basically describe a flow of events that the customer wants to be able to go through to achieve something in the final product. Often since the product might not be directed to a specific customer there are some staff members that act the role of customers such that developers can discuss with them to clarify user stories for their implementation. User stories are manager’s responsibility to break down from the project requirements. Different project roles can be involved in the process of defining user stories depending on the extent of the project and the experience of the
manager leading the extreme programming project. The user-stories are often used as a verification protocol later on in the project and can also be used as a checklist before the final delivery (Beck & Andres, 2004).

2.1.3 Architecture and Design through Increments

One of the most important things in working with source code is that the product developed is going to be shipped after all features are implemented, at least that is how people think when working traditionally. There is a difference with extreme programming because working with stories in this paradigm is instead to make the designing and architectural work incrementally always having a deliverable product. Focusing on having a product available to customers. The incremental design scheme might not always have the features customers demand from the final product but the features that are implemented are carefully tested and according to their respective user story. (Shore & Warden, 2008). Extreme programming addresses

![Figure 2.2: Illustrates a possible incremental change to a class system architecture that aims at simplicity. Simplicity means that a sparse design that only accounts for current features needed. If the design is still lacking of some features (stated in additional user stories) or needs to have more sophisticated behavior the functionality for this is added in an increment later on when the user story describing this is realized.

2.1.3.1 Test Driven Design

Test driven design is the concept of first writing tests that will reflect the programmers need of using a certain construct and after that write the implemented construct that will fulfill those needs. When the construct is completed all
Figure 2.3: Designing both the class and the test class for stressing functionality

the tests will pass and the construct is considered completed. This paradigm enables
discovery of errors and design flaws early in the development process according to
(Shore & Warden, 2008).

2.1.3.2 Feature Driven Design

A different approach from test driven development is to work with
feature driven development where the developers implement features in a traditional
way and test the constructs afterwards.

2.1.4 Continuous Integration

Continuous integration refers to the process of maintaining the codebase such that
at any point in time a programmer can check out the code and get the latest copy
of the code such that he also is able to compile and run it without complications
(Shore & Warden, 2008, pp. 183-185). But what is more to continuous integration
is that a user whom also modifies code should be able to be reasonably confident
that the code he modifies and check in to the codebase will not lead to any major
problems as other developers check in code that modify the same files.

Continuous integration is very useful because it makes errors ev-
ident and easily pinpointed because of its need to merge conflicts and other errors
that occur and can be easily pinpointed to a specific repository version (Miller).

Since the integration is done in small steps potential errors that
could breaks the build or in some other way invalidate automated tests can easily
be narrowed down to which iteration or in what change list the error or invalidation
was introduced. After a flaw is pinpointed the changes introduced in the specific
change list can be modified or reverted to functioning code.

Continuously integrating can avoid having stalls or hiccups when merging huge changes since the traversing of smaller changes often are manageable by automatic code merging tools. In the event of changes made to a block of code by two or more programers then if the integration cycle is relatively short the possible conflicts that could appear are most likely easily resolved (Shore & Warden, 2008, s. 184).

In extreme programming the use of an integration token (Shore & Warden, 2008, s. 190) such as a teddy bear or another kind of mascot or using a separate "integration machine" (Miller) are of high importance as this clearly highlights who are working on currently integrating new code changes into the code-base. The authors (Shore & Warden, 2008) means that this actually serves as a kind of lock that was unwanted in source code management systems but only to the point of just having one team working with the integration of the code base at the same time. This potentially serves a good purpose as the introduction of unwanted behavior to the code-base can be further narrowed down to that of the team currently doing the integrating to be the cause of the inconsistency.

In a system where locking is used, this scenario is not possible since only the one person who locked files for edit are able to edit them. Others have kindly to wait for him to release the locks(Shore & Warden, 2008, p. 170). This might seem convenient but introduces inefficient use of programmer's time as they are not able to edit files as they want. Additional problems with this model of access are that there are limited or no possibilities to work offline and then get online to check in or out code from the repository.

2.1.5 10 Minute Build

This idea aims to maximize simplicity when it comes to building and configuration of developed products (Shore & Warden, 2008, s. 180). To use of scripts that entirely automate the build and configuration steps are of great benefit from many different aspects. For programming teams investing in an automated build script it can be a lifesaver at times where the iterations are stressful and the build and testing process can verify that no direct mistakes are made in the programming (Beck & Andres, 2004). The automated build can also work as to increase team morale and increase productivity since working with the repository code-base aims to be simple and just as easy as typing a few commands to build debug or release code, run tests and deploy. This removed a lot of grief from the programmers, as they previously had to do every step manually.

Very important according to (Shore & Warden, 2008, s. 178)
when aiming for fast building and testing is to build locally. Removing the use of having to rely on external factors such as shared resources and servers that might be down hence putting hinders in the way of building is worth lobbying for. Having a build script that builds the code and can automatically generate intermediate files and code and helps configuring and setting up of services needed for the build to run.

To arrange early in the process of development to have an automated build system will ease the process of developing to great extent and hence the extension of the build scripts can be done iteratively as the product grows. If build times becomes to slow then optimizations can be applied to remove unnecessary stalls. When building the aim should be at most to have a 10 minute building time for the product and if the build exceeds this time the testing might have to be reconsidered and restructured as it is the most likely the cause for the long build times (Shore & Warden, 2008).

Very important to consider is not to rely too much on the development environment IDE as it often has limitations to compiling code, running tests and performing other code related tasks. Often as products mature the requirements on the build process grows and it is very easy to outgrow the IDE capabilities.

2.1.6 Refactoring

When projects grow and applications are developed code-base naturally grows with it, sometimes to such extent that it is not longer manageable by small hacks and fixes (Goodliffe). A practice in extreme programming to adopt that makes the eventual shaping of code and extensible rewrite is to develop in small iterations and to constantly refactor code to fit the needs of the project. This refactoring is done with design closely tied to the process of developing and hence makes the code base more versatile and easily maintained.

To refactor code is basically a formalized way of reorganizing and rewriting code to fit the need of the software project as the software evolves. Refactoring can preferably be done in small steps and sync after every change that the code compiles and tests pass correctly. No major refactoring is ever supposed to take place in extreme programming as this will probably introduce more design than necessary to solve the problem as "over-designing" is never a good idea (Goodliffe). Typical means of doing wrong is to try to anticipate future needs and add new behavior that might be needed in the future, this is probably only a waste of time as requirements often change (Shore & Warden, 2008).
2.1.7 Coding Standards

Coding standards can be agreed upon when projects start about unified guidelines about formatting the code for better readability. Such settings can for example be how many whitespaces an indentation should be counted as for clearly identified indentations. This type of settings can be input as settings into the source code management tools server side such as to enforce these settings for all clients working against the repository. Perforce has further support for client side encodings such as that server side the code is stored in a predetermined encoding but as clients can work on different machines with different encoding standards the server-client communication can handle the conversion silently.

