A Tool for Analyzing the Correlation between Code Complexity and Faults Detected

Seraj Al Mahmud Mostafa
2013

Master of Science (120 credits)
Computer Science and Engineering

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
A Tool for Analyzing the Correlation between Code Complexity and Faults Detected.

Master Thesis Report

SERAJ AL MAHMUD MOSTAFA

Master of Science in Computer Science and Engineering
Department of Computer Science, Electrical and Space Engineering
Luleå University of Technology
Luleå, Sweden

2013
Abstract

Source code file modification and change management is of high importance for large software systems. The regular modification of source code files create an inherent complexity that is hard to analyze and may lead to software systems that are prone to faults. One challenge for reducing faults in software systems is to find a correlation between code complexity and fault detection for change management, such as when handling new features and error tickets.

This master thesis presents a tool developed for the ClearCase version control system, which collects statistical data from files. The existing code complexity tool utilizes the extracted information and generates results for each file for the software system. These results are used to analyze the code complexity throughout the software development lifecycle on the current projects.

The tool will provide the system with information about the software systems correlation between code complexity and detected faults, depending on the role and related task in projects. The potential use of this tool will help achieve more efficient team management, based on managing the code complexity and early fault detection. The vision is to improve estimation of software releases in future and ongoing projects.
# Table of Contents

**Foreword** ........................................................................................................................................................................1

**Acknowledgements** ............................................................................................................................................................2

**Abbreviations** ......................................................................................................................................................................3

**Chapter 1 | Introduction** .........................................................................................................................................................4

1.1 Background ....................................................................................................................................................................4

1.2 The existing revision system .........................................................................................................................................5

1.3 Comparison to GIT revision system ........................................................................................................................6

1.4 Research Area .............................................................................................................................................................8

1.5 Thesis Objective ..........................................................................................................................................................9

1.6 Research Methodology ............................................................................................................................................9

1.7 Outline...................................................................................................................................................................10

**Chapter 2 | General understanding of environment** .............................................................................................................11

2.1 Lookups ..................................................................................................................................................................11

2.2 Software development practice in general ...............................................................................................................11

2.3 Software Maintenance ...........................................................................................................................................11

**Chapter 3 | Existing environment and the new tool** .............................................................................................................14

3.1 Architecture .............................................................................................................................................................14

3.2 Environment..............................................................................................................................................................15

3.3 What the tool should do and support .....................................................................................................................15

3.4 Evaluation of the tool...............................................................................................................................................15

3.5 New Feature/Error Ticket process flow ................................................................................................................16

3.5.1 New feature handling process ..........................................................................................................................16

3.5.2 Version/Build .....................................................................................................................................................17

**Chapter 4 | Implementation** ................................................................................................................................................19

4.1 Data storing procedure: ............................................................................................................................................19

4.2 Result generation.......................................................................................................................................................20
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Evaluation</td>
<td>23</td>
</tr>
<tr>
<td>6</td>
<td>Discussion</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion and Future work</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>7.1 Conclusion</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>7.2 Future work</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>Appendix</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Revision systems</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Understanding VOBs in ClearCase</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Systems design tools</td>
</tr>
<tr>
<td>9</td>
<td>References</td>
<td>41</td>
</tr>
</tbody>
</table>
Foreword

This thesis is submitted to the Luleå University of Technology (Swedish name: Luleå Tekniska Universitet, www.ltu.se) for the partial fulfillment of the requirement of Master degree in Computer Science and Engineering, specialization in Mobile Systems under the Computer Science, Electrical and Space Engineering department. This thesis includes 30 ECTS (European credit transfer system) which is a mandatory part of a 2 years (120 ECTS) Master of Science degree at this university.

Tieto is a large IT consultancy company in Sweden which provides a different level telecom services and product development to its clients. Method and Tools (M&T) is a Tieto maintenance at the Luleå site, which works for supporting the Build Environment for some companies. An agreement of confidentiality and data usage has been signed between the company and the author of this report. This work was produced by M&T group in Tieto to find all necessary information in a low level to enhance their work and plan for ongoing and future projects.

Seraj Al Mahmud Mostafa, is a Master’s student in the Mobile Systems program, by whom this thesis work has been carried out. Before commencing this degree he completed the Bachelor’s degree in Computing Information Systems (CIS) from London Metropolitan University, UK and an associate Post Graduation degree in Information Technology from Jahangirnagar University, Bangladesh. After successful completion of this thesis Seraj Al Mahmud Mostafa will be awarded his Master of Science degree in Mobile Systems from the Luleå University of Technology.
Acknowledgements

I would like to thank the Luleå University of Technology for giving me the chance to complete a Master degree in Mobile Systems and also like to thank Tieto for allowing me to carry out my thesis work.

My enormous gratitude goes to Bengt Axelsson, who was supervising the work on the company’s behalf. He has always been supportive on explaining issues with a clear visibility. I am also thankful to the M&T team member Andreas Mettävainio for his support on some specific matters and also Hassan Bhatti, Sabbir Rahman for their discussion and technical aspects. I am grateful to my internal thesis supervisor Kåre Synnes for his supervision and suggestions time to time to keep my work in track and to help me in keeping my time and task in a shape.

I would like to thank all my class fellows in the program. All of them have been so friendly during all project works, lab works and group-studies during the whole period of my studies.

I would love to say thanks to my Parents, Brother, Sister, Himi and all my childhood friends for their endless support and inspirations at every step of my life.

Seraj Al Mahmud Mostafa
Luleå, August, 2012
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL</td>
<td>Back logs</td>
</tr>
<tr>
<td>RCS</td>
<td>Revision control system</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote procedure call</td>
</tr>
<tr>
<td>SCCS</td>
<td>Source code control system</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software development life cycle</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured query language</td>
</tr>
<tr>
<td>VOB</td>
<td>Versioned object base</td>
</tr>
<tr>
<td>WiP</td>
<td>Work in progress</td>
</tr>
</tbody>
</table>
Chapter 1 | Introduction

Keeping track of every change in a small or large software system has always been a necessity. It is a priority for companies dealing with huge modules of software development and maintenance. The large systems have complications in terms of maintenance as they are prone to regular changes. As a result, difficulty arises in understanding the whole system and its behavior.

It is always hard to control the quality of source codes when complexity arises due to regular modification. These modifications are either adding or removing features to the source code. Changing files is not only about creating complexity in it, but also the commencement of faults in the system. Complex source codes are hard to review, test, maintain and manage which results in the increase of maintenance and product cost. The whole systems become tough to understand and analyze in case of any need in the long run. However, the company (Tieto Sweden AB., www.tieto.com) felt a need of data mining to collect statistical data regarding any changes made into the file(s). Based on this case, the study aims to find out the ‘correlation’ between ‘code complexity’ and ‘faults detection’ upon handling ‘new features’ and ‘error tickets’. The reason behind this is to obtain all necessary information and make precise estimations for ongoing and upcoming projects. The estimations may include, ‘costs’, ‘personnel’, ‘expected required times’, ‘total proposed budget’ for projects and so on. Finding the number of generated new features/error tickets, the new level of complexities in files due to modification in them, is another requirement by the technical experts. Therefore, it becomes an essential recommendation to maintain source code complexity from the beginning of the software development cycle. Software metrics allow us to achieve this goal.

1.1 Background

The functionality changes during the lifetime of a complex software system and so does its performance. It is preferable to have a method that generates historical data, evaluates the correlation between complexity of newly deployed functionality and detected faults with previous source code. With these available historical data, a forecast can be made for how a planned and classed new functionality shall behave in an existing system (e.g. expected faults, estimated hours).

