SOFTWARE QUALITY ASSURANCE IN A REMOTE CLIENT/CONTRACTOR CONTEXT

A thesis submitted in fulfilment of the

requirements for the degree of

MASTER OF SCIENCE

of

RHODES UNIVERSITY

By

ANGUS HUGH BLACK

DECEMBER 2005

Abstract

With the reliance on information technology and the software that this technology utilizes increasing every day, it is of paramount importance that software developed be of an acceptable quality. This quality can be achieved through the utilization of various software engineering standards and guidelines. The question is, to what extent do these standards and guidelines need to be utilized and how are these standards and guidelines implemented?

This research focuses on how guidelines developed by standardization bodies and the unified process developed by Rational can be integrated to achieve a suitable process and version control system within the context of a remote client/contractor small team environment.

Acknowledgements

I would like to gratefully thank the following people for their support and help during the course of this work:

- My supervisor Prof Richard Foss for the guidance and support he gave me throughout this work and all the members of the Audio Engineering Group at Rhodes University.
- The Department of Computer Science for supplying all the equipment and resources required for conducting this research.
- Telkom SA for the bursary I received for this work and all the other sponsors of the Centre of Excellence program.
- Harold Okai-Tettey my lab partner, for all the technical support and help he gave me during the implementation of this work.
- Madeleine Shama for her support and giving me the inspiration I needed during this work.
- My parents for all the support and guidance they gave me throughout my university career.

Table of Contents

List of Figures

List of Tables

Chapter 1 – Introduction

The information technology industry is an ever changing industry in which the demand for quality placed on its software products is increasing rapidly. This quality is important to both the users and the developers. Users require software that will provide them with exactly what they require, and developers must ensure that their products are of a high level of quality, to ensure the survival of their organization in this growing industry.

The key to improving the quality of software developed by an organization lies in improving the processes that are followed for developing software [Nienaber R. & Cloete E. 2003]. These processes can range from activities involving project management, configuration, and change management to implementation or verification and validation. For an organization to gain a certain software standard accreditation, such as that issued by the International Organization for Standardization (ISO), the processes that are involved in conducting the software development effort are assessed. These processes and the organization's process maturity is assessed against a set of defined levels, such as those set forward by the Capability Maturity Model (CMM). In essence, for an organization to ensure that their products are of a high level of quality, they need to embark on a drive for process improvement. Hence quality assurance is not just about ensuring that the final delivered product is of a high standard, but that the processes involved in creating that product are highly effective.

1.1. The Audio Engineering Group

The Audio Engineering Group (AEG) at Rhodes University is a specialized research group within the Department of Computer Science. They engage in contract work for overseas clients. They are a small team of professionals, not all located in the same geographic location. Their need for a process improvement strategy brought about the motivation for this research. The fact that the AEG is a small team of remotely located team members who are conducting work for overseas clients added a unique aspect to this research and the process improvement strategy.

1.2. Process Improvement Strategy

The scope of research for this thesis encompassed areas such as project management, requirements documentation, code documentation, verification and validation, configuration and change management, project cost management, and an overall software development methodology. The particular problem focus of this research was the selection of the mechanisms to aid in improving the AEG's software development process. To solve this problem, a research methodology had to be defined.

The methodology that was followed to resolve this problem and find possible tools or approaches that could lead to process improvement involved the following:

- Looking at the current Quality Assurance techniques for the individual processes.
- Examining how these techniques work in the context of a remote client/contractor.
- Providing enhanced procedures, and if necessary a sufficient tool for this context.

After conducting this initial research methodology it became distinctly clear what areas of the AEG software development process could be improved to aid in achieving quality assurance in this context.

1.3. Process Improvement Areas

As the AEG is a small team working in different locations and conducting contract work for overseas clients, the need for remote process control and the ability to work seamlessly on this contract work irrespective of their geographic location was identified. For this a strategy was devised for the creation of a version management system with integrated status reporting. The system promoted a repeatable process and allowed for remote access, thereby improving the AEG's overall software development process.

Throughout this thesis the major components of this system and the motivation for their integration will be discussed. Each of the major components of the system have an individual chapter devoted to them.

Chapter 2 describes the context in which the AEG conducts their contracts and the approach they take to conducting them. Various tools and methodologies are discussed that could potentially aid their software development process. The requirements for a holistic solution, integrating the remote process control and version management system, with built in status reporting are defined.

Chapter 3 examines issues pertaining to process management and looks at the ISO structure and the assessments done for a certification. The Rational Unified Process (RUP) is introduced, the architecture of the process is detailed, and the manner in which this process could potentially improve the AEG software development process is discussed.

Chapter 4 looks at costing techniques, the Constrictive Cost Model (COCOMO) is examined, and the inner workings of this costing model are described. Why and how this process was utilized by the AEG is discussed.

Chapter 5 discusses the process of configuration and change management, and takes a particular look at version management tools and what they should offer. Five version management tools are discussed and the motivation for selecting the particular version management tool for the system developed is provided.

Chapter 6 provides an in depth discussion into the system developed from this research. The Project Process Control and Versioning System (PPCVS) is introduced the motivation for the creation of this application is given. The system is analyzed and designed and a detailed discussion is given into the applications implementation in C#.

Chapter 7 concludes this research with an evaluation of the system developed for the AEG against the requirements given in Chapter 2 and lists the future work that could extend this research.

Chapter 2 - Remote Client Access Problem and Current Solutions

The sharing of electronic information and making it accessible from remote locations has become common practise with the use of information technology. Project status information, project artefacts, and documents are some of the types of electronic information that can be accessed from remote locations to enhance the project management process. There are numerous techniques and approaches to sharing this information and what follows is a description of a particular context in which the sharing of such information was required.

2.1. Context

The Audio Engineering Group (AEG) at Rhodes University is a specialized group within the Department of Computer Science that focuses its research on the utilization of information technology in the field of Audio Engineering. The AEG was formed in 1987 and initially focussed its research on the transfer of Audio and Musical Instrument Data Interface (MIDI) data over Ethernet. In 1997 their research focus changed towards Firewire and the use of Firewire in the connection management of audio devices.

Since 1999 the AEG has been involved in the implementation of projects for overseas clients/contractors in their research area.

The AEG itself is made up of five main team members, while postgraduates from the department are used occasionally to conduct smaller parts of the contract work. Three of the team members are actually located in the Department of Computer Science, with the remaining two being located in different geographic regions.

Each one of the contracts the AEG conducts typically last about two to four months. The requirements for these projects are obtained from the client/contractor, and have never changed, so there is no 'scope creep' whatsoever.

2.2. Current Approach

Following is a detailed description of some of the key aspects of the process the AEG currently follows to complete a contract.

2.2.1 Communication Channels

Various mediums of communication are utilized in conveying information between the team members and the clients/contractors. Given below are the mediums utilized and a description of their purposes:

- Email – is utilized on a regular basis for communication with both the team members and clients/contractors for the relaying of status information and transfer of relatively small project artefacts.
- Phone Calls – are utilized infrequently for communication between clients/contractors for the relaying of urgent information. Phone calls between the team members are utilized on a regular basis for the transfer of important information pertaining to any project.
- Meetings at Conferences – are conducted whenever possible at Audio Engineering Society Conferences, where team members may be presenting papers or demonstrating systems.
- Visits – are conducted occasionally when a member of the client's organization will visit the department to check on the status of a completed, current or future project.
- File Transfer Protocol (FTP) – is utilized by members of the team and client/contractors to obtain project artefacts. This is being used as a medium of disseminating artefacts and receiving artefacts from remote locations.

2.2.2 Project Costing

As all the work done for the overseas clients/contractors involved tendering for projects, the AEG had to develop a costing procedure in order to create an estimate of how much too quote the client/contractor.

The current procedure is a combination of the two costing techniques known formally as expert judgment and estimation by analogy. Expert judgment involves the consensus of domain experts on the cost of a project, and analogy involves estimating based on previously conducted similar projects [Sommerville I. 1995].

What normally happens is that team members come to a consensus on the cost of the project, or the cost of the project is based on the costing of a previous project done before. These estimates are then relayed to the client/contractor for the tendering of the project.

2.2.3 Process

The basic process that the AEG follows for the development of projects is the traditional systems development life cycle, similar to the waterfall development model [Pollice G. et al. 2003]. Below is a figure showing the basic phases of the life cycle the AEG follows:

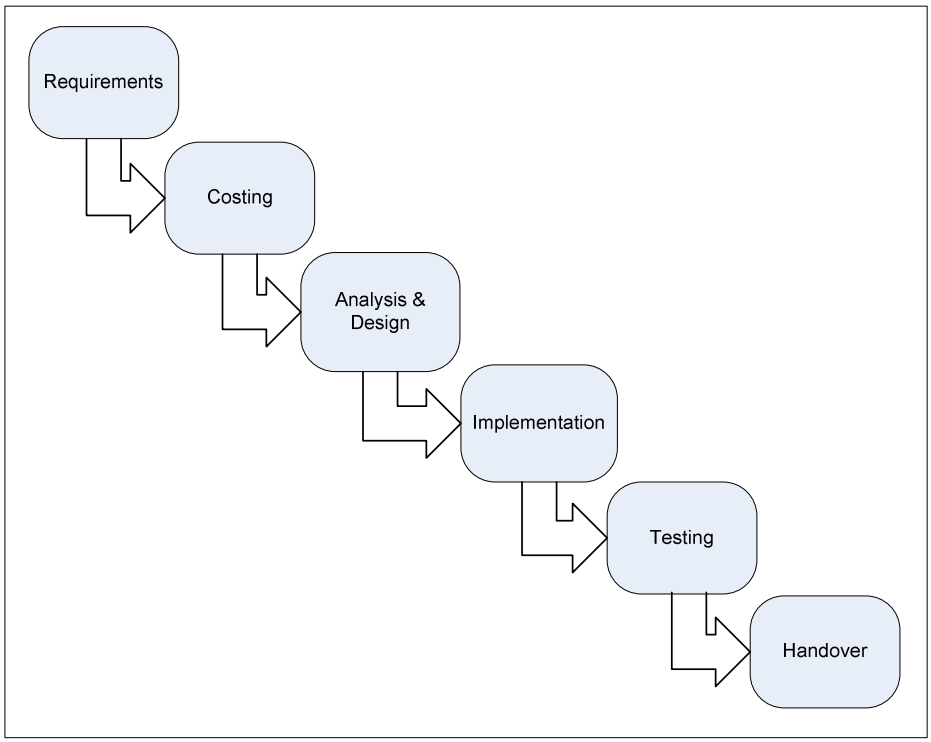


Figure 2.1 : The AEG Software Development Life Cycle

The traditional systems development life cycle involves finishing each one of the phases before moving onto the next one [Hoffer J.A. et al. 2002]. This is how each one of the phases conducted by AEG is carried out, and after each phase is complete, a deliverable is produced. As there is a minimal amount of scope creep, this process could be conducted by the AEG without the process breaking down or experiencing too many problems.

The analysis and design phase of this life cycle process comprises object-oriented analysis. All elements of the systems are modelled as use case diagrams, object models, textual scenarios and sequence diagrams.

2.2.4 Version Control

Currently there are two methods of version control in place, neither of which is utilized by all team members. These two methods are Microsoft Source Safe and an FTP Server.

2.2.4.1 Source Safe

One of the developers on the team utilized Microsoft Source Safe while implementing a project in C# using the Visual Studio .NET development environment. Source Safe was utilized, as it tied in very easily with the development environment. The Computer Science department has an up and running Microsoft Source Safe server from which weekly backups are created.

2.2.4.2 Linux FTP Server

The AEG currently utilizes a Linux FTP Server to store zipped tarballs which contain all implementation and user documentation for the systems developed from contract work. The server provides access to all the team members and accounts are given to clients/contractors to provide them with the ability to download their required material from any contract work conducted by the AEG.

All remaining project artefacts compiled by the project manager are stored on his personal machine in the department. Whenever a new version of a product is released, the new version is placed on the server and the older versions are always kept.

2.3. Tools to Enhance Current Approach

After an analysis of the current approach that the AEG was following to fulfil their contracts, it became possible to determine areas of the approach that needed new tools or processes. Following is a description of the research done in determining which tools or processes were viable new solutions for use in enhancing the current approach.

2.3.1 Costing Questionnaire

It became clear that a more scientific method of costing was required by the AEG in the tendering process. What was required was the use of a standard and generally

accepted costing model, which could be reused on every project, irrespective of the nature of the project.

For this process, a costing questionnaire was developed in Microsoft Excel using the Constructive Cost Model (COCOMO) II, this questionnaire can be seen in Appendix A. The questionnaire collects information based upon project characteristics, and uses this information in a formula to calculate the project cost. This modelling technique is known as parametric or algorithmic cost modelling [Schwalbe K. 2002]. What this provided was a repeatable costing process that could be used on any project, irrespective of the nature of the project or whether a similar project had been done before.

The costing of contract work done by the AEG formed a large portion of the research done for this thesis and will be discussed in detail in a later chapter of its own.

2.3.2 Software Requirements Specification Documentation

The main aim of any project is to ensure that the product developed meets the requirements stipulated by the customer. As the AEG does contract work for overseas clients and they do not always have the opportunity to meet with the clients in person, it is important that this documentation is drawn up correctly and the clients approve the document. These requirements can be obtained in a number of different ways depending on the nature of the project. Once the requirements for a project have been obtained it is important for these requirements to be documented in a formal document. This formal document is commonly referred to as the Software Requirements Specification (SRS).

The SRS is used in many of the stages of any project. It is used in design, implementation, testing, and most importantly in the signing off of the project itself. Since the SRS is used extensively throughout the process of the project, it is important to ensure the completeness and correctness of this document.

The ISO 9000-3 standard has some brief guidelines as to what should be incorporated into the SRS [ISO 1997]. The IEEE has a recommended practice for software requirements specifications, the 830-1998 standard, which details exactly what should be included and how. The remainder of this section will discuss the ISO 9000-3 standard and the IEEE recommendations, as well as a recommendation made by Ian Sommerville [Sommerville I. 1995], [ISO 1997].

2.3.2.1 ISO 9000-3

Most of the ISO 9000-3 standards are tailored towards purchased software and not software developed for one individual customer. Nevertheless, the guidelines set out are still valuable. The guidelines state that the requirements specification should have the following [Oskarsson O & Glass R.L. 1996]:

- The main functional requirements of the product which must be "complete and unambiguous".
- Numerous other characteristics with regard to performance, safety, reliability, security, privacy and interfaces. All of them should be related to what the need of the purchaser may be.

Other than the above points, the standard does inform the reader to some extent what needs to be included, but not a sufficient amount. However the following two recommendations detail exactly what should be in the SRS.

2.3.2.2 The IEEE Recommended Practice for Software Requirements Specification 803-1998

The IEEE recommends that the SRS be written by both a member of the development team and the customer. For obvious reasons this is not always possible and usually the SRS will be written by a member of the development team. The recommendation states that the following issues should be covered by the SRS [IEEE 1998]:

- Functionality – The purpose of the software.
- External Interfaces – How exactly the software interacts will outside sources such as other systems and users.

- Performance – Exactly what is expected of the software on a performance level.
- Attributes – Details the portability, correctness, maintainability and security issues.
- Design Constraints Imposed on Implementation – Such as standards which have to be adhered to, or development languages that must be used or even operating system environments.

The IEEE also recommends that the SRS poses the following characteristics in order to ensure it is of high quality [IEEE 1998]:

- Correct – All the requirements listed in the SRS must be requirements that the system will meet.
- Unambiguous – All the requirements within the SRS must have only one potential interpretation.
- Complete – The SRS must list all major requirements, all responses must be defined and all figures must have labels and references.
- Consistent – The SRS must agree with all the other documentation developed for the project.
- Ranked for importance and/or stability – The importance and stability of each requirement must be stated.
- Verifiable – A method of checking whether the software will meet the requirement must be defined.
- Modifiable – The structure of the document must allow itself to ease of modification.
- Traceable – All requirements must be clearly stated and it must be possible to easily cross reference these requirements from another document.

The IEEE further recommends the parts of the document which should be included in the SRS. These are given below in Table 2.1 from the IEEE 803-1998 Standard [IEEE 1998].

Table of Contents
1. Introduction
1.1 Purpose
1.2 Scope
1.3 Definitions, acronyms, and abbreviations
1.4 References
1.5 Overview
2. Overall description
2.1 Product perspective
2.2 Product functions
2.3 User characteristics
2.4 Constraints
2.5 Assumptions and dependencies
2.6 Apportioning of requirements
3. Specific requirements
Appendixes
Index

Table 2.1 - IEEE Software Requirements Specification Outline

Following is an explanation of each of the three parts of the SRS recommended by the IEEE [IEEE 1998]:

Part 1 - Introduction

The introduction provides an overview and contains the following five sections.

Part 1.1 - Purpose

This should indicate exactly what the SRS is to achieve and who the audience is.

Part 1.2 - Scope

The scope should describe exactly what is being developed, what the product is to achieve, and what benefits this product should provide.

Part 1.3 - Definitions, Acronyms, and Abbreviations

As the intended audience of the SRS is not always technically oriented any ambiguities and technical jargon should be explained.

Part 1.4 - References

Any other documents referenced in the SRS should be listed in this section.

Part 1.5 - Overview

The overview should give a brief description of the rest of the SRS and also describe the organization of the SRS.

Part 2 - Overall Description

This section does not state the exact requirements of the system but simply provides some background for the final requirements detailed in Part 2 of the SRS.

Part 2.1 - Product Perspective

This section should explain the product to be developed with regard to the part it plays in interactions within larger/smaller systems, or whether the product is a stand alone solution. This is usually depicted as a block diagram showing the product's interactions, such as the one given below.

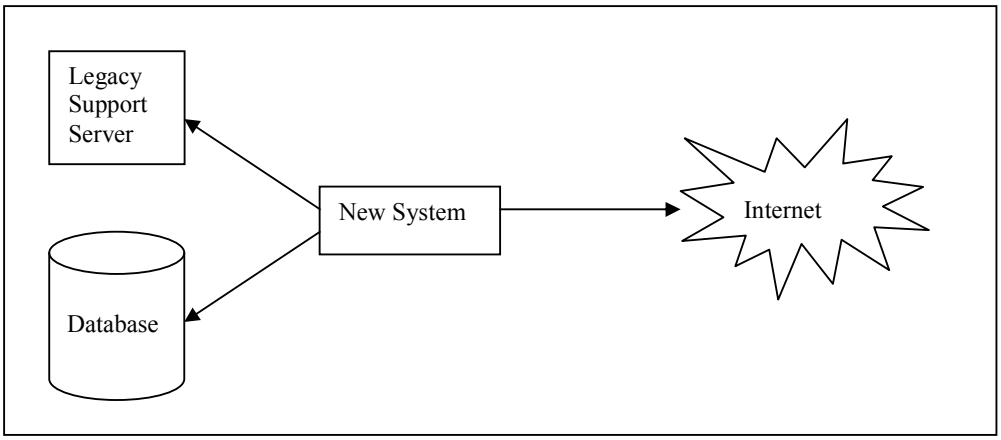


Figure 2.2 - Example Block Diagram

Along with the above, the constraints on the product should be discussed with regard to the following:

- System interfaces – Should detail which functionality will provide these interfaces.
- User interfaces – The characteristics of the interface and how it can optimize the user ability.
- Hardware interfaces – The features of each hardware interaction must be defined.
- Software interfaces – The interfaces with other products must be listed.
- Communications interfaces – The use of interfaces with communication channels such as network protocols must be defined.
- Memory – All limits on the memory constraints for the product must be defined.
- Operations – Both the normal and special operations required by the users of the product must be listed.
- Site adaptation requirements – Any required adaptations to the system that would result from a change in site should be defined.

Part 2.2 - Product Functions

This section is intended to detail the major functions of the product to be developed. This can be represented in a diagrammatic format, detailing each function and it's interaction with others.

Part 2.3 - User Characteristics

In this section the basic characteristics of a typical user should be defined, with regard to their education level, level of computer literacy and other related characteristics.

Part 2.4 - Constraints

Any considerations in the SRS which could place constraints on the development of the product should be listed in this section. They could take the form of the following, as recommended by the IEEE:

- Hardware limitations – If the product is required to run on a particular type of hardware.

- Interfaces to other applications – If the product is required to interface in a particular way with another application.
- Parallel operation – If the product is required to run in parallel with itself.
- Higher-order language requirements – If the product is required to be developed in a particular development language.
- Reliability requirements – If the product is required to provide a particular level of reliability.
- Safety and security considerations – If the product must provided various safety and security mechanisms.

Part 2.5 - Assumptions and Dependencies

Any assumptions which are made that could, if changed, lead to changes in the requirements of the product should be listed in this section.

Part 2.6 - Apportioning of Requirements

This section should detail any requirements which could be required in a subsequent version of the product.

Part 3 - Specific Requirements

In this, the final section of the SRS, the exact requirements of the product are listed under the headings to follow. This should be sufficiently detailed so as to allow designers and testers to use it when performing their required tasks.

Part 3.1 - External Interfaces

In this section the inputs and outputs of the system are detailed and the IEEE recommends it should have the following content and format:

- Name of item
- Description of purpose
- Source of input or destination of output
- Valid range, accuracy, and/or tolerance

- Units of measure
- Timing
- Relationships to other inputs/outputs
- Screen formats/organization
- Window formats/organization
- Data formats
- Command formats
- End messages.

Part 3.2 - Functions

This section should define the inputs and outputs described in the previous section and explain exactly how the inputs and outputs are processed.

Part 3.3 - Performance Requirements

Any performance requirements that the product has to poses should be listed in this section. These requirements could take the form of the number of concurrent users/terminals, or response times required by the system.

Part 3.4 - Logical Database Requirements

Any logical requirements on the use of database should be detailed in this section, such as the type of data that the database will store or the capabilities for access.

Part 3.5 - Design Constraints

Any factors which could lead to constraining the design of the product should be listed in this section. These constraints could take the form of any standards to be adhered to, or operating systems to use, or even hardware constraints.

Part 3.6 - Software Systems Attributes

The attributes of the software are very often used as requirements of the system. These attributes could take the form of the following:

- Reliability – The characteristics required to provide a particular level of reliability.
- Availability – The policies required to provide a particular level of availability, such as checkpoint, recovery and restart.
- Security – List any security mechanism required by the product to ensure the security of the system.
- Maintainability – The level of maintainability of the system should be specified it terms of the ability for maintenance.
- Portability – The ability for the product to be used on different machines or operating systems.

These are the basic points which should describe the requirements of the system within Part 3. There are numerous ways in which these requirements can be organized so as to make the SRS logical and understandable. The IEEE recommends numerous ways of ordering these requirements, some of the most logical are:

- Objects – Ordered by the classes within the system.
- Feature – Ordered by the main features provided.
- Stimulus – Ordered by the inputs into the system.
- Response – Ordered by the responses the system gives to stimulus.

Lastly, at the end of the document, a table of contents and index along with appendices should be provided. The appendices however are not always a necessity and should only be include if required.

This document recommended by the IEEE is very complex and extensive by nature, and the following recommendation is far shorter and less complex. As the above mentioned document is only a recommendation not all sections have to be included in a specific SRS.

2.3.2.3 Software Requirements Specification by Ian Sommerville

The SRS which Sommerville recommends, is far less complex, and incorporates a lot of the sections from the IEEE recommendation into larger sections [Sommerville I.

1995]. Sommerville firstly recommends that this document is not intended to be a design document and should simply state the requirements of a system and not how these requirements are met.

Sommerville also recommends that the document should be broken down into a number of chapters so as to make the process of changing the document relatively easy. He gives a generic structure for a SRS which has the following seven sections:

- Introduction – This should describe the reason for the development of the system, its major functions, as well as how the system will fit into the organization which requires the system.
- Glossary – As the reader is not intended to have technical expertise in computer science, any technical terms should be described here.
- System models – This should diagrammatically explain the system and how it fits into the environment it is required for. This is usually done via object models, dataflow models and semantic data models.
- Functional Requirements Definition – This section should, in readable terms for the customer, explain the major functions of the system. This is usually achieved using natural language or diagrams the reader would understand.
- Non-Functional Requirements Definition – This section should detail the constraints of the system in a very similar nature to the constraints section of the IEEE recommendation. Issues such as standards and performance requirements need to be detailed in this section
- System Evolution – Should any changes be anticipated with regard to the hardware, user requirements, and so on, they should be detailed in this section.
- Requirements Specification – If any further detail needs to be added to the functional or non-functional requirements they should be detailed in this final section.

Sommerville finally suggests that should any further detail be required, it should be included in further chapters within the SRS, or in the appendices section. As can be seen, the format is very similar to the recommendation of the IEEE, and Sommerville

suggests that the IEEE recommendation should be used as a reference when structuring the SRS.

The SRS document, if done correctly, proves to be a very valuable document in the remainder of any software development project. The ISO 9000-3 standard does not require a lot from the document itself, but simply states what the content should be [ISO 1997]. The IEEE 830-1998 Standard is a far more complex and detailed description of what exactly a SRS document should comprise and for this reason it was included in the artefact set built into the system as part of this research.

The best recommendation that could be made when drawing up an SRS, is to ensure that the customer has a clear understanding of what their requirements are, and that they are reflected correctly in the document itself. At the conclusion of a project, the SRS document will prove to be invaluable, as it can almost be used as the contract for the project. Thus it is in the best interests of the developers to ensure that the document is drawn up correctly.

2.3.3 Commenting Procedures

The AEG had no formal process in place for the commenting of source code created to fulfil their contracts. As some of the source code the AEG develops in their contracts is handed over to the clients/contractors for further development in their systems, the use of effective commenting was imperative. Some of the projects done by the AEG are also worked on by post-graduate students who may leave the department, and as these projects may require maintenance at some point, commenting is an essential practice.

The two automated documentation and commenting procedure products, KDOC and Doxygen where evaluated, and their outputs examined [Kang S.S. 2005], [Doxygen 2005]. Both these applications take the approach of producing HTML output of the source code, showing the structure and use of the source code. Specialized comments are placed in the source code, which the application picks out and places at appropriate places in the documentation. Given below is an example of a C header file

taken from one of the AEG projects, with the appropriate comments in place. This will be used in conjunction with both KDOC and Doxygen to produce HTML documentation for the header file.

```
/**
*     A class composed of static routines used for interpreting
*     bridge messages written to CSR addresses 0x3800 and * 0x3A00.
*     There are no application usable routines in this class
*
*     @short Nec bridge command interpreter
*     @author Melekam Tsegaye 'g98t4414@campus.ru.ac.za'
*
*     @version 0.0.1 ( $id )
*     @see Bridge
*/
class BridgeCommandHandler
{
  public:
/**
*    This routine is registed with libraw1394 using the M1394 wrapper
*    class to receive async write messages written to CSR address
*    0x3800 & 0x3A00
*
*    Its function is to recognise each of the 11 nec bridge messages
*    and route them to routines that know how to interpret them.
*
*    @return 0
*    @param  nodeid 16 bit node ID
*    @param  response set if is a response packet
*    @param  data  incoming async write data
*    @param  thefwd a pointer to an M1394 wrapper class being used by
*    an application
*/
  static int bridgecmd_handler
 (nodeid_t nodeid,int response,size_t length, unsigned char *data
 , M1394 * thefwd);
};
```

Figure 2.3 : Code Listing with Commenting Tags

The actual code for the header file is in bold italic and the rest is the comments which are used in producing the documentation. The first of the comments are used to describe the class itself and the author and version. The '@' signs denote tags which are used to describe special comments such as the author, parameters, and return types, etc. There is only one member function within this class, and above it are the required comments for the documentation of the function. The return type is specified with a tag and so are the individual parameters for the function, along with a brief description of the function itself.

Following is a brief description of each one of the two documenting applications, and what they created from the above mentioned source code. Both these applications use the same commenting style for the above class.

2.3.3.1 KDOC

KDOC is a documentation tool for C++. It generates HTML, LaTeX, or Man pages from C++ header files, and is implemented in the Linux environment. KDOC, along with being able to generate the HTML documentation for an individual header file, can also group classes into libraries and generate documentation for all classes within a project. This documentation can easily be cross referenced [Kang S.S. 2005]

The actual process for generating the HTML is quite simple. The KDOC command is called from the command line, passing the header file, and the required HTML is generated. Given below is the KDOC generated HTML page from the code segment in Figure 2.4.

class BridgeCommandHandler

Nec bridge command interpreter. More...

Annotated List Files Globals Hierarchy Index

Definition `#include <bridgech.h>`

List of all Methods

Public Static Methods

- static int **bridgecmd_handler** (nodeid_t nodeid,int response,size_t length, unsigned char *data, M1394 * thefwd)

Detailed Description

A class composed of static routines used for interpreting bridge messages written to CSR addresses 0x3800 and 0x3A00. There are no application usable routines in this class

See also: Bridge

Int **bridgecmd_handler** (nodeid_t nodeid,int response,size_t length, unsigned char *data, M1394 * thefwd) **bridgecmd_handler**

[static]

This routine is registed with libraw1394 using the M1394 wrapper class to receive async write messages written to CSR address 0x3800 & 0x3A00

Its function is to recognise each of the 11 nec bridge messages and route them to routines that now how to interpret them.

Parameters:

nodeid 16 bit node ID

response set if is a response packet

data incoming async write data

thefwd a pointer to an M1394 wrapper class being used by an application

Returns: 0

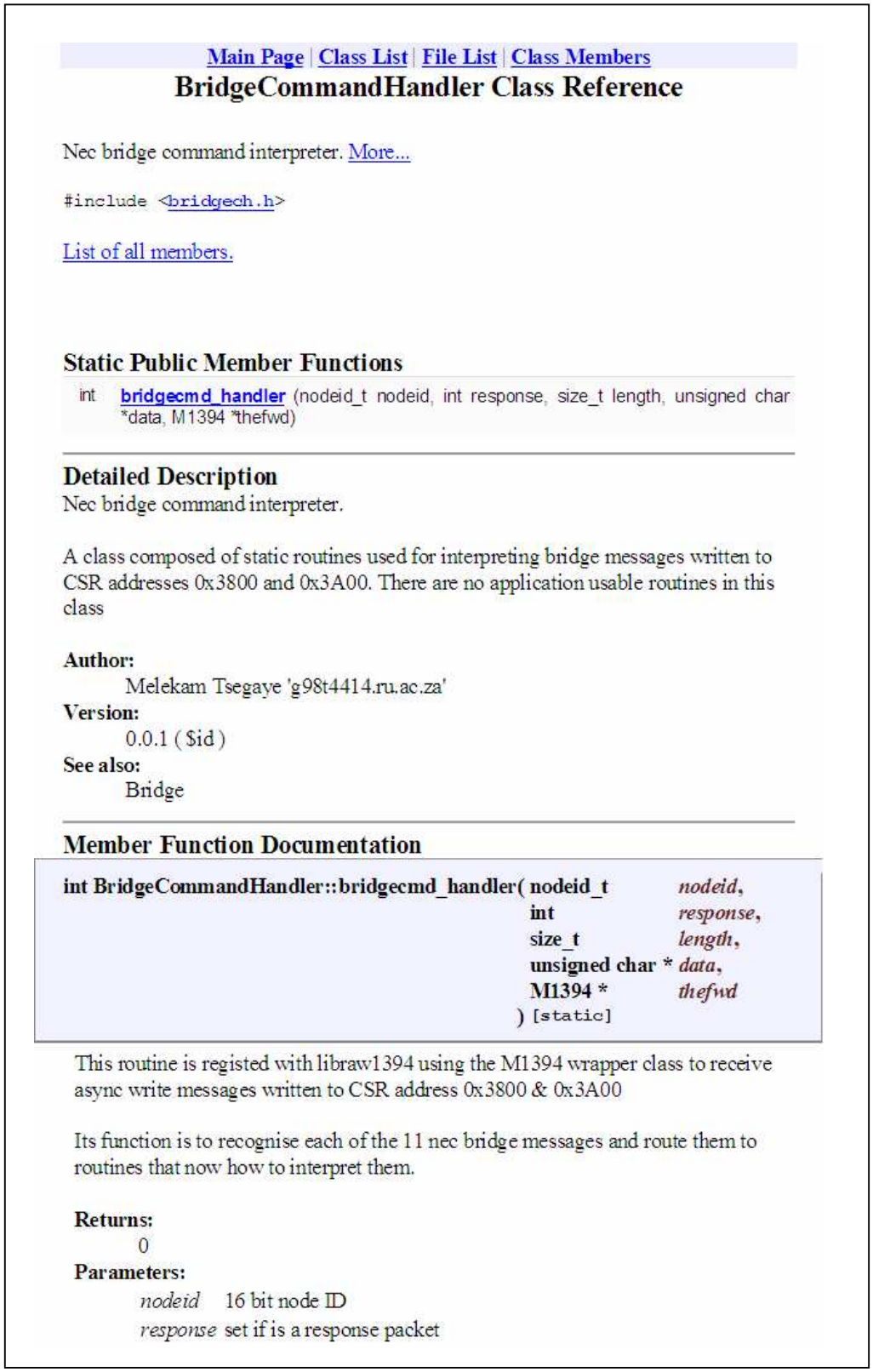
Figure 2.4 : KDOC HTML Documentation

As can be seen, the result is a nicely formatted HTML document detailing the class and the one member function.

2.3.3.2 Doxygen

Doxygen is very similar in nature to KDOC, but provides far more functionality, and runs within the Linux and Windows environments. Doxygen creates documentation for both C++ and Java source code. Along with this it can also generate simple documentation from uncommented code, if required, for quickly navigating through a large piece of code. Doxygen can also generate class diagrams in HTML (as clickable image maps) and LaTeX (as Encapsulated PostScript images). Just as for KDOC, Doxygen groups classes into libraries for easy cross referencing [Doxygen 2005].