2.1.8 Collective Code Ownership

![Diagram of different types of ownership and possible scenarios]

Figure 2.4: Illustrates the different types of ownership and possible scenarios
The concept of your own code or someone else's code is erased with the introduction of collective code ownership; this is although many people think different a very effective way of maintaining code. If ownership is too vague there will be problems where no one can answer for changes made or make any efforts improving the code and hence tend to lead to failed projects or dropped projects. On the other hand if features or code are strongly linked to individual programmers there are high risks involved when a programmer in charge leaves the team or relocates. When changes need to be made, strong ownership tend to lead to slowly evolving code since only the owners are knowledgeable of the specific parts that need changing. Something of importance is thought to be clear about having programmers that are more responsible and strongly tied to some parts of their code, maybe paired with code that are of their special area of interest or expertise. This often leads to better design, but programmers and project should still vouch for having an open discussion and keep the code in such shape that it should be easy for anyone that wants to modify it to easily do so.

One of the very important paradigms to follow when collective code ownership is adopted is to no matter where you find a problem in the code it is the discoverers problem to fix it (Shore & Warden, 2008). The responsibilities implied by collective code ownership are that everyone that roams in the codebase should continuously improve upon the code and fix anything from bugs, name standard confusion and design flaws.

Collective code ownership is not an excuse for programmers being sloppy and untidy and expecting that others will fix their code later on. Such behaviors will most likely end up in unnecessary hours spent from team members correcting mistakes and result in irritation and tension between coworkers. Since the code is everyones code rather than anyones own code programmers have to let go of the pride in having "their own code" and learn to take pride in the contribution of code as a team. Many programmers who have worked on projects adapting the waterfall model where programmers "owned" their own features this can be hard to adapt to and referred to as "strong code ownership". The opposite "weak code ownership" also exist where people have to coordinate with the feature holder when someone wants to do changes in another persons territory but this introduces unnecessary communication to the process of development and are hence often not recommended for practice.

2.1.9 Version Tracking

Keeping track of changes and deliverables are an important step in the development and release cycle as it has a purpose of file and directory history tracking. For version tracking to become powerful it has to reflect changes in a strategic way. This can
be realized through making use of different label and name convention schemes for the delivery process. One possibility of adopting the version tracking schemes is to use the functionality of versioning in the source code management software. The provided way of tracking versions also usually feature tagging labeling for separating versions in a more easily human readable form. These specific labels and numbered version can then be referred to by testers, integrators and release control personnel when correcting defects or releasing deliverables to customers.

As such the version-tracking feature is useful for all internal staff that is working directly or indirectly with code throughout the whole development process. Integration version numbering can also help out when a problem occurs and programmers need help from the original author of the code. This person can be identified and tracked from the integration version number suggested by (Miller).

The frequency of deliveries to the customer has to be decided upon by management preferably making the delivery frequency such that it can be mapped onto development increments. The difference in decision-making on how to bundle developer changes into increments and releases are always a potent question to answer. One of the ways to frequently deliver features to users is to deliver every new feature to customers as it is added. The strategy to define and deliver "value" to customers in this way makes the adding of new features heavily impact the perceived efforts of the software development team. (Shore & Warden, 2008)

Branching Branching can in extreme programming be a useful feature for the tracking and separating of development for major reconstruction or additions to the existing codebase. Decisions on when to branch can in extreme programming be made upon the different project checkpoint or development phases.

2.2 Source Code Management

Source code management is important for numerous different reasons. The main reason is for caretaking of Software Company's most valuable asset that is of course
source code and as such requires great care in keeping it consistent and safe. (Hunt) describes the process of practicing safekeeping of code with source code management software as having a giant UNDO key that will undo all your errors and mistakes. Source code management works as a history tracker that records what everyone has done over time. Furthermore it provides facilities for working in parallel on separate tasks where programmers can merge their developed code with little or no effort. It can also be extended by a numerous of beneficial mechanisms such as having daily or nightly builds scripts running that automatically builds the code and check for errors and warnings and also the system can trigger regression tests making sure that nothing was broken from the last submits made and we can have automated statistics such as programmer effort estimation through tracking sequence of check-ins (Hunt).

2.2.1 Actions

To make clear for further discussion about source code management the baseline in used terminology has to be defined. The terminology for some of the tasks may vary but the main genre uses the following definitions but and is described in the glossary at the end. Some of these operations might be affected by having different access privileges on users that are operation on the source code management server, although this is not discussed in terminology section.

2.2.2 Concurrent Editing

Practicing concurrent editing is rare because it means that multiple users can edit files in real-time. This introduces complications with compilation and other related tasks that are done by each user at his/her own command.

2.2.3 Version Control

One of the most valuable methods to adopt and tools to master is the use of a version control system. This not only makes keeping track of files as everything optimally gets stored in one single place but also eases a lot of day to day working by introducing several valuable features that can be used by not only extreme programming teams but anyone that develops software alike.

Version control basically is the power of tracking every little change that has ever been made to any file or directory that exists in the repository. Even the actions of adding and deleting files from the repository are covered by version control. This absolute power of the contents in version control are to great help for developers both in the perspective of keeping track of changes that are good but as code grows it can be very useful to look back on the initial code
and review design and the initial intent. Furthermore version control keeps track of metadata for a check in that can be used for keeping track of how developers are contributing with their code and to what extent. Version tracking metadata can also be reviewed by developers wishing to know some details or further information about a specific check in that might help them understand the code better and faster. Two main flavors of version control exists, one utilizes file locking mechanism

![Image: Locking Mode](image)

Figure 2.6: Locking Mode

for editing by a specific user only and where he and only he is able to edit the files locked (Shore & Warden, 2008, s. 170). As this type of version control gives the user exclusive rights to edit files it is convenient where such exclusive rights are needed for making changes. However this is seldom the case and the use of exclusive locks more often than not introduces complications to the process of concurrent editing (Sanja Candrlic). Some of the most crucial problems are that you always have to be connected to the repository whenever you need to acquire locks in order to edit files. This unnecessarily limits the flexibility of clients wanting to work offline or on the go. Furthermore the usage of locks also actively slows down the process of working concurrently. Because if programming tasks overlap that are assigned to different programmers than if overlapping files can only be worked on by one part of the time and programmers have to stall until the programmer whom acquired the locks first have released them. The other flavor is where no locks can be acquired, instead everyone is free to check out, edit or modify code and then check in at any time. Although this might seem chaotic because no one can be sure when someone else are making changes to files this is a very practical and a very common model since it has several benefits (Sanja Candrlic). This approach places responsibility on the programmer checking in code into the repository to fix possible collisions where several users have edited the same file since the user last checked out the file. As
such before checking in the modifications done by the programmer he or she should always update to the latest code in order to be sure that all the latest changes to the files he or she has been working on is incorporated. This procedure with many people working on the same files is most often accompanied by the use of merge/diff tools that often make some visual distinguishing of changes to different files.