The company is maintaining their project through a version control tool called ClearCase [8]. Every change takes place in the branches of the version controller, and it is possible to find the modifications at any time through ClearCase as needed. ‘New features’ [7] or ‘error tickets’ [5] occur during the time of development or maintenance. It affects the ClearCase environment in terms of adding, removing or modifying the files in projects. This thesis work emphasizes on the ‘File level’ modification, which will allow us to find more detailed tags regarding any changes made on files, for example, ‘effect of adding, removing or modifying a feature’. The purpose is to get the latest statistics regarding files, new complexity level in the file. Later, this statistical information can be added to the existing code complexity
measurement tool that will achieve overall results on every branch of any file through the Software Development Lifecycle in projects.

The undergone master thesis work is a continuation of a previously concluded master thesis titled ‘Automatic Measurement of Source Code Complexity’ [1]. This previous work involved development of a tool which calculated the complexity level of each file after commit. It gave results for Cyclomatic complexity metrics [12], ABC complexity metrics [13], and Functional Interface complexity (FIC) metrics [14]. In addition to this, it provided information regarding number of lines of code and comments regarding its complexity and not being up to the standard. This tool measured the complexity level for Builds, Packages, and Files. The purpose for continuation of previously carried out work is to gain deeper insight at the ‘File level’ which shall return intended tags regarding any file(s).

1.2 The existing revision system
Presently the whole development system is maintained by the ClearCase tool which has few good aspects to work. ClearCase itself is a comprehensive software configuration management system. It has the ability to manage multiple variants of evolving software systems, tracks down the versions which were used in software builds. It also performs building of individual programs or entire releases according to user-defined version specifications and enforces site-specific development policies.

These capabilities enabling ClearCase to address the critical requirements of organizations that produce and releases software:

**Effective development**- ClearCase enables users to work efficiently, allowing them to fine-tune the balance between sharing each other's work and isolating themselves from destabilizing changes. ClearCase automatically manages the sharing of both source files and the files produced by software build.

**Effective management**- ClearCase tracks the software build process, so that users can determine what was built, and how it was built. Further, ClearCase can instantly recreate the source base from which a software system was built, allowing it to be rebuilt, debugged, and updated all without interfering with other programming work.

**Enforcement of development policies**- ClearCase enables project administrators to define development policies, procedures and to automate their enforcement.

ClearCase contains an extensive set of tools that anyone can use to manage and track software resources such as Web pages, mission-critical data, documentation or source code.
Like other source control tools, ClearCase helps to do the following with your software resources:

- Create new versions of a software resource,
- Compare versions of a software resource,
- Merge changes from one version into another version,
- Control simultaneous changes to a software resource,
- Mark certain versions as stable sources to be used in builds,
- Determine the ‘who’, ‘when’, and ‘why’ of a particular change.

ClearCase also provides some unique advantages:

- Supports team leaders who need to coordinate the activity of people developing products together,
- Gives project manager's a control over the extent and frequency with which team members synchronize their work,
- Supports parallel development where developers work in their personal areas and do not affect the work of team members,
- Assists integrators in combining the efforts of the team in a controlled manner.

Few limitations on ClearCase can be addressed as follows:

- One of the limitation regarding ClearCase is that it cannot be accessed offline. It always requires a server connection to have an access. However, revision system such as GIT [9] does not require a connection all the time. We will see a comparison covered in the next subchapter.
- Files are not updated automatically. We must remember to get the latest files from the VOB on a regular basis or else the changes that others make can break the script. Updating once in a day is probably enough, depending on particular team's needs.
- A snapshot view uses a lot of disk space on the local hard disk drive because it copies every file in the Functional Test project to the local hard disk drive.

On dynamic view:

- In a large team, when many users change scripts, it may take a long time for all the changes to compile when you record or playback a script.

1.3 Comparison to GIT revision system

There are many well-known version control systems available nowadays to maintain the revisions. In compare to ClearCase, GIT can be considered as a good example to know. GIT is distributed version control system focusing on speed, effectiveness and real-world usability on large projects. Its highlights include:
**Distributed development**- Like most other modern version control systems, GIT gives each developer a local copy of the entire development history, while changes are copied from one such repository to another. These changes are imported as additional development branches, and can be merged in the same way as a locally developed branch. Repositories can be easily accessed via the efficient GIT protocol (optionally wrapped in ssh for authentication and security) or simply by using HTTP, we can publish the repository anywhere without any special Web server configuration required.

**Strong support for non-linear development**- GIT supports rapid and convenient branching, merging and includes powerful tools for visualizing and navigating a non-linear development history.

**Efficient handling of large projects**- GIT is very fast and scales well when working with large projects and long histories. It is commonly an order of magnitude faster than most other version control systems and several orders of magnitude faster on some operations. It also uses an extremely efficient packet format for long-term revision storage that currently tops any other open source version control system.

**Cryptographic authentication of history**- The GIT history is stored in such a way that the name of a particular revision (a "commit" in GIT terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed. Also, tags can be cryptographically signed.

**Toolkit design**- Following the UNIX tradition, GIT is a collection of many small tools written in C and number of scripts that provide convenient wrappers. GIT provides tool for both convenient human usage and easy scripting to perform new clever operations.

**The good thing** about GIT is it being accessible offline. It always saves ‘view’ in the local computer so it can be accessed any time without a connection to the server. That means it allows users’ to work when they are not connected to the network. It sends only the changes with an encrypted manner which reduces the overhead and keeps things secure and simple.

---

**At a glance comparison to GIT and ClearCase [10]:**

<table>
<thead>
<tr>
<th>Software</th>
<th>Maintainer</th>
<th>Repository model</th>
<th>Concurrency model</th>
<th>License</th>
<th>Platforms</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearCase</td>
<td>IBM Rational</td>
<td>Client-server</td>
<td>Merge or Lock</td>
<td>Proprietary</td>
<td>Linux, Windows, AIX, Solaris, HP UX, i5/OS, OS/390, z/OS</td>
<td>Non-free</td>
</tr>
<tr>
<td>GIT</td>
<td>Junio Hamano</td>
<td>Distributed</td>
<td>Merge</td>
<td>GPL</td>
<td>POSIX, Windows, OS X</td>
<td>Free</td>
</tr>
</tbody>
</table>
1.4 Research Area

Large software systems frequently change over time. It is essential to keep track of every happening change to predict and measure the effects of updates on the system. The goal is to keep the system administrator, developer and users of the system updated with the changes implemented so that they can differentiate the system from the successor to its predecessor. These updates shall include any changes taking place in the system for e.g. addition or removal of a feature, fixation of a bug in the file. All these changes should be stored in the respective tables of the databases.

During the initial analysis and meetings, we fixed some targets regarding the thesis work. The dominant target was how the results can help the company. For example, does it affect other groups or tasks due to new changes in the system? The work is continuation of the previous work where we go in depth at file level to solve some targeted issues along with some other related issues. Finally, to clarify the working procedure and the usefulness of the product to the different user groups, the following research questions arise:

**RQ 1)** Why is it necessary to find more about all the changes are in each instance of a file, and what different information can be fetched?

**RQ 2)** How the users or the targeted group, admin, team leader or developer, designer could benefit from the findings?

**RQ 3)** How precisely we can manage this information retrieved from the environment considering file(s) and how the opted results can be used?