In order to generate documentation, a configuration file first has to be set up to specify what file is to be parsed and what the output format should be. After the configuration file is set up, the application is called and the HTML documentation created. Given below in Figure 2.5 is the HTML documentation for the above mentioned code segment.



Main Page | Class List | File List | Class Members

BridgeCommandHandler Class Reference

Nec bridge command interpreter. More...

```
#include <bridgech.h>
```

List of all members.

Static Public Member Functions

int **bridgecmd_handler** (nodeid_t nodeid, int response, size_t length, unsigned char *data, M1394 *thefwd)

Detailed Description

Nec bridge command interpreter.

A class composed of static routines used for interpreting bridge messages written to CSR addresses 0x3800 and 0x3A00. There are no application usable routines in this class

Author:
Melekam Tsegaye 'g98t4414.ru.ac.za'

Version:
0.0.1 ($id)

See also:
Bridge

Member Function Documentation

int BridgeCommandHandler::bridgecmd_handler(nodeid_t *nodeid*, int *response*, size_t *length*, unsigned char * *data*, M1394 * *thefwd*) [static]

This routine is registed with libraw1394 using the M1394 wrapper class to receive async write messages written to CSR address 0x3800 & 0x3A00

Its function is to recognise each of the 11 nec bridge messages and route them to routines that now how to interpret them.

Returns:
0

Parameters:
nodeid 16 bit node ID
response set if is a response packet

Figure 2.5 : Doxygen HTML Documentation

As can be seen, the actual HTML documentation created is almost exactly the same as that produced by KDOC.

These are both very valuable tools for the creation of implementation documentation. Doxygen however provides far more functionality, as well as support for both the Microsoft Windows and Linux environments. It can produce documentation for a wider range of languages than KDOC.

2.3.4 Documentation Templates

For each one of the projects the AEG conducted, there were certain documents that were found in all of them. What was clearly required was a template document for each of these recurring documents that could be used in every project conducted by the AEG. These templates would then help aid in each and every project that the AEG conducted, irrespective of the nature of the project. The following documents where found to be recurring in each project:

- Costing Document
- Requirements Specification
- Analysis and Design Documentation
- Testing Plan
- Installation and Release Documentation

The actual templates created and why these templates where selected, will be discussed in detail in chapter six.

2.3.5 Version Control

The use of version control on all project artefacts, and not just source code, has become a standard practice in software development projects today [Pollice G., Augustine L., Lowe C., & Madhur J. 2003]. The AEG is currently using two methods of version control, and a more appropriate method was required that would allow for the unification of all project artefacts. In Chapter five the benefits offered by Version

Control Management tools and the need for such an activity in the large and complex field of software engineering will be discussed.

2.3.6 Project Management

Microsoft Project is a resource and task tracking tool which can enhance and simplify a project managers tasks within a project [Microsoft 2005a]. Microsoft Project provides a graphical representation of resource and task tracking, and is a simple and comprehensive product. The program itself provides a project wizard that helps a user set up a project schedule at start up.

The two main areas of interest the AEG had in the use of Microsoft Project was task tracking, and resource tracking. Each one of these components can provide valuable information as to whether a project runs under or over budget.

2.3.6.1 Task Tracking

Tasks in a Microsoft Project schedule can be entered via the graphical interface. A name, estimated task time, resource assigned, task dependency, and other information can be entered for each individual task. This is graphically represented in a Gantt Chart, which can be seen in Figure 2.6 below.

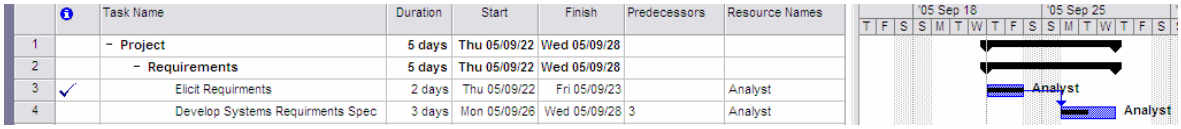


Figure 2.6 : Example Task Tracking Gantt Chart

The left hand column in Figure 2.6 represents information about the tasks. There are two tasks entered below the 'Requirements' task underneath the Project task. For each of these tasks, there is a duration set for each, which represents in days how long the task is estimated to take. There is also a Predecessor set, which means that the task can only start upon the completion of a previous task. The resource for an individual

task can be set, based upon resources entered into the schedule in the resource sheet. The resources for a project are typically team members and hardware resources.

All this information is then used to draw the Gantt Chart seen on the right of Figure 2.6. The blue bars represent the tasks, and their size is proportional to how long the task is. The calendar for the project is shown at the top of the Gantt Chart. The black bars within the blue bars represent how much of the task has been completed. This type of information has to be entered during the actual project.

The Gantt Chart has numerous other valuable graphical indicators, such as showing when a task goes over schedule, or where the milestones are in a schedule, or how much slack a particular task has.

2.3.6.2 Resource Tracking

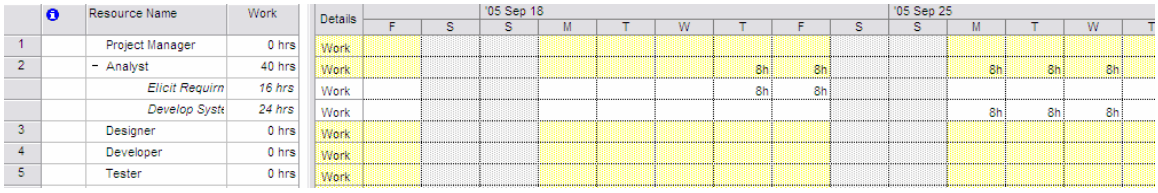
Just as the tasks can be tracked in a Microsoft Project schedule, so can the resources involved in those tasks be tracked. For each resource in a project schedule, important information is entered to detail the resource name, type, standard rate of work, and over time rate. This can be seen in Figure 2.7 below.

	Resource Name	Type	Material Label	Initials	Group	Max. Units	Std. Rate	Ovt. Rate	Cost/Use	Accrue At	Base Calendar	Code
1	Project Manager	Work		P		100%	R 400.00/hr	R 500.00/hr	R 0.00	Prorated	Standard	
2	Analyst	Work		A		100%	R 500.00/hr	R 600.00/hr	R 0.00	Prorated	Standard	
3	Designer	Work		D		100%	R 450.00/hr	R 550.00/hr	R 0.00	Prorated	Standard	
4	Developer	Work		D		100%	R 400.00/hr	R 500.00/hr	R 0.00	Prorated	Standard	
5	Tester	Work		T		100%	R 400.00/hr	R 500.00/hr	R 0.00	Prorated	Standard	

Figure 2.7 : Example Resource Tracking Sheet

As can be seen in Figure 2.7, there are five resources entered, each one of these being of type ‘work’, with a rate set. Resources can be set as either type ‘work’ or ‘material’. Work items represent team members and material items represent hardware or some type of resources other than a team member. Each one of these resources, once entered into this resource sheet, can be assigned tasks in the project schedule shown in Figure 2.6. This is done by selecting the required resource for a task from the combo box in the ‘Resource Name’ field of a particular task. Once

resources have been utilized in a particular task within a schedule, this information is reflected in a resource usage sheet, which can be seen below in Figure 2.8.



	Resource Name	Work	Details	'05 Sep 18							'05 Sep 25					
				F	S	S	M	T	W	T	F	S	S	M	T	W	T
1	Project Manager	0 hrs	Work														
2	– Analyst	40 hrs	Work							8h	8h			8h	8h	8h	
	Elicit Requirn	*16 hrs*	Work							8h	8h						
	Develop Syst	*24 hrs*	Work											8h	8h	8h	
3	Designer	0 hrs	Work														
4	Developer	0 hrs	Work														
5	Tester	0 hrs	Work														

Figure 2.8 : Example Resource Usage Sheet

The sheet shown in Figure 2.8 shows exactly what tasks a particular resource was utilized for, as well as when it was utilized in the project schedule, and exactly how many hours the resource was utilized for. This is of particular use to the AEG, since team members are paid by the hour, and being able to simply track team member hours at the end of a contract helps immensely in the project manager's role.

Microsoft Project provides a project manager with a valuable resource and task tracking tool, and it is simple and easy to use. The integration of Microsoft Project into the final versioning system will be discussed in chapter six.

2.3.7 The Rational Unified Process

The AEG software development life cycle process was discussed in section 2.2.3 and follows a traditional waterfall model. The AEG needed something that was more of an iterative and evolutionary development process, that could be easily implemented, and provided a repeatable process. The Rational Unified Process (RUP) provides an organization with a repeatable configurable software development process that can be tailored for an individual organization's needs [IBM 2005c].

RUP will be discussed in detail in chapter three.

2.3.8 Testing Procedures

The process of testing software is the cornerstone for developing quality software products and ensuring final user satisfaction. The research performed on testing for this thesis has been focussed on the final testing of a finished product. This information on testing techniques has been obtained from the book by Oskarsson and Glass that describes best practices for developing International Organization for Standardization (ISO) certified software [Oskarsson O. & Glass R.L. 1996].

There are two basic approaches to testing software, these being a static and dynamic approach. The static approach tests the system without having to run the actual software itself. Two of the frequently used static methods are peer code review and structural analysis. The dynamic approach is the most commonly used approach, and is done while the software is executed. Some of the regularly used methods are source language debug, assertion checker, performance analysis, requirements driven testing and structure driven testing. Following is a brief description of each of the above mentioned testing methods.

2.3.8.1 Static Methods

The first of these methods is the peer code review, which involves two or more people, with one of them being the actual developer. This team of testers will attempt to review the code to find any errors the developer may have made, and find any sections with faults or deviations from what is required. Structural analysis involves the use of a tool to check the correctness of the code. This may typically be viewed as a compiler or another tool that provides detailed information with regard to the potential faults a section of code may have.

2.3.8.2 Dynamic Methods

The first of the dynamic methods is the source language debug, which is used to test the system and any outputs the system may produce. This may take the form of a trace of the system, or a break point set in the system to pause and check system status.

Assertion checkers involve setting certain assertions within the code on particular variables. The checker then records any deviations from these assertions set, and reports on them after the execution of the code.

Performance analysis takes a different approach to the previously mentioned methods, and is not involved in finding errors, but more the overall performance of the system. It involves checking the performance of the system in certain areas, and recording how long certain processes take. This information is then used to find potential problems within the code with regard to performance issues.

The final two dynamic methods, requirements driven testing and structure driven testing, involve testing whether the system can meet a certain goal.

Requirements driven testing is potentially the most important of the testing methods. It involves taking the requirements provided by the customer during the analysis/design phase of the project, and ensuring that the system meets these requirements. Requirements driven testing is often referred to as black-box testing, as it tests the entire system with no regard to the inner workings of the system. Structure driven testing on the other hand, is often referred to as white-box or clear-box testing and tests the internal structure of a system. This is done by implementing test cases which are known to thoroughly test all aspects of the system.

2.3.8.3 Acceptance Testing

Following the above mentioned testing methods, the system is ready to move into the final testing phase before the final hand over of the system. This phase is known as the acceptance phase, and has a number of testing methods associated with it. These are unit testing, integration testing, system testing, independent testing, beta testing, and finally the actual acceptance test itself. The three methods which focus on the final testing of a finished product are the independent test, beta test and acceptance test.

The independent test is carried out by testers who have had nothing to do with the actual development of the system, and are deemed to have no emotional attachment

with the system, and hence will find it a lot easier to take an objective look at the system when trying to find possible faults.

The beta test is the second last of the tests to be carried out, and involves handing the system to a set of potential users, who knowing the system is not completely finished, use the system in an effort to find faults. The system is then tested in a real world environment with real world data. This test eliminates the possibility of any mismatch between the data that the developers were testing the system with, and the data that will actually be used in conjunction with the system.

The final test is the acceptance test. This test is the final phase, where the customer will sign off the project, and ensure that the system meets their every need. There should not be any problems with this test if the preceding tests were conducted properly.

2.3.8.4 Tools in the Testing Process

Numerous tools are available for automating the testing process. Some of these tools may need to be adapted to fit certain problem domains, or a tool developed specifically for the problem domain, but their use greatly aids in speeding up the testing process. Following is a description of three such tools, know as test coverage analyzers, test case managers, and test data generators.

Tools known generally as test coverage analyzers are the first of the tools that aid the test process, by informing the tester how much of the system a particular test tested. This is useful in being able to ascertain how much of the system is guaranteed to work. This is achieved by the analyzer setting certain counters in the various sections of the system, keeping track of their counts, and finally delivering a comprehensive report after the test was conducted. A particularly useful such product was found for C++, which is developed by Bullseye Testing Technology, and tracks the coverage of code within a C/C++ program statically or dynamically [Bullseye Testing Technology 2005].

Test case manager software tools are responsible for the running of tests on a system. They prove particularly useful when numerous tests need to be conducted. They manage the running of the tests, the selection of the sections to be tested, and the generation and analyzing of reports generated during the testing process. Numerous software packages are available for use in this area of testing. Applied Testing and Technology sell a product 'ApTest Manager'. This is a web based test management tool, which collects the test definition, and is responsible for the execution of tests. It can generate reports on any series of tests conducted [Applied Testing and Technology 2005]. PBSys sell a product 'Test Case Manager' (TCM) that enables test managers to configure test cases and execute test case while storing the output of any such test case. Through the wizards and Graphical User Interface (GUI) test engineers can very easily configure and manage test cases [PBSys 2005]. PassMark Software also sells a test management tool 'TestLog' that manages software testing cases through a GUI interface and allows for the creation and management of test plans and cases. This tool is designed to allow for flexibility and is able to operate with any testing methodology in any industry [PassMark Software 2005].

Test data generators are tools which can create a random set of data, populate a database, and use this database in testing a system. The data is usually generated by using data which was previously used in tests, randomly selecting data from the database to test with, or analyzing the structure of the system and generating data that would be practical for use. One such tool is developed by Grid-Tools called GT DataMaker [Grid-Tools 2005].

The above mentioned tools are not the only ones available for use. Numerous other tools are available that can aid in other aspects of the testing process, and in some instances it is found to be a lot easier for developers to design and implement their own tools. Test and Data Services are a testing consulting company that offer a broad range of testing packages and customer testing solutions beyond the scope of the above mentioned testing tools [Test and Data Services 2005].

2.3.8.5 Test Documentation

Documenting the testing phase is possibly the most important part of the testing process. It aids the developer in the fixing of errors and the client/contractor in ensuring all detected errors were removed. The test document needs to contain the following three components:

- Test plan - details exactly what is to be tested
- Test procedure - details exactly how the test is to be conducted
- Test report - details the exact running of the test and if any problems were encountered.

Test documentation enables the tracking and correction of bugs.

2.3.8.6 Recommended Practice for the AEG

The first recommendation is for the AEG to document the test process as effectively as possible. This will be achieved through the use of one of the templates created for the AEG, which will be discussed in chapter six.

With the nature of the hardware used in the AEG projects, it is suggested that a detailed description of the connections made, layout of the hardware, and necessary equipment required, be provided with the test cases. This could be done in the form of digital photographs, and would provide any potential tester with all the required information.

The two testing techniques that need to be carried out on every single project are the peer code review and requirements driven testing, and then acceptance testing to ensure the final end product meets the client/contractors needs.

2.4. Requirements for a Solution

After analyzing the AEG's approach to contract work, and after examining the possible tools that could help in aiding the project process, it was possible to create an effective requirements plan for a holistic solution. This solution would have to encompass 5 key aspects:

- Remote Process Control
- Repeatable Process
- Version Control
- Status Reporting
- Time Tracking

Following is a discussion of each of these aspects and the requirements for each.

2.4.1 Remote Process Control

As this work is done by a team of professionals not all located in the same geographic location, it was important that they could all work together, irrespective of their location. The solution would have to allow for the unification of all project artefacts into a single location, so that each individual team member could access these artefacts, modify them, and make these modifications available to the other team members.

2.4.2 Repeatable Process

The solution would have to allow for and promote the creation of a process that was repeatable on every project conducted by the AEG. As all standards certifications are focussed on the actual process involved in conducting a project, and not solely the outcome, the creation of a high quality repeatable process would lead to a quality product. For this repeatable process, all documentation would have to be taken from a pool of resources available for every single project.

2.4.3 Version Control

The solution would have to incorporate remote process control with an effective version management system, which would track versions and be flexible enough to allow the AEG team to work simultaneously on projects without any problems.

2.4.4 Status Reporting

As the contracts are done for overseas clients/contractors a suitable status reporting tool would have to be built into the system, to allow for easy access to status information by the clients/contractors.

2.4.5 Time Tracking

As the contract work done by the AEG is mostly for short two to four month contracts, the effective time management of resources and the project itself is imperative. The solution should allow for the tracking of resources and project tasks.

After each one of these key aspects where explored, it was possible to create a requirements statement for the solution:

"The integration of remote process control and version management that would allow for status reporting and time tracking"

2.5. Chapter Summary

In this chapter an investigation into the context of this research was conducted. Through the course of this investigation the context was evaluated and various investigations into tools and process improvements discussed. The potential improvements these tools and processes could give to the AEG in their contract work were examined.

The current approach taken by the AEG in conducting their contract work was discussed and issues relating to communication channels, project costing, process and version control were highlighted.

Various tools and processes where examined that could aid the AEG, these tools and processes being:

- Costing Questionnaires
- Commenting Procedures
- Documentation Templates
- Version Control
- Microsoft Project
- The Rational Unified Process
- Testing Procedures

Finally the requirement for a holistic solution that could aid the AEG in their contract work was listed and examined.

In the next chapter process management will be discussed, a product evaluated, and there will be an analysis of software process improvement through the use of the ISO standards set.

Chapter 3 – Process Management

According to the 'Cambridge Dictionaries Online' the word 'process' means *"A series of actions you take in order to achieve a result"* [Cambridge University Press 2005]. According to Sommerville, the meaning of the word process when related to software development projects is *"the set of activities and associated results which produce a software product"* [Sommerville I. 1995].

Each individual major activity within a software development project represents what is termed a process. Typically processes found within software development organizations are activities that involve many other inter-related activities. For example the following are processes:

- Project management
- Quality assurance
- Documentation
- Configuration Management
- Measurement

According to Royce there are three distinct perspectives when referring to processes [Royce W. 1999]:

- Metaprocess – which are related to processes regarding organization wide policies, procedures and best practices. The metaprocess is focussed on long term strategies.
- Macroprocess – which refer to policies, procedures and best practices related to a particular project. The macroprocess is focussed on creating an adequate abstraction of a metaprocess for a particular project.
- Microprocess – being the policies, procedures and best practices related to creating a particular project artefact. The microprocess is focussed on creating a quality artefact as quickly and as economically as possible.

The focus of this chapter will be on macro and micro processes and the use of a product which defines an organization wide macro and micro process framework.

Following is a discussion of the use of software standards as a method of process improvement.

3.1. Software Standards for Process Improvement

While the use of software standards has not been the focus of this research, it has played a vital role in selecting, understanding, and implementing the process selected for the integration of process control and versioning in the system developed.

Following is a general discussion of software standards, focussing on the standards developed by ISO. Unless otherwise referenced, all the information for this section (3.1) has been obtained from the course notes of a course entitled *"Software Engineering Standards – A framework for software process improvement"* given by the Software Process Improvement Laboratory at the South African Quality Institute and presented by Professor Alistair Walker [Software Process Improvement Laboratory 2004].

According to Walker in his course notes on software process improvement, a software standard is *"guideline documentation that reflects agreements on products, practices, or operations by nationally or internationally recognized industrial, professional, trade associations or governmental bodies"*. There exist numerous standardization bodies that are involved in the development of standards in the area of software engineering. Some of these being [Peters J.F. & Pedrycz W. 1999] :

- The Institute for Electrical and Electronic Engineers (IEEE)
- International Organization for Standardization (ISO)
- American National Standards Institute (ANSI)
- U.S. Department of Defense (DoD)
- British Standards Institute (BS)
- Institute of Electrical Engineers in the UK (IEE)
- Common Request Object Broker Architecture (CORBA)
- Object Management Group (OMG)

ISO is one of the largest developers of international standards and has a work programme that ranges from standards in the agricultural and construction field to the medical and information technology field. These standards developed by ISO aim to ensure that products and services manufactured and supplied are more efficient, safer and cleaner. If a software development company has an ISO certification, it gives them a distinct advantage in the global community, since numerous clients in the IT field require a contractor to be ISO certified [ISO 2005]. A standards certification also gives an organization the ability to assess their capability based on this certification and not solely on previously completed projects [Oskarsson O & Glass R.L. 1996]

3.1.1 International Standardization Body Structure

ISO has another partner in the international standardization community, the International Electrotechnical Commission (IEC). ISO works with the IEC in creating and disseminating their standards. In 1988 they formed a Joint Technical Committee (JTC1) which became responsible for standardization in the IT field.

In this partnership, these two groups have the following responsibilities:

- ISO – are responsible for the promotion and the development of standards to help aid in the international exchange of goods and services around the globe.
- IEC – are responsible for the preparation and publishing of the standards developed by ISO.

In Figure 3.1 below the structure of the international standardization body that is responsible for the creation of the standards in the IT field is shown.

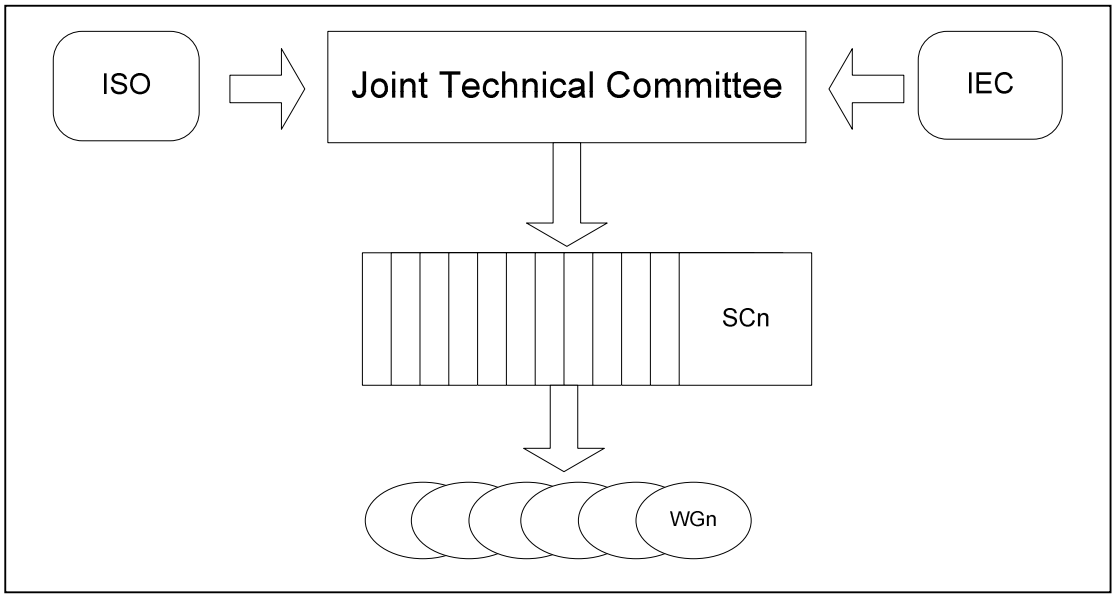


Figure 3.1 - The IT International Standardization Body Structure

As has been mentioned, the JTC1 is made up of ISO and IEC. Within the JTC1 there are numerous subcommittees (SC) which have numerous working groups (WG) associated with them. The WG's are the actual groups of people who are responsible for the development of standards in a particular field.

In Table 3.1 below the various SC's and WG's in JTC1 within their various application directions are listed. A sub committee of particular interest in the software engineering field is the SC7 which is responsible for software and systems engineering.

Application Directions	JTC1 Subcommittees and Working Groups
Application Technologies	SC 36 - Learning Technologies
Cultural and Linguistic Adaptability and User Interfaces	SC 02 - Coded Character Sets
	SC 22/WG 20 - Internationalization
	SC 35 - User Interfaces
Data Capture and Identification Systems	SC 17 - Cards and Personal Identification
	SC 31 - Automated Identification and Data Capture Techniques
Data Management Services	SC 32 - Data Management and Interchange
Document Description Languages	SC 34 - Document Description and Processing Languages
Information Interchange Media	SC 11 - Flexible Magnetic Media for Digital Data Interchange
	SC 23 - Optical Disk Cartridges for Information Interchange
Multimedia and Representation	SC 24 - Computer Graphics and Image Processing
	SC 29 - Coding of Audio, Picture, and Multimedia & Hypermedia Information
Networking and Interconnects	SC 06 - Telecommunications and Information Exchange Between Systems
	SC 25 - Interconnections of Information Technology Equipment
Office Equipment	SC 28 - Office Equipment
Programming Languages and Software Interfaces	SC 22 - Programming Languages, their Environments and Systems Software Interfaces
Security	SC 27 - IT Security Techniques
Software Engineering	SC 07 - Software and Systems Engineering
TBD	SC 37 - Biometrics

Table 3.1 - JTC1 Subcommittees and Working Groups

3.1.2 Subcommittee 7 – Software and Systems Engineering

The mandate of the JTC 1/SC7 is the *"standardization of processes, supporting tools and supporting technologies for the engineering of software products and systems"*. The SC7 has various programs of work and they are as follows:

- Software and systems engineering processes - describe standards detailing good practises regarding this area, as well as standards for assessing these practises against certain benchmarks.
- Software systems products - aid in the measuring of software quality for buyers and the sizing and documenting of these products.
- Enterprise architecture - is a partnership with the OMG, aimed at integrating IT and business systems definitions in an effort to provide software and systems engineering tools for the implementation of enterprise information systems.
- Software engineering environment - standards which are aimed at making it easier to re-use and re-deploy the data contained in software engineering environments.

- Software and system engineering formalisms - are standards detailing the formal representation and modelling of software and systems.
- Software engineering body of knowledge - is a partnership with the IEEE Computer Society in an effort to publish a technical report detailing the IEEE's Software Engineering Body of Knowledge (SWEBOK).
- Management of software assets - is a standard currently under construction detailing the management of software assets such as operating systems, development tools, and numerous other such assets owned by software and systems engineering organizations.

The main focus of all the standards developed by SC7 is on the processes that are involved in creating software. Activities such as documentation, configuration management, quality assurance, verification, and other such software development activities are all deemed to be processes.

The motivation behind focussing the standards on processes and not individual documents or artefacts is that if an organization has processes that conform to these standards, any future projects that use these processes will also conform.

The way this is achieved is by building a central repository of all the organization's processes, which would house the organization's best practises and templates for particular documents within each process. This then makes it very easy for a new project to simply acquire the required processes from the central repository and utilise it for the particular project. This concept can be seen in Figure 3.2 below:

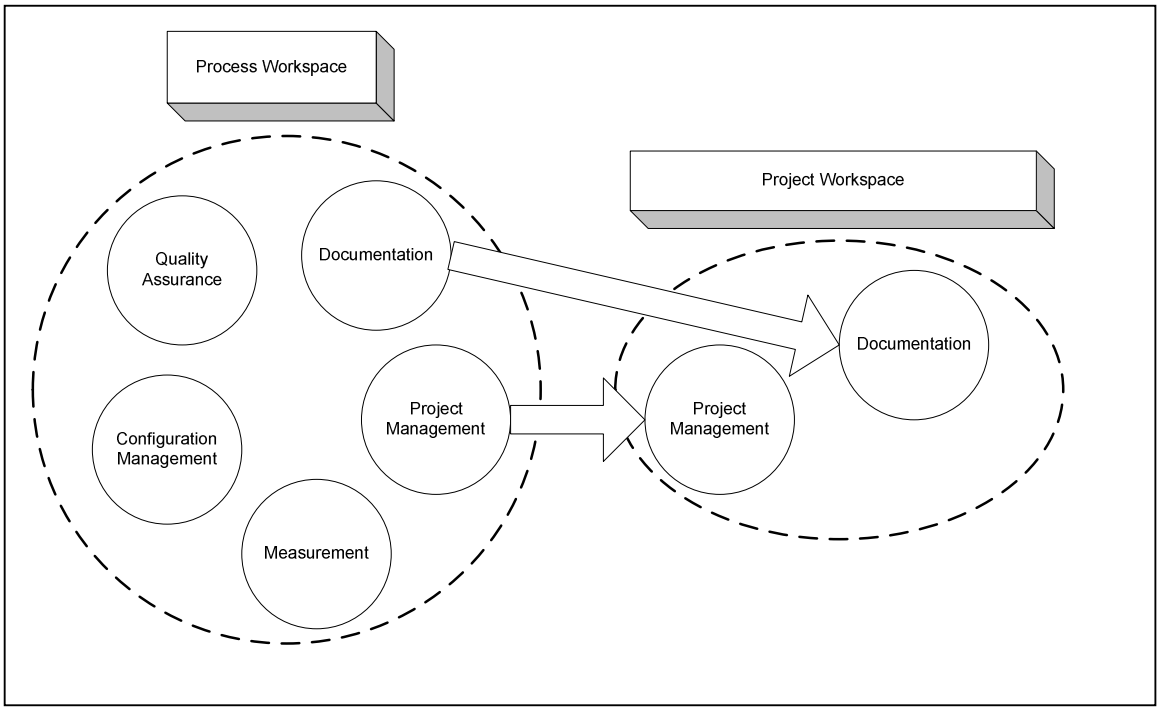


Figure 3.2 - The Process Repository

If an organization conforms to this concept of a process repository then once a new project begins, it is a simple matter of acquiring the required conformant processes from the process workspace and utilising them in the new project workspace. The end result of this is a project utilising known best practises and documents that conform to the particular standard set required. This can then lead to decreased project start-up time and make the actual project process more efficient.

The SC7 has published various standards in the Software and Systems Engineering field. Figure 3.3 below shows the standards that the SC7 has developed and the relationships between each of them.

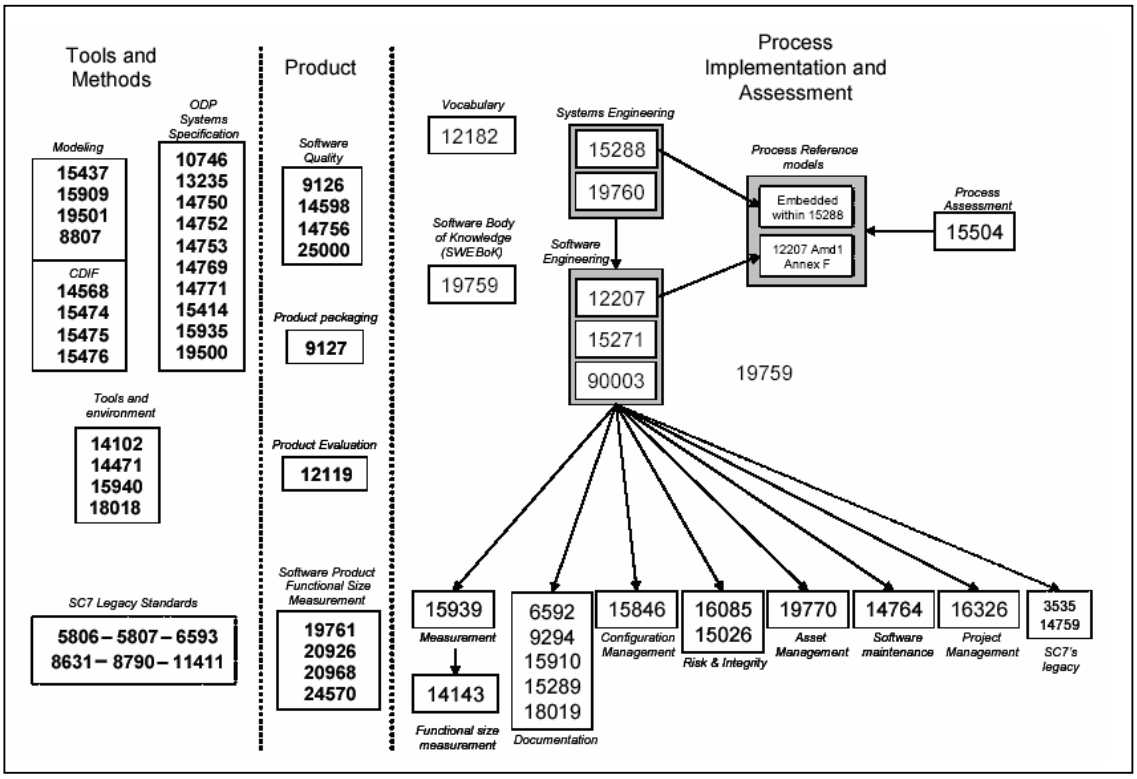


Figure 3.3 - The SC7 Standards Set

The two standards of importance when examining an organization's processes are the 15288 standard for system life cycle processes and the 12207 standard for software life cycle processes. The 12207 standard is focussed on the processes, activities, and tasks that are involved in a software life cycle while the 15288 standard is a framework for describing the life cycle of systems. As such, the 12207 standard focuses on lower level issues pertaining to software development projects. These lower level issues being creating documents, and the attributes that certain processes should contain.

The standards are developed in a structural hierarchal fashion. In Figure 3.3 the 15288 standard is higher in the hierarchy of standards than the 12207 standard and as such 15288 focuses on issues at a higher conceptual level in systems and software engineering.

In a similar way the 19760 and 15271 standards are guidelines for the implementation of their respective higher level standards, as they are directly below their respective standards in Figure 3.3.

3.1.3 Process Assessment and the Capability Maturity Model

To acquire/re-acquire an ISO certification, a process of assessment needs to be conducted according to a standard assessment process. This assessment of processes within an organization involves an examination of the processes, in conjunction with a measurement scale, a set of standards, and a method to represent the results. Process assessments are conducted by qualified assessors, and the assessor will examine each individual process within the organization for conformance to the required standard.