To make checking in your changes with this model less painful programmers are encouraged to frequently check in code that is changed to make possible collisions affecting as few lines of code as possible (Shore & Warden, 2008). This approach is also encouraged for the reason that changes that might affect stability are discovered early on. And as a result programmers making use of work by other programmers can faster start working on their tasks.

Consistently making use of having incremented change-list numbers and build numbers are common in all of the phases of continuous integration.

### 2.2.4 Project Documentation

The author in (Miller, s. 109) concludes that well written software tests are the best possible documentation for software that is written and submitted to a source code repository. As the tests will show the usage of software components they give a brief and concise way of explaining the relevant code. Another great benefit according to (Miller) is that tests will always be up to date, as they otherwise would break the build hence it works as always having a current version of the documentation.

Traditional documentation as the product documentation consisting of API documentation, manuals and handoff documentation is also among the important documentations needed according to (Shore & Warden, 2008). Such
documentation cover the communicational needs between the involved programmers and other stakeholders of the project. They (Shore & Warden, 2008) mean that the project documentation among all other relevant files for the product should co-exist in the repository together with the code for ease of version control and updating.

2.2.5 Files in Source Control

In extreme programming the authors (Shore & Warden, 2008) means that every file that is part of the project belongs to version control. Furthermore everything relevant to any stage in the development process is consolidated and easy to find if it also exist in the repository. This means it is a way of keeping track of the changes to files that are in closely relation even if they are made by different people such as requirements worked on by customers and management but are read by programmers who actually need them to be updated and in close relation to the code that it involves. Furthermore technical writers benefit from this as well where they can store their writings in a repository closely related to the actual product they are documenting. If new stakeholders such as new programmers are introduced they can quickly find material that cover parts they are unfamiliar with. Basically every person working on a software project benefit from the consolidation of having all the files in one place.

According to (Shore & Warden, 2008) the way of having all files integrated into one repository often raise questions by people working with it stating that the repository is slow to download, as this might be the case with a repository downloaded from scratch. However it is rarely the case that the whole repository changes at all times and hence incremental updates after the initial syncing are a rare occasion (Sink, Source Control HOWTO, 2004).

2.2.6 Versioning

Versioning files that exist in the repository are a way of keeping track on how many times a file or directory have been modified in one way or another. This can effectively reflect where most changes occur in the development process. The use of this versioning are vast, it ranges from notifying programmers of changes in files related to their work to having testers and integrators investigate functional logic and stressing the new changes as to unveil any design or implementation flaws. Furthermore the version history can even be interesting for customers that with each release can receive a list of submissions/ corrections and their respective content. Furthermore versioning also plays another important role in the release cycle as the integration process can be tracked, increments can be monitored and specific differences can be made of the product releases. Different versioning systems are often applied for the developing product and the released product. This could
be confusing to involved personnel but ultimately the distinguishing of products being under development and live products can be very important to highlight when working on a specific version of the software.

It is common practice to have specific releases followed by a description and a label. The description is basically an explanation of what changes where made and for what purpose. This should serve as a brief explanation for other programmers that work on the same codebase. The label then serves as a mark of a specific snapshot in time of a file or a repository. This can then be referred to easier by for example developers working together. The label can also be used for tagging of a build for example stating that a specific label of the source code repository correspond to a shipped version of the product to clients.

The label can be used to achieve a few different tasks. One of them is to mark a label upon a release version of the application code. Mostly because the labeling of the released version can be very useful later on in the interaction with the client, and product post mortem development. A second very good time in the development stage when to make a label is when something big is about to change. Maybe a reconstruction of some part of the code is going to happen and to safeguard possible reverts a label can be set for ease of going back.

When several files are moved from one place to another in the repository it can be useful for verbosity to set a label to refer to when several different people might use the same set of files and code. The labels can then serve as a sort of informal or internal release versioning control. Another important use of labels is the application of labels by build robots that build automated builds at certain snapshots in time. This can in a successful both for automated build scripts to perform a build and then set a label at time of success or for developers to set a label that the automated build picks up and compiles. (Sink, Source Control, 2004).

2.2.6.1 Branching

As soon as projects starts to grow a demand of being able to maintaining stable code while still trying out newly developed code and brave new features emerges. To be able to cope with these requirements on the source code management tools developers can utilize branching. The branching ability basically gives users the possibility to make a linked copy of files that exist in the original repository also known as trunk to also live coexisting in parallel directory in the repository. This is often referred to as a branch. While branching can be quite useful for developing and integrating features parallel there can be a big overhead for the final merging of the branching with the original repository. This calls for making wise decisions about when to branch. There are several different branching possibilities on organizing code development. One of the decisions when to use a branch can be making
a branch on a freeze of the code from new changes and only accept minor bug fixes to the code stream as to stabilize the build for customer release. This is however not very recommended, as the code fixes that attempts to stabilize the build branch are as much useful to the main repository development as to the branch. In XP the developers should always try to focus on stabilizing the mainstream of the code. Newly development features should instead be the subject of making a branch. In the newly created branch any big changes can be developed and when the code has matured enough it can be reintegrated back with the code mainstream. Often the tagging, labeling procedure is applied in coalition with the creation of a branch of the code. This is because the ease of finding the point at the branch easier in the revision history. This could be just to be able to revert to some point where a feature was not already implemented or just for the sake of having a version to diff against as to see what major reconstruction was done since a last tagged version in history.
Chapter 3

Perforce

Perforce was founded in 1995 by Christopher Siewald with the commitment to produce SCM software that is committed to performance and reliability. Today Perforce is with over 280000 users in 4700 companies classified as one of the biggest and most renowned source code management systems for customers using Perforce for their source code and digital asset management (Perforce, 2008).