**RQ 4)** How and on what perspective individual people, group of people can find the new tool as helpful?

Additionally, how to make the best use of the tool to be used by the company in future.

*To achieve this goal we need a clear understanding of few things such as:*

- Source code size (standard, usual practice)
- Source code complexity measurement
- Error reporting system
- Adding new feature process

There are some important issues to be known before the definition of the exact goal, research boundary and designing of the solution. To make the goal achievable it is very important to analyze the following key factors:

- The main reason for code complexity measurements
- Key considerations during designing and including environmental issues
- Current systems behavior due to the changes made (if anything added/removed) and analyzing the changes if any
- Impact analysis of the system and comparing current behavior with its previous state
1.5 Thesis Objective

The main objective of this study is to find correlation between code complexity and faults generation based on regular modification of files. It shall allow storage of all the results regarding the changes made in files from time to time. These results generated from the developed prototype should demonstrate selected measurements in action. The prototype should be a standalone implementation suitable for integration into an existing automated build environment or an integrated module in the same automated build environment.

The above defined objectives of research can be mentioned as follows:

**System review:** This include study regarding earlier existing source code complexity measurement tool and further hands on ClearCase environment existing at the company. It also represents the working procedure at Tieto, *for example*, how the tasks are divided into groups, handling of changes and database storage management through SDLC.

**Data mining and defining parameter:** Collection of data from changes in the source code files. This is considered as an input to the proposed tool to generate results. Thus, it becomes essential to select the right parameter from the extracted data to act as input to proposed tool for it to generate results.

**Tool development and integration into real system:** To develop tool and integrate into real system with generation of expected results from extracted data. All data in the database should be accessible through the Web interface for producing results.

1.6 Research Methodology

The work involved development of a tool for the ClearCase version control tool to allow system users to retrieve expected outcomes from modified version of file level. The thesis work was carried out using *Agile* development with scrum process. The work is an extension of previously concluded master thesis carried out at Tieto. The company provided an existing prototype which was earlier developed to measure code complexity to extend functionality in this thesis work. Weekly meeting were conducted with the team (Methods and Tools) along with the supervisor. Final presentation of the concluded thesis work was carried out at the university.

The company deals with large software systems which includes design, development, maintenance, integration, testing and support. There are several teams to handle the different phases of SDLC for each of them. They are divided according to the task based on the project. Each of the work is stored in a different set of database along with the necessary source files. From these number of databases, we further worked on ‘one’ large package to implement the newly developed tool.
The thesis work shall provide following contribution as continuations to existing work which in explained in remaining sections of the thesis.

Pre Study analysis:

- Understanding the previous research and the present environment.
- Understanding the version of each build and the predecessor.
- Investigate and identify the appropriate way to retrieve the maximum potential output. For instance, there are some unnecessary characters or words which we need to avoid, and there are some important comments that we need to handle carefully during storing data in tables.

Design, development and implementation of the proposed tool:

- Development of tool which shall allow to store every detailed data on files from Solaris environment to some database in Windows environment.
- Creation of Web interface which accesses the database to produce expected results.
- Testing of developed tool for its possible usage in the main system to access it anytime, anywhere by all users and team members through a Web interface.

Test, verification and validation of results on real system:

- Run different kinds of test run by the team members to accumulate proper results.
- Several user complexities from different relevant group helped us to choose the right way of developing the tool.

1.7 Outline

The thesis work carried out is categorized into seven chapters. Chapter 1 provides an introduction and summary to the master thesis work. Chapter 2 gives a detailed description of the environment to analyze and deploy the proposed method. Chapter 3 looks into the scope of implementing the tool. Chapter 4 describes the implementation part meanwhile Chapter 5 gives a quick view on Evaluation process. Chapter 6 discusses the achieved results for the whole work. Chapter 7 concludes the master thesis and indicates possible future work as an extension to the thesis. The appendix provides information regarding the architecture and working methodologies of ClearCase.
Chapter 2 | General understanding of environment

2.1 Lookups
This work investigates the task by the M&T (Methods and Tools) team which is responsible for maintaining builds and all environmental issues regarding them. As this is regarding maintenance, we had to keep in mind that there will be lots of issues regarding new changes and error handlings. Thus, any extracted data from the ClearCase environment shall be considered important. We determined to store all historical changes made at file level into the databases generated by the tool according to the user's choice. The generated results can assist the administrator, scrum master or the team leader to make clear decisions. It is very common practice to use past experiences and historical record for building a better solution to the present need. However, it became necessary to store all relevant information regarding a file for current and future use. As an example, if one error ticket is solved, we can see how many people worked for how many days, hours etc. Also, it is possible to see if there is any additional error tickets/new features added to the system due to solving that error.

2.2 Software development practice in general
The human brain is capable to process and store massive information at a time compared to the most powerful computers and most elegant software in the world. However, it has its own limits as well. These limits are unfortunately evident in the world of software development. Development is the most important thing in software creation and maintenance. Working on a small piece of software and analyzing the codes is not hard for a developer to have its perfect understanding. For instance, how each of its elements work together, what each section does, and what the effects of changes or any additions made. However, this perfection can quickly decay in a number of ways. Simply leaving the code and returning to it weeks or months later can greatly reduce this level of comprehension. Adding another developer can lead to confusion and conflicts, while increasing the size can exceed the brain's ability to maintain a good understanding of the whole. Unfortunately, in real life development all of these situations can occur. To count large lines of code, number of tools and models exist to help developers gain and maintain an understanding of software at a range of levels of abstraction, from low-level code to high-level overviews.

2.3 Software Maintenance
One of the immutable laws of software is that ‘it changes’. The software invariably grows larger and more complex as it is updated on a regular basis due to the requirements. A typical software project can consist of extensive lines of code, which is far beyond the ability of an experienced developer to maintain a complete understanding. There are several fields of software engineering based around these tendencies, for example, Software Evolution, Software Maintenance, Program Comprehension, Impact Analysis, Change Propagation,
Software Visualization, Software Timeline etc. The thesis work focuses on discussing Software maintenance.

Software maintenance has been defined as the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment. Maintenance is one of the most costly and lengthy aspects of software development. It often accounts for much as two thirds of all development efforts, however it is one of the badly supported aspect. Many personnel believed that the maintenance activities have low prestige, poorly supported by management and have a low priority at the corporate level. However, maintenance is an integral part of software evolution. In fact, the RISE definition of software evolution implies that the maintenance is the set of activities that make up evolution, since in many instances the majority of work that changes a piece of software takes place in the maintenance phase. Bugs in the code files is other big issue which refers to correction in system that lead the maintenance into a complex situation. The distinction between evolution and maintenance is not always clear. However, at a very high level, evolution is the changing of software over its life while maintenance is the set of activities that comprise those changes.

Software maintenance can be divided into following main types:

- Perfective maintenance involves adding or improving the functionality
- Corrective maintenance is fixing errors or bugs in the software
- Adaptive maintenance refers to updating the software to meet a changed environment, such as a new operating system
- Preventative maintenance concerns with altering the software to facilitate future maintenance efforts.

As we are dealing with the effect of adding new features/error tickets at any time of the SDLC and maintenance, it becomes important for the team to be always ready to handle them in productive approach. It is good to be careful while solving a new feature. The new code is expected to be tested and implemented, so we are not leaving any chance of creating a new error ticket for that or adding a new feature just because of the previous ticket is solved. The team should always be ready to accept any new ticket at any time due to a change. The maintenance team is the one which works for maintaining the Build environment.