The 15504 standard, which can be seen in Figure 3.3 on the far right, is a standard for process assessment and is utilized in assessing an organization's capability. In Figure 3.4, the basic interactions between the standards and certifications in a process assessment can be seen.

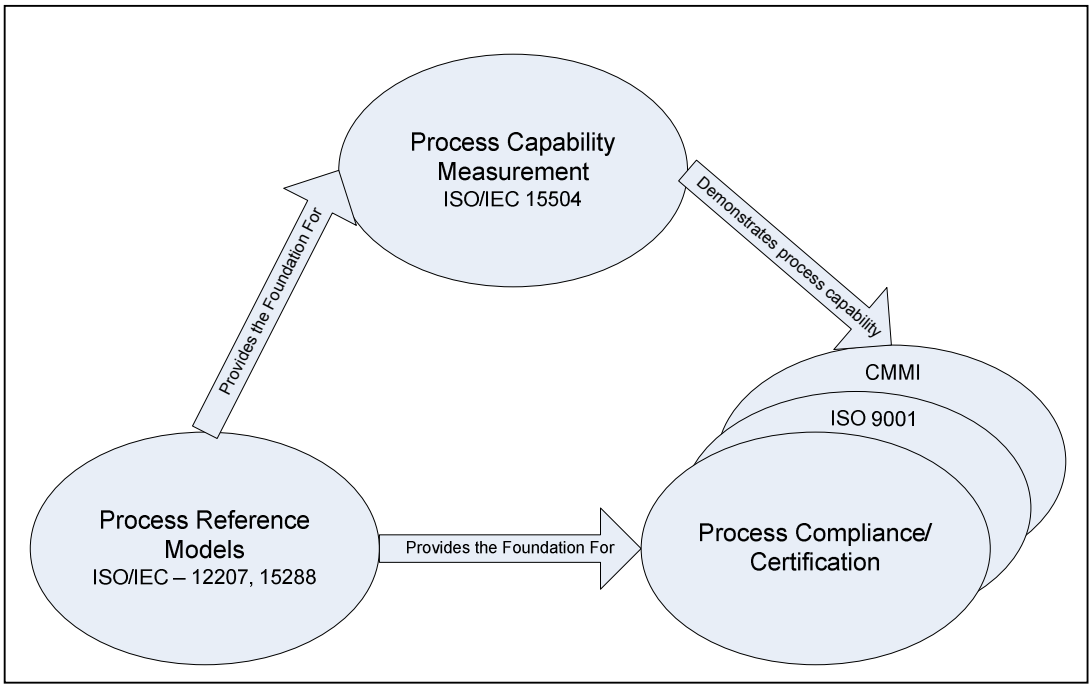


Figure 3.4 - Standards and Certifications Interactions

In Figure 3.4 the relationship between the standards and the certifications can be seen. The use of the 12207 and 15288 standards provides the foundation for the certifications such as ISO 9001, and this is achieved through the use of the 15504 standard to asses the organization's process capabilities.

When an assessment is conducted, the process assessor will rank the individual processes according to a capability scale, which usually has 6 ranked levels increasing in capability:

0) Incomplete process - The process is not complete and lacks purpose.
1) Performed process - The process is complete but there is no evidence of repeatability or control within the process.
2) Managed process - The outcome of the process conforms to required standards and the process itself is managed and monitored and tailored for best practise.
3) Established process - The process itself is well defined and tailored for individual use, and well documented and supported.
4) Predictable process - The process outcome is predictable and there exists data to monitor the performance of the process.
5) Optimised process - The process is consistently monitored for conformance to current and future needs.

Each individual process is then assessed and the attained capability level established. After this is done, it is then possible to assign a capability level for the organization as a whole.

This capability scale is very similar to the Capability Maturity Model (CMM) developed by Carnegie Mellon University's Software Engineering Institute, which defines five maturity levels:

- Initial
- Repeatable
- Defined
- Managed
- Optimizing

Each of these maturity levels, except level 1, has associated key processes, which if accomplished help prove the capability of an organization at a particular maturity level. These key processes are listed below in Table 3.2 [Manzoni L.V. & Price R.T. 2003].

Maturity Level	*Key Process Area*
2 - Repeatable	Requirements Management
	Software Project Planning
	Software Project Tracking & Oversight
	Software Subcontract Management
	Software Quality Assurance
	Software Configuration Management
3 - Defined	Organization Process Focus
	Organization Process Definition
	Training Program
	Integrated Software Management
	Software Product Engineering
	Intergroup Coordination
	Peer Reviews
4 - Managed	Software Quality Management
	Quantitative Process Management
5 - Optimizing	Process Change Management
	Technology Change Management
	Defect Prevention

Table 3.2 - Key Process Areas for Each Maturity Level

For an organization to be rated at a particular maturity level it has to be able to show that it is performing the key processes at an acceptable level. This is achieved by assessing it against the common features. The common features are used to help group and order the processes for ease of assessment. Listed below are the common features [Manzoni L.V. & Price R.T. 2003]:

- Commitment to Perform
- Ability to Perform
- Activities Performed
- Measurement and Analysis
- Verifying Implementation

If the organization can successfully prove it achieves the common features for each of the key processes, then that maturity level can be assigned to the organization.

While a certification from CMM, ISO9000, or any other standards body will help in aiding the establishment of a capable process model, it does not guarantee software quality and the organization must ensure that the focus does not shift from developing software to developing processes [Van Vliet H. 2000].

In the next section of this chapter a project process framework will be discussed and detailed as to how it can help achieve a certain level of process capability in an organization.

3.2. The Rational Unified Process

The Rational Unified Process, initially developed by Rational Software, and then bought by IBM, is one of the most widely used software development processes. It provides a very extensive level of detail for the software development process; it is well documented and very concise, providing templates and examples for numerous project artifacts [Hirsh M. 2002].

RUP, unlike many traditional waterfall development processes, follows an iterative development approach, which aids in the early detection and correction of any possible problems within a development project [Kruchten P. 2003].

The iterative development approach involves the use of a series of development disciplines in what is termed an ‘iteration’. A development project may incorporate numerous iterations for successful completion. Earlier iterations will focus on requirements, analysis, and design, while later iterations will shift their focus to development and testing. This concept can be seen below in Figure 3.5 [Kroll P. & Kruchten P. 2003].

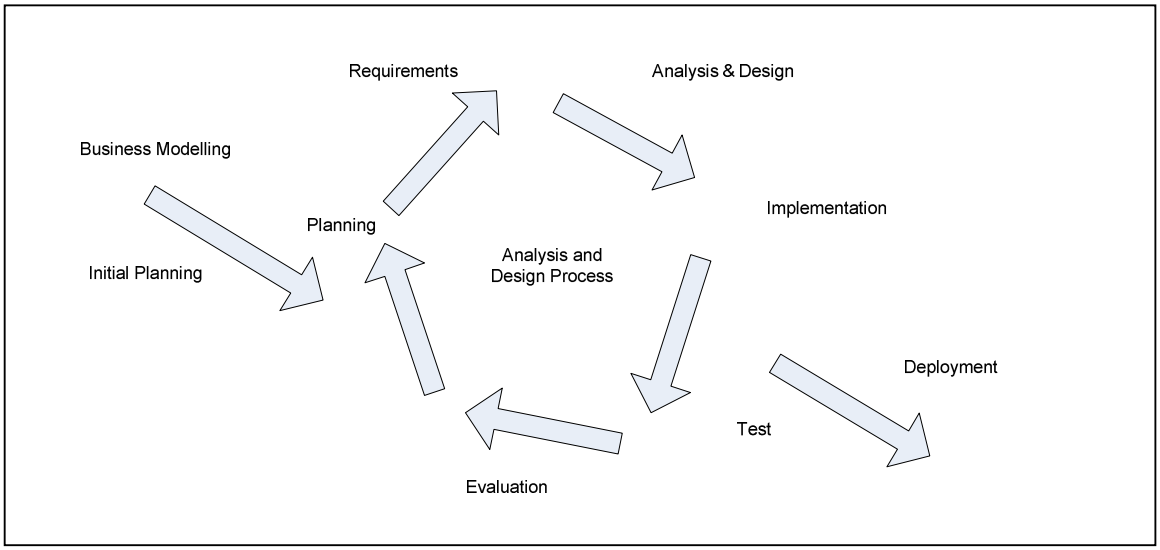


Figure 3.5 - Iterative Development in RUP

The iteration for the 'Analysis and Design' process can be seen in Figure 3.5. Each of the iterations for this process involve a little bit of each of the development disciplines found in development projects. Kroll and Kruchten suggest in their book entitled *The Rational Unified Process Made Easy – A Practitioners Guide To The RUP* that this approach to development is far superior to any waterfall approach for the following reasons [Kroll P. & Kruchten P. 2003]:

- It accommodates changing requirements
- Integration is not one "big bang" at the end of a project
- Risks are usually discovered or addressed during early integrations
- Management has a means of making tactical changes to the product
- Reuse is facilitated
- Defects can be found and corrected over several iterations
- It is a better use of project personnel
- Team members learn along the way
- The development process itself is improved and refined along the way

3.2.1 The RUP Architecture

The RUP architecture can be seen in Figure 3.6. The process itself is broken into various phases, iterations and workflows.

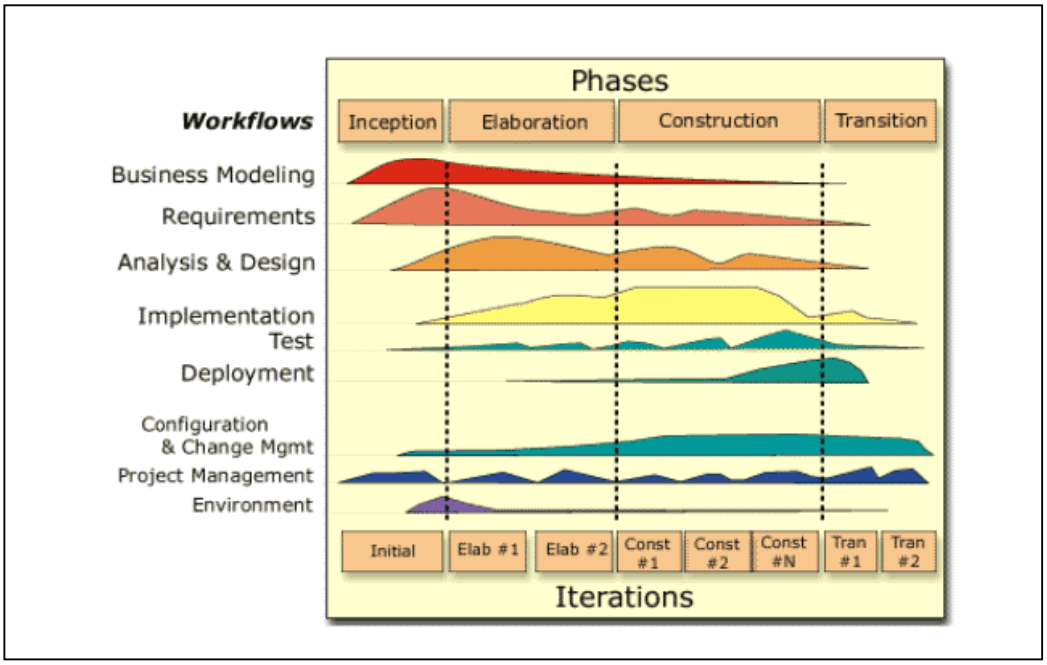


Figure 3.6 - The RUP Architecture

The horizontal and vertical axes in Figure 3.6 represent the following [Kruchten P. 2003]:

- The horizontal axis represents the life of the project and is broken into phases and iterations. Theses phases are the four main stages of the project, and iterations are the steps within each phase to complete the phase.
- The vertical axis represents the workflows. These are the main activities within the project and each workflow involves various activities, artifacts and workers.

The horizontal axis is broken into four main phases, and each of these phases represents a particular stage in the project [Kruchten P. 2003]:

- The inception phase involves defining the project as a whole and initiating plans for the progression of the project

- The elaboration phase involves building on the project plans from the inception phase and developing certain design and architecture analysis of the problem
- The construction phase involves solving the actual problem the project has set out to achieve
- The transition phase involves the handover of the end product and the correction of any problems that may arise.

As can be seen from Figure 3.6, certain workflows involve more work in certain phases. For example the implementation workflow involves more work in the construction phase than the requirements workflow and vice versa. Within each of these phases there are iterations, which are progressive iterative steps through the phase to ensure its completion.

As previously mentioned, the vertical axis represents the workflows and within each workflow various activities, artifacts, and workers are involved, which can be seen in Figure 3.7 below.

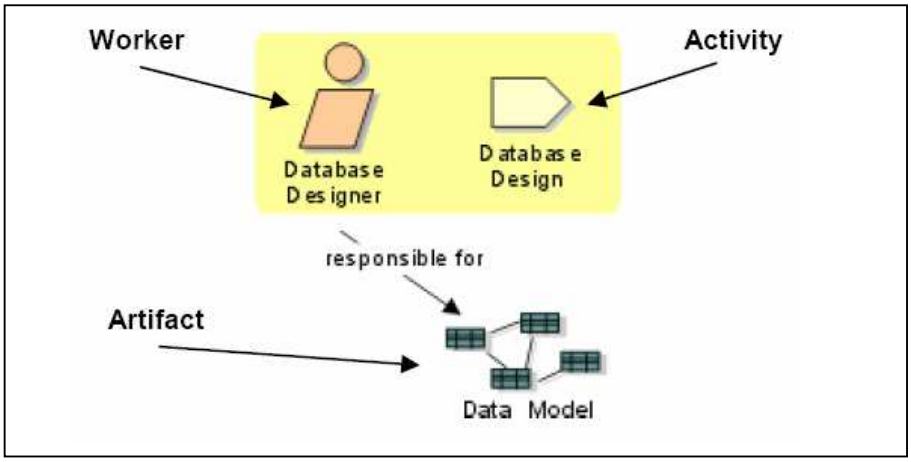


Figure 3.7 - Workers, Activities and Artefacts in RUP

The worker represents a team member responsible for a particular role in the project, in this instance the 'Database Designer'. The activity is something that the worker is required to perform to fulfill his role. In this instance the worker is required to perform the 'Database Design'. The artifact is the product of a worker accomplishing

his role and performing his activities. In this instance the 'Data Model' is required to be produced.

3.2.2 Creating a Tailored Process with RUP

In Version 2002 of RUP there were 80 major artefacts, 150 activities and 40 roles. This volume of information and activities cannot be undertaken by most small teams utilizing RUP [Hirsh M. 2002]. For this reason RUP advocates that the process needs to be tailored towards the needs of an individual organization or project [Kroll P. & Kruchten P. 2003]. This 'tailoring' is achieved through the use of the RUP Builder. The RUP Builder provides a step by step procedure for creating a tailored process tree.

The RUP builder will create this customized process based upon the selections made when creating the customized tree. Given below in Figure 3.8 is a selection panel from the RUP Builder application.

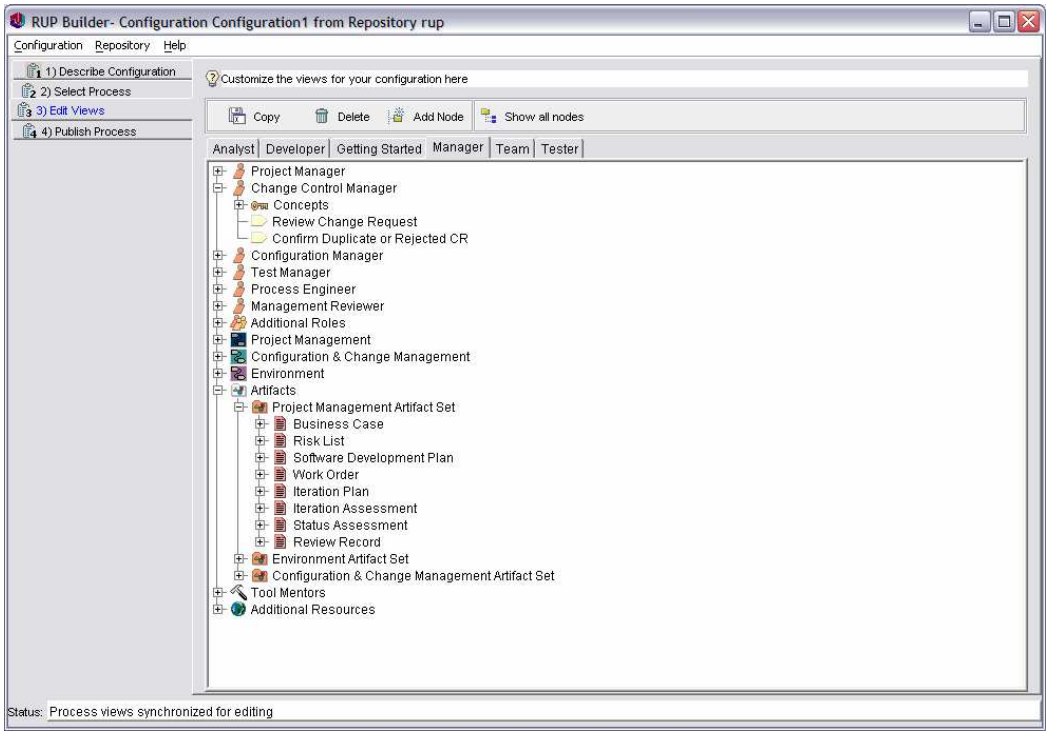


Figure 3.8 - The RUP Builder

In Figure 3.8 above the Manager Role is open. This view allows the user to select which manager roles, activities and artefacts are required by the tailored process. The selected items are then built into the customized view when the tailored process is published.

3.2.3 Utilizing a Tailored Processes

Once the tailored process has been published, it creates what could be termed a 'hyperlinked book' [Henderson-Sellers B. et al. 2000] or tree containing information pertaining to project team roles and activities. This RUP tree can be viewed through a web browser and made accessible to all members of a team, to clearly define their role and activities, as well as the necessary artefacts they are required to produce.

An example RUP tree defining the role for a system analyst can be seen below in Figure 3.9.

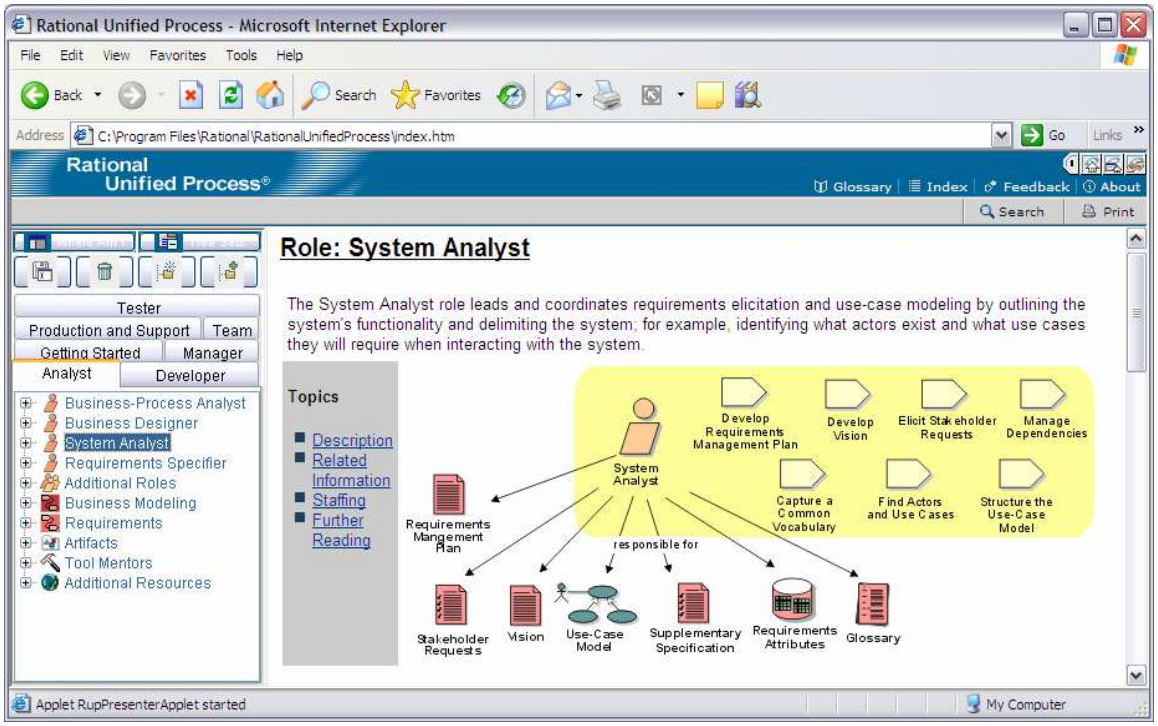


Figure 3.9 - RUP Tree for a Systems Analyst

In Figure 3.9 above the figures represent the same information as in Figure 3.7. By clicking on any of the items around the systems analyst's logo, the necessary information for that activity or artefact will be shown.

The appropriate information will be shown for a particular activity shown in the right of the systems analysts logo in Figure 3.9 by a user clicking on any of the activities, in so doing a link to information regarding the selected activity will be shown. This information usually details exactly what the activity involves, which artefacts are required before the activity can commence, and what artefacts will be created during the activity.

In a similar manner as the activity selection above a user selecting an artefact will open a link to information regarding that particular artefact. The information provided for each artefact can vary depending upon the nature of the artefact. Information regarding creating the document and essential details that need to be included is always shown. For some artefacts, example and template documents are provided.

In the same manner, information can be obtained for any of the other analyst roles by clicking on a name in the left hand window in Figure 3.9, and information pertaining to any other general role such as manager or developer etc, by clicking on the necessary tab above the window.

3.2.4 RUP Capability

RUP can provide a solid foundation for creating a process framework, but if the framework itself is flawed, then the processes will fail. Manzoni and Price in a paper entitled *'Identifying Extensions Required by RUP to Comply with CMM Levels 2 and 3,'* analyze RUP, and how it conforms to the CMM. Their major findings were as follows [Manzoni L.V. & Price R.T. 2003]:

- RUP meets most of the requirements of the CMM

- Not many of the managerial aspects were supported such as cost management, human resource management, communications management and contracts management
- To meet CMM level 2 or 3 with RUP, the processes need to be enhanced to support various lacking key processes

On the whole, Manzoni and Price were very satisfied with the level of support RUP gave in the areas of project management and software engineering process.

For this reason, and previously mentioned benefits, RUP was chosen as a process tool to be utilized in conjunction with the version control and tracking system developed in the course of the research.

3.3. Chapter Summary

In this chapter the standards set developed by the SC 7 for Software and Systems Engineering were introduced and discussed, as well as how these standards can lead to process improvement.

The task of process assessment was investigated, the CMM introduced, and the various levels of the maturity within CMM examined. How each one of these levels of maturity can be assigned was detailed.

RUP was introduced and discussed in detail, particularly with regard to the architecture and how RUP can be used to build a tailored process framework for an organization.

In the next chapter, project costing will be discussed, and a costing model examined and discussed in detail.

Chapter 4 – Project Costing

As previously mentioned in Chapter 2 the costing of projects is a vital process in any of the contracts conducted by the AEG. The costing information generated by this process is used in the tendering and budgeting of projects. These two activities are of importance to the AEG since if the tendering process is not conducted properly it could potentially cause the AEG to lose a contract. The budgeting information creates the timeline which the AEG follows to complete the project.

For any IT project costing is a highly important activity as the information is used in numerous financial and strategic planning processes. The information generated by the costing process predicts the amount of effort, time, and staffing requirements a project may require. This information is generally obtained by conducting costing procedures on information obtained in requirements specification documentation [Saliu M.O., Ahmed M., & AlGhamdi J. 2004]. Schwalbe calls this costing process 'project cost management' and it involves four main tasks [Schwalbe K. 2002]:

- Resource planning – involves determining the quantity and type of resources required to conduct the project. This information is presented in the form of a list of resource requirements.
- Cost estimating – involves developing a cost management plan which details the costs which should be incurred by the use of the required resources.
- Cost budgeting – involves the development of a cost budget for the project, which should detail the project baseline which can be used to determine a project's performance.
- Cost control – involves controlling any changes to a cost budget, and revising a cost estimate if required.

As costing information is used in so many strategic and financial decisions during a project, it is a process that needs to be conducted with utmost attention to detail. The person who should generally be responsible for the costing of projects should be a member of the organization who has had the most experience, such as a domain

expert in the area of the proposed project, and not an accountant or financial manager [Schwalbe K. 2002].

There are numerous techniques and procedures for costing, such as the one listed below in Section 4.1, and some unfortunate problems that hamper the performance of these techniques. In this chapter a particular technique and its potential problems will be examined.

4.1. Costing Techniques

The approach used in the costing process can vary from simply making an educated guess, to actually using models and formulas to estimate the cost of the project. Sommerville suggests that the following five techniques exist [Sommerville I. 1995]:

- Algorithmic cost modelling - This is the process of using formulas developed from previously completed projects to estimate the effort required for a future project.
- Expert judgement - This involves the collaboration of several domain experts in the particular suggested development environment coming to a consensus on the cost of a project.
- Estimation by analogy - This technique is used when previous projects of a similar nature have been undertaken before, and an analogy is drawn between the completed project's cost and the proposed one's cost.
- Parkinson's Law - This law states that the amount of free time available to complete the project is filled by the work. Hence, if a project is to take 12 months and 5 persons are available to work on the project the cost will be 60 person-months.
- Pricing to win - This simply involves the suppliers estimating the cost to be whatever the customer is willing to spend.

Sommerville states that the most 'systematic' of these techniques is the algorithmic cost modelling technique, as formulas can be built using information obtained from

previously completed projects to aid in predicting similar future projects [Sommerville I. 1995].

One of the most well known and documented algorithmic costing models is the COCOMO costing technique developed by Barry Boehm [Sommerville I. 1995]. Following is a detailed examination of this costing model and the results it yielded on some of the completed AEG projects.

4.2. COCOMO

COCOMO was first released in 1981 and is used by thousands of professionals in the industry. Unlike most other algorithmic costing models, it is an open model, and this allows for the following information to be published [Softstar Systems 2005]:

- All the equations within the model.
- All assumptions involved within the model.
- All the definitions pertaining to the model.
- All of the costs which are involved within the model.

This published information helps any organization utilizing the COCOMO model in tailoring this model for their individual use, and hence providing any organization with a very valuable costing tool.

4.2.1 The COCOMO Costing Formulas

The COCOMO model was developed by Boehm by analyzing a database of 56 projects and developing a costing model which provided accurate cost estimates based upon these 56 projects. The costing model has two basic formulas for calculating project cost [Royce W. 1999]:

$$\text{Effort} = C1 \; EAF \; (\text{Size})^{P1}$$

$$\text{Time} = C2 \; (\text{Effort})^{P2}$$

Where

Effort = number of staff months.

C1 = a constant scaling coefficient for effort.

EAF = a factor which is used to represent the nature of the project.

Size = the number of final source lines of code (generally).

P1 = an exponent which represents the economies of scale inherent in the process of developing the product, with particular reference to activities such as rework, bureaucratic delays and communication overheads which make up non-value-adding activities.

Time = the actual project time measured in months

C2 = a constant scaling coefficient for schedule.

P2 = an exponent used to represent the inertia and parallelism of managing the software project.

From these two formulas, three generic basic effort and schedule estimating formulas were generated, with values for C1, P1, C2 and P2 assigned to represent the complexity of the project. Not all software development projects are the same, and so Boehm developed these three modes for the model for the three different types of projects [Royce W. 1999]:

- Organic projects – refer to simple projects developed by small teams in a well understood domain area.
- Semidetached projects – are more complex than organic and typically comprise team members with limited experience in the development environment.
- Embedded projects – are the most complex, and are characterised by issues regarding reliability and real-time processing with a tightly controlled schedule.

Each one of these modes have values for C1, P1, C2 and P2. Given below are the equations with their respective values [Royce W. 1999]:

- Organic
  - Effort = 3.2 EAF (Size) $^{1.05}$
  - Time = 2.5 (Effort) $^{0.38}$
- Semidetached
  - Effort = 3.0 EAF (Size) $^{1.12}$

- Time = 2.5 (Effort) $^{0.35}$

- Embedded
  - Effort = 2.8 EAF (Size) $^{1.2}$
  - Time = 2.5 (Effort) $^{0.32}$

As is evident from the above formulas an increase in project complexity leads to an increase in both the exponent values for P1, P2 and a slight decrease in the C1, C2 variables. This will have the effect of increasing the value for 'Effort' and 'Time', leading to a longer estimate of project duration.

4.2.2 COCOMO External Inputs

There are two types of external inputs into the COCOMO formulas. The first is the 'Size'/'KSLOC' variable which represents the estimated source lines of code, measured in thousands. The second type of input is the information gathered from a series of characteristics which describe the proposed system.

The variable EAF is calculated as the product of the characterization parameters shown below in Table 4.1 and then used in the COCOMO formula to calculate the 'Effort' [Royce W. 1999], [Softstar Systems 2004].

Identifier	*Effort Adjustment Factor*	*Parameter Range*
RELY	Required reliability	0.75 - 1.40
DATA	Database size	0.94 - 1.16
CPLX	Product complexity	0.70 - 1.65
TIME	Execution time constraints	1.00 - 1.66
STOR	Main storage constraints	1.00 - 1.56
VIRT	Virtual machine volatility	0.87 - 1.30
TURN	Computer turnaround time	0.87 - 1.15
ACAP	Analyst capability	1.46 - 0.71
AEXP	Applications experience	1.29 - 0.82
PCAP	Programmer capability	1.42 - 0.70
VEXP	Virtual machine experience	1.21 - 0.90
LEXP	Language experience	1.14 - 0.95
MODP	Use of modern practises	1.24 - 0.82
TOOL	Use of software tools	1.24 - 0.83
SCED	Required development schedule	1.23 - 1.10

Table 4.1 - COCOMO Project Characterization Parameters

Each of these characterization parameters has a range, and depending upon the proposed project's resources capability for a particular characterization parameter, a value is selected from the parameter range shown in Table 4.1. The parameter range is usually broken down into about five levels of ability for each characteristic, ranging from very low, low, nominal, high, to very high, with each characterization parameter representing the following[Softstar Systems 2004], [University of Southern California 2005b]:

- Required Reliability (RELY) – The effect that failure of the product could have.
- Database Size (DATA) – The volume of data required to test the final product.
- Product Complexity (CPLX) – How complex the code required to develop the product will be.
- Execution Time Constraints (TIME) – The amount of CPU time the product will require.
- Main Storage Constraints (STOR) – The amount of main memory the product will utilize.
- Virtual Machine Volatility (VIRT) – The complexity of the interaction between the virtual machine and the hardware and software required.
- Computer Turnaround Time (TURN) – The average response time experienced by the developer, i.e. the time it takes for results of a job to be returned by the system to the developer, during implementation or testing.
- Analyst Capability (ACAP) – The capability of the system analysts for the project.
- Applications Experience (AEXP) – The level of experience the project team has within the product domain.
- Programmer capability (PCAP) – The capability of the programmers for the project.
- Virtual Machine Experience (VEXP) – The level of experience the project team has with the hardware and software to be incorporated within the product.

- Language Experience (LEXP) – The level of experience the project team has with the development language used for the product.
- Use of Modern Practises (MODP) – The degree to which modern practices are used in developing the product.
- Use of Software Tools (TOOL) – The types of tools used, ranging from low level assemblers to higher level programming languages.
- Required Development Schedule (SCED) - Whether or not the schedule has been compressed from the nominal schedule.

4.2.3 COCOMO Assumptions

When Boehm developed this model he had to make certain assumptions in order to develop the model effectively. These assumptions are [Royce W. 1999]:

- Size refers to lines of uncommented source code.
- The development life cycle begins with the product design and ends with acceptance (hence excluding requirements analysis).
- Staff months are 152 hours.
- The project is managed well.
- The requirements will remain constant throughout the duration of the project.

4.2.4 Effort Allocations for Activities in COCOMO

Once a COCOMO estimation is made, which will predict a project's estimated total time, it is useful to know how much of this time should be allocated to the various activities within the project. COCOMO has a set of standard activities and provides estimates as to what percentage of time should be spent on each activity. These can be seen below in Table 4.2 [Royce W. 1999].

Activity	Percentage Project Time
Requirements Analysis	4%
Product Design	12%
Programming	44%
Test Planning	6%
Verification and Validation	14%
Project Office	7%
Configuration Management and Quality Assurance	7%
Manuals	6%

Table 4.2 - Effort Allocations in COCOMO

This can provide a guideline upon which a project schedule could be developed for a project.

As the COCOMO model was originally developed by Barry Boehm in 1981 and the nature of software development projects has changed drastically since, the formulas and models required updating. This lead to the development of the COCOMO II model. COCOMO II incorporated changes to certain parameters, the removal of certain other ones, and the addition of further parameters. Some of the characteristics listed in Table 4.1 do not pertain to software development projects today, and as such they had to be removed and some characteristics which are pertinent to current software development projects had to be added. Following is a description of the COCOMO II model, and the changes made from the original COCOMO model.

4.3. COCOMO II

The first release of COCOMO II was in 1997 and was intended to provide a revised and current model, which could account for projects performed within software development environments very different to what they were in 1981. There have been a number of subsequent releases, and the plan was to release a new calibration of the model annually [University of Southern California 2005a]. Boehm states that COCOMO II *"is a an updated and recalibrated version of the Constructive Cost Model"* [Boehm B. 2000].