Perforce comes with multiple different tools committed to source code management, for the client side applications Perforce supply both command line clients and visual clients for operation. Visual clients further include tools for merging and tools for debug tracking. Also plug-ins for working directly with Perforce SCM from popular integrated development environments such as Visual Studio is available.
3.1 Features

Perforce is a cross platform tool making the Perforce environment favorable to programmers spanning multiple environment and platforms in their daily operation. The cross platform solution relies on low-level frameworks for communication allowing it to work on every computer that implements TCP/IP for communication and popular platforms like Unix, Linux, Mac OS X and Windows are supported. Furthermore Perforce have different choices for developers that interface with perforce because there are command line clients available for scripting and advanced usage. There is also a GUI application that visually browses the source code repository and allows for actions and commands in the repository to be manageable by GUI benefits such as drag and drop actions.

Perforce features branching and branching can work on any number of files in the repository and can continuously track changes of files between branches. Branching can be done from any of the clients interfacing with Perforce server and Perforce server automatically keeps track of the incremental changes made in the repository and across all branches. Perforce does not support keeping track of branches files for integration and hence if branches are to be integrated it has to be done manually.

Merging features in Perforce is available when files are in version and content conflict with files already in the repository. In this scenario the user can either rely on the auto merge option available or choose to manually merge files.

Perforce includes a simple bug-tracking feature called jobs that can handle simple requests for enhancements or code fixes for developers needing a basic featured bug tracking tools. Additionally Perforce have an APIs for interfacing with the Perforce suite for 3rd party developers to integrate their tools with Perforce. The idea is to make APIs available to customize jobs such that it can interface with any 3rd party tool interested in having support for Perforce. Such tools that are available for bug tracking capable of interfacing with Perforce are IBM Clear Case.

Perforce also has support for client workspace configuration and storing the client workspaces on the Perforce server. This makes client side migration to new computers and replication of build environment for the Perforce client side installation and settings to multiple computers convenient and easy.

According to (Meredith, 2007) the Perforce workspace that is used for storing client and build specifications are shortcomings of the flexibility of the development and build process since the ability to reproduce lost or damaged client workspaces are not possible. Perforce lacks support for disconnected operation which might be useful when working in a disconnected state, this means that a user of for example Subversion can still diff/merge against the "local" version of the server file, of course changes made after the cached copy is retrieved is not comprised (Catal,
3.2 Technical Specification

3.2.1 Internationalization

Internationalization or sometimes referred to as localization is supported in many ways, both Perforce product itself is available in multiple language but of most importance is the support for working with multiple language and file formats that support a wide range of character code standards. This support is not only client side but also server-side (Perforce, 2008). Furthermore Perforce have support for storing files in UNICODE format on the server-side while at the same time working with local language setting on client side computers.

3.2.2 File support

For a source code management software to support file changes and rapid synchronization of file changes to multiple clients a technique referring to the mathematical definition of "deltas" as file differences (Sink, Source Control HOWTO, 2004).

There are two types of tracking deltas, one is to at a given time in file history mark as a revision and then for every change made of that revision only save the changes made at these times. The other way of tracking deltas is to mark the revision most recent in history and calculate every change historically as a delta. The latter alternative has the benefit of not having to calculate deltas on every "sync to latest". Such calls against the file repository are in the developer’s interest of calling most often and hence the way Perforce make use of deltas. Perforce uses the later way of working with file deltas since it have the benefit of performance and a decrease in performance when an older version of a file is retrieved. Perforce furthermore have the ability to track changes to files of different format. The formats that are supported are in addition to plain text files, text and binary data with executable bit set, text with keyword expansion, binary files, symbolic links, Unicode files and Macintosh resource forks. Support for transparent replication of the main repository are supported by Perforce by having special servers that act as so called "cashes" that quietly replicates code from the main repository and replicates it to caches. The caches act as the real server effectively reducing network load and support multiple geographically dispersed teams developing against the same code base (Perforce, 2008). The Perforce system uses TCP/IP for communication and hence induces very lightweight and widely supported network communication for good performance in networked environments (Perforce, 2008, s. 2).
3.3 Exclusive Locking and Concurrent Editing

The Perforce SCM tool has support for two ways of working with files in a collaborative environment. One possibility is to allow programmers to set exclusive rights for editing and writing to files in the repository. And also to the way of having multiple users checkout and work with the same files. The latter case requires users to merge their changes if someone else has possibly made changes to the same lines of code in the same files.

3.4 Usage Areas

The Perforce tool has focus on source code management and digital asset management as these two areas are of similar nature and require similar technology for determining and tracking version history. Together with branching and the possibilities to store different file types makes Perforce a versatile platform for storing important digital assets. The capabilities supporting virtual copies also makes it possible to store code stream branches and supporting files such as entire products making them available for end user purposes such as demos and audits.

3.5 Tool stakeholders

Many different type of stakeholders benefit from SCM tools and Perforce have support for many of them.

Developers have the richest feature set at hand on the Perforce tool as they are the key area of people meant to use the Perforce tool as aid while collaboratively writing code. Developers can with Perforce make use of the code fetching, editing, updating and committing of code to the repository that is required
by SCM as their basically functionality. Furthermore developers have access to advanced features such as exclusive locking of files. When using a collaboration mode where multiple people can commit code the Perforce tool provides options and sub tools for merging possible file conflicts. Perforce also delivers visual tools of displaying several valuable features according to (Perforce, 2008, s. 15) shows file history and revisions.

For management the Perforce toolset can offer centralized management of users, group and respective permissions. Furthermore statistics for management is collected from user’s activity such as number of check-ins of code, lines of codes in submissions, time tracking and descriptions of submissions where submissions occur. Additional users can be tailored from administrative users to let different key people access to different parts of the repository. Custom tailored reports can also be generated directly from sources by the Perforce toolset p4report that have abilities to directly generate reports on formats such as for Crystal Reports.

3.6 Different Branch Conventions

Perforce uses branch on release convention and technology called Inter-File Branching technology (Perforce Software Development Lifecycle, 2008, s. 1) to support this work. The support for branching in Perforce works such as to branch when an expected release has to do some special bug fixes from the main development code-stream (Perforce Software Development Lifecycle, 2008, s. 1). As such compared to other source code management systems the main code-stream is always considered the development stream whereas branches are made for releases and bug fixes. Making use of branching this way all the relevant source code files that belongs to a intended product release are marked for branching, not just the file that possibly will differ in the release (Perforce Software Development Lifecycle, s. 2). Perforce furthermore has support for merging back branches into the main code stream and have merge tracking possibilities (Jeffries).

3.7 Product Version Tracking

Like other source code management software suites Perforce versions its files in the repository against check-ins. For each check in where a modification to a file has been done the file revision increments. Furthermore each file gets linked to a change list. A change list is basically a receipt of all the files that where changed at a certain check in and the change list number also increments as people commit check-ins.