The whole maintenance work is carried out on an UNIX platform and users are supposed to log into the ClearCase through UNIX. Once the users are in this environment, they are allowed to work on files for any changes or creation of new instances if needed. The figure below shows how the system is configured and the highlighted part is the area where we worked to get all our expected results.
In figure 1, we can see the flow of accessing the files and the use of them by different sources. The highlighted part is the area where we implemented our code to retrieve all related information regarding a file from its creation till any future updates/modifications.

The configuration files can sometime have the same name, which can be used in other packages and builds. It was also an essential task to find the number of occurrences of those configuration files and to which Packages and Builds they belong.
Chapter 3 | Existing environment and the new tool

3.1 Architecture
The following figure (figure 2) gives us a clear view of the whole system and its working procedures in respect to its user-files-views in the system.

![Figure 2: system architecture](image)

The ClearCase runs both on the UNIX systems and Windows systems. In the Windows platform, it is accessible and viewable but to create a new instance or modifying an existing one we have to be in the UNIX platform. The users need to have their own view to create, modify or update a file. One user can have multiple views, single file can be accessed through different views and multiple users can work on the same file at the same time. Thus, the user-view-files have a many-to-many relationship while they are on a development or modification stage. All the works are saved in the ClearCase environment and are accessible later by different user or different system through personal login credentials. The data can be accessed by different sources such as; some users want to use them through the UNIX shell, ClearCase in Windows or via the Web pages.
3.2 Environment
Presently all kinds of development, adding/removing features or troubleshooting is done through the UNIX (Solaris) environment. So, it is quite important to consider the boundary during the design of the proposed tool, such as Unix as well as Windows operating system environment. The UNIX system is used for all kinds of development, commit, integration and final testing. Viewing of results, analysis and pre-testing (like, viewing the new tree structure) are in between done from the Windows environment. We can also have the same view in the Windows environment in ClearCase but it is not possible to make any changes in it. Therefore, UNIX is the only environment where all the development is done.

3.3 What the tool should do and support
The tool is developed due to a demand of the company for collecting some historical and modification related issues in the file systems. These results are targeted to support different user, user groups for fulfilling their future goals.

Before commencing to this step we have following considerations:

- The tool is capable to access MySQL databases and text files in the ClearCase.
- The tool is being available for Solaris.
- Windows platform allows the tool to print results from ClearCase.
- Data can also be stored into MySQL tables from both environments.
- Also, the tool can merge with other tools if necessary.

The preferred tool should offer more than just fulfilling the core requirements. Some additional features are:

- The tool can be integrated with ClearCase.
- The generated report should be viewable through a Web page
- All the transactions are stored in the database on a regular basis.
- The application can be run from command prompt as ‘Batch job’/’Crontab job’.
- The tool can be updated and evaluated due to upgrade/changes of systems.

3.4 Evaluation of the tool
The tool would mostly be used by the team leader, Scrum master or developer rather than a general user. The management can also use the produced results for organizational purpose and planning upcoming projects.

As expected, the tool is capable to extract all the necessary information from file(s) from the root level of the system and further store it in the database. Different types of results can be accumulated based on the data retrieved and to the needs of the team leader, developer or a user. Some of the result types which can be fetched are shown as below:
- How many tracks are created due to this correction?
- How many people are involved?
- How many days are needed?
- How many numbers of files are affected?
- How many number of new features/error tickets generated while solving them?
- Is there any special comment stored for a file?

Our main aim is to find every single state of a file with its different properties. Hence, a detailed investigation with every file containing track was carried out. *For example*, in case of a trouble or a requirement of change in the regular flow within a system, either an error ticket or new feature is added to some tracks. After an arrival of this error ticket or new feature, it goes in development or modification stage. After the modification and some more stages (e.g. testing and verification) they are merged back into the main track again. These processes are done under some certain rules and regulations. For this, we have to understand the processing of an error or feature addition. In the next section we will see how an error ticket or new feature is added and processed.

### 3.5 New Feature/Error Ticket process flow

As companies strive to manage their Software Development Lifecycle (SDLC) projects, they are inundated with its requirements, defects, software and data changes. In order to manage customer expectations and priorities, adding new features is common in general, therefore we need to have a regular follow up on it. Since change is inevitable, it would be insane to know it is going to happen and to do nothing to manage it. Adding features those are required in a system is a regular process to be used throughout the SDLC.

#### 3.5.1 New feature handling process

If change happens, whether one leaves things alone or not, then it behooves the person to define a process for managing change. Below is an attempt to define a process derived from these parts.

**Submitting**

This is the process of formally documenting requests and submitting them for an evaluation. Submitting is accomplished via an online ticket system in companies where quality tracking tools have been implemented. The data is typically stored to a commonly accessed database.

**Evaluating**

During this process, a person or group with the authority to determine the priority and evaluates the request. The user can later set the timeframe for its implementation to achieve the highest probability of contributing to the success of the current/future project or a release.
Reviewing
This function is usually performed by a team of people who are responsible for defining the content of releases in order to meet or contribute to the product roadmap. In addition to re-assessing pending proposals (requests for change), they are re-evaluated for their effect on the release schedule, resource availability and improvement to the process. This group must judiciously determine that these changes that will contribute to the product success with the least impact on schedule and resources.

Scheduling
Once the priorities and functional considerations with respect to the product roadmap have been evaluated, a schedule must be determined. A committee needs to schedule requirements by gathering cut-off dates, design time boxes, review types, development order and verification timelines.

Implementing
During the implementation phase, the Quality Assurance team and Development team works together to define the extent of each change. New test cases are devised, inspection of content occurs with review and update of the documentation.

The system proposed below, uses new feature as the name for a change. The label, new feature is neutral with respect to blame, cause, egregiousness and priority of the change described. It is important to form a strong working relationship between those people that can issue a new feature and those responsible for resolving a new feature. To this end, a new feature is not always a defect. To refer to every change as a defect, incident, problem or bug indicates that the creator of the software or system made a mistake. Normally, there are many types of new features. Some indicates to a solution or a defect and others are enhancements for a new work.

3.5.2 Version/Build
This field represents the version or a build of software where the new feature was detected. There may be several kind of these fields. Other might hold the version of software, where the fix resides and yet another might represent the version actually released. This field presumes that the software delivered has a build or release version that pertains to all of the software.

Similarly, we have a same process for handling an error ticket. The below diagram show the flow for an error ticket for how it is handled. We are not going to cover every detail on the error ticket as this is almost the similar kind of process. But we go through a description which allows us to understand the whole scenario.

The figure below describes the process as it is implemented today in an agile/lean setting. Candidates are error ticket/new features that are not yet on any backlog, but awaiting an administrator level decision. Backlogs (BL) are new feature/error tickets that have a
clearance from the Administrator but that are not yet in progress. Backlogs should be prioritized either by the CCB (*process authorization body*) or by a delegate. Via internal pull, the *new features/error tickets* go into Work in Progress (WiP) as soon as there is free capacity.

![Error Ticket process flow in the system](image)

**Figure 3: Error Ticket process flow in the system**

This figure (figure 3) describes the main *error ticket* process flow in the UNIX system. But to place a *new feature/error ticket*, we have some process on a user friendly Web environment. The following chapter describes how to place a new *error ticket* in the system through the Web environment. As for example, whether they are in different track, date, time and created by many different users. Also, it shows many other properties that can be seen from the labels of the image (*Figure 3*).
Chapter 4 | Implementation

The whole implementation procedure is divided in two parts. First part is continuous data collection and storage in respective tables upon changes made in files. The other part is producing results on the Web interface according to the user's choice. We will see how these two parts are implemented in the following sub chapters.