4.3.1 COCOMO II Scaling Factors

One of the significant changes to the original model was the creation of one single formula for all the project modes, and the discarding of the three modes of projects (Organic, Semidetached and Embedded). To account for the significant differences in projects, they added five scaling factors, which can be seen below in Table 4.3 [University of Southern California 1998], [Softstar Systems 2004].

Identifier	*Effort Adjustment Factor*	*Parameter Range - P1*	*Parameter Range - P2*
PREC	Precedentedness	0.0124 - 0.062	0.0024 - 0.0124
FLEX	Development Flexibility	0.0102 - 0.0507	0.002 - 0.0101
RESL	Architecture/Risk Resolution	0.0142 - 0.0707	0.0028 - 0.0141
TEAM	Team Cohesion	0.011 - 0.0548	0.0022 - 0.0109
PMAT	Process Maturity	0 - 0.78	0 - 0.0156

Table 4.3 - Scaling Factors in COCOMO II

These scaling factors are used to modify the values for the exponents P1 and P2 in the COCOMO formulas. The assigned parameter range is selected in exactly the same manner as the characteristics in Table 4.1, by selecting a level of competence in the range, this being divided into five. The selected value is then subtracted from the nominal value for P1 or P2 and the updated value is used in the final formula for the costing. Each of the scaling factors in Table 4.3 represent the following [University of Southern California 1998], [Softstar Systems 2004]:

- Precedentedness (PREC) – The comparability of the proposed project with any previous projects done.
- Development Flexibility (FLEX) – The level of flexibility in the requirements.
- Architecture/Risk Resolution (RESL) – The degree to which the architecture has already been defined in the industry.
- Team Cohesion (TEAM) – The level of cohesion that exists amongst all the stakeholders in the project.
- Process Maturity (PMAT) – The Maturity rating for the organization, much the same scale as discussed in Section 3.1.3 with regard to CMM.

The factors PREC and FLEX are intended to incorporate the major differences between the three project modes in the original COCOMO [University of Southern California 1998].

4.3.2 COCOMO II Project Characterization Parameters

In COCOMO II, the parameter ranges shown in Table 4.1 were recalibrated and changed. Furthermore some of the characterization parameters in Table 4.1 were removed and others added. The original COCOMO project characterization parameter set had 15 parameters, and COCOMO II has 17. The project characteristics VIRT, TURN, VEXP, LEXP and MODP where removed, as they were not deemed to be characteristics of current software development projects [Royce W. 1999]. The following characteristics in Table 4.4 below were added to the project characteristic set [Royce W. 1999], [Softstar Systems 2004].

Identifier	*Effort Adjustment Factor*	*Parameter Range*
RUSE	Required Reuse	0.95 - 1.24
DOCU	Documentation	0.81 - 1.23
PVOL	Platform volatility	0.87 - 1.3
PEXP	Platform experience	0.85 - 1.19
PCON	Personnel Continuity	0.81 - 1.29
LTEX	Language/Tool Experience	0.84 - 1.2
SITE	Multiple-site Development Team Communications	0.8 -1.22

Table 4.4 - New COCOMO II Project Characterization parameters

Some of the characteristics listed above in Table 4.4 are totally new, and some are a combination or modified versions of the removed ones. Each of these parameters represent the following [Royce W. 1999], [Softstar Systems 2004]:

- Required Reuse (RUSE) – If the product is going to be used in other systems.
- Documentation (DOCU) – The volume of documentation required for the product.
- Platform Volatility (PVOL) – The anticipated changes in platforms such as Operating System or Database Management System.
- Platform Experience (PEXP) – Level of experience the project team has with the target platform. This was a modification of the characteristic VEXP.

- Personnel Continuity (PCON) – The turnover rate for the organization annually.
- Language/Tool Experience (LTEX) – The level of experience the project team has with the language and tools to be used. LTEX is a modification of LEXP to include both the tool and language.
- Multiple-site Development/Team Communications (SITE) – The location of team members and the types of communications.

Boehm suggests that the change between the two models is not as significant as expected, this being attributed to the addition of two 'people-related' scaling factors, namely TEAM and PCON, and that personnel and team capability are the strongest influences on any software development projects productivity [Boehm B. 2000].

4.4. COCOMO II and Functional Points

As project costing is invariably done at the start of a project, it is almost impossible to accurately estimate the volume of uncommented lines of code the project will entail, this is the most important factor adversely affecting the accuracy of theses formulas.

In the COCOMO II model, functional points can be used to calculate the estimated source lines of code and can be used in the formulae as the 'Size' variable. It is very difficult to estimate the uncommented lines of code before implementation even begins. However the functional point measurements are done using the requirements specifications for a system and this documentation is available very early on in the project life cycle [University of Southern California 1998].

One of the leading organizations in the world involved in the promotion and the use of functional point measurement is the International Functional Point Users Group (IFPUG). They hold annual conferences, issue certifications, and sit on numerous standardization bodies to promote functional point measurement as an international standard [International Functional Point Users Group 2005].

4.4.1 Functional Point Measurement Process with COCOMO II

Functional point measurements are conducted by counting the number of information processing activities that are performed from within and outside the system. These are broken down into five user function types [University of Southern California 1998]:

- External Input (EI) – Counts the number of inputs of user data or user control input which crosses the external boundary into the system, or modifies/adds data in a logical internal file.
- External Output (EO) – Counts the number of outputs of user data or user control output which crosses the external boundary of the system.
- Internal Logical File (ILF) – Counts the number of major logical groups of user data or control information.
- External Interface Files (EIF) – Counts the number of files shared with other systems other than the system itself.
- External Inquiry (EQ) – Counts the number of occurrences that an input into the system leads to an immediate output.

COCOMO uses an unadjusted function point approach because the project characterization parameters do not effect the functional point measurement at all. This is not the usual functional point procedure [University of Southern California 1998]. Regular functional point costing approaches directly calculate the costing from the functional point measurements, adjusting them with the necessary characterization parameters.

There is a four step process which is used to calculate the unadjusted functional point count. This count is then used to calculate a substitute value for the 'Size'/KSLOC' variable in the COCOMO formulas [University of Southern California 1998]:

1. Determine function counts by type – This involves the process previously mentioned where counts are calculated for each one of the five user function types by assessing the design and requirements documentation. This process is intended to be conducted by the lead technical official on the project.
2. Determine complexity-level function counts – Once the count of each individual user function type has been done, the complexity level of each is

classified according too Low, Average and High. This is done by referencing the counts of the data element types and the number of file types using the table below.

For ILF and EIF				For EO and EQ				For EI			
Record Elements	Data Elements			File Types	Data Elements			File Types	Data Elements		
	1 to 19	20 to 50	50+		1 to 5	6 to 19	20+		1 to 4	5 to 15	16+
1	Low	Low	Avg	0 or 1	Low	Low	Avg	0 or 1	Low	Low	Avg
2 to 5	Low	Avg	High	2 to 3	Low	Avg	High	2 to 3	Low	Avg	High
6+	Avg	High	High	4+	Avg	High	High	3+	Avg	High	High

Table 4.5 - Complexity Level Function Counts

Key:

***ILF**: Internal Logical File (Files)*

***EIF**: External Interface Files (Interfaces)*

***EO**: External Output (Outputs)*

***EQ**: External Inquiry (Queries)*

***EI**: External Input (Inputs)*

3. Apply complexity weights – After computing the complexity-level function counts these complexity levels are then used for each type to calculate the complexity weight. This is done with the table below.

Function Type	Complexity-Weights for Complexity Levels		
	Low	Average	High
Internal Logical Files	7	10	15
External Interface File	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

Table 4.6 - Function Type Complexity Weights

4. Compute Unadjusted Functional Points – This involves adding all the counts for the complexity weights calculated with Table 4.6 to obtain the unadjusted functional points.

Once the above four-step process is completed, it is possible to calculate the 'Size'/'KSLOC' variable for use in the costing formulas. This is done by multiplying the unadjusted functional point count by the corresponding development environment value, using a conversion table such as the one below.

Language	Source Lines of Code/Unadjusted Functional Points
Assembly	320
C	128
C++	29
Pascal	91

Table 4.7 - Unadjusted Functional Points to Source Lines of Code Table

This provides a thorough approach to estimating the source lines of code and should help to get a more accurate value and hence a more accurate final estimate for the project duration.

4.5. Potential COCOMO Shortfalls

The COCOMO costing model has since its release provided many professionals with a valuable tool for software development systems costing. However this model, as with all models, is flawed in a few ways.

Saliu, Ahmed and AlGhamdi believe that the COCOMO model, like many other algorithmic costing models, relies on accurate information such as source lines of code, complexity, interfaces etc. They believe that this information is highly uncertain, particularly when it is needed for costing at the start of a project [Saliu M.O., Ahmed M., & AlGhamdi J. 2004]. They also believe that COCOMO fails to provide an adequate solution which takes into consideration technological advancements [Saliu M.O., Ahmed M., & AlGhamdi J. 2004]. For the COCOMO model to provide for this shortfall they will have to annually recalibrate the model to account for the changes in the environment and this they try and do.

Helm states that the COCOMO model does not provide for rework to systems due to flawed requirements, and that any errors in the initial 'Size' estimation will lead to a

inaccurate costing estimation [Helm J.E. 1992]. As one of Cuomo's primary assumptions is that the requirements for the system will not change the model unfortunately does not provide for this. All of the algorithmic costing models available rely on accurate information to be input into the model to create the most accurate estimation. Unfortunately there is no way around this, and as Helm states the *"garbage in, garbage out"* principle is unfortunately all but true in this instance.

Besides the obvious flaws that COCOMO may have, it is a very valuable tool and in the next section a comparison done on AEG projects comparing the COCOMO estimated cost and the actual cost will be discussed.

4.6. COCOMO and the AEG

As the AEG has been doing contract work for numerous years, and information pertaining to project costs and the source code was available, it provided an excellent platform for testing the COCOMO model.

The COCOMO II costing equations were used in their native form and none of the variables were calibrated in any way. For the actual costing process a costing questionnaire was created using Microsoft Excel, an example questionnaire done for one of the AEG projects is shown in Appendix A. The questionnaire allows for the user to select the levels of capability for each individual project's characteristics for both the scaling factors shown in Table 4.3, and the general project characteristics shown in both Table 4.1 and 4.4. The source lines of code were derived from the final source code for the project by using a tool created to strip out all of the comments from within the code. The questionnaires were filled out by the members of the team who developed the actual systems. The AEG measure all their project costs in hours, and the estimates generated from the questionnaires had to be adjusted accordingly. In Figure 4.1 below, the difference between the estimated and actual costs are shown.

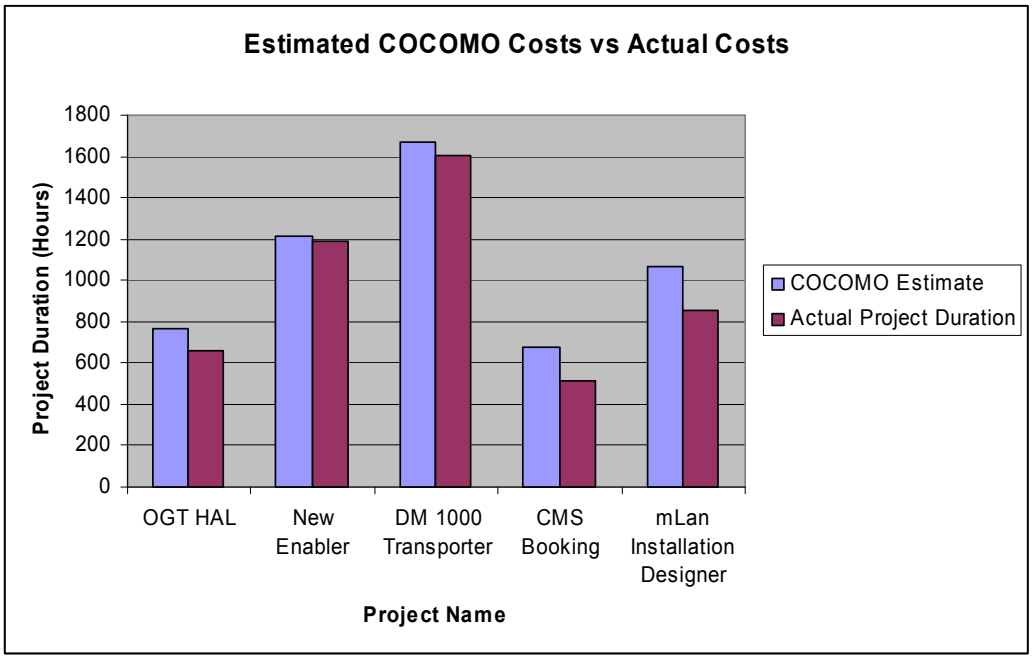


Figure 4.1 - Estimated COCOMO Costs vs Actual Costs

As can be seen from the above figure, the estimated cost is marginally higher than the actual project costs. The only comparison that is extremely close is the 'New Enabler' project and the greatest difference is the 'mLan Installation Designer' project. This marginal difference in project costs can be attributed to the following:

- The model was not calibrated in any way to fit the AEG's development environment. This is usually done by most organizations to ensure the most accurate cost estimations.
- The AEG does not engaged in a formal testing process and the testing is done by the developer. This would make a significant change, as the COCOMO activity break down assigns test planning, verification, and validation as 20% of a project cost. This can be seen in Table 4.2.
- As some of the contracts are done as research and development for clients/contractors, there was not a demand for user manuals. This is assigned a 6% total project cost in the COCOMO activities break down shown in Table 4.2.
- As the costing questionnaires were filled in by the actual developers, it was found that for characteristics such as ACAP and PCAP they were rather modest in their assignments.

- As the AEG is a small team and work assignments are broken down into tasks that each individual can complete on their own, the effectiveness of the team is increased. This notion is supported by Hale, Parrish, Smith and Dixon, who in a paper entitled *Enhancing the Cocomo Estimation Models* conclude that if tasks are broken down into pieces of work that can be accomplished by individuals, it aids in decreasing the development effort [Hale J. et al. 2000].

After the above analysis was conducted, a test on the mLan Enabler project was done to test the accuracy of the functional point approach. This method also yielded the same outcome, a minor over estimation in the COCOMO model.

The COCOMO approach to costing has given all costing professionals a tool to utilize in their efforts to predict project costs. For this model to be utilized effectively however, the model requires calibration for the target organization, as no matter how well the COCOMO model is defined, they will never be able to account for every single software development organization's environment.

4.7. Chapter Summary

In this chapter, costing techniques were introduced and a well documented algorithmic costing model discussed. The original COCOMO model was examined, after which the new COCOMO II model was introduced and the major differences between the original and new model shown.

The functional point count method was introduced and its implementation in the COCOMO context detailed. The potential shortfalls that the COCOMO model has were listed, and the AEG's costing analysis on the COCOMO model discussed and evaluated.

In the next chapter version control management will be discussed and how the tools available for this process can potentially aid any software development organization. There are many of these tools available and a number of them will be discussed and evaluated.

Chapter 5 - Version Management

Version Management is an activity which forms part of the software engineering process known as Configuration and Change Management. As the name suggests, it involves managing changes and configurations within a software development project, in an effort to ensure the integrity of project artefacts with the use of a Version Management System. [Kruchten P. 2003].

Version Management is an important activity within the development projects conducted by the AEG as they require all project artefacts to be available to team members irrespective of their geographic locations. The artefacts handled by the version management system need to be maintained in such a manner as to ensure all changes to any artefacts can be accounted for. Furthermore allow for the simultaneous accessing and modification of artefacts within the version management system.

5.1. Version Management System Functionality

According to Beck, version management systems should poses the following general functionality [Beck J. 2005]:

- Management of the modifications made to artefacts.
- The ability to report on the history of the artefacts and to view previous versions. The ability to view differences between two versions and to roll back to a previous version of any artefact maintained by the system.
- Allow for collaborative work on any artefacts in the system, and implement necessary conflict resolution mechanisms in such an environment.
- Allow for the branching of developments into more than one level of parallel sets and the ability to merge these parallel levels together.

Most, if not all version management tools offer very similar functionality. They have the ability to create what is termed a “repository” on a local or remote machine, and store and access all project artefacts from this repository through the version management tool. Below is a list of some of the common functionality offered by version management tools:

- Login – The ability to connect and authenticate with the version management system on a remote machine
- Checkout – The ability to checkout an artefact from the repository for use.
- Commit – The ability to send back a checked-out artefact with or without modifications to the artefact itself.
- Display Differences – The ability for the version management system to display differences between different versions of artefacts within the repository, i.e. the differences between the latest version of an artefact and a previous version.
- Display History – The ability for the version management system to display the history of accesses and modifications made to an artefact.
- Logout – The ability to close a connection to a remote machine.

5.2. Version Management Tools

The importance of configuration management cannot be underestimated. Pollice, Augustine, Lowe and Madhur state that if they could only choose one tool to use in the course of their projects, it would be a configuration management tool [Pollice G., Augustine L., Lowe C., & Madhur J. 2003]. In a paper entitled the *"Impact of the Research Community On the Field of Software Configuration Management"* the authors believe that Software Configuration Management (SCM) is an essential tool in the success of any software development project [Estublier J. et al. 2002]. Shown below is a table created by the International Data Corporation displaying the growth of these tools in 2000.

Company (product)	Annual Revenue $M	Share %	Annual Growth %
Rational (Atria ClearCase) [IBM 2005a]	293	32.4	50.3
MERANT (Intersolv PVCS & Harvest CCC) [Serena 2005c]	115	12.6	14.9
Computer Associates (Endevor) [Search Networking 2005]	113	12.5	5.2
SERENA (ChangMan) [Serena 2005a]	94	10.4	38.2
Telelogics (Continuus) [Telelogic 2005]	65	7.1	23.1
Microsoft (SourceSafe) [Microsoft 2005b]	31	3.4	2.3
Total (with Others)	906	100	22.7

Table 5.1 - Worldwide SCM Tools ($M)

As can be seen from Table 5.1 above the annual growth of these products is significant, and proves that the industry as a whole is adopting these products into there software development projects. The products shown in Table 5.1 unfortunately only show the proprietary products and not products such as CVS and Subversion which are established and accepted non-proprietary solutions.

In this chapter a number of version management tools, both proprietary and non-proprietary will be discussed and evaluated. Apart from the normal capabilities of a version management system, we were looking also looking for an Application Programmer Interface (API) that would allow for the integration of process information into the version system.

5.2.1 Microsoft Source Safe

Microsoft SourceSafe is a propriety version management tool developed by Microsoft. It is available only on the Microsoft Windows Platform and integrates with Microsoft Visual Studio .NET. SourceSafe offers the following important features [Microsoft 2005b]:

- Easy to use Graphical User Interface – The user interface for accessing all files within a local or remote repository is simple and easy to use. The interface can be seen below in Figure 5.1.

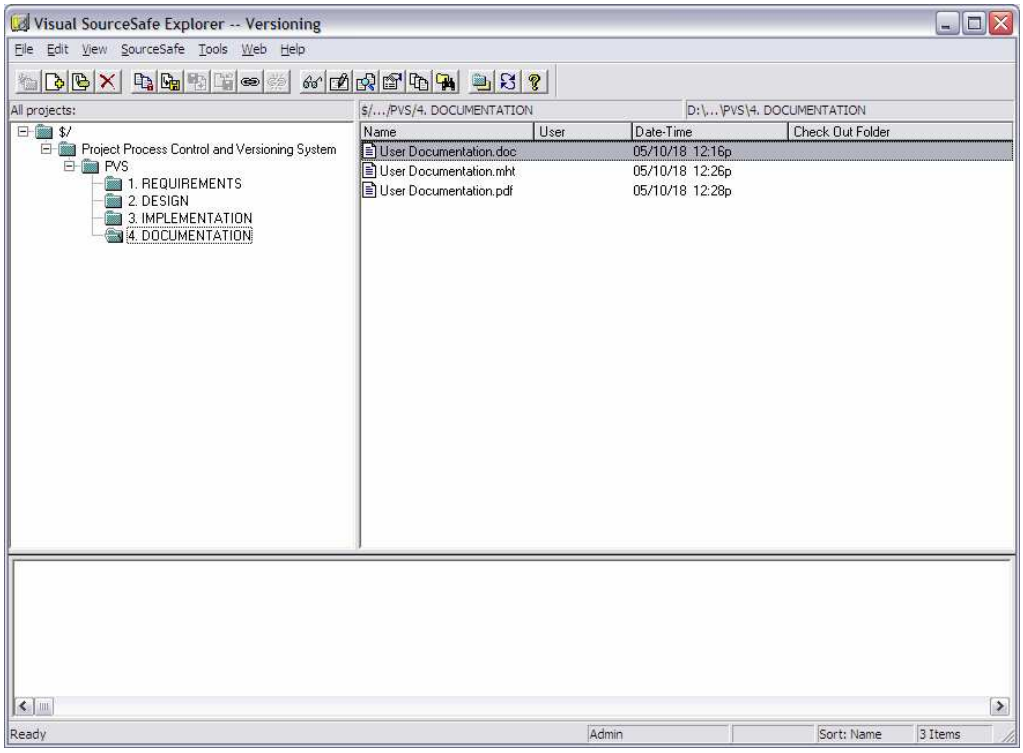


Figure 5.1 - Microsoft Visual SourceSafe Interface

The interface in Figure 5.1 shows a single project the 'Project Process Control and Versioning System' in the repository, shown in the left hand pane. The right hand pane of Figure 5.1 shows the files within each individual project within the folders. These files can be checked out and worked upon, and then checked back to the repository from within this interface.

- Integration with Visual Studio .NET – The source files from within a Visual Studio .NET project can be placed under version control using SourceSafe and accessed from within the Visual Studio .NET environment. This can be seen below in Figure 5.2.

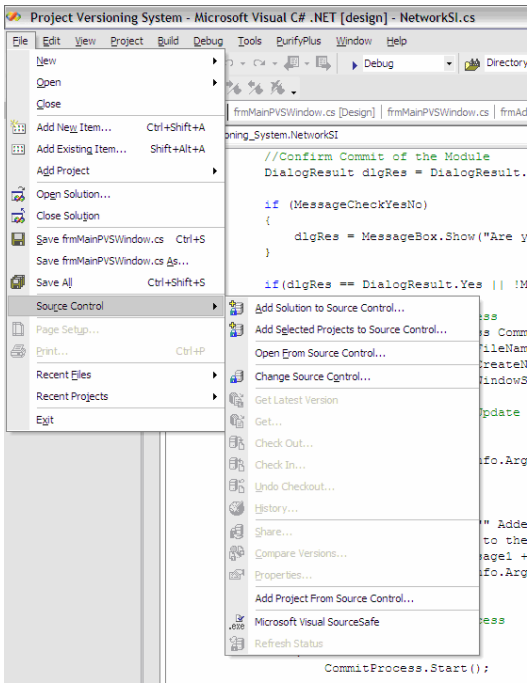


Figure 5.2 - Microsoft Visual SourceSafe within Visual Studio .NET

By clicking on the 'File' tab in the .NET environment and selecting the 'Source Control' option users can perform version management activities on any file, this can be seen above in Figure 5.2. This allows for individual files to be worked upon by a team member while another member may be working on another file within the same project.

Here are some of the important versioning related features of Microsoft SourceSafe:

- File Protection – Locking provides a mechanism to prevent work from being overwritten by disallowing more than one modification to any one file at one time.
- Visually View Differences in Files – SourceSafe can visually show the differences in versions of files within SourceSafe.
- Fork Projects – Allows for a project to have two parallel versions, particularly useful if one version is for release and another for a beta version bug fix.
- Merge Files – Allow for the merging of two different versions of the same file, particularly usefully if the same file was worked on simultaneously.
- Maintain an Audit Trail – Can produce reports and shows records of who accessed and modified particular files.

There do exist some disadvantages of SourceSafe and listed below are a some of the more important issues [Bolton M. 2005]:

- As Microsoft SourceSafe is a proprietary product is cost money as apposed to the non-proprietary products.
- Numerous features found in other version management tools are lacking in Microsoft SourceSafe, such as branching support and cannot be safely extended.
- It has reliability issues on busy or slow network connections.
- Apart from the issues above no acceptable API could be found for the integration of SourceSafe and the process required for the system developed from this research.

5.2.2 Merant Professional

Serena markets the product ChangeMan [Serena 2005a] and recently bought the company Merant who markets the product Merant Professional [Serena 2005c], originally called PVCS. Both these products are proprietary Version Management Tools.

The Merant Version Manger product has both a User Interface similar to that of Microsoft SourceSafe and a Web Client, which can be used to login to a remote machine and get artefacts from the server. Given below in Figure 5.3 is the web client which is used to connect to the remote client server and obtain artefacts.

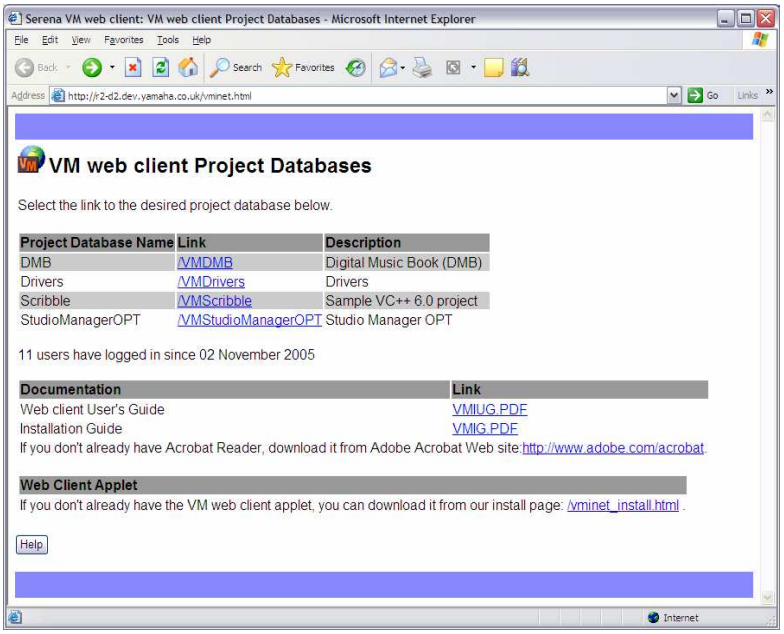


Figure 5.3 - Merant Version Manager Web Client

Through this web client users can select the particular project database they wish to login to and then access the necessary artefacts. The Merant Version Manager offers all the same functionality that standard version management tools offer. Users are able to:

- Check Out – Artefacts from the repository and work on them.
- Get – Artefacts from the repository in read only mode from examination and not editing, this option does not lock any files in the repository as the check out would.
- Check In – Send back any artefacts that have been modified to the repository, this will unlock all files that the user may have hade checked out.
- View Differences of Artefacts – Display the differences between a current and previous version of an artefact.

Merant Version Manger also integrates with the Visual Studio .NET development environment and also works on the UNIX, Linux and Windows platform [Serena 2005c].

Merant also had in its suite of products a tool called TeamTrack [Serena 2005b], TeamTrack is a process and issue management system which also has a Web Client

the allows users to login to the TeamTrack system and post queries and comments on a particular project or project artifact's status.

The two products Version Manager and TeamTrack were almost exactly what the AEG required for a solution, unfortunately they had following disadvantages:

- As they are proprietary products, there was no API that could be used to extend the system for any further functionality needed by the AEG.
- The Version Manger and TeamTrack tools where very disjoint from each other and the AEG needed one tool that could provide the solution required.

5.2.3 Rational ClearCase

Rational ClearCase is a configuration and change management tool with integrated version control that is available on the Microsoft Windows, Linux and Unix platforms. It also integrates with the Microsoft Visual Studio .NET and the open source Eclipse development environments [IBM 2005a].

The ClearCase version management tool provides for interactions with the repository through client GUI's providing local and remote access, as well as a web interface and command line interactions.

ClearCase provides functionality for users to conduct the following important operations on a repository:

- Creating a repository or what is termed a Versioned Object Database (VOB) in ClearCase.
- Checking In and Out of artefacts from the VOB.
- Branching and merging of different versions of a particular artefact.

5.2.3.1 Creating a VOB

Each VOB within a repository represents a set of artefacts that represent an individual project within the repository. The VOB itself can be created through a wizard or at the command line. When a VOB is created, the user can specify the required type of

VOB, whether or not the VOB is intended for public access or private access, and even specify exactly which users are allowed access. To gain access to a VOB, it needs to be mounted from either the ClearCase Explorer or the command line.

5.2.3.2 Checking In and Out Artefacts

Before any artefacts can be checked out or into the VOB it is required to be mounted. ClearCase offers two modes to check out files from a VOB, by reserved or unreserved checkout. When a user checks out an artefact as reserved, it locks the artefact in the VOB and no one else is allowed to checkout the artefact in reserved mode until the user checks back the reserved artefact. When a user checks out a file in reserved mode, users can still check out the artefact in unreserved mode. In an unreserved check out, the users can access the artefact, but they are not given exclusive rights to check back the file before any other user, as in the case of a reserved checkout.

To check an artefact back into the VOB, the user can select the artefact in the ClearCase explorer and select check in, or alternatively the artefact can be checked in from the command line.

5.2.3.3 Branching and Merging Artefacts

ClearCase offers support for branching, which enables users to create a parallel version of a particular development effort. Branching is particularly useful if work that is out of the ordinary is required to be done to a version of artefacts and can then be merged back into the version later. For instance in the case of bug fixing certain parts of a development effort. Branching allows for the ordinary development to continue, while simultaneously fixing any bugs or conducting work not affecting the ordinary development.

Merges are done in ClearCase by using the ‘Merge Manager’. ClearCase adopts the following strategy for merging artefacts, one artefact is defined as the base contributor, and this is usually the oldest common version of the artefact before a branch was created. Any other new artefacts that are being merged are contributors. When merging, ClearCase scans through the artefacts line by line. For any particular

line, it looks at the base contributor, and if there are no changes between the base contributor and any of the other contributors, the line is added to the new merged artefact. If there is a change between the base contributor and only one of the contributors, the new line is added to the new merged artefacts. However if there are numerous changes between the base contributor and other contributors then the user is required to determine which line is to be used.

Besides offering the above support, ClearCase also offers support for displaying the history of any particular artefact within a VOB, and displaying any files that are currently checked out from the VOB. ClearCase is an easy to use version management tool that offers all the functionality required of a version management tool. ClearCase has an API for Java and Perl and the Java API is hosted on SourceForge [SourceForge 2005a].

5.2.4 Concurrent Versions System

CVS is an open source network-transparent version control system and is available on both the Microsoft Windows and Linux environments. It has numerous front-ends developed for these environments.

CVS has a client/server architecture which allows developers to access the repository from anywhere with an Internet connection [CVS 2005]. The version management system tracks all access to the repository and builds log and history files. It also has the capability to show the differences between different versions of a file stored within the repository.

5.2.4.1 CVS Benefits

Beck lists the following arguments for the use of CVS as a version management tool [Beck J. 2005]:

- CVS is Language Neutral – CVS can be used irrespective of the development environment, unlike some version management tools which are integrated into

various development environments and can only be used on a project if that particular development environment is used.

- CVS is Widely Deployed and Used – CVS comes standard with all GNU/Linux distributions and CVS offers free and commercial versions of the protocol on the Linux, Windows and Macintosh platforms. CVS is used in industry by large corporations such as Caterpillar Inc, The Boeing Company, Apple Inc, U.S. Department of Agriculture and Pixar Inc.
- CVS is Easy to Administer – CVS is easy to set up and extensive documentation and public forums are available to help in the process of setting up a CVS Server.
- CVS is Easy to Use – For basic interactions with a CVS Server very few commands are required and the commands themselves are very basic.
- CVS is Not Commercial – CVS itself does not use any proprietary file formats and the CVS project itself is open source.
- CVS is Free of License Cost – CVS can be obtained and used free of charge.

5.2.4.2 CVS Interactions

In the interactions between a CVS Server and a client, the user can issue command line calls to communicate or can use one of the numerous GUI clients available. CVS communications can be done from a Windows client to a Linux CVS Server. Interactions between a Microsoft Windows client and a Linux CVS Server can be done using CVS NT commands in a console window. The CVS NT command is an executable that can be used to interact with a Linux or Microsoft Windows based CVS repository. The syntax required in executing commands for both the Linux CVS and the Microsoft Windows CVS NT are exactly the same, which makes the CVS system extremely platform independent.

All the below information about the CVS interactions is described in the *'Version Management with CVS'* document [Cederqvist P. 2004]. All information stored within a CVS Server in stored in what is termed the 'CVSROOT'. This is the parent directory for what is termed the 'modules' held within the CVS Server, and the modules are folders which have artefacts pertaining to a particular project. Modules themselves may have more modules within themselves.

From the command line users can:

- Login into the CVS Server
- Checkout modules
- Commit any modified artefacts
- View differences of particular versions of artefacts
- Display the history of activities on a particular artefact
- Logout from the CVS Server

Once a commit command is executed and if any of the artefacts within the module have been modified the default text editor will be opened and the user can enter a log entry to record what changes were made to the artefact, as seen below in Figure 5.4.