It is then possible to compare file or change list versions and revert
when needed to change list or revision numbers of files or revert whole changes made in change lists overall. The ability to tag a certain change list is also possible as to make it clear what type of changes a certain change list contains.

Branching support in Perforce has abilities to track branching and branch merges more extensively than competing suites such as Subversion. Mostly because of Perforce abilities to track file and directory revisions over multiple branches and branch merges. The Perforce suite has branded this mechanism under the name of Inter-File branching mentioned previously. Tags or labels can also be used with branching as to mark a specific branch or merge with a more easily detected human readable form of a label.

Perforce supports branching and merging and additionally perforce can track merges done historically between the different branches and the trunk. The general procedures for source code management involving product ver-

Figure 3.1: Shows the visual revision graph of file branch and merge history

sion management are supported using Perforce. This is realized through the tagging/labeling support of branches or change lists. And used by configuration managers setting tags to specific snapshots of the repository to mark them as a release snapshots. Furthermore if developers use branch on release it is possible to use branches in such a way as to label a branch and stabilizing only this branch although the extreme programming in this case doesn’t recommend it.
Chapter 4

Method

4.1 Choice of method

Following the method of doing quantitative studies, the intention is to gather material to get a deeper understanding of the user’s perspective and knowledge about the relevant methodologies and tools in use. Conducting quantitative interviews with users makes it easier come to a conclusion whether or not the tools and methodologies are applicable and favorable. (Alan Bryman, 2003)

This study chose to conduct qualitative interviews with people working directly with the tools and have experience of working with the paradigms of agile software development. The knowledge these people exhibit is both due to long-term usage of the tools paradigms and from their expert viewpoint as they have been developing software for many years. Thus the interviewees are experienced in a broader perspective of software development as well. So if there are some problematic with the tools and paradigms they have most likely experienced them and learned to avoid them and thus can share with us.

The interview method of choice is to do semi-structured interviews since the nature of the questions and problematic can be derived in such a way that regarding to some significant answers received further investigation is needed. At the same time leave space for freely talking about the subject (Alan Bryman, 2003). The questions used for interviewing have been outlined from the problem formulation with the theoretical framework as support. Furthermore they can be categorized as questions concerning the tool, concerning paradigms and concerning the application of tool usage for specific paradigms. Furthermore the question formulation consists of main questions in each area and also pursuing questions if the answer received needs more clarifications.
4.2 Gathering of data

The material gathered for this study is in the form of primary source because the empirical study is conducted through interviews. Only statistics about Perforce usage and the theoretical material gathered in the theoretical part of this thesis consist of secondary data sources.

4.3 Selection of correspondents

At the host company Autodesk we try to evaluate daily operation and from a deductive reasoning together with worksite interviews make a conclusion as to benefit the company in-house development processes.

Correspondents selected specifically for this study has been carefully chosen because of their long careers and experienced background in the relevant areas of this conducted study. We believe that targeting persons in the industry developing software practicing agile methods are most likely to have good knowledge of the possible tool and method benefits and shortcoming’s. Through these interviews we want to gather their knowledge and opinions to form us a greater understanding of this specific area.

During the interviews the purpose is to gather information in a transparent way, not having the interviewer interfere with questions and the way of questioning, but at the same time try to establish a non-hierarchic relationship such that allow for two way communication with high level of reciprocity and verbosity (Alan Bryman, 2003).

To find candidates for interview company managers in related companies have been asked for possible candidates since they have greater understanding of company structure and employee background. Requirements for finding good candidates have been that relevant background in the areas are present, that no nondisclosure agreement or other business conduct procedure are in violation with our questions.

The number of participants in the study was limited to eight software developers that have been interviewed on site. Candidates have then been asked to participate in recorded interviews, recordings made only for the purpose of dictating the answers by the interviewer for review and analysis.

4.4 Credibility of the study

To achieve high credibility the interview objects are selected from the industry with relevant knowledge and several years of experience in the industry (Alan Bryman, 2003). This would give them sufficient background in the area of expertise that
is required for this study. As these people are experienced software developers they have been in the industry such that they have worked in several different environments and used several different tools that give them the expert status.

To achieve high reliability of this study throughout the interviews and working with relevant material have been done according to established criteria. The interviews have followed a pre-determined frame with pre-compiled questions. To ensure validity further the interviews are conducted in a resembling environment with the same interviewer in all of the cases (Alan Bryman, 2003, p. Chap 3). But the social situation is something that can be a factor to variance in the study and impact on the reliability. The study have also been conducted during a short time span such that the environment and interfacing variables such as working conditions would not vary and affect the interviewed people such that they are not applicable for the interview.

4.5 Interview

4.5.1 Before the interview

Before the interview the candidates have been asked to provide some background information that verifies their credibility as a competent interview candidate.

4.5.2 During the interview

The interviews have been of about 40 minutes duration each and have been taken place in the office at Autodesk. The interview outline questions that have been used throughout all the interviews conducted. During the interviews a recording device have been used to capture sound as to let the interviewer concentrate on the interviewee instead of making notes during the interview.

4.5.3 After the interview

After the interviews the recorded material have been transcribed and analyzed to capture the consensus as correctly as possible.

4.6 Possible Criticism Against The Method

The selected method may have some limitations. A potential problem can be that the interviewer is directing the interviewed person to get the answers that the interviewer wants. In this way the behavior of the interviewer might affect the interviewee by body language or facial expressions. This is something considered while conducting interviews and everyone have been interviewed through a common approach.
Furthermore the interviewee wants to meet the interviewers expectations and therefore might speak above their own limitations and experience for the interviewed belief of satisfying the interviewer. To counter this; the interviewed people have been selected from a wide range of projects and with different backgrounds but all of them with the relevant knowledge and experience in the field.
Chapter 5

Empirical Data

5.1 Extreme Programming

All the participants agree on that extreme programming suits most of their needs from a methodology for developing software. Adapting to new or changed requirements in software development with extreme programming is agreed on by several interviewees as one of the most powerful benefits of extreme programming.

5.1.1 Continuous Integration

Continuous integrating is an important part of practicing extreme programming and the source code management tools have to support regular usage and frequent integrations according to one of the interview objects. The concurrent locking model is effective for the continuous integration according to all of the interview objects.