4.1 Data storing procedure:
The developed tool will be integrated with the existing system to collect targeted data from the ClearCase environment. This tool will help the system users in many ways. The complexity level will have new values because of changes in the code files(s) and the database will store all the new values in respective tables. On the other hand the version tree keeps adding the new instances of files to the successors. This helps us to find out what are the exact reasons for the new complexity values from these branches.

Some functions, loops, classes, comments can be added or removed because of the demand of an update. Man-hour used, number of people, time to fix an error or an integration of a new change can be easily calculated as well. In the long run, the version tree includes them in the respective branches as historical data which can help in deployment of other related tools in the systems.

There are few databases to store all the data collection. The different tables on the database have a different set of data value. For instance, there is a database which grabs all possible information from a package for all files that it has. The potential data sets we got in a table are name of the creator of the file, date and time of creation, name of developer who modified the file time to time and time required to develop or modify. Other data sets include which package does it belongs to and where to commit back, list of predecessors and successors if any, read write permission to the user, any special comment and also some confidential records.

Other than created tables, more tables have been merged from different sources. These tables will help to compare or match new and old values for result calculation to see the differences. There are also few tables which contains information regarding ‘Builds’ and ‘Packages’. The following table describes the type of information we have stored as an example.
One table outline in the database:

<table>
<thead>
<tr>
<th>Field_name</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>File_name</td>
<td>Fixed length</td>
</tr>
<tr>
<td>Track</td>
<td>Fixed length</td>
</tr>
<tr>
<td>Daycreated</td>
<td>Fixed length [date format]</td>
</tr>
<tr>
<td>Timeopened</td>
<td>Fixed length [time format]</td>
</tr>
<tr>
<td>Createdby</td>
<td>Fixed length</td>
</tr>
<tr>
<td>Branchtype</td>
<td>Dynamic length</td>
</tr>
<tr>
<td>Master_replica</td>
<td>Dynamic length</td>
</tr>
<tr>
<td>Ref_for_mastership</td>
<td>Fixed length</td>
</tr>
<tr>
<td>Ep_user</td>
<td>Fixed length</td>
</tr>
<tr>
<td>Ep_group</td>
<td>Fixed length</td>
</tr>
<tr>
<td>Ep_other</td>
<td>Fixed length</td>
</tr>
<tr>
<td>Element_type</td>
<td>Fixed length</td>
</tr>
<tr>
<td>Comments</td>
<td>Dynamic length</td>
</tr>
</tbody>
</table>

Here, the ‘File_name’ is the name of the modified file and ‘Track’ is the branch of ClearCase in which the file belongs to. ‘Daycreated’ and ‘Timeoppened’ are the date and time of creation of the file respectively. ‘Createdby’ refers to the initiator of the file and ‘Branchtype’ states whether it is a main or sub branch. ‘Master_replica’ is about the location of the file and ‘Req_for_mastership’ means it awaits a decision to be in the main track after modification of all necessary changes. ‘Ep_user’, ‘Ep_group’ and ‘Ep_others’ are the fields which indicate the employee, which project they belong to, group and what type of access permission they have respectively. ‘Element_type’ is the type of file, e.g. .txt/.c/.cpp files etc. ‘Comments’ is the field where there are some comments stored and due to this comments some actions can/cannot be taken.

The tables in the database are containing refined data along with some few more tables to compare data when necessary. The Web interface will use these data to produce results according to the user’s choice. These choices may differ from user to user upon their need.

4.2 Result generation

Creation of new feature or error ticket is always branched under its main track with a new name on a temporary basis. It is distributed either by the team leader or automatically assigned on a web page to the corresponding developer. Once the assigned task is carried out, it then awaits for a test. If any faults are detected during testing, it goes back to the modifier/developer in a new subtract which is to be corrected. After a successful test completion the administrator or team leader receives notification to add that final version back in the main track.

It is very important to know that, if there is a new feature added, then there can be several error tickets to solve it completely. Same thing can happen to any error ticket as well. Even, there can be a chance of no error tickets occurrence. Sometimes, due to hardware upgrade or switch over to a new system, addition of new feature to the error ticket occurs. Usually they
are inserted as a patch file to the systems. These patch files can also be treated as new feature or error ticket depending on what types of additions or corrections they had to make.

No matter with the total number of error tickets or new features, they are properly branched under the generating track. Those tickets are assigned a unique ID from the temporary ID, after finalizing which are traceable through the ClearCase where all the activities are saved in the database. The following images represent the details regarding some extracted information as we have discussed earlier.

In the first row of the following image (figure 4.a) represents the file names. and the other rows contain different information. For example, they are in different track, in different date and time and created by many different users. Also it shows many other properties of the files as seen from the field labels of the tables.

![Figure 4.a: shows that how all data are stored in database](image)

![Figure 4.b: shows that how all data are stored in database (continued to 4.a)](image)
In figure 4.a the names are blurred with black mark due to confidentiality issue. Figure 4.b is the continuation to the figure 4.a.

The main goal of the Web interface is to provide a single and common platform which may reduce the complicacy in changing OS from Windows to UNIX or bidirectional. The different users of the system can get the expected results according to their need. For example, the information we have extracted are:

- Number of **new features/error tickets** occurred in a year or by some person.
- Number of **new features/error tickets** occurred for solving a **new feature** or correcting an **error ticket** by year and person.
- Time (how many man hours) required for one task.
- Number of people involved in the task.
- Number of steps was taken (for instance, only admin level can see this).
- How many files were affected due to them and such issues?

There are many options on the page where a user can select the required tasks to see their results for a file. For example, a user wants to see one file’s property like, how many people were involved, how many **new feature** were added to that file and how many **error ticket** occurred to solve it. The result will change upon different types of query.

The following image (figure 5) shows results for error handling. We can see how due to modifications requirement, number of people, errors and changes are related. Same type of accumulated results can also be retrieved by adding multiple projects and different queries in the source code files.

![Figure 5: Related issues for an error](image-url)
Chapter 5 | Evaluation

From the beginning as our prerequisite, all our activities in ClearCase are handled through the UNIX environment where the developed tool has been tested. It is made easy to change for any kind of modification or implementation in the need of adding extra features, with a possibility to be implemented in almost any of the version. The data fetching part can be deployed in Windows operating system. All necessary data is stored in the database whenever any change is made in the file. The database is available to the systems user and also as the source to the Web interface to generate results.

The tests have been conducted in phases. For instance, the raw Perl files were run through configuration files on the UNIX systems by the admin or same level users. These files can be run automatically as a regular task by configuring them in the Crontab job. Once the task is initiated, the DB will immediately allow an access to it. Other databases can be accessed or data can be stored on them by providing database details (e.g. name, port, password) while configuring the Crontab.

According to the user evaluation the tool was capable to produce the following shown expected results. Screen shots of the Web interface are shown as result to see how a user can get information as required. For an instance, the user can see a comparison between new feature (cr) and error ticket (tr) among various options (Figure 6) and the other image (Figure 7) shows the total number of people participated for solving several new features.

![Figure 6: comparison of Error Ticket and New Feature are handled per year.](image)
However, the above statistics shows us the view of an overall situation on code complexity and fault related issues. It is quite helpful for a team leader in decision making for the future and ongoing projects, handling with new tasks, time planning, number of people expected, time distribution for individual new feature or error ticket etc. It will be easier to add new features on it by the developer if the code files were will written, properly commented and documented.