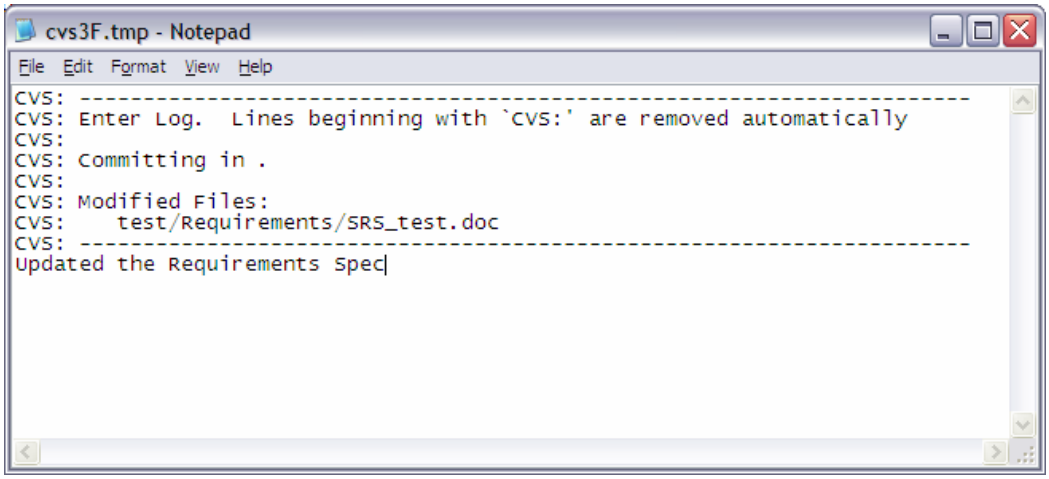


Figure 5.4 - CVS Commit Log Message

All this information taken from log files is kept in a file on the CVS Server and they can be accessed very easily to check what changes were made and by whom, to any artefact stored in the repository.

The CVS commands all have the same form making them very straightforward to use and the more complex operations on a CVS have the same form too. One of the major benefits of a CVS Server is that a module can be setup for anonymous access, giving anyone read only access to a module on the CVS Server. This has been one of the major factors contributing to CVS being used in the open source community.

5.2.4.3 TortoiseCVS Client

TortoiseCVS is a Windows client for CVS that integrates directly into Windows Explorer and allows users to perform CVS commands on files directly in Windows Explorer [TortoiseCVS 2005]. By simply right clicking on a file in a Windows Explorer pane, the file can be added to a module. If the file is already in the repository such as the files shown below in Figure 5.5 further operations can be done.

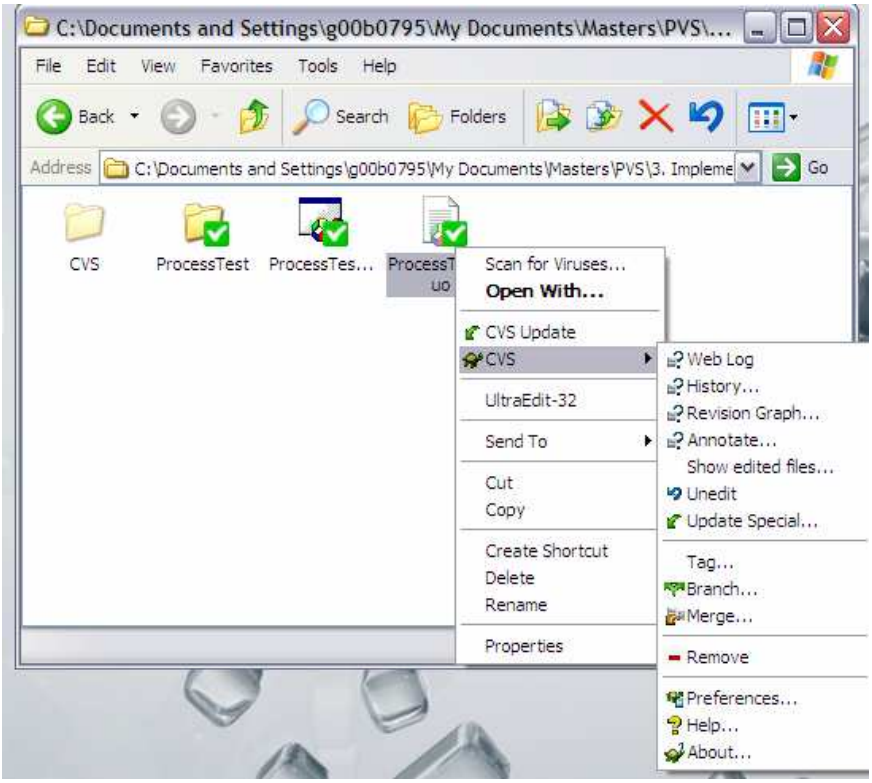


Figure 5.5 - TortoiseCVS Explorer Interface

CVS, as with many other industry standard tools, does have problems. The next section contains a discussion of a recent tool in the version management community, Subversion, which was designed to improve on the features offered by CVS.

5.2.5 Subversion

The Subversion project was started in 2000 by the company Collabnet, and the first release of Subversion was in 2002. The goal of the Subversion project was to improve on the features that CVS has, and possibly add some extra functionality to the system [Collins-Sussman B. 2002]. Within the Subversion FAQ Collins-Sussman indicates "w*e aren't (yet) attempting to break new ground in SCM systems, nor are we attempting to imitate all the best features of every SCM system out there. We're trying to replace CVS"* [Collins-Sussman B. 2002].

In a paper by Glassy who investigates the effect of using version control in tertiary education system, he chooses Subversion over CVS for the following reason [Glassy L. 2005]:

- Security – CVS uses text files to store repository information, making it possible for students to edit history information about modules. Subversion in contrast uses a binary repository, hence improving the security of all repository information and making it un-editable.
- Robustness – CVS has potential problems when conducting commits which fail, which could lead to all parts of a module not being committed correctly. Subversion will only fully commit a module if the commit is conducted fully and correctly. Subversion also has built in mechanisms for repairing any damaged repositories.
- Feature – As Subversion was built to correct any errors and build on any features CVS has, it proved more useful particularly with the ability to rename, copy, and move module files and directories.

The official Subversion documentation cites the improvements that Subversion offers over CVS. Given below are some of the important features [Collins-Sussman B., Fitzpatrick B.M., & Pilato C.M. 2005]:

- Directory Versioning – Subversion tracks any changes to directories and files, whereas CVS only tracks changes and keeps history information for individual files.

- True Version History – As Subversion offers the capability to rename, copy, and move files and directories, history can be kept for a file or directory irrespective of its location within the repository or any name changes.
- Atomic Commits – This is the same issue that Glassy raises with a commit not being totally completed in the second point above.
- Choice of Network Layers – Subversion is more flexible and offers more interoperability in accessing the system than CVS.
- Consistent Data Handling – Subversion makes use of a binary differencing algorithm and makes it possible to view differences in both text and binary files while CVS only offers support for text files.

As CVS is one of the industry standards, and has been for a number of years and there are numerous API's available and a large volume of documentation available it was chosen as the version management tool to be used for the system developed for this research. The API's and the CVS NT executable could be very easily used in the selected development environment to integrate the process information required by the solution with the CVS.

Rational ClearCase was a viable tool which could have been used as the version management tool for the system developed from this research. Unfortunately however the API's where not available for use with Microsoft Visual Studio .NET in which the system was going to be developed and there was always and inclination to use an open source non proprietary product as it would provide more documentation and the underlying architecture and workings of the system could be explored.

With all the benefits and improvement Subversion offers over CVS it is a more robust and reliable solution for version control. Unfortunately at the start of this research Subversion was still a relatively new solution so was not considered for integration in the system developed from this research.

5.3. Chapter Summary

With all the benefits and features that Version Management Tools can offer to software development teams in easing the Configuration and Change Management Process, they should form an integral part of any project.

In this chapter, the basic functionality that a Version Management Tool should offer was discussed, and the role that a Version Management Tool should play in any project was highlighted.

Two proprietary and two non-proprietary tools were introduced, and the features each offers were explored.

The CVS solution was discussed in detail, and example interactions given. The Subversion system that was developed to replace CVS was discussed, and the benefits that it offers over CVS listed. The choice of CVS as the version management tool to be utilized was motivated.

In the next chapter the system that was developed from this research, 'The Project Process Control and Versioning System' will be discussed in detail, and how each component from the previous chapters were integrated to form the final system.

Chapter 6 - The Project Process Control and Versioning System

The Project Process Control and Versioning System (PPCVS) was developed to meet the process and versioning needs of the AEG. The PPCVS provides RUP based process control and the version control offered by CVS, all in one solution, which is accessed through the use of a GUI interface. The PPCVS allows for the seamless interaction of all team members on a project. The PPCVS also provides valuable status information to clients on the project as a whole, and the ability to check on the status of certain project artefacts within a project.

The system was developed to solve a business need of the AEG, and has been developed to meet the requirements listed in Section 2.4. This business need is detailed in a RUP Vision Document, which can be seen in Appendix B. The Vision Document details all the requirements the AEG has for the system, which is done by showing the high level major functionality of the system.

The complete client/server PPCVS application comprises the following major components:

- The Artefact Set, comprising of all the necessary artefacts required by the AEG to start a new project.
- The Linux CVS Server running Red Hat Linux with kernel 2.4.27 and CVS version 1.11.17.
- The PPCVS Client Application, comprising network components, the PPCVS GUI, and the Project Status Bar (PSB).

6.1. The Artefact Set

With every project that is created using the PPCVS, the user is able to select from a set of template artefacts, which have been tailored for the AEG to include in their project. These artefacts have been selected from RUP, based on their effectiveness in

the project process for projects undertaken by the AEG. Not all the template and reference material provided by the system is taken from RUP, some of the documentation is from the IEEE and ISO standards set, and some created from scratch.

As discussed in Chapter 3, an organizations level of quality or a adherence to standards, is not measured by end products, but rather by the processes that are followed in creating products. Using standard project artefacts for every project should aid in improving the overall project process. Besides using some of the artefacts recommended by RUP, the actual RUP methodology is followed. This is achieved by the utilization of the PSB, which will be discussed in Section 6.3.3.5.

In Table 6.1 below, the project artifact set can be seen, the inspiration for this set being partly taken from a paper written by M. Hirsh entitled *"Making RUP Agile"* [Hirsh M. 2002]. In this paper Hirsh cites that *"many professionals do not consider RUP practical for small, fast paced projects"* the reason for this being that in Version 2002, RUP had 80 major artifacts, 150 activities, and 40 roles [Hirsh M. 2002]. As the AEG is indeed a small team involved in fast paced projects, RUP in its entirety was not viable for them either.

Costing	Costing spreadsheet
Requirements	Software Requirements Specification
	Use Case Model
Analysis and Design	Software Architecture Document
	Design Model
Implementation	Implementation Model
Testing	Defect List
Deployment	Release Notes
	Installation Artefacts
Project Management	Microsoft Project Status File
	Iteration Plan
	Iteration Assessment

Table 6.1 - PPCVS Artefact Set

Each one of the processes in the left hand side of Table 6.1 represent the workflows that have been taken from RUP, and the right hand side of Table 6.1 are the selected artefacts. As can be seen, the workflows selected have been modified, and only a subset of the artefacts used. This is in fact exactly what RUP recommends teams do in order to get the most out of RUP. Support is given by the use of the RUP builder, in which teams can select the set of roles, activities and artefacts required by a particular project. This process of selecting only the required workflows and artefacts leads to a tailored process. The workflows selected will be discussed in Section 6.3.3.5 when the PSB is discussed. Each of the template artefacts supplied by the PPCVS have a cover page that enables the team member working on the document to update the version of the document and state any major changes they make to the document, with the date. The artefacts in Table 6.1 represent the following:

- Costing Spreadsheet – The costing questionnaire developed using COCOMO II discussed in Chapter 4 and in Appendix A.
- Software Requirements Specification – The requirements document created from the IEEE 830-1998 recommendation discussed in Section 2.3.2.2.
- Use Case Model – An empty document in which the team members can place the Rational Rose Use Case Diagram.
- Software Architecture Document – Details the high level design of the system using Unified Modelling Language (UML) diagrams.
- Design Model – An empty document in which team members can place the Rational Rose Object Model and Sequence Diagrams. Rational Rose is a UML modelling tool that enables users to very easily model systems in terms of use case diagrams, object models and sequence diagrams [IBM 2005b].
- Implementation Model – A document detailing all the artefacts that are utilized in the implementation of the system and the actual implementation artefacts themselves.
- Defect List – An empty document in which lists of defects and information pertaining to the defect can be placed and updated when defects corrected. Used to keep track of any defects found by any team member in the system.
- Release Notes – Details any release information pertaining to a particular version release of a product.

- Installation Artefacts – An empty document in which the details pertaining to the installation of a release can be placed, along with the installation files.
- Microsoft Project Status File – A template Microsoft Project Schedule in which the project manager can detail all information pertaining to the project schedule. This document will be discussed in detail in Section 6.3.3.5.
- Iteration Plan – A template in which the project manager can detail the plans for a particular assessment and the milestones to be attained.
- Iteration Assessment – A template in which the project manager can assess the previous iteration against the milestones attained and information pertaining to any problems encountered.

All these artefacts help in creating a standard, repeatable process using best practises. These documents are used in conjunction with the actual tailored process, which will be discussed in section 6.3.3.5.

6.2. Linux CVS Server

The CVS Server is housed on a Linux machine in the Audio Engineering Lab at Rhodes University. As previously mentioned, the Linux CVS Server is running Red Hat Linux with kernel 2.4.27 and CVS version 1.11.17. The CVS Server will manage and store all project artefacts and the PPCVS Windows GUI will access this server through the network components of the system.

To setup a CVS Server, the following steps need to be carried out in order for the server to respond to any remote calls over the network. All the following information is obtained from the official CVS Documentation [Cederqvist P. 2004].

6.2.1 Creating a Repository

This is the first action that needs to be performed. Firstly the location of the repository needs to be decided upon, this being commonly referred to as the CVSROOT. Once

the location of the CVSROOT is decided upon, the CVSROOT needs to be initialized. This is done by running the following command in a Linux shell:

```
cvs –d /usr/local/cvsroot init
```

The above command is broken down into the following parts:

- cvs – the cvs command
- -d – denotes that you are naming the repository explicitly in the command line by directory
- /usr/local/cvsroot – the location of the desired CVSROOT on the CVS Server
- init – the command to be executed, in this instance the initialization command

Once this command is executed it will create a folder within the CVSROOT called CVSROOT which contains a number of administrative files for the handling of modules and files, and the general settings of the repository.

6.2.2 Setting up the CVS Server for Password Authentication

Once the repository has been created, the CVS Server needs to be configured for password authentication if remote connections are going to be made to the CVS Server. This is done by editing a configuration file, 'cvspserver', on the Linux machine such that if a connection is made on a particular port, the cvs command is executed. The file '/etc/xinetd.d/cvspserver' needs to be created and the following information placed inside it:

```
Service cvspserver
{
port        =     2401
socket_type =     stream
protocol    =     tcp
wait        =     no
user        =     root
passenv     =     PATH
server      =     /usr/local/bin/cvs
server_args =     -f -allow-root=/usr/local/cvsroot pserver
}
```

Figure 6.1 - The CVS Server Configuration File

Once this initialization file is configured, as above in Figure 6.1, any network interactions with the CVS Server on port 2401 will use the cvs system to respond, with the above settings.

6.2.3 Setting up CVS Server Passwords

There are two options for an administrator of a CVS Server to consider when setting up the CVS Server. The CVS Server will by default give all users with user accounts on the Server access to the repository, or the administrator can set up a password file which gives users in the password file access to the repository only, and not the entire system. If the password file option is chosen, the file 'passwd' within the administrative folder CVSROOT inside the repository needs to contain information similar to this example:

```
anonymous:
user1:abc123
user2:123abc
```

The above password file creates three users, anonymous, user1 and user2. The anonymous user has no password, and anonymous users are by default given read-

only access to the repository. The user's user1 and user2 have passwords shown on the right of the colon.

6.2.4 Non-Binary File Settings on the CVS Server

As CVS stores and manipulates files within the repository as text files, special cases need to be made for binary files. This is done by listing the extensions of the files which are binary files, and CVS will manage them as binary and not text files. This is done by editing the file 'cvswrappers' in the administrative folder CVSROOT of the repository, and placing the following information in the file:

```
*.DOC -k 'b' -m 'COPY'
*.doc -k 'b' -m 'COPY'
*.XLS -k 'b' -m 'COPY'
*.xls -k 'b' -m 'COPY'
```

The above settings allow for Microsoft Word (*.doc) and Microsoft Excel (*.xls) files. The above settings are broken down into the following:

- *.DOC – The extension of the files
- -k 'b' – Indicates that this file type is binary
- -m 'COPY' – Indicates to the CVS that files of this type can not be merged together and that merges need to be done by the user, as the CVS cannot merge binary files.

Once the above mentioned steps have been concluded, the CVS Server is ready to operate fully and interact with any Windows clients.

6.3. The PPCVS Client Application

In this section the analysis, design and implementation of the PPCVS client application will be discussed, and the functionality it provides, examined.

6.3.1 The PPCVS Client Application Analysis

When conducting the analysis for the application the major functionalities or use cases for the system where extracted from the PPCVS Vision Document found in Appendix B. Given below is the Use Case Diagram for the PPCVS application.

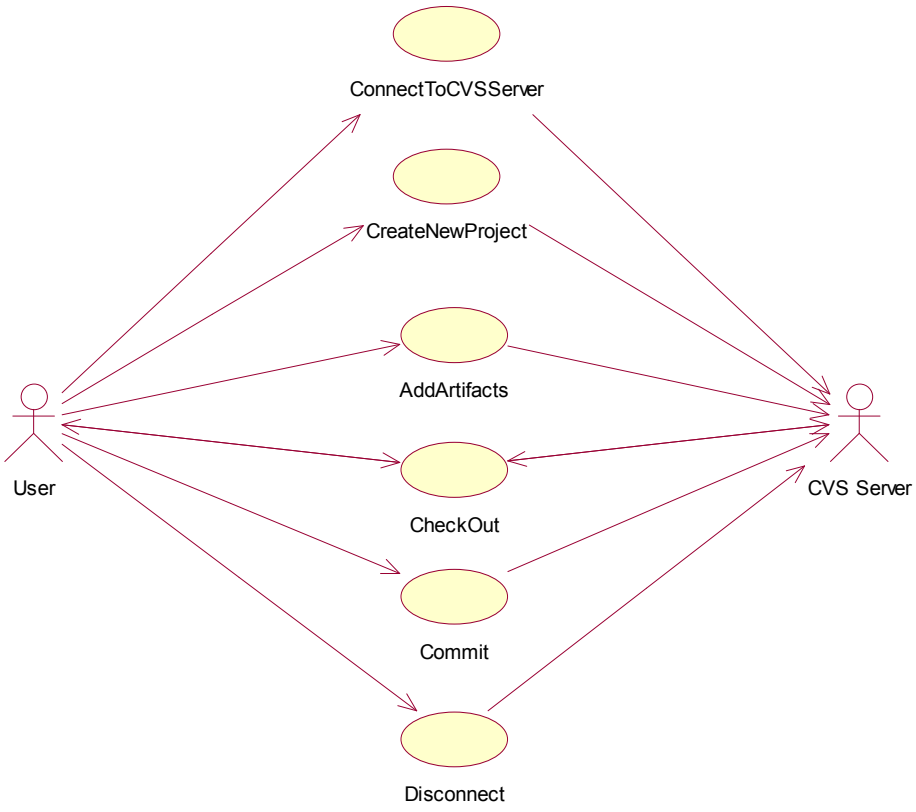


Figure 6.2 - The PPCVS Use Case Diagram

Each of these use cases given in Figure 6.2 provide the main functionality of the PPCVS application. The actors 'User' and 'CVS Server' are the only two actors identified in this system. Each of the use cases in Figure 6.2 provide the following functionality:

- ConnectToCVSServer – Allows the user to connect to the CVS Server, authenticating themselves with their username and password through the PPCVS client application.

- CreateNewProject – Gives the user the functionality to create a new project on the CVS Server and select from a set of tailored templates to start the project with.
- AddArtefacts – Allows the user to add any number of artefacts to the CVS Server, both files and entire directories can be added.
- CheckOut – Provides the functionality for the user to checkout a module or entire project from the CVS Server.
- Commit – Allows the user to send back any modified artefacts to the CVS Server.
- Disconnect – Allows the user to close the connection between the PPCVS client application and the CVS Server.

All the use cases shown in Figure 6.2 were detailed in a flow of events document that can be found in Appendix D. A flow of events document details the functionality of use cases. Information pertaining to the typical behaviour of use cases are shown and the details as to what the system should do are highlighted [Quatrani T 1998].

6.3.2 The PPCVS Client Application Design

From the use case diagram in Figure 6.2 and the flow of events in Appendix D the object model for the PPCVS application was created. The object model for the PPCVS client application showing the major 'CVSNetwork' class and the windows within the application can be seen below in Figure 6.3.

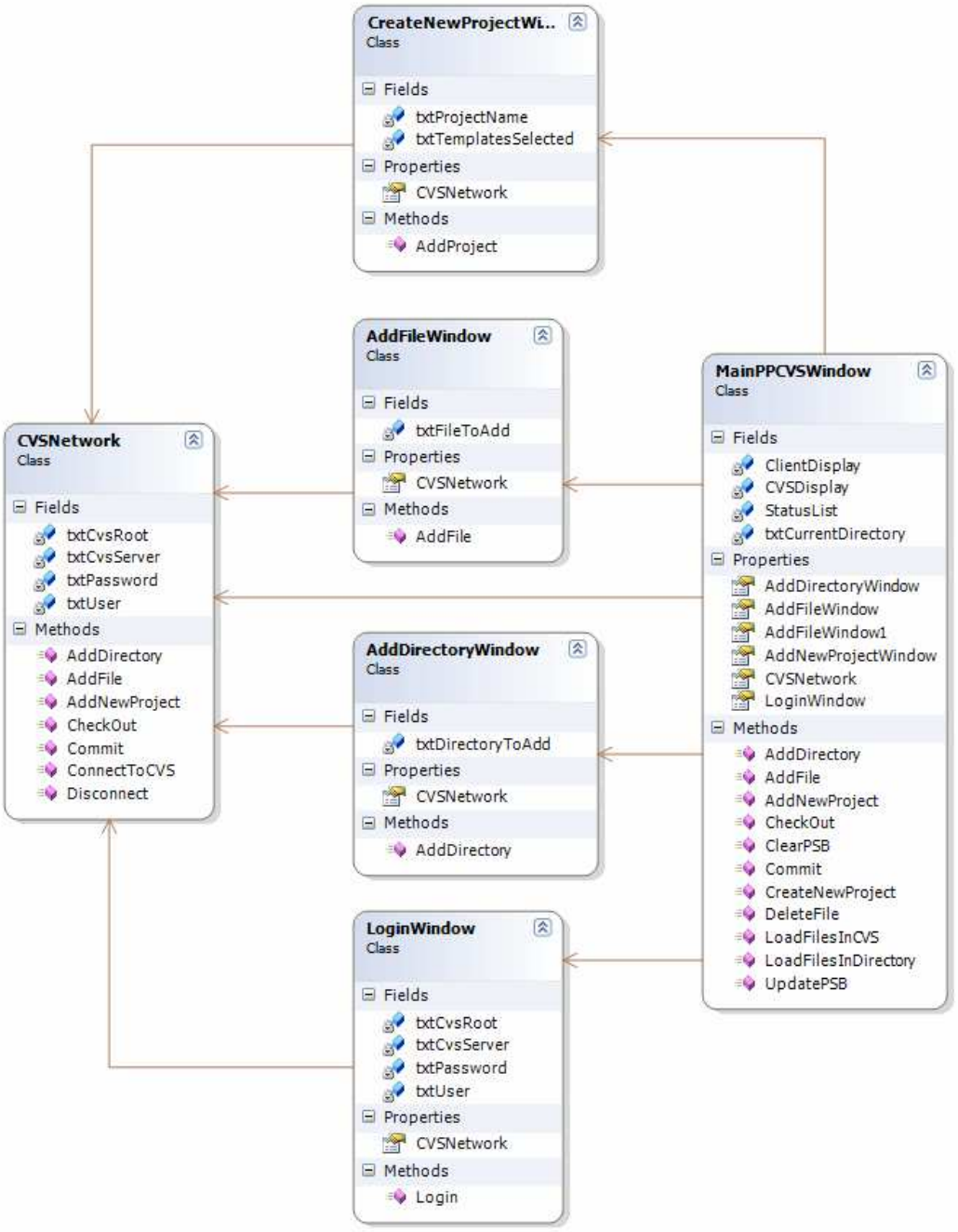


Figure 6.3 - The PPCVS Object Model

The class 'CVSNetwork' is responsible for providing all the network functionality to the PPCVS application. The other five objects shown above in Figure 6.3 are all windows from which the user interacts with the PPCVS application and these objects in turn communicate with the 'CVSNetwork' class to perform the necessary

interactions with the CVS Server. Following is a discussion of the design of each window given in the Object Model in Figure 6.3 above and the functionality these windows provide.

6.3.2.1 The PPCVS Client Application GUI

All the network interactions with the CVS Server are initiated via the PPCVS Client GUI. The GUI was developed in Visual Studio .NET using C#. The full user documentation for the system is given in Appendix B. The PPCVS application has the following windows within the client GUI:

- The LoginWindow – The first form that is opened, which allows the user to login to a selected CVS Server.
- The MainPPCVSWindow – The main form in which most of the interactions with the CVS Server are done. This form also has the PSB on it, which details project status information.
- The CreateNewProjectWindow – This form allows a user to create a new project and select the required template information from the artefact set through a wizard.
- AddFileWindow – This form enables the user to select where in the repository they wish to add a selected file.
- AddDirectoryWindow – This form enables the user to select where in the repository they wish to add an entire directory.

6.3.2.1.1 The Login Window

In this form users can enter information regarding the following:

- The CVS Server Address
- The CVSROOT location
- The User Name
- The User Password

Figure 6.4 - The PPCVS Login Window

Once this form is completely filled in, as shown above in Figure 6.4, the user can attempt to login to the system. If the login process is successful the 'MainPPCVSWindow' will be opened.

6.3.2.1.2 The MainPPCVSWindow

Upon opening the 'MainPPCVSWindow' the system performs a check out to obtain the CVSROOT\modules file, which is an administrative file that contains the list of modules in the repository. This information is then used to populate the list of modules available to the user. In Figure 6.5 below the 'MainPPCVSWindow' is given.

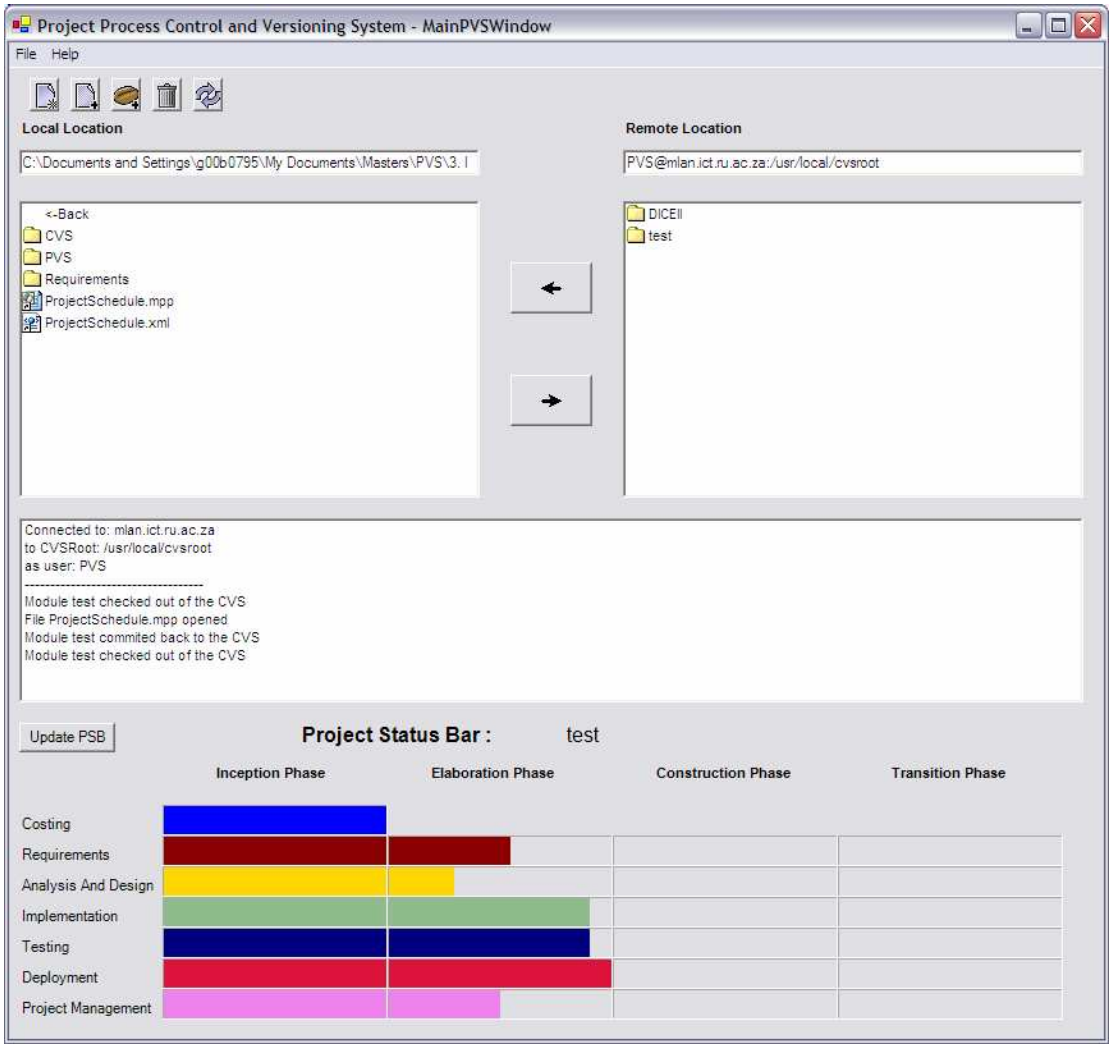


Figure 6.5 - The PPCVS MainPPCVSWindow

In Figure 6.5 above, the left hand pane represents the files on the local client machine, know as the client side display, and the right hand pane shows the project artefacts on the CVS Server, know as the remote side display. In the remote display the projects, 'DICEII' and 'test', are in the repository of the CVS Server. The user is able to select a module or individual project artefact and check it out onto the local machine by clicking on the arrow pointing towards the left hand pane and selecting a module from the remote side display. Modules are items shown in the remote display, they may be the root directory of a project as shown in the remote display in Figure 6.5 or one of the sub folders within the project, which are accessed by double clicking on a project within the remote display. Once the user has finished editing or viewing a project artefact found within the checked out module, he is able to select the artefact and click

on the arrow facing towards the CVS Server artefacts. This will commit any changes to the artefacts back to the server. By double clicking on any folder within each of the panes, that folder (client side display) or module (remote side display) will be opened and the files/artefacts within the selected item shown.

The pane below the client and remote displays is the status window, which shows information pertaining to the connection, and any interactions with artefacts. Below the status window is the PSB which provides status information of a particular project to the user. Via the 'MainPPCVSWindow' a user can perform all the necessary operations on artefacts, on the client machine and the CVS Server.

6.3.2.1.3 The CreateNewProjectWindow

When the user decides to create a new project within the repository, he is taken through a series of graphical selection panes from which he can select which of the artefacts from Table 6.1 are required in the new project. Given below is a selection pane from the 'CreateNewProjectWindow'.

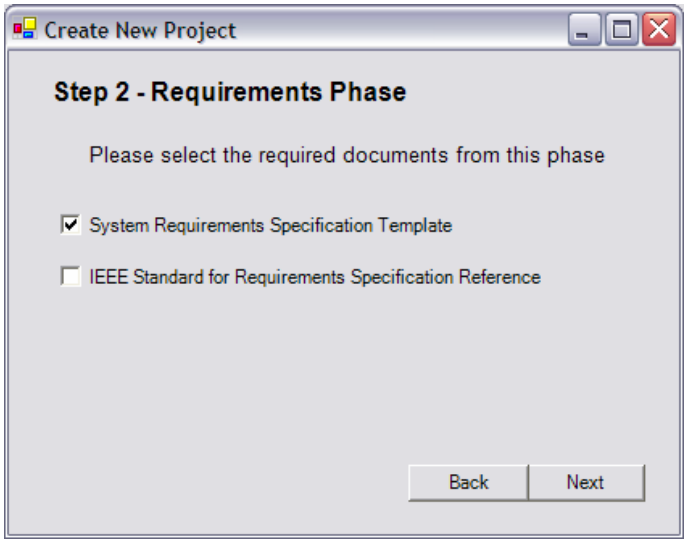


Figure 6.6 - The CreateNewProjectWindow

Once the selected artefacts are chosen, the new project is created in the repository containing all the required documents. These artefacts made available to the user

through this window are the artefacts shown in Table 6.1. This new project with all the selected artefacts is then available to anyone with access to the repository.

6.3.2.1.4 The AddFileWindow and AddDirectoryWindow

Both these windows allow users to add artefacts to a module within the repository, the file or directory to be added is chosen by the user selecting the desired item in the client display shown in Figure 6.5 and clicking the relevant 'Add' button above the client display. The process for selecting the location for the file/directory to be added is identical. Given below in Figure 6.7 is an 'AddDirectoryWindow'.

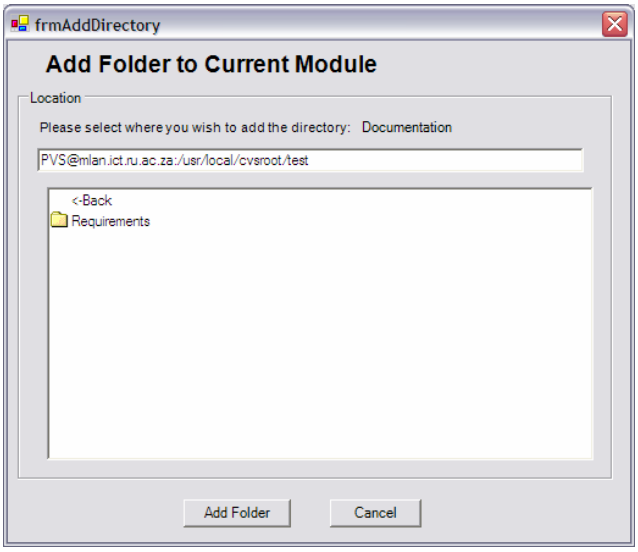


Figure 6.7 - The AddDirectoryWindow

From this window a user can browse a selected module by double clicking on a module name or directory within the module, and select the desired location for the file or directory to be added.