5.1.2 10 Minute Build

After checking out code from the codebase the build and run process should be done in less than 10 minutes and according to interviewees they mean this relies heavy on the ingeniously of build engineers. One interviewee also states that having the possibility to rebuild specific modules that a programmer is currently working on is very important since often changes only affects part of the codebase. If every build and run or build and debug would take 10 minutes it would not suffice according to one interview object. Important in the 10-minute build is that the checkout of code should take an insignificant amount of time according to one interviewee. And compared to various other SCM systems the interviewees has been working on they state that Perforce is the fastest and most responsive tool they have used. Although all of the interviewees agree it is fast one interviewee remark on that while Perforce is performing an operation such as a checkout the whole application is locked for interaction. The interviewed persons regarded the build automation
as mandatory as it saves valuable time and a local building environment is crucial for development. But as interviewees stated there can often be different versions of components that have to work together and different settings might be applicable to the environment so environment setup might work better by setting up at least key variables manually to ensure correct behavior. Interviewees think that this as long as the shared resources are not directly affecting runtime behavior of applications it is okay to rely on shared resources for build and running even though it is not recommended by extreme programming paradigms.

5.1.3 Collective Code Ownership

The collective code ownership is a great ideal but becomes hard to apply according to the interviewees since people tend to know their own code best and therefore tend to care most for.

5.1.4 Architecture and Design through Increments

All of the interviewed also mentions the importance of clearly defined user stories when working on a product. Having the user stories available when starting the development the implementation may becomes easier to realize says one of the interviewees. Furthermore two of eight mentioned the possibility to associate a user story towards a Perforce change list.

The iterative way of developing is efficient if user stories can be worked on simultaneously means the interviewees. But its important to of have tasks without dependencies according to one interviewee.

5.2 SCM and Perforce

5.2.1 General

Backup is agreed upon by all of the interview objects as being the most important aspect of source code management. Source code management may help the company to keep source code safe in the event of data corruption or technical error that might occur in client side systems according to all of the interviewees.

Source code management may also help protect digital assets as assets can be distributed by the source code management systems with additional copies replicated to different geographical locations. Source code systems can make the code and data exist in a more global scope according to one of the eight interviewees. Two interview objects mention that in the event of a conflict or natural disasters source code management systems might help the safekeeping of code especially when distributed geographically. Client side source code management could
be made easy by project and company guidelines reflecting the way the code should be handled as to minimize risks according to one of the interviewees. These kinds of guidelines may contain information on how to proceed with managing common tasks that affect source code and digital assets according to one interview object. To remind users of frequently making check-ins of their code may also be valuable according to one interview object. Source code management may also be useful for protecting the company assets from digital fraud or theft according to one interview object.

5.2.2 Users

All the interviewed mentions that developers are the main and most important users of the source control management tools and as such should be developer oriented. They also agree that other project roles may benefit from working with the source code system such as test developers, technical writers and managers.

5.2.3 Functionality

All interview objects agree on the fact that the basic crucial functionality of a source code management toolset is the functionality of version control with the possibility of doing check-ins and checkouts. But all of the interviewed people furthermore agree on that in most cases it will not be enough. The need for doing more complicated operations and extending the basic behavior are most often required. Such functionality can be merging; branching and comparing file differences that are thought of as crucial by the interviewed people. The importance of Perforce in the workflow is as important to the development process as a keyboard is to writing code on a computer says one of the interviewed. And with the motivation that Perforce has an implementation that compared to other source code management software used by the interviewed people are extremely fast in comparison. Added to that is that Perforce can maintain the file history even if files are branched, this is clearly a benefit according to one interviewed person that states that this functionality makes it easier to work against previously existing files. Working against existing files the evolution of the files is also there and can be viewed on demand, something that isn’t possible on branching with other SCM systems.

Furthermore as the check-ins and checkouts go about there are possibilities to form labels for certain snapshots of the code and at any period of time its possible to make a checkout by label, date or by any change list ever submitted to the repository. This functionality is something that is highly appreciated by everyone interviewed.
5.3 Perforce applicability for Extreme Programming

5.3.1 General
Throughout all the interviews the interview objects all agree on that in order to successfully apply extreme programming they need tools aiding in the daily operation and the tools need to handle key functionality in software development. These problems can be according to the interviewees the problem of working in parallel with code, working on different projects simultaneously and supporting the various aspects of extreme programming.

5.3.2 Communication
Extreme programming focus on communication but the information and documentation that is needed may be placed in source control for easy access according to four out of eight interview object. Code review is not supported directly by Perforce but can be very important for some projects working with extreme programming and Perforce recommend 3rd party software interacting with Perforce for this functionality to all of the interview objects. One of the interview objects mentions that code review may be very important for extreme programming projects since the pair-programming alternative of code review may not always be possible. If code review is in place the process may utilize the Perforce change list feature for integration with the development workflow according to one of the interviewed.

Something that six of eight of the interviewed also talked about was Perforce option to make rollbacks of change-lists or labels that may help developers revert erroneous code check-ins.

Increased communication and communication cost in extreme programming is something that everyone agreed on but stated it as motivated by the fact that in the interviewees opinion the increased communication ended up in shorter development times. When developing software in small teams in a single location the communication cost was not very high according to one of the interviewees, but when communication is required between teams spread out over different time zones this most likely increases the communication cost, agrees all the interviewed. One interviewee stated that if key personnel work in different time zones this might mean problems since the communication loop would be longer. The key approach according to three out of eight interviewed persons was to have XP applied locally and globally to work with only the XP paradigms that could apply, but the important thing is a unified value base and planning process. To unify terminology globally was also important to eliminate misunderstandings according to one interviewee. One of the interviewed stated the problem that XP emphasizes on communication while traditional software models such as the waterfall model emphasizes on excessive
documentation. This interviewee thought the communication was very important but stated that the shortage of documentation from previous integrated projects or components was clearly a problem in the process of continually evolving software.

5.3.3 Risk Management

Extreme programming could potentially reduce the project error rate if developers work together and developers overlook each others individually written code according to one interviewee. Experienced programmers may need to watch and mentor novice programmers since they tend to sit and try to solve issues by themselves even if they are not able since they are unknown of their limitations according to one interview object. Novice programmers may therefore cause stalls and problems working in extreme programming teams because they may not be used to programming in general and the overhead of working agile with XP is very high according to one interviewee.

5.3.4 Features

What the interviewed persons all agree on are that extreme programming is a highly applicable for most kinds of software development needs but high pressure is put on a proper and versatile toolset. The Perforce source code management tools according to the interviewed supply most of the necessary toolset. Many of the features that Perforce supplies to users can be of great use in general programming but also for extreme programming that redefine the requirements on SCM tools somewhat mentions one interviewee. This redefinition of requirements are mainly that extreme programming needs to have extensive support for concurrent programming and have a satisfying model for handling conflicting code says one interviewee. Half of the interviewed state also that they have used branching extensively and that Perforce branching capabilities are of much use to them.