There are many options to select on the new Web tool which allows the user to choose from multiple options according to their choice. Few snapshots for web tool have been discussed here which is useful for overall understanding to the reader. Some things could not be disclosed in the report due to confidentiality agreement with the company.

**Figure 7: number of people each year solving new features**

![Graph showing number of people varying every year for Handling CR](image_url)
Chapter 6 | Discussion

The results presented in this report are investigated and produced due to the requirement by the company. At the beginning of the work, many issues to work on were discussed and had fixed goals to reach. Finally, after discussion and deep analysis on this topic, the objective of the thesis has been achieved. We had to find all transactions in the version tree regarding a new feature and error ticket creating and solving. These occurrences happen in file level whenever any modification is made on file(s).

For a quick review on this task, the research questions described earlier can be now answered. The following paragraphs discuss the findings based on the research issues.

RQ 1: Why is it necessary to find more about all the changes are in each instance of a file, and what different information can be fetched?

We know that, the new features, error tickets can occur anytime of the SDLC during the maintenance period of the project. These occurrences leave a high chance of creating multiple instances of files and being added to the different tracks in the version controller. Once we have the result for the number of instances regarding one file, it becomes easier to find more about it. For example, from each instance we get ‘date and time of the file creation’, ‘name of the person who originated’, ‘the previous originating track’, ‘next track to be merged in’, ‘reason to create this new instance’, ‘man hours occupied by the developer’, ‘how many people were involved’, ‘total number of new features/error tickets’ and more importantly if it is a feature to be added or an error to be handled.

RQ 2: How the users or the targeted group, admin, team leader or developer, designer can have the benefit of the findings?

This bunch of information helps the team leader in making proper estimations for the next project such as time calculation, number of developer involvement or estimated man hours to be used. At the same time, these information can be useful to the management to make decision from a general point of view. Historical data analysis is made easier through this process including finding relevant information on any similar project that were developed long ago. On the other hand there may be some closed projects and for them we need to retrieve and store all required information for some raised issues in some present situation. Overall, it can also help in budgeting for the future projects in various ways.

RQ 3: How precisely we can manage this information retrieved from the environment considering file(s) and how the opted results can be used?

This is one of the implementation of the investigation which is about retrieving specific information on each file. The information is stored in ways that are accessible in any circumstances and representing them to the targeted user groups as they want.
We have created database for this issue and is well managed, so there is no chance of redundancy in data storing. The types of information are separated in table with proper key initialization, providing easier access to merge information and proper usability through Web interface. These are presented in a user friendly way for users demand centric query. *For example*, it is possible for someone to know the number of people involved in some specific task or handling some error issues. The user groups can access all needed information at any point according to their access right.

**RQ 4: How and on what perspective individual people, group of people can find the new tool as helpful?**

The tool was made for some particular results to be generated and for some specific groups. However, this retrieved information is not open to all user groups and not even fully accessible for any of the other groups or users. Only the administrator or team leader can set the access priorities accordingly to required tasks. In general, the results from this tool are not generalized for each of the user group. *For example*, a developer may not need to know how many people worked in this similar task last year, but to the team leader it is forms an essential information. The Scrum master may look for how many people were hired for a number of *new feature* addition and correction in a specific time, which may not be an essential information to the management team. The management team may be interested in the man-hour used and cost involved. So, whatever information we retrieve is specifically usable to specific groups of people.

*Additionally*, if these types of information is used in every project by each group, then it becomes a cumulative result providing an overall estimation to the company. It is on the company or administrator level to make the best use if this information according to their specific needs. It can be either for calculating budget, time estimation or fixing number of developer required for the next period of project with estimation on company’s business practice.
Chapter 7 | Conclusion and Future work

7.1 Conclusion
The tool has been developed from this research to fulfill some specific requirements by the users of the system to collect statistical data from the ClearCase environment. The thesis work has been concluded with reference to the thesis objective as follows:

Detailed study shows that ClearCase provides extensive set of tools which anyone can use to manage and track software resources, either they are web pages or source code. However it cannot be accessed offline as in the compare of GIT. GIT further supports saving view while working offline and can be integrated later when it is connected to the network. The thesis work however involved with tool development in existing ClearCase solution at the company.

Data collection was carried out from changes in the source code files. The essential parameters were extracted from the available data to act as a feed to develop the tool and produce the results. The main research questions have been addressed in the discussion which forms the objective addressed by the thesis work.

The tool developed works on any UNIX environment where the results can be displayed graphically through Web interface. It fetches source code modifications from ClearCase environment and stores in the respected tables. It is able to plot number of error tickets, new tickets, man hour, number of instances of file modification for a specified time frame. The management can be benefited through it in case of budget calculation e.g. estimation of time, personnel depending on tasks. However, this tool can be a great help for such type of organization where the company deals with large software systems.

7.2 Future work
Various functionalities could be added other than considered for our specific target. Essential addition could be to detect unused code blocks such as functions or classes in the source code. These remain in the code due to regular modification by the developers during the addition of features or solving errors. Unused code blocks raises complexity to the code files. Hence, company can focus to work on normalizing the code files.

The work on tool can be further extended for both Windows and UNIX operating system with added flexibilities. Development in C++/C# platform can bring maximum development and maintenance feasibilities for future use.
Chapter 8 | Appendix

1 | Revision systems
It is very important to know the system where the work will be performed and how it will be
done. So, we will discuss quite elaborately the revision systems and the subversion ClearCase
that we have worked on.

Versioning is a system that tracks incremental versions or revisions of files and, in some
cases, directories over time. What makes revisions successful is the fact that it allows user to
explore the changes which resulted in each of those versions and facilitates the arbitrary
recall of the same. It’s become easy for its lifetime to track all the records as required.

*What are the most used versioning/revision system?*

Here is list of most used revision systems with few of their properties.