Having viewed all the main windows in the PPCVS application, it is now time to explore the Project Status Bar.

6.3.2.1.5 The Project Status Bar

In Figure 6.5 the PSB is located at the bottom of the window. This bar is used to display the status of a current project. The bar takes a similar form to that of the RUP architecture shown in Figure 3.6. The workflows have been edited to only show the required workflows necessary for the AEG projects. The selection of these workflows will be discussed in the implementation section detailing the PSB in section 6.3.3.5. The PSB can be seen in Figure 6.8 below.

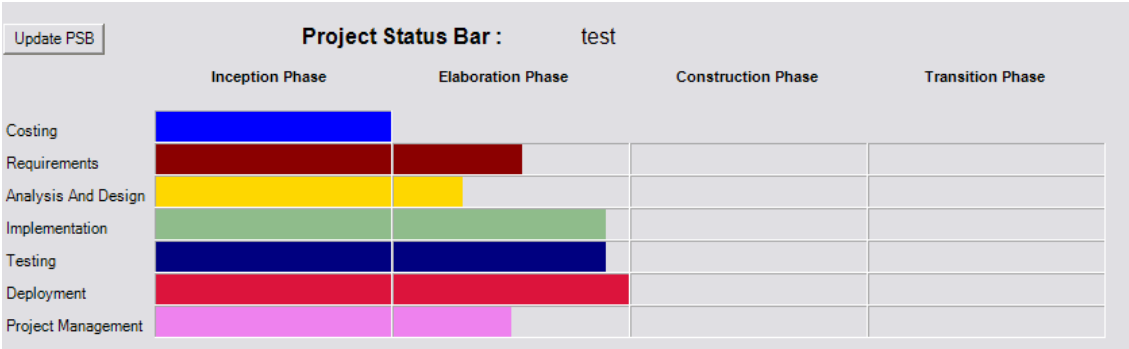


Figure 6.8 - The Project Status Bar

The PSB is populated whenever an entire project is checked out from the repository. This status information is obtained from a schedule file supplied with each project created using the PPCVS application. How the PSB obtains this status information will be discussed later in Section 6.3.3.5.

6.3.3 The PPCVS Client Application Implementation

The entire PPCVS application was developed in Microsoft Visual Studio .NET. All the windows discussed in the previous section interact with the 'CVSNetwork' class to communicate with the CVS Server. Following is a discussion of how these interactions with the 'CVSNetwork' were implemented to enable the PPCVS client GUI to communicate with a CVS Server.

6.3.3.1 SharpCVSLib

Initially, the implementation of the interactions of the PPCVS was done using a library hosted by SourceForge called SharpCVSLib [SourceForge 2005b]. This library is developed in C#, and is an API for C# that allows developers to utilize the library to access CVS repositories from within any application developed using Visual Studio .NET. SharpCVSLib is developed and maintained by Mike Krueger and Clayton Harbour. The SharpCVSLib project was started in 2003 and is still in its Alpha version of development.

At the start of the implementation of the PPCVS application, the API provided functionality for connecting to a CVS repository and checking out files. When the application required the API to commit or create new modules in the repository, the API proved to be very unstable, and at most times did not provide the functionality it was supposed to.

As no other suitable library could be found for C#, the only other alternative was to use the CVS NT command line application as discussed in section 5.2.4.2, directly from within the PPCVS application, and develop an API specifically for the PPCVS application. Before the 'CVSNetwork' class and its implementation can be fully explained, the functionality and syntax of the CVS NT command line application must be explored. In the next section is a discussion of the functionality of this application.

6.3.3.2 The CVS NT Command Line Application

Before any commands can be issued to a remote CVS Server, the user is required to perform authentication by logging onto the system. This is done by using the following command in a console window:

```
cvs –d:pserver:user@server.ac.za:/usr/local/cvsroot login
```

The above command is broken down into the following parts:

- cvs – The cvs command line application.

- -d – Denotes that you are naming the repository explicitly in the command line by directory.
- pserver – States that connection protocol to be used is password authentication.
- server.ac.za – The name of the CVS Server.
- /usr/local/cvsroot – The location of the CVSROOT in the CVS Server.
- login – The operation to be executed.

After this is executed, the user will be required to enter the password associated with a particular user name on the CVS Server. Once this is done the user can then execute a command to check out a module from the CVS Server as follows:

```
cvs –d:pserver:user@server.ac.za:/usr/local/cvsroot checkout module1
```

This command will check out the module 'module1' into the folder from where the cvs command was executed through the console window. 'module1' represents an entire project module in the repository of the CVS Server, meaning it is the root folder for the project in the repository. All the files within the module 'module1' can then be accessed and modified using any tool desired on the client machine. As can be seen, the login and checkout commands are very similar. The commit command is similar too:

```
cvs –d:pserver:user@server.ac.za:/usr/local/cvsroot commit module1
```

As mentioned in Section 5.2.4.2 any commits executed on modified artefacts will open the default text editor, and the user can enter a log message for the modifications made to the artefacts as seen in Figure 5.4. This provides a mechanism in which a certain level of accountability can be achieved and any changes to the system will be well documented.

To logout from the CVS Server, the logout command is used:

```
cvs –d:pserver:user@server.ac.za:/usr/local/cvsroot logout
```

This will close the connection between the client and the CVS Server. Given below in Figure 6.9 is the complete list of the commands used the PPCVS application in interacting with the CVS Server directly from within C#. Following is a discussion of how this was achieved.

```
Login
cvs -d:pserver:user@server.ac.za:/usr/local/cvsroot login
Checkout
cvs -d:pserver:user@server.ac.za:/usr/local/cvsroot checkout module1
Commit
cvs -d:pserver:user@server.ac.za:/usr/local/cvsroot commit module1
Logout
cvs -d:pserver:user@server.ac.za:/usr/local/cvsroot logout
```

Figure 6.9 - The CVS NT Command List

6.3.3.3 The CVS NT Command Line Application Within C#

The CVS NT commands are executed from within the C# application by opening a console window and passing the required information to the window to execute the operations. This is achieved by creating a process within C# and passing the required parameters to the console window and executing this process. The implementation of this in C# for the login operation can be seen in the code segment given below.

```
System.Diagnostics.Process LoginProcess = new
      System.Diagnostics.Process();

LoginProcess.StartInfo.FileName = "cvs";
LoginProcess.StartInfo.Arguments = "-d:pserver:" + strUser + ":" +
      strPassword + "@" + strServer + ":" + strCvsRoot + " login";

LoginProcess.StartInfo.CreateNoWindow = true;
LoginProcess.StartInfo.WindowStyle =
System.Diagnostics.ProcessWindowStyle.Hidden;

//Execute Login Process
WaitMsg.SetMessage("Logging In To: " + strServer);
WaitMsg.Show();

try
{
      LoginProcess.Start();
      while (!LoginProcess.HasExited);
      WaitMsg.Close();
}

catch(Exception)
{
      MessageBox.Show("Could not connect to CVS", "Bad Password or
      Server Unavailable", MessageBoxButtons.OK,
      MessageBoxIcon.Error);

      WaitMsg.Close();
}
```

Figure 6.10 - Example CVS NT Process Execution in C#

In the code segment above the process 'LoginProcess' is created and given the file name 'cvs' as the command to run and is also given the arguments for a cvs login operation. The process is set up to not create a window and to hide the process window itself, so that the user of the PPCVS system will not see the actual console window which is opened during the process. The process is then executed in a try-catch code block so that if the process fails, an error message is displayed to the user and the application does not crash.

All of the member functions within the 'CVSNetwork' class are implemented in the same manner, as shown in the code segment in Figure 6.10. They all create a new process for the operation to be carried out passing it the required command and arguments and then execute the process in the same manner as shown in Figure 6.10. The only difference being in the way the processes are declared and some minor differences in the way in which the methods are built up and the arguments passed to

the console window. For instance some of the commands are required to be executed from a location other than the working directory of the PPCVS application, in these instances a batch file is created which moves to the location where the operation is required to be executed and executes the CVS command line application.

6.3.3.4 The PPCVS Behavioural Model

In this section there will be an explanation of how GUI interactions are handled and how the CVS Server is accessed from the GUI client. The interactions are modelled in terms of UML sequence diagrams.

6.3.3.4.1 Connect To The CVS Server

The connect to CVS Server interaction provides the functionality for the user to connect to a CVS Server repository. This interaction obtains all its information from the 'LoginWindow', the sequence of events for this process can be seen below in the sequence diagram given.

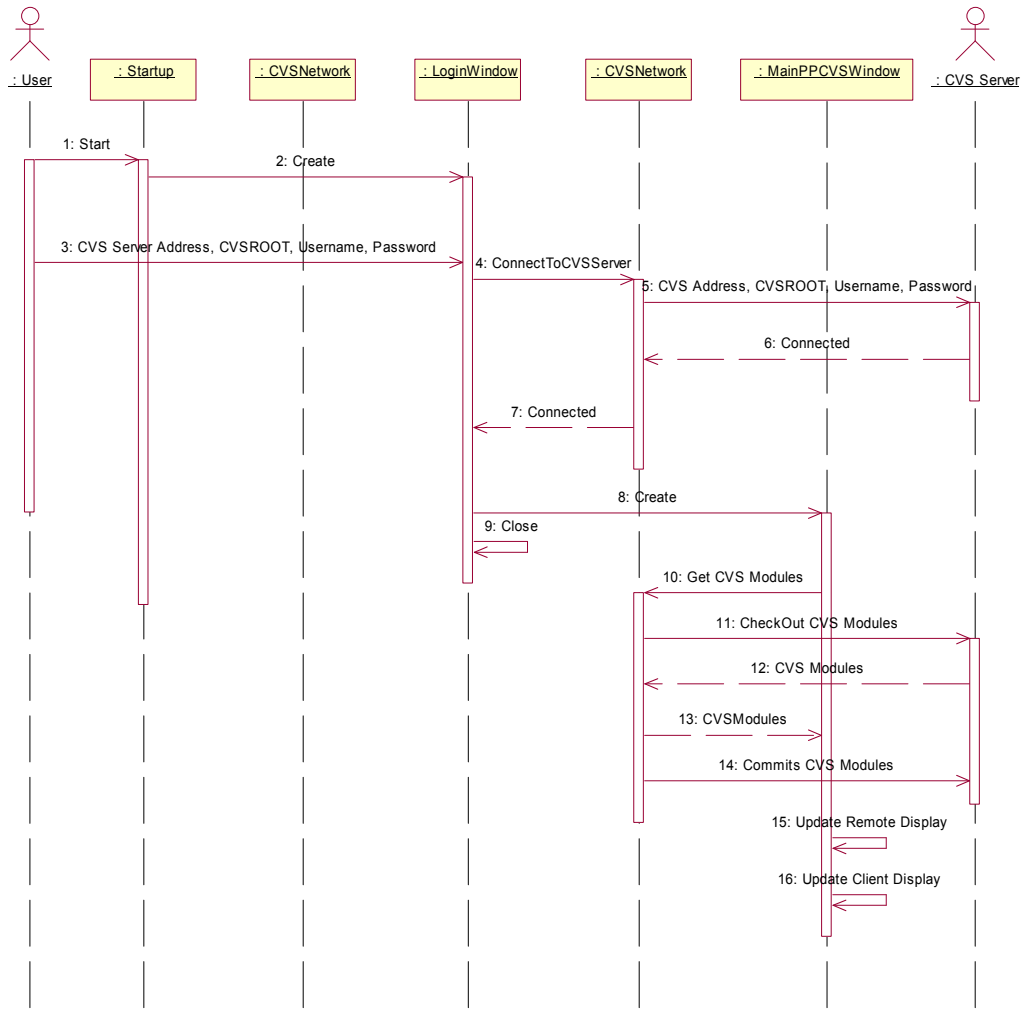


Figure 6.11 - The Connect To CVS Server Sequence Diagram

The user supplies the 'LoginWindow' with the necessary details this then supplies the 'CVSNetwork' class with this information and the login process is executed. After the successful connection, the 'MainPPCVSWindow' is opened and the available modules on the CVS Server obtained through the 'CVSNetwork' class, which is then used to update the remote display.

6.3.3.4.2 Create New Project

When the user decides to create a new project, the 'CreateNewProjectWindow' is opened and the user is then able to select the artefacts he needs in the new project. The artefacts are included into the new project when the new project is created by the

'CreatNewProject' method of the 'CVSNetwork' class. This method also checks out the modules file and updates this file to include the new project name, and then commits the new module file back to the repository on the CVS Server. Since the module file was modified, a log message is required by the CVS Server. A default message is passed detailing the user who created the new project. The 'MainPPCVSWindow' then gets the latest modules file and updates the remote display. The actual sequence of events involved in this operation can be seen in the sequence diagram given below in Figure 6.12.

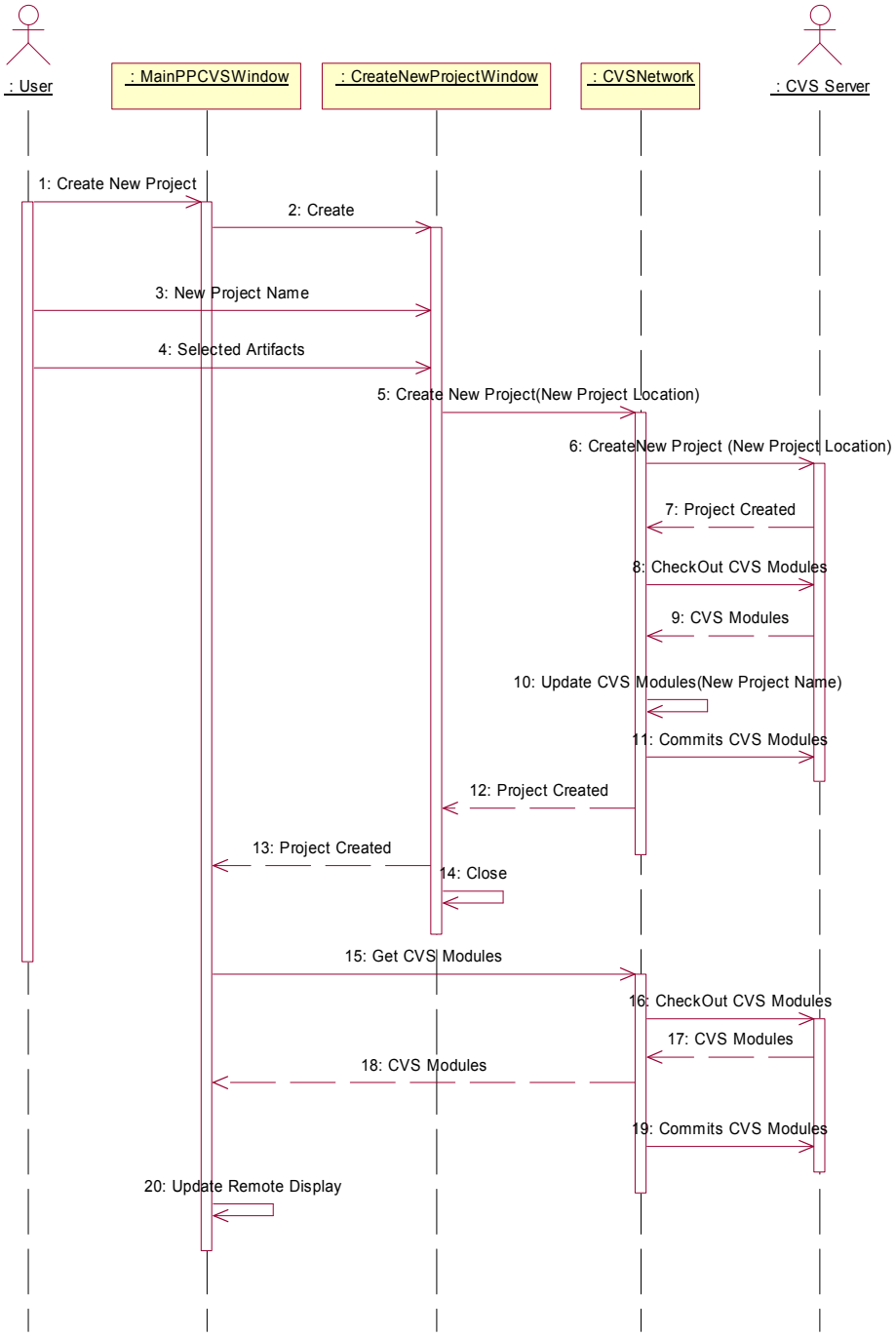


Figure 6.12 - The Create New Project Sequence Diagram

The 'CreateNewProject' method provides the functionality for the PPCVS application to create a new project in the repository. This method will create the new project workspace in the repository and copy across any artefacts selected for inclusion in the new project. Once the creation of the new project is complete, the administrative file

'modules' found within the CVSROOT folder in the repository needs to be updated to include the new project. This is done by checking out the file, opening a text writer, appending the new project name to the file, and then committing the file back to the repository. This process is implemented by the code segment as seen below.

```
//Update the modules file on the CVS
this.CheckOut("CVSROOT/modules", false);
TextWriter newModuletw = new StreamWriter("CVSROOT\\modules",true);
newModuletw.WriteLine(strNewProjName + "  " + strNewProjName);
newModuletw.Close();
this.Commit("CVSROOT/modules", false, true, true);
```

Figure 6.13 - The Update Modules File Code Segment

Once a new project is created a log message should be created for the new project. This is done in the same manner as shown in Figure 5.4 for a commit.

6.3.3.4.3 Add a File

The add a file interaction allows for the addition of a file to any particular module in the repository. The sequence of events involved in adding a file to a module can be seen below in figure 6.14.

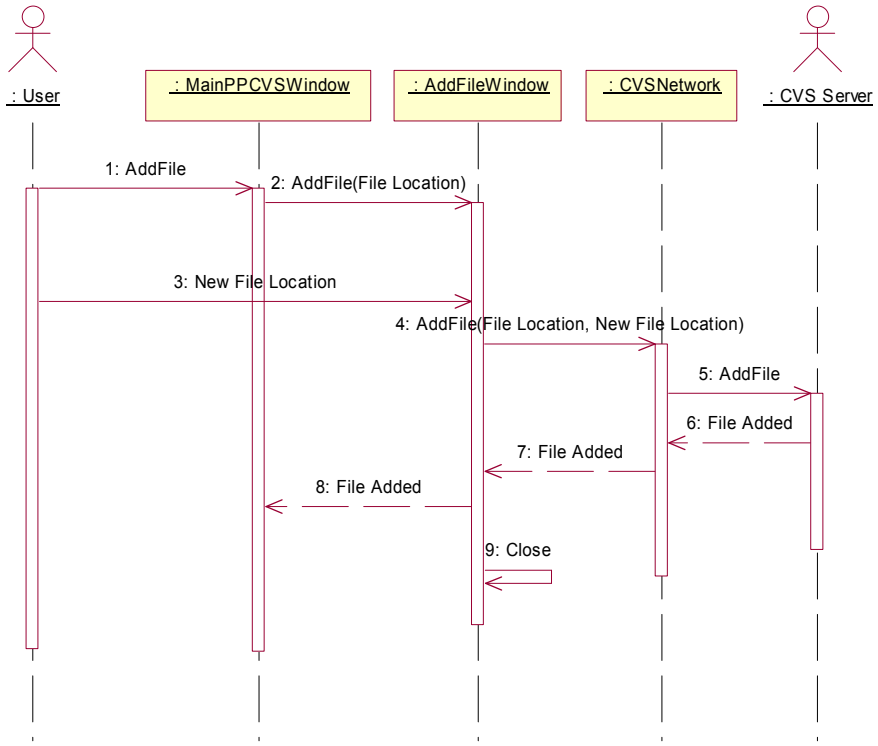


Figure 6.14 - The Add a File Sequence Diagram

In the sequence of events above in Figure 6.14 the user selects to add a file to the repository on the CVS Server. The user selects the new file location in the repository and the 'CVSNetwork' class adds this file to the repository on the CVS Server. For any addition of files to a CVS Server repository a log message is required and the system passes a default message to the CVS Server detailing which user added the new file.

6.3.3.4.4 Add a Directory

The CVS command line application does not actually provide functionality for adding an entire directory and this was implemented by firstly creating the new directory in the desired location within the repository, and individually adding each file from within the directory to be added. This process can be seen below in the sequence diagram given.

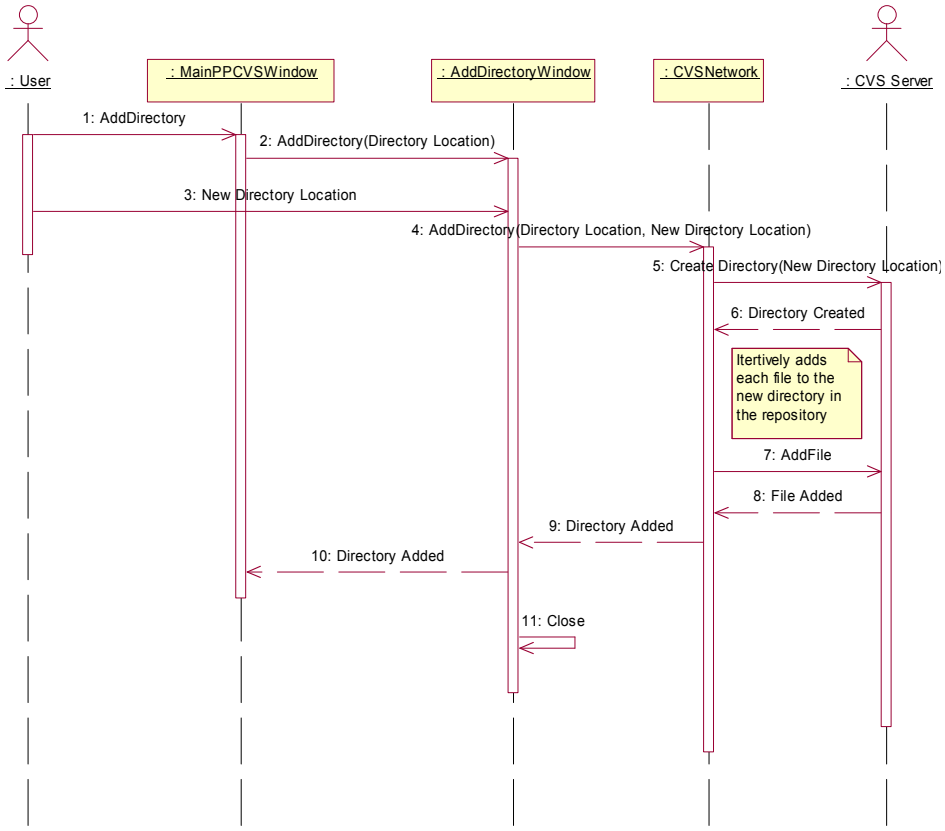


Figure 6.15 - The Add a Directory Sequence Diagram

In the sequence of events above in Figure 6.15 the user selects to add a directory to the repository on the CVS Server. The user selects the new directory location in the repository and the 'CVSNetwork' class adds this directory and the files within to the repository on the CVS Server. As the files inside the directory are added individually, a log message is passed to the CVS Server detailing the user who has added the new directory.

6.3.3.4.5 CheckOut

The checkout interaction provides the user with the ability to check out a selected module from the repository on the CVS Server to the client machine. After this is completed the user can then work on the selected module. The sequence of events involved in this operation can be seen below.

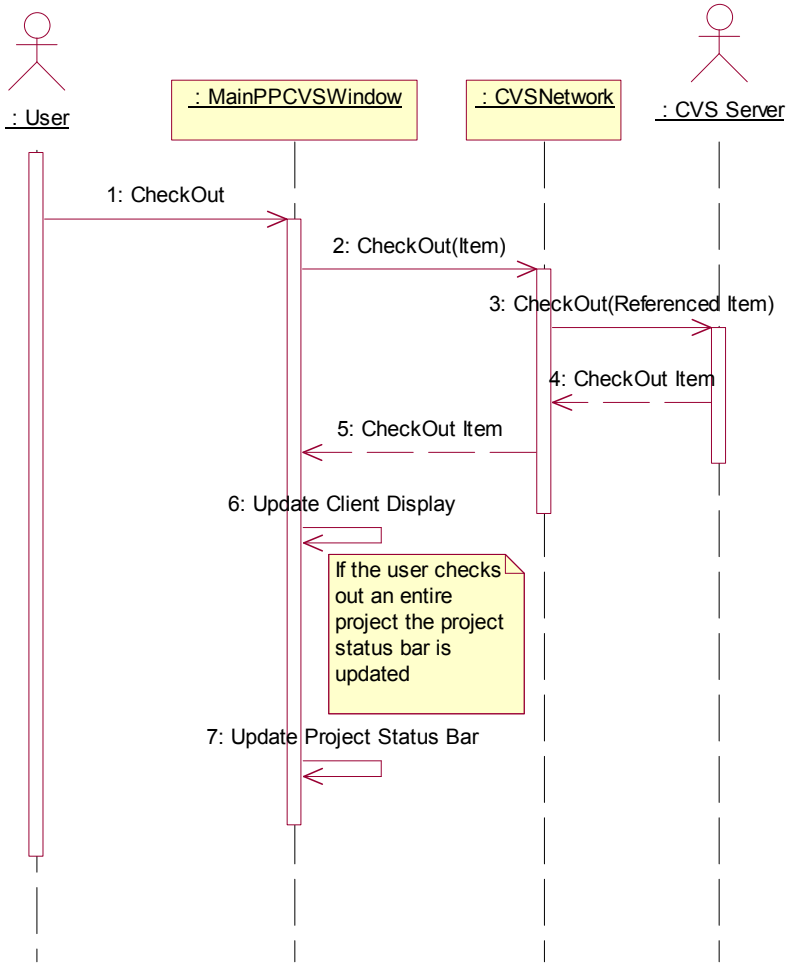


Figure 6.16 - The CheckOut Sequence Diagram

In the above sequence of events the user selects the desired item to be checked out from the repository, this information is sent to the 'CVSNetwork' class which checks out the item from the CVS Server, after which the client display is updated and the checked out item will be shown. If an entire project was checked out, the system will populate the PSB with the information in the project schedule file.

6.3.3.4.6 Commit

The Commit interaction provides the functionality for the user to select a checked out module in the client display to commit back to the CVS Server. If any modifications were made to the artefacts within the module the default text editor is opened and the user can enter a log message to detail any changes made as shown in Figure 5.4. The

sequence of events involved in this operation can be seen below in the sequence diagram.

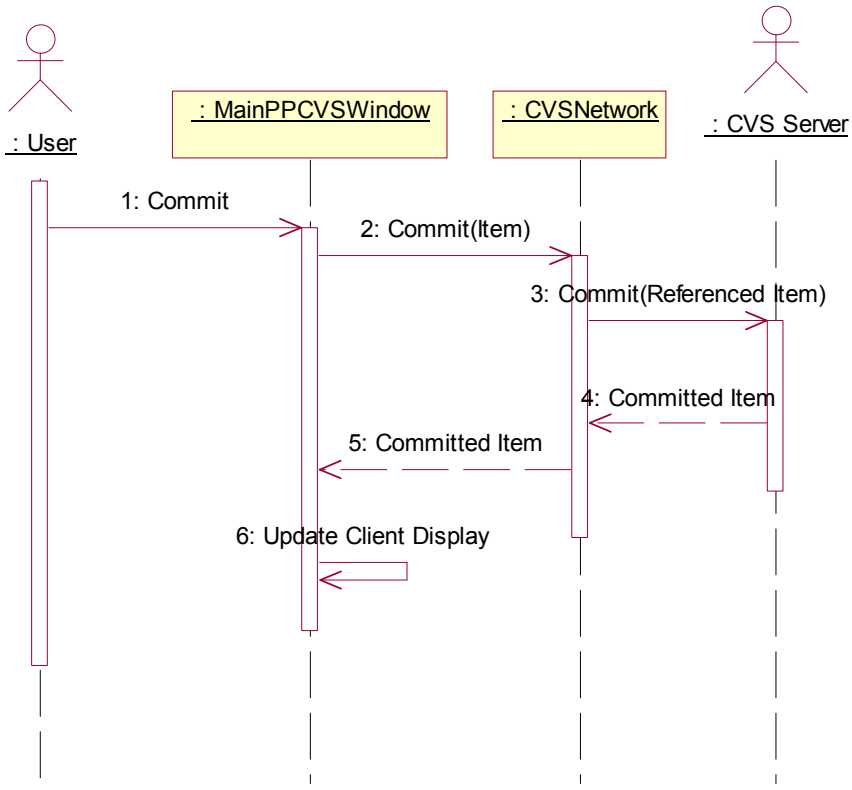


Figure 6.17 – The Commit Sequence Diagram

After any commit the client display is updated to shown the removal of the selected module from the client machine.

6.3.3.4.7 Disconnect

The disconnect interaction provides the user with the functionality to close the PPCVS client application and close the connection to the CVS Server. The sequence of events involved in this operation can be seen in the sequence diagram below.

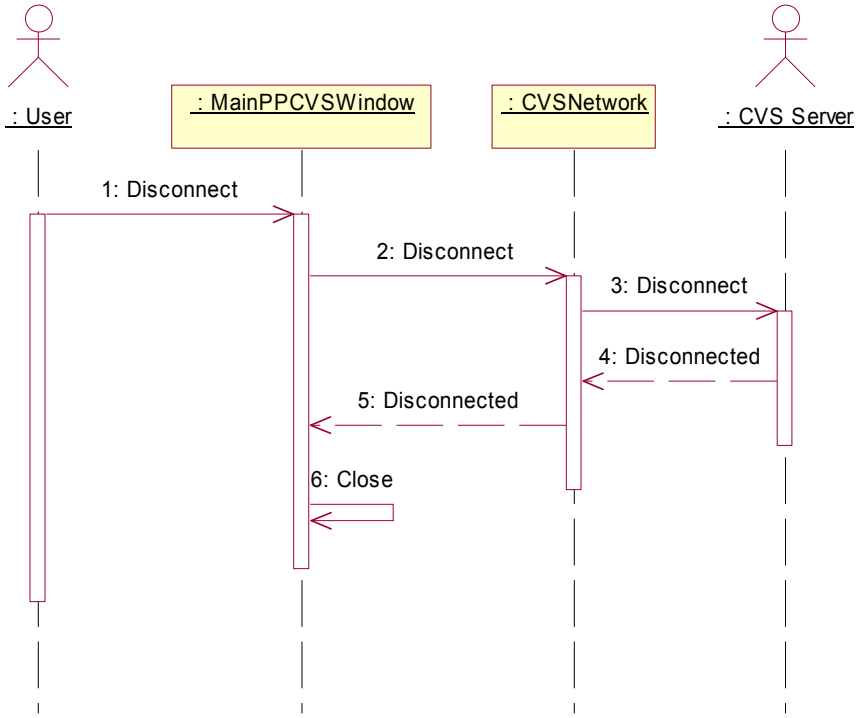


Figure 6.18 - The Disconnect Sequence Diagram

These are the interactions that were created for the PPCVS application to interface with the Linux CVS Server. They provide all the functionality required by the use cases shown in Figure 6.2. In the next section the implementation of the PSB will be discussed and the functionality it provides detailed.

6.3.3.5 The Project Status Bar Implementation

The PSB shown at the bottom of Figure 6.5 is used to detail the status of any project when an entire project is checked out from the repository. Given below is a discussion of the workflows included in the PSB, and how the PSB obtains this status information.

6.3.3.5.1 The Project Status Bar Workflows

The seven workflows shown in Figure 6.19 below are those which have been included from the workflows selected from the RUP architecture. The Costing workflow has been added to the workflow set.