5.3.5 Files in Source Control

An important requirement on the source code management tool is also to have support for storing and version tracking different kind of digital resources. Storing other kinds of data besides source code is a very important aspect of extreme programming as to keep everything related to the project in one unified place according to one of the interviewees.
5.3.6 Version Control

Features of Perforce that in addition to basic functionality such as the ability to perform check-ins and checkouts may help developers five out of eight says answers with the feature of visual tagging of a build or change list. According to one interview object this give them the possibility to easily navigate important repository versions without remembering exactly the repository version number, as the label may often reflects what kind of changes that is significant in each tagged/labeled corresponding version.

Furthermore on the question about how teams may work with Perforce before a release the interviewed answered they could have different approaches in different projects and while some may stabilized the code in the mainstream in some projects others could stabilize for build in a branch. Two interview objects that had worked on various sized projects stated that in the different phases of a project there could be different approaches on how to interact with the source code. One interview object also mentioned that in web related projects the whole development cycles may be shorter than traditional software and therefore the main code stream was never used for stabilizing products and instead branched on release.

5.3.7 Product Versioning

Product version numbers can be chosen to reflect different setups for individual customers or configurations according to one interviewee. One possibility of product versioning may follow with different configurations of modules rather then a specific customer released version according to interviewees. Different customers requesting different functionality may then instead get a tailored package of different modules. The maintaining of products sold to customers can then be tracked by module version and if modules are updated every customer holding the specific module can receive the specific updates. Software configurations may in this way be labeled as specific products and sold as configuration although they are referred to by individual module contents according to one interviewee. One interviewee suggest that maintaining and fixes of code for products older than 1 year are merely done for compatibility issues with newer products in their own product family or if critical bugs are discovered.

Instead of supporting and updating old major versions of the configurations some may use a classic mode of their application for customer satisfaction while the programming of modules internally and externally are evolving for future versions. This may be useful for applications only wanting to support a main line of development instead of maintaining multiple concurrent versions of an application according to one interviewee.
Chapter 6

Analysis

6.1 Extreme Programming

Based on the models of defining extreme programming that are stated in various literature and material such as (Beck & Andres, 2004), (Jeffries) and (Shore & Warden, 2008) and the collected material from interviews one of the most important feature of extreme programming compared to traditional software development models such as the waterfall model (Rerych, 2002) is the increased need of communication. Communication that on the downside according to (Shore & Warden, 2008) can be seen as costly and resource inefficient. On the upside it results in faster decision making, faster processes and workflow for keeping people informed and provide clarity about their tasks and the average time to complete projects decreases. Using extreme programming together with source code management tools such as Perforce enables the adopting of new or changed requirements in the software during development. A further benefit of version tracking of source code is also that the file history is always preserved. Interviewees state the need for guidelines as to be able to have best practices in place for performing of various common activities. But the extreme programming literature doesn’t provide this material as extreme programming defines values and not processes. Continuous Integration Important from the interviews and from the literature there is a common understanding that the frequency of check-ins should be high. Interviewees mean that code should be checked in as often as possible. Of course the code has to be functioning and should preferably add some functionality or complete some task in each check-in as it is commonly agreed that check-ins should be mapped against user stories. Furthermore the tracking of changes a high frequency of check-ins make the integrations more easily managed and fewer lines of code is easily tracked when browsing repository history. (Miller) also states that the high frequency of check-ins makes it easier for the test developers to easily pinpoint errors that break the build or break functionality. Finding the errors faster leads to faster loopback to the responsible developer
who can start work on a solution. Literature such as Shore & Warden talk about the usage of an integration token for passing around in the teams but the interviewed persons saw this as a resource inefficient and unnecessarily time consuming process to adopt. The interviewees referred to the code review process as a substitute to the singleton integration step in the process and after go signal from the reviewer there is no problem integrating the code with the rest of the codebase.

6.1.1 10 Minute Build

According to (Shore & Warden, 2008) the aim is to have scripts that complete all the tedious tasks of setting up the environment, configuring services and compiling the source code. This is to make the often-repeated process to be a quick way of getting started or further interacting with the project. Another important aspect of programming that can be automated to great extent is different variants performing automated tests (Beck & Andres, 2004). Another important aspect of the "10 minute build" is that the build should be performed locally as to avoid dependencies on shared resources that might not be redundantly available. The product should be able to run and debug correctly on the local machine but if the data it works with is not available at any point in time it is not very important. Perforce have support for the 10 minute build through its possibility of replication of data to local caches making the checkout part of 10 minute build time an insignificant part. Furthermore storing of all the necessary files for the build such as scripts and additional software makes it corporate with the requirements of extreme programming to be able to store everything related to a build in one place.

6.1.2 Refactoring

Following the extreme programming paradigm the user is expected to complete small tasks of minor complexity in a short amount of time. The code is expected to be simple increments of low complexity as otherwise it would have been furthermore split into smaller tasks to reduce individual task workload. Working and modifying existing code is termed refactoring and the refactoring of code is often necessary in the agile way of working with software development (Shore & Warden, 2008). What should be noted though is to make sure that refactoring is never extensive. Extensive refactoring of existing code might solve more to the problem than there is and this is according to (Shore & Warden, 2008) a problem. The interviewees have never had any problems with "overdesigning" while refactoring code as stated by literature. More often than not they implicate that it is a benefit as designing for future need often is what expected later and eases future work expanding the code. Perforce have features for working with the merging of code in case of conflicts or edits in the same files but refactoring is up to the user and while Perforce can detect
if chunks of code are not edited but moved it can keep track of the unchanged code and not mark it as new in case of a refactoring.

6.1.3 Architecture and Design through Increments

Iterative development is based in the theory a fundamental paradigm to follow for successful deployment of extreme programming. Change list orientated source code management makes it easy to navigate the repository’s different implemented user stories.

6.1.4 Coding Standards

Working in a large project that spans multiple people increases the need for unified guidelines regarding formatting and other source code related settings needs to be agreed upon by the whole project. Special support for formatting options are embedded in the Perforce suite (Perforce, 2008) and according to interviewees the support that Perforce provides is very useful for maintaining code base formatting and internationalization & localization settings. People who work on different locales can do so without interfering with each other and as code is send to the server the server-side settings are enforced.