<table>
<thead>
<tr>
<th>Software</th>
<th>Platforms supported</th>
<th>Programming language</th>
<th>Network protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccuRev SCM</td>
<td>Most Java Platforms (Unix-like, Windows, Mac OS X)</td>
<td>C++, Java</td>
<td>Custom, HTTP, SFTP, FTP, custom, custom over ssh, custom over HTTP, email bundles, WebDAV (with plugin)</td>
</tr>
<tr>
<td>Bazaar</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>Python, Pyrex, C (nb 8)]</td>
<td>BK protocol, rsh, ssh, HTTP, email</td>
</tr>
<tr>
<td>BitKeeper</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>C</td>
<td>TCP/IP, HTTP</td>
</tr>
<tr>
<td>CA Software Change Manager</td>
<td>Unix, Linux, Windows, i5/OS</td>
<td>C, C++, Java, HTML</td>
<td>HTTP, custom (CCFS), custom (MVFS file system driver)</td>
</tr>
<tr>
<td>ClearCase</td>
<td>Linux, Windows, AIX, Solaris, HP UX, i5/OS, OS/390, z/OS</td>
<td>C, Java, Perl</td>
<td>HTTP, custom (CCFS), custom (MVFS file system driver)</td>
</tr>
<tr>
<td>Code Co-op</td>
<td>Windows</td>
<td>C++</td>
<td>e-mail (MAPI, SMTP/POP3, Gmail), LAN</td>
</tr>
<tr>
<td>Codeville</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>Python</td>
<td>Unknown</td>
</tr>
<tr>
<td>CVS</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>C</td>
<td>sshserver, ssh</td>
</tr>
<tr>
<td>CVSNT</td>
<td>Unix-like, Windows, Mac OS</td>
<td>C++</td>
<td>sspi, sserv, gserver, pserver, sserver,</td>
</tr>
<tr>
<td>Software</td>
<td>Platforms supported</td>
<td>Programming language</td>
<td>Network protocols</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------</td>
<td>----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>darcs</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>Haskell</td>
<td>HTTP, custom over ssh</td>
</tr>
<tr>
<td>Fossil</td>
<td>POSIX, Windows, Mac OS X, Other</td>
<td>C</td>
<td>HTTP</td>
</tr>
<tr>
<td>Git</td>
<td>POSIX, Windows, Mac OS X</td>
<td>C, shell scripts, Perl</td>
<td>Git Server Protocol[15] over TCP or ssh, rsync, HTTP/HTTPS, email, bundles</td>
</tr>
<tr>
<td>GNU arch</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>C, shell scripts</td>
<td>WebDAV, HTTP</td>
</tr>
<tr>
<td>IC Manage</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>C++, C</td>
<td>Custom</td>
</tr>
<tr>
<td>LibreSource Synchronizer</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>Java</td>
<td>HTTP, File-System</td>
</tr>
<tr>
<td>Mercurial</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>Python, C</td>
<td>HTTP, custom over ssh, email bundles (with standard plugin)</td>
</tr>
<tr>
<td>MKS Integrity</td>
<td>Unix-like, Windows</td>
<td>C, Java</td>
<td>HTTP, custom</td>
</tr>
<tr>
<td>Monotone</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>C++</td>
<td>custom (netsync), custom over ssh, file system</td>
</tr>
<tr>
<td>Perforce</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>C++, C</td>
<td>Custom</td>
</tr>
<tr>
<td>Plastic SCM</td>
<td>Linux, Windows, Mac OS X</td>
<td>Java</td>
<td>REST services over HTTP/HTTPS</td>
</tr>
<tr>
<td>Rational Team Concert</td>
<td>Linux, Windows, AIX, Solaris, HP UX, i5/OS, OS/390, z/OS</td>
<td>C++, Java, C#</td>
<td>SOAP over HTTP or HTTPS</td>
</tr>
<tr>
<td>SCM Anywhere</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>C++, Java</td>
<td>TCP/IP</td>
</tr>
<tr>
<td>Sourceanywhere Standalone</td>
<td>Unix-like, Windows, Linux, Mac OS X</td>
<td>C++, C, Java</td>
<td>custom, TCP/IP</td>
</tr>
<tr>
<td>StarTeam</td>
<td>Windows and cross-platform via Java based client</td>
<td>C</td>
<td>custom (svn), custom (svn) over ssh, HTTP and SSL (using WebDAV)</td>
</tr>
<tr>
<td>Software</td>
<td>Platforms supported</td>
<td>Programming language</td>
<td>Network protocols</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------</td>
<td>----------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Subversion (SVN)</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>Perl</td>
<td>Unknown</td>
</tr>
<tr>
<td>SVK</td>
<td>Unix-like, Windows, Mac OS X</td>
<td>Java</td>
<td>HTTP, custom over ssh, custom</td>
</tr>
<tr>
<td>Synergy</td>
<td>Linux, Windows, Unix-like</td>
<td>C++ and C#</td>
<td>SOAP over HTTP or HTTPS</td>
</tr>
<tr>
<td>Team Foundation Server</td>
<td>Server: Windows Server 2003 or later, Windows 7 or 8 (for Express edition only); Clients: Windows and Web included</td>
<td>C#</td>
<td>HTTP, HTTPS</td>
</tr>
<tr>
<td>Vault</td>
<td>Unix-like, Linux, Windows</td>
<td>C++</td>
<td>NFS</td>
</tr>
<tr>
<td>Vesta</td>
<td>Tru64, Linux</td>
<td>C</td>
<td>SMB, DCOM</td>
</tr>
<tr>
<td>Visual SourceSafe</td>
<td>Windows</td>
<td></td>
<td>Network protocols</td>
</tr>
</tbody>
</table>

As this is a company thesis and they use the IBM’s rational ClearCase, so all the discussions and evaluation will be on ClearCase on the following chapters.
2 | Understanding VOBs in ClearCase

Data structure of ClearCase:

Figure A1 shows a development environment managed by ClearCase. At its heart is a permanent, secure data repository. It contains data that is shared by all users: this includes current and historical versions of source files, along with *derived objects* built from the sources by compilers, linkers, and so on. In addition, the repository stores detailed “accounting” data on the development process itself: who created a particular version (and when and why), what versions of sources went into a particular build, and other relevant information.
Only ClearCase commands can modify the permanent data repository. This ensures orderly evolution of the repository and minimizes the likelihood of accidental damage or malicious destruction. Conceptually, the data repository is a globally accessible, central resource. The implementation, however, is modular: each source (sub) tree can be a separate *versioned object base (VOB)*. VOBs can be distributed throughout a local area network, accessed independently or linked into a single logical tree. To system administrators, modularity means flexibility; it facilitates load-balancing in the short term, and enables easy expansion of the data repository over the long term.

*Version control in ClearCase:*
The most basic requirement for a software *configuration management* system is *version control* maintaining multiple versions of software development objects. Traditional version-control systems handle text files only; ClearCase manages *all* software development objects: *any* kind of file, and directories and links, as well.

Versions of text files are stored efficiently as *deltas*, much like SCCS or RCS versions. Versions of non-text files are also stored efficiently, using data compression. Version control of directories enables the tracking of changes to the *organization* of the source code base, which is just as important as changes to the contents of individual files. Such changes include creation of new files, renaming of files, and even major source tree “cleanups”.

*Versioned object bases (VOBs):*
ClearCase development data is organized into any number of *versioned object bases (VOBs)*. Each VOB provides permanent storage for all the historical versions of all the source objects in a particular directory tree. As seen through a ClearCase *view*, a VOB seems to be a standard directory tree the “right” versions of the development objects appear, and all other versions are hidden (Figure A2).

![Figure: A2: VOBs Appear in Views as Ordinary Directory Trees](image-url)
A version-controlled object in a VOB is called an *element*; its versions are organized into a *version tree* structure, with *branches* and sub-branches (Figure A3). As this figure shows, branches have user defined names, typically chosen to indicate their role in the development process. All versions have integer ID numbers; important versions can be assigned *version labels*, to indicate development milestones *for example*, a product release.

![Figure A3: Version Tree of an Individual Element](image)

**Parallel development:**

Each (sub) branch in an element’s version tree represents an independent “line of development”. This enables parallel development, creating and maintaining multiple variants of a software system concurrently. Creation of a variant might be a major project (porting an application to a new platform), or a minor detour (fixing a bug; creating a “special release” for an important customer). The overall ClearCase parallel development strategy is as follows:

- **Establish a base level**— Development work on a new variant begins with a consistent set of source versions, identified (*for example*) by a common version label.
- **Use dedicated branches** — All changes for a particular variant are made on newly created branches with a common name.
- **Isolate changes in views**— Development work for a particular variant takes place in one or more views that are configured to “see” the versions on the dedicated branches.

*For example*, changes to several source files might be required to fix bug #819, which was reported in Release 2.6. For each file element, the changes are made on a new branch (named *fix819*), created at the “baseline” version (labeled *RLS2.6*). The view in which a user works to fix the bug sees the *fix819* branch versions, or else “falls back” to the baseline *RLS2.6* version.
(“View 1” in Figure A4). For contrast, this figure also illustrates another view, configured to select different versions of the same file elements.