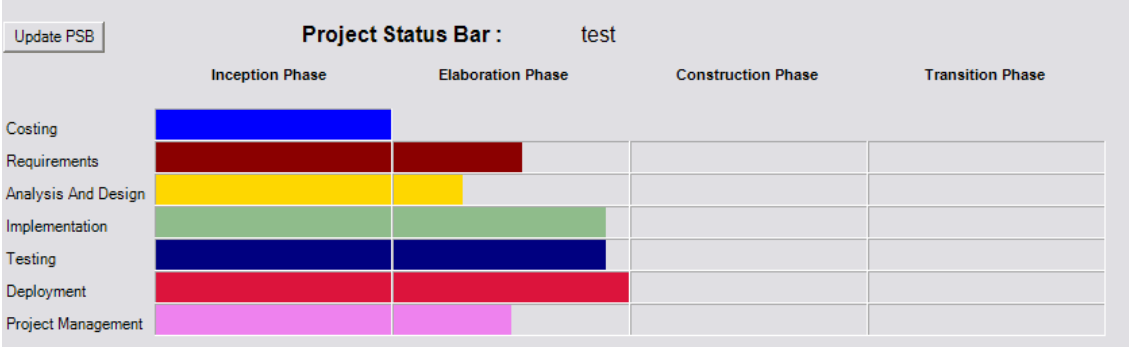


Figure 6.19 - The Project Status Bar

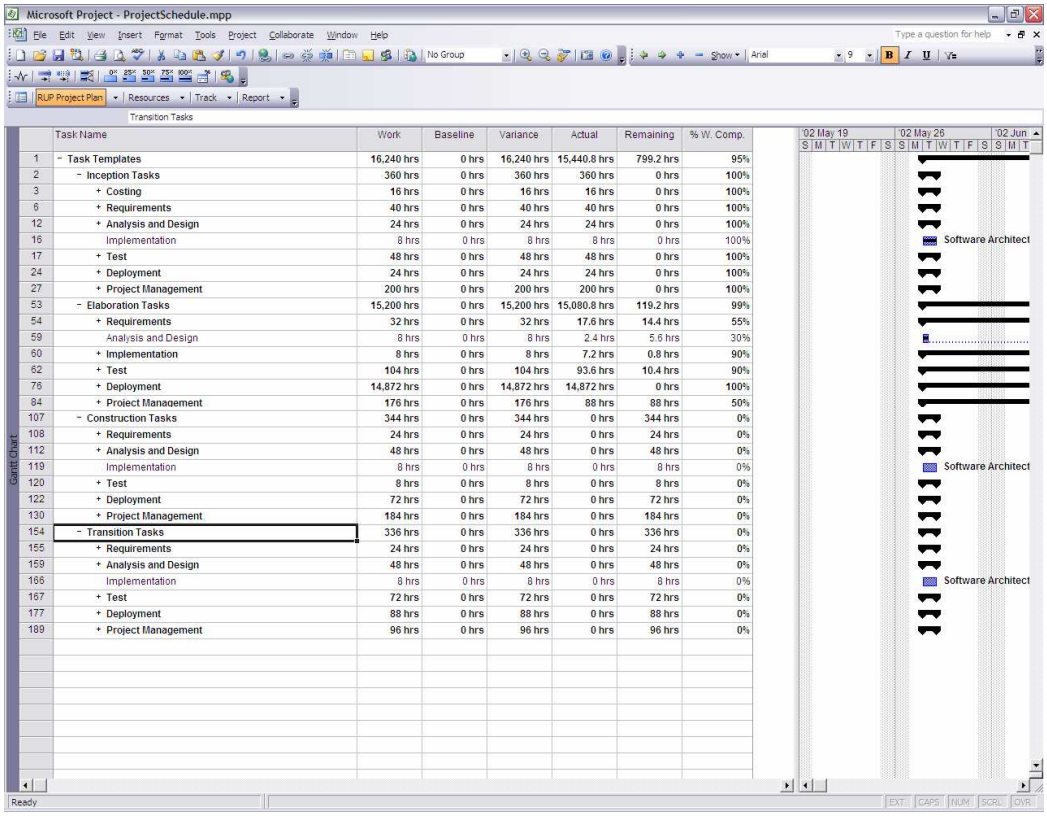
The costing workflow was added to this set of workflows since it is an important activity in the AEG project development life cycle, and RUP had no activities directly related to this process. Manzoni and Price, in a paper in which they evaluated RUP against the CMM, found that RUP was lacking in certain areas of systems management and made particular reference to the lack of support for cost management [Manzoni L.V. & Price R.T. 2003]. The costing workflow was only added to the inception phase of the architecture, as it is an activity which takes place only at the start of any project.

The workflows for business modelling, configuration and change management, and environment were removed from the original RUP architecture as they added no value to the life cycle of the projects conducted by the AEG. The workflow configuration and change management was excluded, as the PPCVS application is intended to automate all the necessary activities required by the workflow.

6.3.3.5.2 The Microsoft Project Schedule Template

In the artefacts set in Table 6.1 the template artefact Microsoft Project Status File is provided with all projects created using the PPCVS. This schedule is a template in which the information obtained from the costing questionnaire can be used to set out the overall project schedule by taking the estimated total project hours an applying this to the schedule. This schedule is configured as an iterative schedule in which the four phase's inception, elaboration, construction and transition are defined, and the activities and artefacts which are involved in each phase detailed. After this schedule

is completed, it is used throughout the duration of the project as the baseline upon which the project is conducted. The Microsoft Project RUP Schedule that corresponds to the PSB information shown in Figure 6.19 is given below.



	Task Name	Work	Baseline	Variance	Actual	Remaining	% W. Comp.
1	Task Templates	16,240 hrs	0 hrs	16,240 hrs	15,440.8 hrs	799.2 hrs	95%
2	Inception Tasks	360 hrs	0 hrs	360 hrs	360 hrs	0 hrs	100%
3	Costing	16 hrs	0 hrs	16 hrs	16 hrs	0 hrs	100%
6	Requirements	40 hrs	0 hrs	40 hrs	40 hrs	0 hrs	100%
12	Analysis and Design	24 hrs	0 hrs	24 hrs	24 hrs	0 hrs	100%
16	Implementation	8 hrs	0 hrs	8 hrs	8 hrs	0 hrs	100%
17	Test	48 hrs	0 hrs	48 hrs	48 hrs	0 hrs	100%
24	Deployment	24 hrs	0 hrs	24 hrs	24 hrs	0 hrs	100%
27	Project Management	200 hrs	0 hrs	200 hrs	200 hrs	0 hrs	100%
53	Elaboration Tasks	15,200 hrs	0 hrs	15,200 hrs	15,080.8 hrs	119.2 hrs	99%
54	Requirements	32 hrs	0 hrs	32 hrs	17.6 hrs	14.4 hrs	55%
59	Analysis and Design	8 hrs	0 hrs	8 hrs	2.4 hrs	5.6 hrs	30%
60	Implementation	8 hrs	0 hrs	8 hrs	7.2 hrs	0.8 hrs	90%
62	Test	104 hrs	0 hrs	104 hrs	93.6 hrs	10.4 hrs	90%
76	Deployment	14,872 hrs	0 hrs	14,872 hrs	14,872 hrs	0 hrs	100%
84	Project Management	176 hrs	0 hrs	176 hrs	88 hrs	88 hrs	50%
107	Construction Tasks	344 hrs	0 hrs	344 hrs	0 hrs	344 hrs	0%
108	Requirements	24 hrs	0 hrs	24 hrs	0 hrs	24 hrs	0%
112	Analysis and Design	48 hrs	0 hrs	48 hrs	0 hrs	48 hrs	0%
119	Implementation	8 hrs	0 hrs	8 hrs	0 hrs	8 hrs	0%
120	Test	8 hrs	0 hrs	8 hrs	0 hrs	8 hrs	0%
122	Deployment	72 hrs	0 hrs	72 hrs	0 hrs	72 hrs	0%
130	Project Management	184 hrs	0 hrs	184 hrs	0 hrs	184 hrs	0%
154	Transition Tasks	336 hrs	0 hrs	336 hrs	0 hrs	336 hrs	0%
155	Requirements	24 hrs	0 hrs	24 hrs	0 hrs	24 hrs	0%
159	Analysis and Design	48 hrs	0 hrs	48 hrs	0 hrs	48 hrs	0%
166	Implementation	8 hrs	0 hrs	8 hrs	0 hrs	8 hrs	0%
167	Test	72 hrs	0 hrs	72 hrs	0 hrs	72 hrs	0%
177	Deployment	88 hrs	0 hrs	88 hrs	0 hrs	88 hrs	0%
189	Project Management	96 hrs	0 hrs	96 hrs	0 hrs	96 hrs	0%

Figure 6.20 - Microsoft Project RUP Schedule

This schedule shown above in Figure 6.20 shows a project that is currently in the elaboration phase and has tasks relating to the workflows shown in the PSB. Through the utilization of this RUP schedule the tailored RUP process is followed.

The information shown in the bars of the PSB is obtained from the XML version of this artefact. All Microsoft Project files can be saved as XML files in which the details of the file are accessible. For every project that is checked out from the repository the XML scheduled is pulled across along with the project files. An XML reader parses the file, populating the PSB. The tag ‘PercentageWorkComplete’ is used to obtain the information for the PSB. The pseudo code for this process can be seen below.

```
XmlReader Object created with the schedule file
Read First XML node
While(There are still nodes of XML to read)
       If Node = Percenatge Work Complete
              Update the relevant Bar in the PSB with Node Value
       Read Next XML Node
Close the XMLReader
```

Figure 6.21 - PSB XML Reader Pseudo Code

As the project schedule is updated regularly by the project manager, this information is always up to date, showing the current status of the project for any interested parties to log onto the system and see that status of a particular project.

Besides providing status information by the clicking on any of the bars in the PSB, the relevant artefacts are opened by the PPCVS application, and the user can see the progress of a particular artefact. This can be seen in Figure 6.22 below where the entire project 'test' is checkout from the CVS Server and the user double clicks on the requirement bar and the SRS is opened.

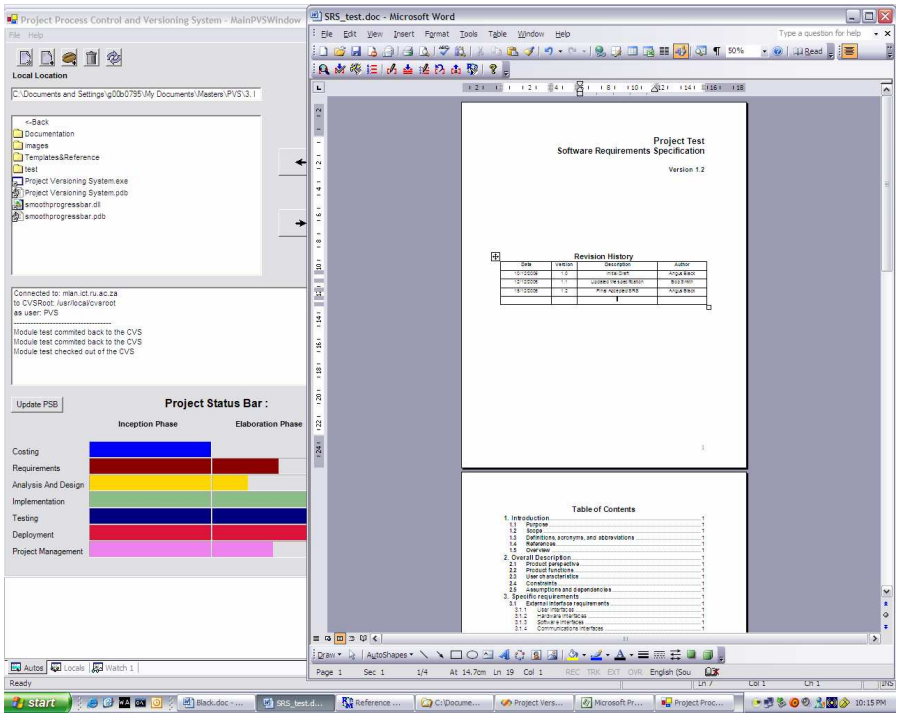


Figure 6.22 - PSB Document Opening

With the use of the version management system, project status reporting, and the process provided by this application, the PPCVS provides a platform on which the AEG can use a RUP process oriented development methodology using best practises. It also has the ability to provide for sufficient version management and status reporting.

6.4. PPCVS Additional Features

Following is a discussing into the features that the PPCVS application is lacking in comparison to typical version management tools.

6.4.1 Branching and Merging

As binary files are predominately utilized by the PPCVS application, and CVS does not provide support for the merging of binary files, this functionality was not included into the PPCVS application. Should, however, the version control tool utilized be changed to a tool that could provided support for this, such as Subversion discussed in Section 5.2.5, then this functionality should be provided.

6.4.2 Viewing History Logs

The viewing of history logs provides a mechanism in which the history of an artefact can be examined and the details of the log messages supplied for each commit examined. This is functionality that could be added to the PPCVS application. As this operation can be very easily conducted from the command line, it was not added to this first version of the PPCVS application.

6.4.3 Displaying Differences in Versions

This provides a mechanism in which the difference between two versions of an artefact can be viewed. As with the branching and merging support, this functionality was not added, as predominately binary artefacts are utilized, and CVS does not support this. As with branching and merging, should the versioning tool be changed, this functionality should be added.

6.4.4 Locking Failure

In the course of testing the PPCVS application, it was found that the CVS NT application was not locking any checked out artefacts on the Linux CVS Server. This was the case for both the checkouts from within the PPCVS application and checkouts conducted at the command line in a console window. Mechanisms need to be manually built into the PPCVS application to ensure this functionality is provided, to avoid any commits overwriting updated artefacts.

Apart from binary file support, the viewing of history logs and an effective locking mechanism needs to be added to the next version of the PPCVS application, to ensure that the versioning mechanism is effective.

6.5. Chapter Summary

In this chapter the PPCVS application was introduced and the functionality the system offers to the AEG discussed.

The artefact set which is available to all new projects created by the PPCVS was introduced, and the details of each artefact described.

The configuration of a Linux CVS Server enabling it to communicate with the PPCVS application was shown.

The analysis and design of the PPPCVS client application was discussed in extensive detail, and certain appendices referenced showing how the design was conducted. The associated GUI windows with the application were shown and the functionality of each introduced and discussed.

The version management tool CVS which is used by the PPCVS application was discussed, and the interactions between the client PPCVS application and the CVS Server was shown by means of sequence diagrams.

The PSB, which provides for the status reporting, was shown, and the process upon which it is based was discussed.

In the concluding chapter the important contributions this research has made will be discussed and possible future work that could be done based upon this research will be detailed.

Chapter 7 - Conclusion

This research has considered how quality assurance can be achieved in a remote client/contractor context through the utilization of process improvement techniques. It has focussed on techniques to improve the Audio Engineering Group's software development process by improving project management, costing techniques, configuration and change management, and an overall project process. All these processes were selected for improvement after an extensive evaluation of the context in which the AEG conducts their projects. In each chapter various important areas were researched and the following was explored in each chapter:

- In Chapter 2 the context in which the AEG projects were conducted was introduced. Various tools were introduced and particular importance was place on the IEEE recommended practise for requirements elicitation, the KDOC and DOXYGEN automated commenting tools were evaluated, resource and tasking tracking using the Microsoft Project application was discussed, and various testing procedures explored. Finally the requirements for the PPCVS application were listed.
- In Chapter 3, process management was discussed and how software standards can help in process improvement explained, particular reference was made to the ISO standards set and the CMM.
- In Chapter 4 costing techniques were explored and the COCOMO costing model was examined in detail. The improved COCOMO II model was introduced and the differences between itself and its predecessor explained. How the COCOMO costing model was utilized by the AEG was explained. The potential pitfalls the COMOCO model has were listed.
- In Chapter 5 the typical functionality of version management tools were discussed. The products Microsoft SourceSafe, Merant Professional, Rational ClearCase, Concurrent Versions System and Subversion were introduced and their functionality discussed. The motivation for the selection of CVS into the PPCVS application was given
- In Chapter 6 the system developed for this research was introduced and its analysis, design and implementation discussed.

The Project Process Control and Versioning System application was created to solve a business need of the AEG. This need was formalized as the ability for their team to work seamlessly in a remote context, with a process built on best practises and having integrated version control and status reporting mechanisms. The PPCVS solves this business need and supplies them with an effective process that can be used on any project they undertake. The PPCVS application provides a structured mechanism, allowing a project to be quickly initiated and promotes the use of the well known RUP methodology. Given below is a block diagram showing the overall flow of work for a project using the PPCVS.

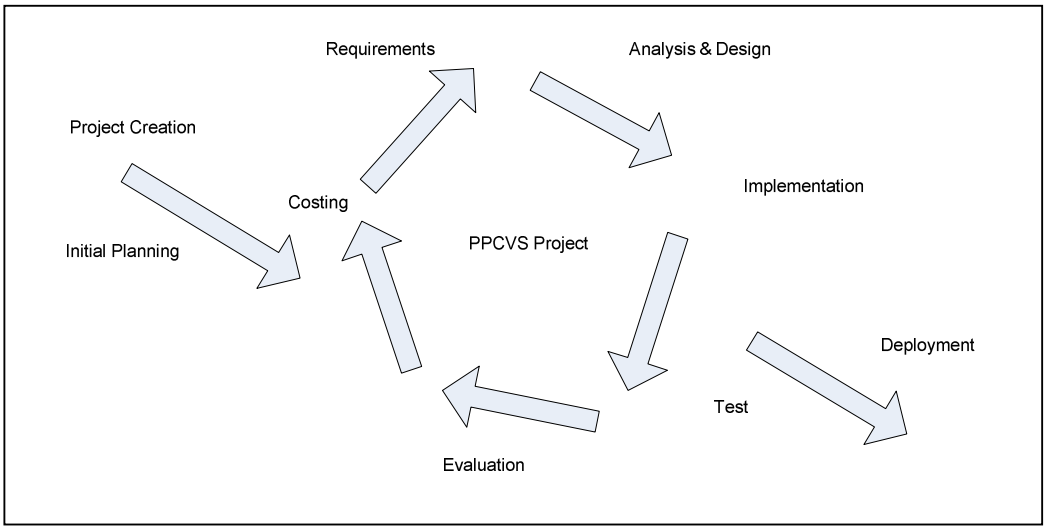


Figure 7.1 - The PPCVS Project Process

Through the use of the Microsoft Project RUP Schedule, which is tailored for the project process shown above in Figure 7.1, an iterative development model is promoted. This iterative development model promotes the use of techniques to improve the processes conducted by the AEG and ensures that any projects carried out by the AEG are conducted with quality in mind.

7.1. PPCVS Application Evaluation

For the PPCVS application to be considered an effective solution for the AEG, the requirements that created the business need for the solution discussed in section 2.4

must be evaluated against the functionality of the PPCVS application. Five key requirements were identified:

- Remote Process Control
- Repeatable Process
- Version Control
- Status Reporting
- Time Tracking

7.1.1 Remote Process Control

The PPCVS application allows any user with an internet connection to log onto the CVS Server and access any artefacts, irrespective of their geographic location. This gives the users the ability to work simultaneously on any project, and engage any project process remotely.

7.1.2 Repeatable Process

As the process defined for the PPCVS application is tailored for the AEG and this process is available to be used on every project, a repeatable process built on best practices has been defined. When any new project is created using the PPCVS application, the user can select from the artefact set listed in Table 6.1, this having the result of improving the repeatability of the overall process.

7.1.3 Version Control

The PPCVS application provides an effective, easy to use version control mechanism which interacts with a CVS. The version control mechanism built into the PPCVS applications allows the user to:

- Connect to a CVS Server.
- Create new projects on the CVS Server.
- Add artefacts to any project on the CVS Server.

- Check out artefacts from the CVS Server.
- Commit any modified artefacts to the CVS Server.
- Disconnect from the CVS Server.

These mechanisms provide the basic functionality that a typical version control tool should provide. The PPCVS application doesn't however, provide the mechanisms for branching and merging, viewing history logs, and displaying differences in versions. Apart from these functions there was also a fault found in the locking mechanism between a Windows CVS NT client and a Linux CVS Server. All these additional features and the locking failure were discussed in section 6.4.

7.1.4 Status Reporting

The PSB provides a detailed status reporting mechanism in which users can very easily check the status of any particular process within a project and view the progress of any artefact within a project. This will provide very valuable information to any client, and improve the communication between the AEG and any of their clients, as they can very easily view the progress of any artefact through the PPCVS application.

7.1.5 Time Tracking

Through the use of the RUP Microsoft Project schedule template supplied with every project created by the PPCVS application, the project manager of the AEG is able to track the utilization of any resources as shown in Section 2.3.6.2.

The PPCVS application provides and effective solution to the problem that initiated this research. It successfully integrates the remote process control, version control and status reporting mechanisms into a single easy to use application.

7.2. Future Work

As a result of this research, a number of additional research topics have been discovered, which if pursued could further improve the software development process:

- The development of an integrated version management system within the RUP application.
- The effective integration of cost management into RUP.
- The integration of COCOMO directly into RUP and the utilization of Rational Rose to calculate functional point counts.
- The development of a concise and reliable C# CVS API that could handle all the interactions offered by the CVS protocol.

Even though the PPCVS application may not provide all the functionality of a typical version management tool, it does provide the AEG with something they were never using before. Through the use of this application their project process will definitely be improved as they are able to work together on any project with a repeatable project process that was tailored for their context and provides status reporting. Ian Sommerville states in his book "*Software Engineering*" that good project management will not ensure the success of the project but bad management will most definitely result in project failure [Sommerville I. 1995]. It is hoped that in providing this unique integration of process management and version control this research has contributed to good project management.

Appendix A – Costing Questionnaire

Project Name	mLan Installation Designer
Number of Uncommented Lines of Code (rough estimate)	10774
Number of hours spent on the project	855
Process Exponent Parameters	
	If there is any uncertainty with the selection please select the one stated Nominal
Precedentedness(PREC)	
How comparable is the project to previous projects done by this team	Largely Familiar
Development Flexibility(FLEX)	
How flexible are the requirements	General goals
Architecture risk resolution(RESL)	
Degree to which the architecture has already been defined	Often 60%(Nominal)
Team Cohesion(TEAM)	
Degree of cohesion in interactions with stakeholders	Highly cooperative
Process Maturity(PMAT)	
SEI Maturity rating for organization	Level 2+(Repeatable)(Nominal)
Project Characterization Parameters	
Required Reliability(RELY)	
The effect of failure of the product	Moderate, easily recoverable losses(Nominal)
Database size(DATA)	
Volume of data required to test the product	Low
Product Complexity(CPLX)	
The degree of coding complexity	Nested code, standard math routines, multiple files(Nominal)
Required reuse(RUSE)	
Is the product going to be used in other systems	Across project(Nominal)

Documentation(DOCU)	
The volume of documentation is going to be produced	Right-sized to life-cycle needs(Nominal)
Execution Time Constraint(TIME)	
The amount of CPU time the product will need	Nominal
Main Storage Constraints(STOR)	
The amount of main memory the product will utilise	Nominal
Platform volatility(PVOL)	
The anticipated change in platform(OS, DBMS, etc)	Major change every 12 months; Minor change every month
Analyst capability(ACAP)	
The capability of the analyst for the project	High
Applications experience(AEXP)	
Level of experience with the product domain	6 years
Programmer capability(PCAP)	
The capability of the programmers for the project	Very High
Platform experience(PEXP)	
Level of experience with the target platform	6 years
Personnel continuity(PCON)	
The turnover rate for the organization	3 % per annum
Language/tool experience(LTEX)	
Level of experience with the language and tools to be used	6 years
Multiple-site development/Team communications(SITE)	
The location of the team members and type of communications	Multi-city or multi-company. E-mail(Nominal)
Use of software tools(TOOL)	
Type of tools used	Strong, mature life-cycle tools, moderately integrated
Required development schedule(SCED)	
Whether or not the schedule has been compressed from the Nominal Schedule	100 % of nominal schedule(Nominal)
EAF	0.213257856
C1	2.94
C2	3.67
P1	1.0226

P2	0.3027
Functional Point Estimate for Uncommented Lines of Code	0
Effort	7.12789117
Total Project Time(Months)	6.7
Total Project Time(Hours)	1064
Project Phase Breakdown	
Phase	***Phase Estimate in Hours***
Requirements analysis	43
Design	128
Programming	468
Test Planning	64
Verification and Validation(Testing)	149
Project Office	74
Configuration Management and Quality Assurance	74
Documentation	64

Appendix B – PPCVS Vision Document

Project Process Control and Versioning System

Vision

1. Introduction

In this document the business need that brought about the development of the Project Process Control and Versioning (PPCVS) system will be described. The high level functionality of the PPCVS application will be explored and the details as to how the PPCVS application fulfills this business need explained.

The remainder of this document will described the positioning of the PPCVS in the Audio Engineering Group (AEG), the stakeholders within the AEG and an overall product perspective.

2. Positioning

As the AEG needed a formal software process methodology to follow in order to improve their process management, as well as a version control and status reporting system, this product provides a single solution that can solve these business needs.

2.1 Problem Statement

The problem of	Lack of an easy to use efficient version management system. A non repeatable software development process. No status reporting.
affects	All team members of the AEG.
the impact of which is	Disjoint or no version control with no status reporting and a software process lacking in repeatability and formalization.
a successful solution would be	an improved repeatable software development process in which team members can simultaneously work on versioned artefacts with built in status reporting.

2.2 Product Position Statement

For	AEG team members.
Who	Are involved in software development projects for overseas clients/contractors.
The Project Process Control and Versioning System	Is a software product.
That	Is a software process improvement tool with built in support for version control and status reporting
Unlike	Any other disjoint process improvement, version control and status reporting software.
Our product	Provides all the functionality of process improvement, version control and status reporting into a single easy to use product.

3. Stakeholder Descriptions

This section describes the users that will use the PPCVS application, there are three typical types of users:

- Project Manager
- Analyst
- Developer

3.1 User Summary

Name	Description	Responsibilities
Project Manager	Responsible for the management of all projects conducted by the AEG.	Opening any new projects within the PPCVS application and ensuring that they engage in all the necessary activities required by their role.
Analyst	Responsible for the design of any system developed by the AEG.	Fulfilling any activities required by their role within the processes included in the PPCVS application. And ensure that all artifacts worked upon are versioned by the PPCVS application and made available to all the team members.
Developer	Implementation of all systems created as contract work for overseas clients/contractor	To ensure that their implementation artifacts are sufficiently versioned by the PPCVS application.

3.2 User Environment

The users of the PPCVS application are highly educated professionals with a high level of computer literacy.

The PPCVS application will be used by users located in different geographic locations around the country, communications will be done via standard internet communications using the Concurrent Versions System (CVS) NT protocol.

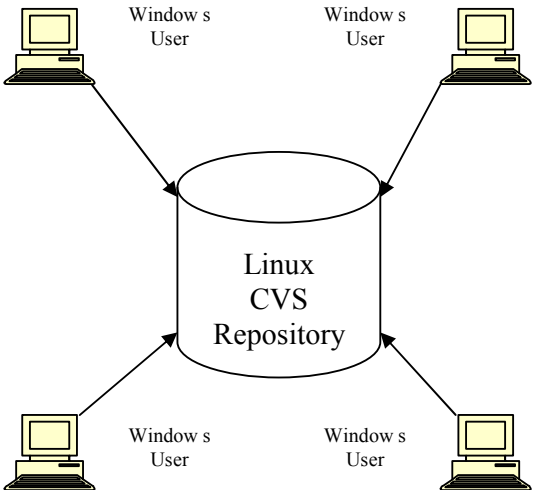
Users will use the PPCVS application in the Microsoft Windows environment and interact with products in the Microsoft Office Suite and Microsoft Visual Studio .NET environment in modifying any artifacts.

4. Product Overview

This section details the high level view of the PPCVS application discussing the products capabilities and the interfaces it makes with other applications, protocols and environments.

4.1 Product Perspective

The PPCVS application allows numerous users to communicate with a Linux CVS Server repository through the use of the PPCVS client Graphical User Interface (GUI) over a network using the CVS NT protocol. This concept can be seen in the diagram below.



4.2 Assumptions and Dependencies

The following assumption and dependencies are pertinent to the PPCVS application:

- The PPCVS application will be used from within a Microsoft Windows environment.
- The CVS Server used for the repository accessed by the PPCVS client GUI will run cvs on a Linux operating system.
- The PPCVS application requires CVS NT to be installed on the client machine before the application can be utilized.
- All artifacts other than the implementation artifacts will be modified in a tool provided by the Microsoft Office suite or suitable alternative.

4.3 Needs and Features

Need/Feature	**Description**
Connect to a CVS Server	Provide the ability for users to connect to a remote CVS Server through the PPCVS client GUI

Create a New Project on the CVS Server	Provide the capability for users to create a new project on the CVS Server. In so doing allowing users to select from a set of template and reference material to include in the new project on the CVS Server.
Add Artifacts to the CVS Server	Provide the user with the ability to add any further artifacts to a project within the repository of the CVS Server.
Check Out	Provide functionality in which the user can obtain artifacts from the CVS Server and modify or view them.
Commit	Provide functionality in which the user can commit any changes to artifacts back to the repository.
Disconnect from the CVS Server	Provide the ability for the user to close the connection between the PPCVS client GUI and CVS Server

4.4 Alternatives and Competition

At this time there is no alternative product which provides this tailored process management with built in version control and status reporting for the AEG context.

Alternatively the AEG could make use of separate products that could solve these business needs individually and not from one product such as the PPCVS.

Appendix C – PPCVS Installation and User Documentation

Project Process Control and Versioning System

Installation Guide and User Documentation

August 2005

Ver 1.0

Release Notes

The Project Process Control and Versioning System (PPCVS) has been tested running Microsoft Windows XP Professional with Service Pack 2.

The PPCVS utilizes a Linux CVS and has been tested with CVS version 1.11.17 running on Red Hat Linux with kernel 2.4.27.

The PPCVS utilises the CVSNT 2.5.01 command line executable to communicate with the Linux CVS and is packaged with the installation.

All files and folders checked into the repository must not contain spaces in their names.

When adding files or folders from within a checked out module please ensure that the module is moved away from the PPCVS installation folder, that additions are done, and then the module is moved back to the PPCVS installation folder for the commit.

All check outs are done to the PPCVS installation folder.

All commits must be done from the PPCVS installation folder.

1. Installation

1.1 System Requirements

The Project Process Control and Versioning System (PPCVS) requires Microsoft Windows XP on the Client Side and a suitable Linux distribution running CVS version 1.11.17 or later.

The system requires a minimal amount of hard disk space, approximately 15 Mb. The amount of space required is dependant on the size of the modules being worked on. The system also requires a suitable network connection for communication between the PPCVS and the Linux CVS.

For project status reporting, the system requires Microsoft Project 2003 and the templates require Microsoft Word and Excel 2003.

1.2 Installation - PPCVS

Installation of the system is done by double clicking on the *Project Versioning System.msi* file in the installation folder provided.

The installer is a standard Microsoft Windows installer package. Please select the location you wish to install the system to and which user you wish to give access to the system on your machine.

This installer will create a PPCVS folder within the Programs Tab in the Start Menu.

To run the application click on the PPCVS icon within the PPCVS folder in the Programs Tab.

1.3 Installation - CVSNT

After the PPCVS is installed it is necessary to install CVSNT, this has been supplied with the PPCVS installation.

To start the CVSNT installation double click the *cvsnt-2.5.01.1927.msi* file in the installation folder. Once the installer has been opened please select the typical installation, this will install all that is required by the PPCVS.

Once the installation of CVSNT is complete it is necessary to reboot your computer.

2. Logging into the System

After selecting the PPCVS icon from within the PPCVS folder, the Login Window will appear, as seen below.

There are four fields which need to be filled in before you can log into the PPCVS system:

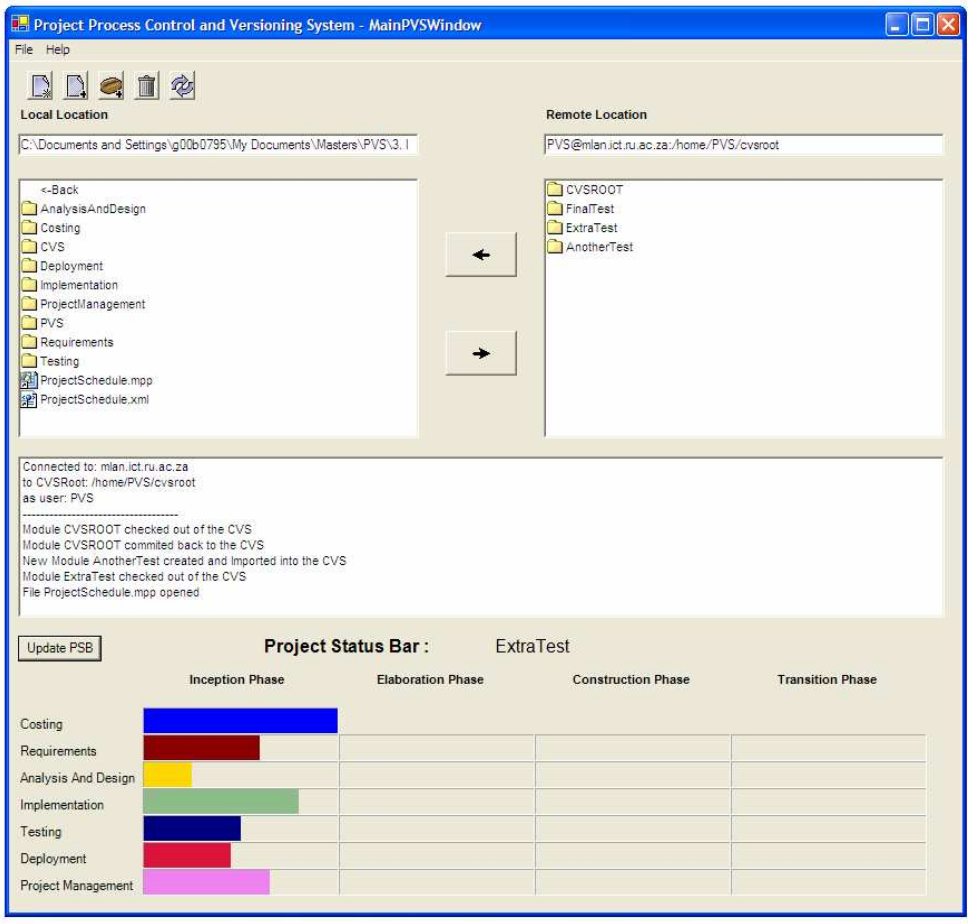
- CVS Address – This represents the name of the CVS server you wish to connect to, entered in the form as seen above servername.domain.location
- CVS Root – This represents the location of the CVSROOT on the server you are connecting to, entered in the form /location/cvsroot.
- Username – This represents the name of the user whose account on the server is going to be used to connect to the server.
- Password – This represents the password for the selected user.

Once all the above fields have been filled in, click the login button to connect to the CVS Server and open the Main PPCVS Window.

If the connection to the server fails the Main PPCVS Windows will be opened, and an error message displayed. Please close the application and open it again to connect.

3. Navigating the MainPPCVSWindow

Once the login has been completed successfully the Main PPCVS Window will be displayed as seen below.