6.1.5 Collective Code Ownership

The concept means that everyone working with code is equally responsible of the quality of code and if code exist that might be a security threat or in some other way can be improved it is the user finding the error that is responsible to fix it or report it to the original author. The collective code ownership is good in theory but in practice it can be hard to enforce as programmer interest often reside in the code they have written themselves. Interviewees think that the Perforce tools have the appropriate functionality that supports checking versions of file regardless of user submitting them and no file security is imposed that gives the file creator the exclusive right of modifications such files. Only in the rare case of files coming from 3rd party no-contributable software that exclusive read and no write can be forced on file basis to prevent users from modifying 3rd party code (Perforce, 2008).

With source code management another important aspect is the need for integrating the code review process with the overall workflow of extreme programming. Code review can populate a change list with files that have been changed and can mark the change list available for review before the actual commit of code. Code review peers can fetch the distributed change list for review and if the reviewer approved the change list. Then the change list can be submitted to the Perforce repository by the responsible programmer or by the reviewer. Perforce
supports the code review functionality through change lists and 3rd party tools and according to both the theoretical framework and in the interviews the code reviews are useful for correctness and completeness of submissions.

6.2 SCM and Perforce

6.2.1 General

Perforce supports the common functionality for version control such as check-ins and checkouts, branching, merging and tagging. Perforce supports the two common models of editing files that exist. These modes are the exclusive locking model and the concurrency model. The exclusive locking model is to be used when absolute control needs to be exhibited on files.

The concurrency model is preferred both by literature (Shore & Warden, 2008) and by the interviewees and this model allow programmers and other users of the source code management systems to work in parallel to great extent.

From the interviews the message was that they preferred an equal amount of communication and documentation since communication among programmers lead to faster development times but at the same time documentation has to be available for programmers to overlook when they move around in the code base. All the files relevant to the developing process and related to the products being developed should be in source control is a common agreement in both theory and from interviews.

6.2.2 Users

Most importantly developers have to be confident and familiar with the features of Perforce and use it properly as to not violate the paradigms and successfully deploy Extreme Programming. Related to users are the security requirements that can be put on files and different parts of the repository. Security requirements span from support of having users with different security levels and access rights depending on their category of user following the different roles of extreme programming to the storing of valuable information to counter various threats including natural disasters. Users have various access rights on the resources such that managers have access to statistics and data from any other type of user and developers have access to the code relevant to their projects. Test developers have access to their respective projects run files and test cases accordingly. The support for security roles such as customers can also ease the communication process with customers as they can observe relevant parts of the repository and object if the products or files they can access are in conflict with their needs or requirements. Perforce have support for
both security roles and restrictions on a user level and on a file level and can also cooperate with existing user domains in both UNIX and Windows environments.

It may also aid in the development process to have experienced uses look over novice users as to minimize risks during the projects. Furthermore since changes committed to the repository propagate to everyone else working on a project the result of a bad submission may affect many people. Hence the importance of the code reviews double-checking before submissions that can be performed by Perforce through 3rd party software. The bad karma associated by people breaking the build also makes the programmers more careful when to integrate their code into the repository.

Testers with access to source code can apart from black box testing i.e. testing the code against different test scenarios also perform white box testing i.e. the testing procedure where test developers actually track down and debug code to see if there are any errors where they have more easily can do low-level testing. The purpose of white-box testing is to make sure that all the code that are present is working. The roles of testers have no specific support in Perforce but the Perforce feature of storing complete client workspaces with environmental variables makes it easy for testers to have a uniform environment or to replicate programmer workspace where errors might occur in order to investigate.

Technical writers can also have use of accessing the source code through Perforce as to make the documentation easier. Managers can review employee efficiency and read different key values such as number of check-ins, number of files, lines of code per employee and additional statistical data that might help management in decision-making processes.

6.2.3 Functionality

Of great importance in the process of developing software is that software should aid in the development, not limiting the development, as such the speed and responsiveness of the application and Perforce server repository is crucial and according to interviewees Perforce has the benefit of speed.

6.2.4 Version Tracking

A source code management tools is essentially made for the purpose of maintaining a track on versions of code (Sink, Source Control, 2004). However the features that work with the versioning have different uses to different kind of development when different type of procedures are put in place. Common practices exist but the use of source code management tool features such as version history, branching and tagging are diverse. In both the theory and in the interviews the common understanding is that the branch on release and branch on feature are the main two ways of using
branching and merging. The one to prefer is depending on the project scope and project settings and either way Perforce can support this. Furthermore Perforce maintains history of branches and merges.

User stories are an important feature that has to be easy to use and maintain with the source code management tools. Using Perforce to maintain the code and related files the user stories can be matched against source code change lists.
Chapter 7

Conclusion

This thesis tries to answer how the Perforce toolset aids in software development using the extreme programming methodology. Proper usage of source code tools and mandatory guidelines for how to operate the process of working with source code can make significant speed increase and other benefits such as lowering or eliminating risks.

Furthermore Perforce can support the necessary needs that follow with the extreme programming procedure of working with source code features such as version tracking, code collaboration and continuous integration. Continuous integration is supported via features such as code review possibilities, change list mappings and is agreed by both interviewees and in literature as easy to work with.

The 10 minute build is supported both by the storing of build related files and through the possibilities of automation of actions in Perforce and the technical solutions such as local repository caches that speeds up check-ins and checkouts. Perforce has support for storing and versioning a versatile set of file formats and instead of including important features such as tools for code review and automated builds it has APIs available that can be used for integration with other components such as code review and automated builds convenient when working with projects.

Perforce handles the requirements of isolated development of new features and stabilizing builds with the usage of branching and merging. Support for managing customer versions containing different feature sets is also covered by Perforce in several included features such as version tagging, multiple workspace configurations for client build setups that enables different build configurations and view settings of the repository. Working with user stories and having customer involved in the development process is supported by user story to change list mapping and customer involvement is supported by security roles and custom views of repositories. The emphasizing on more communication and less documentation also makes the code repository vital for having customers directly involved in the code
Extreme Programming also set demands on different user group support on the source code management such as to support managers, developers, testers and technical writers. These roles can also be supported by the features of Perforce through view organizations and security settings of the repository. Different statistical features are also enabled for users who require it. Perforce has support for all the features necessary for successfully working with extreme programming paradigms such as iterative development and continuous integration.
References


http://subversion.tigris.org/servlets/ReadMsg?list=users&msgNo=34298


http://www.kaboodle.com/reviews/subversion-vs.-perforce


http://www.perforce.com/perforce/corp.html


http://www.perforce.com/perforce/branch.html

http://www.perforce.com/perforce/doc.073/user/i18nnotes.txt


http://www.perforce.com/perforce/life.html

http://cartoon.iguw.tuwien.ac.at/fit/fit01/wasserfall/entstehung.html


http://www.ericssink.com/scm/source_control.html