This strategy enables any number of views and thus any number of development projects to be active concurrently. All the views access the required source versions from the shared data repository.

Figure A4: Parallel Development

*Merger branches*
There is an additional important aspect of the ClearCase parallel development strategy. Work performed on sub-branches should periodically be reintegrated (merged) into the main branch, the principal line of development. ClearCase includes tools that automate this process.
**Extended namespace**

Most of the time, a user needs just the one version of an element that appears in his view. In some situations, however, he needs convenient access to other versions. Examples include merging the changes made on a sub-branch into the main branch, and searching all the versions of an element for an old phrasing of an error message.

ClearCase makes access to historical versions easy, by extending the standard file/directory namespace. In essence, the entire version tree of every element is embedded under its standard pathname. Most of the time, the version tree remains hidden, but special version extended pathnames allow any program to access any (or all) of an element’s versions (Figure A5).

![Figure A5: Version-Extended Pathnames](image)

**Meta-data annotations**

To supplement the information automatically captured by ClearCase, users can explicitly annotate file system objects. Such annotations are termed meta-data. The hyperlinks and merge arrows discussed just above are one form of meta-data. Attributes provide yet another annotation facility, in the form of name/value pairs. For example, an attribute named Comment Density might be attached to each version of a source file, to indicate how well the code is commented. Each such attribute might have the value "unacceptable", "low", "medium", or "high".

**ClearCase Interfaces**

ClearCase has both a command-line interface (CLI) and a graphical user interface (GUI). The CLI is implemented as a set of executables, stored in /usr/atria/bin. (Each user should add this directory to his or her search path.) The “first among equals” of the CLI utilities is cleartool; through a set of subcommands, it provides the functions performed most often by users: checkout, checkin, list history, display version with annotations, and so on.
The ClearCase GUI includes several point-and-click programs:

- `xclearcase` provides a “master control panel” that is both easy to use and thoroughly customizable. Users can examine and select both their file system data and ClearCase metadata, with a variety of browsers.
- `xlsvtree` displays the version tree of an element, making it easy both to determine how an element has evolved, and to select particular versions for comparison or merging.
- `xcleardiff` is a flexible tool for comparing and/or merging the contents of multiple versions of an element, or any other files.

Figure A6: ClearCase Graphical User Interface
Many version-control systems require users to perform their day-to-day work on copies of data, only occasionally accessing the permanent data repository. ClearCase allows users to work directly with the repository: that is, directly with VOBs. Direct access is implemented coherently and securely in the multiple-user, multiple-host environment by combining several mechanisms:

- **View context** — Any program, not just a ClearCase program, can read a VOB’s data. But a program must use a ClearCase *view* to access a VOB; otherwise, the VOB appears to be empty. Through the view, the VOB appears to be a standard directory tree.

- **Client programs** — A VOB can be modified only by special ClearCase client programs. Most version-control operations are implemented by `cleartool` (command-line interface), and by (graphical user interface). Audited builds are performed with `clearmake` and `clearaudit`.

- **VOB activation** — Typically, a VOB is located on a remote host, rather than on the user’s own host. A VOB is made available on the user’s host by *activating* it there, through operating system networking facilities. *For example*, on a UNIX system, a VOB is activated by mounting it as a file system of type MVFS — ClearCase’s *multi-version file system* type.

![Figure A7: How ClearCase Users Access a VOB](image)
**VOB Data Structures**

A VOB is implemented as a *VOB storage directory*, a directory tree whose principal contents are a set of storage pools and an embedded database.

![Figure A8: VOB Storage Directory](image)

**VOB Storage Pools**

A *VOB storage pool* is a subdirectory that stores users’ file system data. Some storage pools provide a repository for historical and currently-used versions of source files. Other storage pools provide a repository for object modules, executables, and other *derived objects* created during the development process. Each VOB is created with an initial set of pools, located within the VOB storage directory. These can be supplemented (or replaced) with pools located on the same disk, on another local disk, or on a remote host. This affords administrators great flexibility, enabling data storage to be placed on hosts where ClearCase itself is not installed (*for example*, a very fast file server machine).

ClearCase can store individual versions of files in several ways, using such techniques as data compression and line-by-line *deltas*. The data storage/retrieval facility is extensible users can supply their own *type managers*, implementing customized methods for storing and retrieving development data.

**VOB Database**

Each VOB has a *VOB database*, a subdirectory containing information managed by the database management system embedded in ClearCase. The VOB database stores version-control information, along with a wealth of other information, includes:

- user-defined annotations on source file versions
• complete “bill-of-materials” records of software builds
• event records that chronicle the changes that users have made to the VOB: Creation of new versions, adding of annotations, renaming of source files, and so on
This information is termed meta-data, to distinguish it from file system data.

Elements, Branches, and Versions
Each ClearCase VOB stores version-controlled file system objects, termed elements. An element is a file or directory for which ClearCase maintains multiple versions. The versions of an element are logically organized into a hierarchical version tree, which can include multiple branches and sub-branches.

Figure A9: Version Trees of ClearCase Elements

Some elements may have version trees with a single branch the versions form a simple linear sequence. But typically, users define additional branches in some of the elements, in order to isolate work on such tasks as bug fixing, code reorganization, experimentation, and platform-specific development.

Figure A9 illustrates several features of ClearCase version trees:
• Each element is created with a single branch, named main, which has an empty version, numbered 0.
• ClearCase automatically assigns integer version numbers to versions. Each version can also have one or more user-defined version labels (for example, REL1, REL2_BETA, REL2).
• One or more branches can be created at any version, each with a user-defined name. (Branches are not numbered in any way.)
• ClearCase supports multiple branching levels.
• Version 0 on a branch is identical to the version at the branch point.
3 | Systems design tools

- **Solaris 10** – was the OS on which the tool is developed
  OS family is UNIX, Programmed in C. Available in both open source and closed source. Work both in Server and Clients. The supported platforms are SPARC, IA-32, X86-64, Power PC. Interfaces are CDE, GNOME.

- **Perl** – was the programming language to code for retrieving data
  Perl is a platform independent, high level, general purpose, interpreted, dynamic programming language. On the other hand it can contain UNIX command in it and in this case it is not platform independent. So, it is a language which can run through cross platform OS. The different paradigms are: multi-paradigm, functional, imperative, Object oriented (class based), reflective, procedural and generic.

- **MySQL** – was used to store all databases
  MySQL is the most used RDBMS. It acts as a multi-server provides multiple user access to a number of databases. It is also a good choice for Web applications.

- **SQL** – is used to make all queries, insertion and modification
  SQL is the special purpose query language for managing data in RDBMS. It is originally based on tuple relation. The scopes include insertion, query, update and deletion of data. Schema creation and modification, data access control are the features also.

- **PHP** – was used to develop the Web application
  PHP is an open source server side scripting language designed for Web development to produce dynamic Web pages. It can easily be embedded into HTML so that it does not need to call other external files to process data. The code is interpreted by a Web server with a PHP processor module which generates the resulting Web page. It also has evolved to include a command-line interface capability and can be used in standalone graphical applications. PHP can be deployed on most Web servers and also as a standalone shell on almost every operating system and platform, free of charge.
Chapter 9 | References


[5] Internal company documentation on “Error ticket Writing”.

[6] Internal company documentation on “ISP Codes”.

[7] Internal company documentation on “Correction Handling”.

[8] Relational ClearCase documentation – model and data, IBM.