The left hand display panel in the MainPVSWindow represents the local files on the client machine. The right hand display panel represents the modules stored on the connected CVS server. The buttons on the top from the left represent the following:

Create New Project

Add File To Module

Add Folder To Module

Delete File or Folder

Refresh the Client Side Display and Remote Side Display

Each one of these buttons and functions will be discussed in a later section.

Below the two displays is the status window. This window will display any information about the connection to the CVS Server and the actions performed on files, folders and modules on the CVS Server.

Below the status window is the Project Status Bar (PSB), this bar is used to display information pertaining to the status of the project. The PSB will be discussed in detail in a later section.

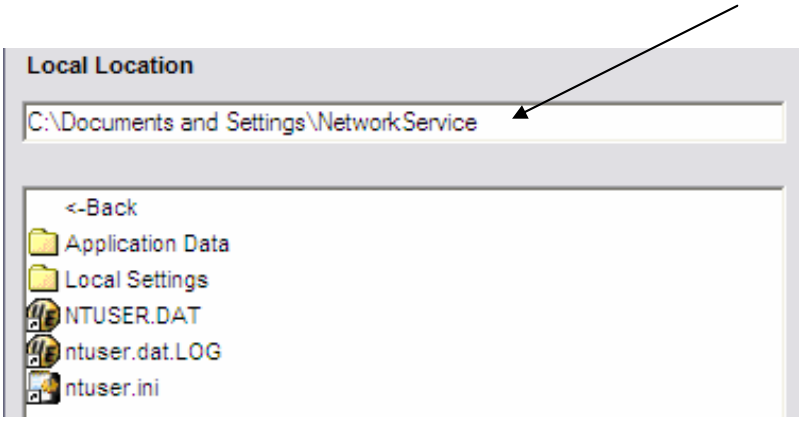
3.1 Navigating the Client and Remote Side Files

The navigation of both the client and remote side displays is very similar in nature to that of the Windows Explorer.

3.1.1 Navigating the Client Side

On application start up, the client display will always be in the PPCVS installation folder. By double clicking on any folder within the display it will open that folder in the display. By double clicking on <-Back in the display it will go one folder up in the directory structure.

All movements in the client display update the text box above the display which shows the current location on a drive, see below in the picture.

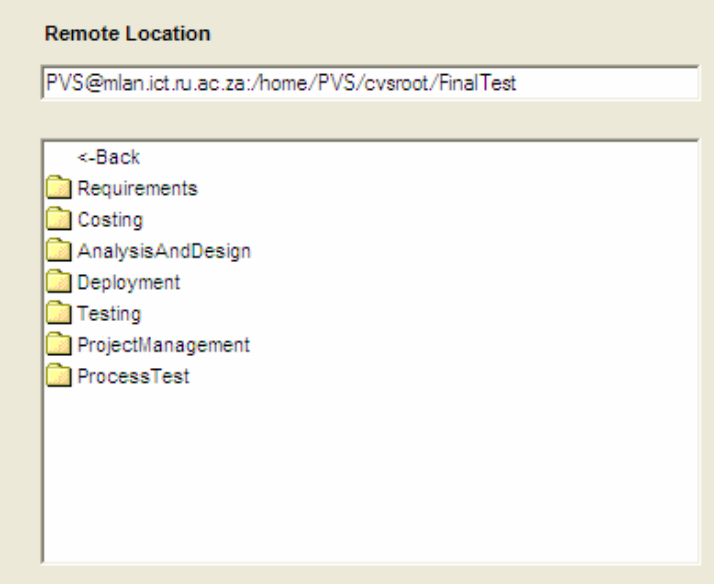


By double clicking on any file in the client display, the file will be opened with the appropriate application registered with Windows for that file type.

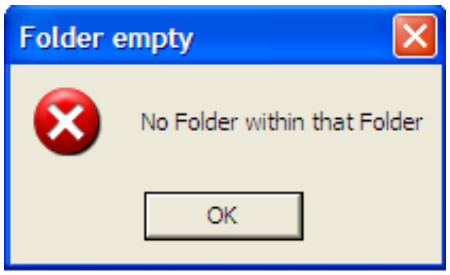
3.1.2 Navigating the Remote Side

The navigation of the remote side is very similar to that of the client side except the display only displays the folders held in a certain module on the CVS Server.

On application start up, the remote display shows the modules stored in the CVSROOT of the CVS Server. By double clicking on a module the application will update the display to show the folders held within that module. Below is a picture of the remote display after the module FinalTest has been opened.



By double clicking on a sub folder within a module you can further explore the folder structure within a module, if however there are no folders within a selected folder an error message will be displayed as seen below.



All movements in the remote display update the text box above the display which shows the current location in the CVSROOT of the CVS Server.

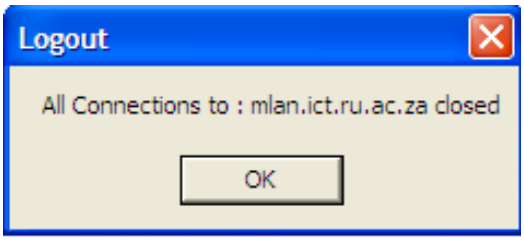
By double clicking the <-Back in the remote display the display will go one folder back the in the folder structure, this can be done until the display returns back to the CVSROOT.

3.2 Refreshing the Client and Remote Displays

By clicking the (Refresh) button both the client and remote displays will be updated. The client side will get the latest folders and files held within the current folder and the remote side will return to the CVSROOT of the CVS Server.

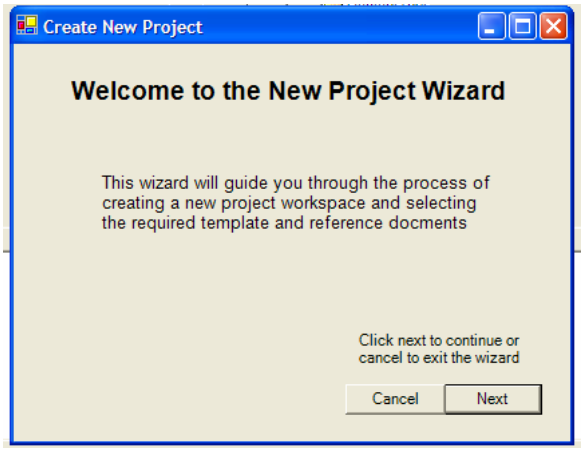
3.3 Exiting the PPCVS application

By clicking on the close button in the top right corner of the PPCVS application or selecting Exit from the File pull down menu, the PPCVS application will be closed. All connection to the CVS Server will be closed and if the exit is executed successfully, the exit message given below will be shown, and after clicking OK the PPCVS application will close.



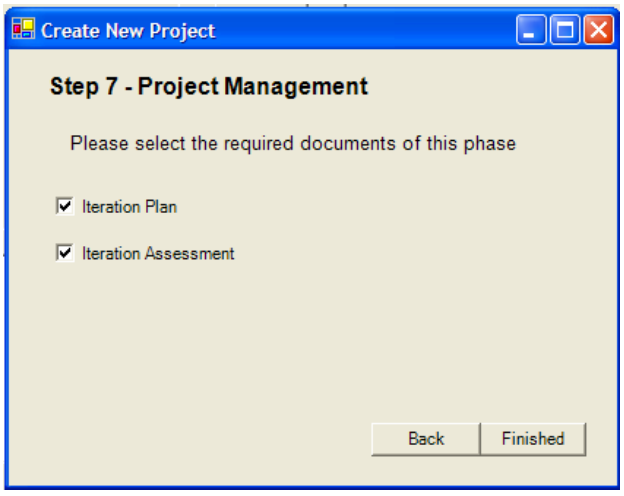
4. Creating a New Project

To create a new project on the CVS Server click the (Create New Project) button, by clicking this button the New Project Wizard will be opened, as seen below.

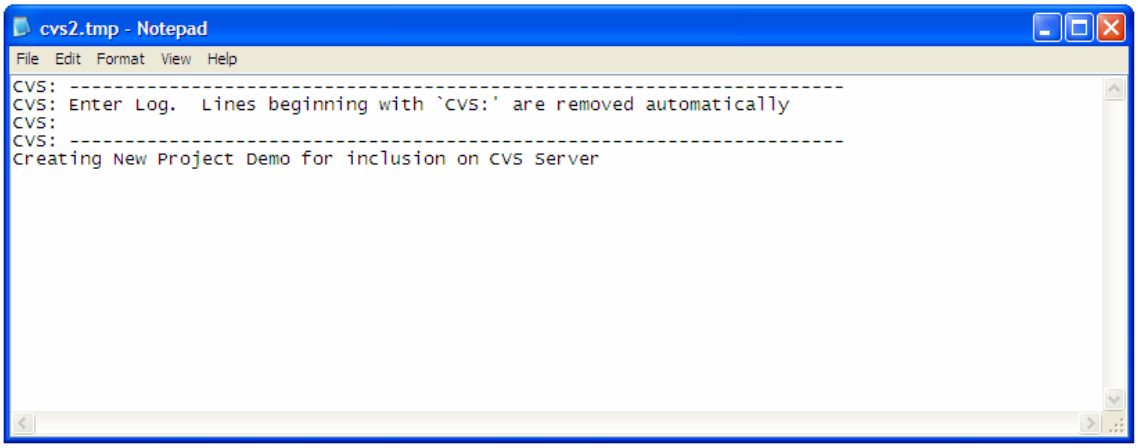


The wizard will take you through a series of screens where the project name can be entered and template and reference material required for the project selected. The first screen requests the name of the new project, please don't leave any spaces in the project name as this will create errors on the CVS Server.

After the project name has been entered a series of screens will be shown with check boxes and template and reference material. To select an item for inclusion in a new project simply click on the check box next to the item. On the last screen of the wizard the Finish Button will be displayed as shown below.



By clicking this button the new project will be created and all the selected items sent to the CVS Server. The default Windows text editor will be opened and a log message can be entered for the new project, this can be seen below in the example log.



After the log has been entered please close the log and choose to save it. If the project is created successfully a Message Box will be shown stating the project's creation was successful and the application will return to the Main PPCVS Window. These log messages are all kept on the CVS Server and store information pertaining to who creates or edits files and folders on the CVS Server as well as information such as time, from what machine etc.

All the files for a particular project are stored within a root folder on the CVS Server, this root folder is referred to as a module. Following is information explaining how to access these files within a particular module.

5. Checking Out Modules and Folders

Both entire modules and sub folders inside modules can be checked out from the CVS Server.

In order to check out an entire module from the CVS Server, select the desired module in the CVSROOT in the remote display and click the ← (Check Out) button found between the client and remote displays. This will then check out the entire contents of the module to the PPCVS installation folder.

To check out a sub folder from within a module browse to the location of the desired folder in the remote display and select the folder and click the ← button. This will check out the entire contents of that particular sub folder from within the module to the PPCVS installation folder.

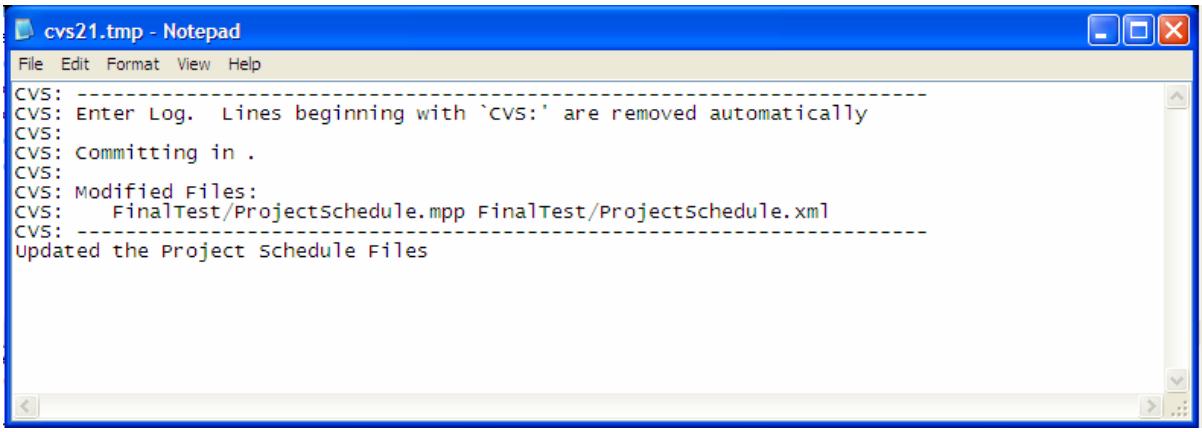
Once a check out has been executed the application will display a Message Box stating that the desired items have been checked out.

Once a check out has been executed it is possible to browse the checked out items and open and edit any files within those folders by browsing the folders in the client display.

6. Committing Modules and Folders

Once a module is required to be committed back to the CVS Server, the module must be in the PPCVS installation folder. Whether the checked out items are entire modules or sub folders, the main module root must be selected from the PPCVS installation folder, and the → (Commit) button clicked. The application will prompt for confirmation of the commit.

If any of the files within the checked out module have been edited, Windows' default text editor will be opened and a log message can be entered for the edited files. An example of a log message can be seen below. The log will show which files have been edited since the last check out.



After the log is entered please close the log and choose to save it. Once a commit has been executed successfully a Message Box will be shown stating that the desired items have been committed to the CVS Server. After the OK button has been clicked, the committed items will be removed from the client display.

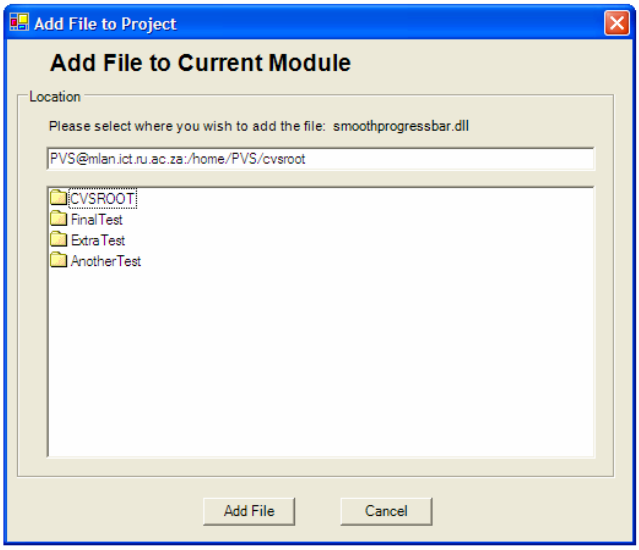
7. Adding Files and Folders to a Module

Please ensure that when adding Files and Folders from within a checked out folder to a module, that the checked out folder is copied away from the PPCVS installation folder!

Files and folders from anywhere on the client machine can be added to a module on the CVS Server.

7.1 Adding a File to a Module

To add a file to a module, select the desired file to be added in the client display and click the (Add File) button. This will then open the add file screen, as seen below.



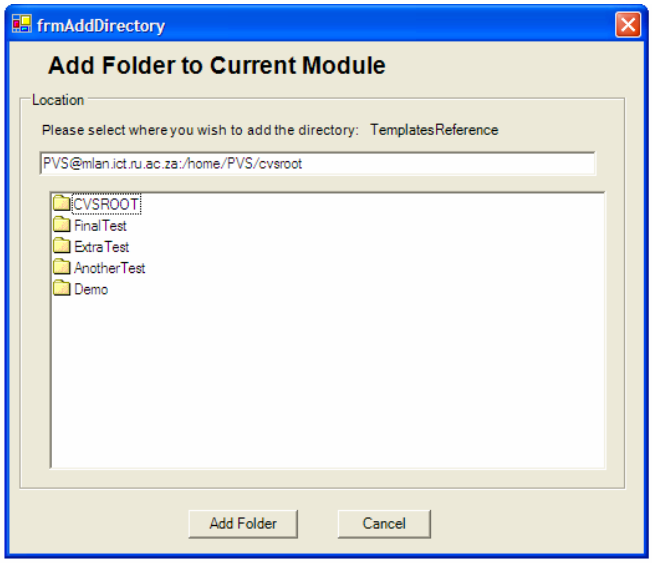
The Add File Screen shows the name of the file to be added, and a window where the desired location of the file to be added can be selected, known as the *'add file display'* window. The *'add file display'* window can be browsed in much the same way as the remote display found on the Main PPCVS Window.

Once the desired folder is found into which the file should be added, click the *'Add File'* button after selecting the folder. If the file is required to be added to the root of

the module, select the module name only. After the *'Add File'* button is pressed a log message much the same as the one found when creating a new project, or committing edited files will be opened. After the log has been entered, the file will be added to the folder within the desired module, and the Main PPCVS Window will be displayed.

7.2 Adding a Folder to a Module

The adding of a folder to a module is much the same as adding a file. To add a folder to a module, select the desired folder to be added in the client display, and click the  (Add Folder) button. This will then open the add folder screen, as seen below.



The Add Folder Screen shows the name of the folder to be added, and a window where the desired location of the folder to be added can be selected, know as the *'add folder display'* window. The *'add folder display'* window can be browsed in much the same way as the remote display found on the Main PPCVS Window.

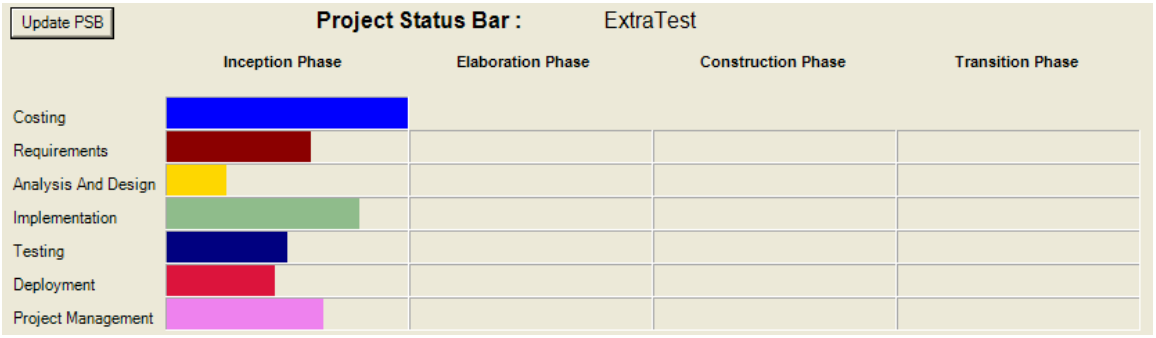
Once the desired folder is found into which the folder should be added, click the *'Add Folder'* button after selecting the folder. If the folder is required to be added to the root of the module, select the module name only. This will then add that folder to the folder within the desired module and return to the Main PPCVS Window.

8. The Project Status Bar

The Project Status Bar (PSB) is located at the bottom of the MainPPCVSWindow and as previously mentioned, it displays the status of projects.

To utilize any functions of the PSB it is necessary to check out an entire module

The project status bar is broken into certain workflows and phases of a project. The items in the vertical axis display the workflows in a project, and the items in the horizontal axis show the phases of a project. All the information shown in the PSB is obtained from files created with any new project created by the PPCVS. In order for the PSB to display project status information, an entire module needs to be checked out from the CVS Server. Below is an example of a PSB with status information.



The files *ProjectSchedule.mpp* and *ProjectSchedule.xml* are located in the root of a module and are where the PSB gets its information from.

In order for the current status of a project to be displayed, it is necessary to check out the entire module and double click on the *ProjectSchedule.mpp* in the client display. This will open the file in Microsoft Project. Once the status has been successfully updated, the file needs to be saved, and then further saved as *ProjectSchedule.xml* by selecting the *'Save As Tab'* in the File pull down menu, and choosing to save the file as XML.

The PSB can then be updated to check if the changes to the status were correct by clicking the *'Updated PSB'* button found on the top left of the PSB. After the module

is no longer required, it can be committed back to the CVS Server. The next time the entire module is checked out, the new project status information will be displayed in the PSB.

By clicking on any of the coloured bars found within the PSB, it will open the relevant documents associated with that workflow. These files can be browsed or worked on, and will be checked back to the CVS Server with the rest of the module when a commit is performed.

Appendix D – PPCVS Use Case Flow of Events

Project Process Control and Versioning System

Use Case Flow of Events

1. Flow of Events for the ConnectToCVSServer Use Case

1.1 Preconditions

There are no preconditions to this flow of events

1.2 Main Flow

This use case begins with the user opening the PPCVS client application and the 'LoginWindow' opening. The user can type in the CVS Server Name, CVSROOT location, Username and Password into the 'LoginWindow'. The user can then LOGIN or CANCEL the 'LoginWindow':

If the activity selected is LOGIN, the S-1: Login subflow is performed.

If the activity selected is CANCEL, the S-2: Cancel Login subflow is performed.

1.3 Subflows

S-1: Login

The system connects over the network to the desired CVS Server and authenticates the user using the username and password supplied in the 'LoginWindow'. If the authentication is successful the 'LoginWindow' is closed and the application is opened and the 'MainPPCVSWindow' is displayed (E-1).

S-2: Cancel Login

The system closes the 'LoginWindow' and the application is exited.

1.4 Alternative Flows

E-1: An invalid username, password, CVS Server address or CVSROOT is supplied and the connection fails or a connection could no be established between the PPCVS client and desired CVS Server. The error will be displayed and the user will have to reconnect.

2. Flow of Events for the CreateNewProject Use Case

2.1 Preconditions

The Login subflow of the ConnectToCVSServer use case must execute successfully before this use case begins.

2.2 Main Flow

This use case begins when the user opens the Create New Project Wizard. The user supplies the system with the desired new project name. The user is then able to select the desired artefacts to include in the new project by selecting them from a series of selection panes. The user can then FINISH or CANCEL the Create New Project Wizard:

If the activity selected is FINISH, the S-3: Create subflow is performed.

If the activity selected is CANCEL, the S-4: Cancel New Project subflow is performed.

2.3 Subflows

S-3: Create

The system creates the new project workspace on the CVS Server and adds all the selected template artefacts (E-2). The Create New Project Wizard is closed and the 'MainPPCVSWindow' is displayed. The list of available projects in the Remote CVS Server Display is updated.

S-4: Cancel New Project

The user selects to cancel the Create New Project Wizard. The Create New Project Wizard is closed and the 'MainPPCVSWindow' is displayed.

2.4 Alternative Flows

E-2: The import new project fails or a connection between the client PPCVS application and CVS Server could not be established. The error is displayed and the Create New Project Wizard closed and the 'MainPPCVSWindow' displayed.

3. Flow of Events for the AddArtifacts Use Case

3.1 Preconditions

The Login subflow of the ConnectToCVSServer use case must execute successfully before this use case begins.

3.2 Main Flow

This use case begins with the user selecting a file or directory to add to the repository on the CVS Server from the client display window of the 'MainPPCVSWindow'. The respective 'AddFileWindow' or 'AddDirectoryWindow' is opened. The user can then select to ADD FILE or ADD DIRECTORY to the repository or CANCEL:

If the activity selected is ADD FILE, the S-5: Add File subflow is performed.

If the activity selected is ADD DIRECTORY, the S-6: Add Directory subflow is performed.

If the activity selected is CANCEL, the S-7: Cancel Add subflow is performed.

3.3 Subflows

S-5: Add File

Through the 'AddFileWindow' and the user is able to select the location in the repository where the file is desired to be added. The system then adds the file to the desired location in the repository on the CVS Server (E-3).

S-6: Add Directory

Through the 'AddDirectoryWindow' and the user is able to select the location in the repository where the directory is desired to be added. The system then adds the directory to the desired location in the repository on the CVS Server and then individually adds any files within the selected directory.

3.4 Alternative Flows

E-3: The add file or directory fails or a connection between the client PPCVS application and CVS Server could not be established. The error is displayed and the 'AddFileWindow' or 'AddDirectoryWindow' is closed and the 'MainPPCVSWindow' displayed.

4. Flow of Events for the CheckOut Use Case

4.1 Preconditions

The Login subflow of the ConnectToCVSServer use case must execute successfully before this use case begins.

4.2 Main Flow

This use case begins with the user selecting an entire project or module within a project to checkout from the repository on the CVS Server from the remote display window of the 'MainPPCVSWindow'. The system checks out the selected item from the repository (E-4). The client display window is updated to shown the checked out item. If the checkout is an entire project the Project Status Bar is populated with status information found in the schedule from the project artefacts (E-5).

4.3 Subflows

There are no subflows for this flow of events.

4.4 Alternative Flows

E-4: The checkout fails or a connection between the client PPCVS application and CVS Server could not be established. The error is displayed to the user.

E-5: The project schedule could not be found within the project artefacts and the Project Status Bar cannot be populated and the error is displayed to the user.

5. Flow of Events for the Commit Use Case

5.1 Preconditions

The Login subflow of the ConnectToCVSServer use case and CheckOut use case must have executed successfully before this use case begins.

5.2 Main Flow

This use case begins with the user selecting a file or directory to commit to the repository on the CVS Server from the client display window of the 'MainPPCVSWindow'. The system commits the selected item back to the repository (E-6). If any of the artefacts being committed have been modified since the checkout the operating systems default text editor will be opened and the user must supply a log message to detail these modifications. After the log has been entered the client display window in the 'MainPPCVSWindow' is updated to shown the items have been committed.

5.3 Subflows

There are no subflows for this flow of events.

5.4 Alternative Flows

E-6: The commit fails or a connection between the client PPCVS application and CVS Server could not be established. The error is displayed to the user and the user will have to retry and commit the items.

6. Flow of Events for the Disconnect Use Case

6.1 Preconditions

The Login subflow of the ConnectToCVSServer use case must execute successfully before this use case begins.

6.2 Main Flow

This use case begins with the user selecting to close the 'MainPPCVSWindow'. The system closes the connection between the PPCVS client and the CVS Server and the 'MainPPCVSWindow' is closed (E-7).

6.3 Subflows

There are no subflows for this flow of events.

6.4 Alternative Flows

E-7: The connection between the client PPCVS application and CVS Server cannot be established, the user is supplied with an error message and the user will be required to open the PPCVS client again and successfully disconnect before the user is disconnected correctly.

List of References

Applied Testing and Technology. 2005, *ApTest Manager Homepage*. http://www.aptest.com/atm2/.

Beck J. 2005, *Using The CVS Version Management System in a Software Engineering Course*. Journal of Computing Sciences in Colleges 20[6], 57-65.

Boehm B. 2000, *Safe and Simple Software Cost Analysis.* IEEE Software, vol. 17, no. 5, pp. 14-17.

Bolton M. 2005, *Visual SourceSafe Version Control: Unsafe at any Speed?* http://www.michaelbolton.net/testing/VSSDefects.html.

Bullseye Testing Technology. 2005, *Bullseye Coverage Homepage*. http://www.bullseye.com/index.html.

Cambridge University Press. 2005, *Cambridge Dictionaries Online*. http://www.dictionary.cambridge.org/.

Cederqvist P. 2004, *Version Management with CVS for cvs 1.11.17*. http://www.cvshome.org.

Collins-Sussman B. 2002, *The subversion project: building a better CVS*. Linux Journal 2002[94], 3.

Collins-Sussman B., Fitzpatrick B.M., & Pilato C.M. 2005, *Version Control with Subversion*.

CVS. 2005, *CVS Homepage*. http://www.cvshome.org.

Doxygen. 2005, *Doxygen Home Page*. http://www.stack.nl/~dimitri/doxygen/.

Estublier J., Leblang D., Clemm G., Conradi R., Tichy W., van der Hoek A., & Wiborg-Weber D. 2002, *Impact of the Research Community On the Field of Software Configuration Management*. ACM Sigsoft 27[5], 31-39.

Glassy L. 2005, *Using Version Control To Observe Student Software Development Processes*. Journal of Computing Sciences in Colleges 21[3], 99-106.

Grid-Tools. 2005, *GT DataMaker Homepage*. http://www.grid-tools.com/gt-datamaker.html.

Hale J., Parrish A., Dixon B., & Smith R.K. 2000, *Enhancing the Cocomo Estimation Models*. IEEE Software.

Helm J.E. 1992, *The Viability of Using Cocomo in the Special Application Software Bidding and Estimating Process*. IEEE Transactions on Software Engineering 39[1], 42-58.

Henderson-Sellers B., Due R., Graham I., & Collins G. 2000, *Third Generation OO Processes: A Critique of RUP and OPEN from a Project Management Perspective*. Presented at the Seventh Asia Pacific Software Engineering Conference.

Hirsh M. 2002, *Making RUP Agile*. Presented at the 2002 Conference on Object Oriented Systems Languages and Applications, Seattle, Washington.

Hoffer J.A., George J.F., & Valacich J.S. 2002, *Modern Systems Analysis & Design*. Prentice Hall, Upper Saddle River.

IBM. 2005a, *Rational ClearCase*. http://www-306.ibm.com/software/awdtools/clearcase/index.html.

IBM. 2005b, *Rational Rose XDE Modeler*. http://www-306.ibm.com/software/awdtools/developer/modeler/.

IBM. 2005c, *Rational Unified Process Web Page*. http://www-306.ibm.com/software/awdtools/rup/index.html.

IEEE. 1998, *IEEE Recommended Practise for Software Requirements Specifications*. IEEE Std 830-1998 New York.

International Functional Point Users Group. 2005, *About IFPUG*. http://www.ifpug.org/about/.

ISO. 2005, *Overview of the ISO System*. http://www.iso.org/iso/en/aboutiso/introduction/index.html.

ISO. 1997, *Guidelines for Applying ISO 9001:1994 to Computer Software*. ISO Std 9000-3, Geneva.

Kang S.S. 2005, *KDOC -- C++ and IDL Class Documentation Tool*. http://www.ph.unimelb.edu.au/~ssk/kde/kdoc/.

Kroll P. & Kruchten P. 2003, *The Rational Unified Process Made Easy - A Practitioner's Guide To The RUP*. Addison-Wesley, Boston.

Kruchten P. 2003, *The Rational Unified Process An Introduction*. Addison-Wesley, Boston.

Manzoni L.V. & Price R.T. 2003, *Identifying Extensions Required by RUP (Rational Unified Process) to Comply with CMM (Capability Maturity Model) Levels 2 and 3*. IEEE Transactions on Software Engineering, vol. 29, no. 3, pp. 181-192.

Microsoft. 2005a, *Microsoft Project Website*. http://www.microsoft.com/uk/office/project/prodinfo/overview.mspx.

Microsoft. 2005b, *Microsoft Visual SourceSafe: Features Overview*. http://msdn.microsoft.com/vstudio/previous/ssafe/productinfo/features/.

Nienaber R. & Cloete E. 2003, *A Software Agent Framework for the Support of Software Project Management*. *SAICSIT* pp. 16-23.

Oskarsson O. & Glass R.L. 1996, *An ISO 9000 Approach To Building Quality Software*. Prentice Hall, Upper Saddle River.

PassMark Software. 2005, *TestLog - Test Case Management Software*. http://www.testlog.com/

PBSys. 2005, *Test Case Manager (TCM)*. http://pbsys.tripod.com/products/devtools/TCM.html

Peters J.F. & Pedrycz W. 1999, *Software Engineering: An Engineering Approach*. Wiley, New York.

Pollice G., Augustine L., Lowe C., & Madhur J. 2003, *Software Development For Small Teams - A RUP-Centric Approach*. Addison-Wesley, Boston.

Quatrani T. 1998, *Visual Modeling With Rational Rose and UML*. Addison-Wesley, Reading.

Royce W. 1999, *Software Project Management - A Unified Framework*. Addison-Wesley, Reading.

Saliu M.O., Ahmed M., & AlGhamdi J. 2004, *Towards Adaptive Soft Computing Based Software Effort Prediction*. Fuzzy Information 1, 16-21.

Schwalbe K. 2002, *Information Technology Project Management*. Course Technology, Canada.

Search Networking. 2005, *AllFusion Endevor Change Manager*. http://searchnetworking.bitpipe.com/detail/PROD/1067262190_564.html&src=searchnetworking.bitpipe.com.

Serena. 2005a , *Serena ChangMan*. http://www.serena.com/Products/changeman/Home.asp.

Serena. 2005b, Serena *TeamTrack*. http://www.serena.com/Products/teamtrack/Home.asp.

Serena. 2005c, *Serena Version Manager*. http://www.serena.com/Products/professional/vm/home.asp.

Softstar Systems. 2004, *Costar Costing Application*. [7.0].

Softstar Systems. 2005, *Overview of COCOMO*. http://www.softstarsystems.com/overview.htm.

Software Process Improvement Laboratory. 2004, *Software Engineering Standards - A Framework For Software Process Improvement*. Course Notes. CMR-038-SLD.

Sommerville I. 1995, *Software Engineering*. Addison-Wesley, Essex.

SourceForge. 2005a, *Clearcase for Java*. http://sourceforge.net/projects/clearcase-java/.

SourceForge. 2005b, *SharpCVSLib*. http://sourceforge.net/projects/sharpcvslib.

Telelogic. 2005, *Telelogic Synergy*. http://www.telelogic.com/corp/products/synergy/index.cfm.

Test and Data Services. 2005, *Software Testing Services*. http://www.testdata.co.za/services.htm

TortoiseCVS. 2005, *TortoiseCVS Homepage*. http://www.tortoisecvs.org/.

University of Southern California, C. F. S. E. 1998, *COCOMO II Model Definition Manual*. http://sunset.usc.edu/research/COCOMOII/.

University of Southern California, C. F. S. E. 2005a, *COCOMO*. http://sunset.usc.edu/research/COCOMOII/cocomo_main.html#conferences.

University of Southern California, C. F. S. E. 2005b , *COCOMO 81 Intermediate Model Implementation*. http://sunset.usc.edu/research/COCOMOII/.

Van Vliet H. 2000, *Software Engineering - Principles and Practice*. Wiley, New York.