

دیس
Com
1265
c-1

۱۹/۱۵/۲۰۰۲

Cross-platform Version Control System VCS Server



Muhammad Ali

A report submitted to
the Department of Computer Sciences, Quaid-I-Azam University
as a partial fulfillment of the requirement
for the award of the degree of M.Sc in Computer Sciences.

January 2002

QUAID-I-AZAM UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Dated: March 16, 2002

FINAL APPROVAL

This is to certify that we have read the project report submitted by Mr. Muhammad Ali and it is our judgement that this report is of sufficient standard to warrant its acceptance by the Quaid-i-Azam University, Islamabad for the degree of Master of Science in Computer Science.

COMMITTEE:

1. EXTERNAL EXAMINER

Ms. Naseem Akhtar Bhatti
Director,
Computer Training Centre,
UGC Campus,
Sector H-9,
Islamabad.

Naseem
16-3-2002

2. SUPERVISOR

Ms. Mudassira Arshad
Lecturer
Dept. of Computer Science
Quaid-I-Azam University
Islamabad.

Mudassira Arshad

3. CHAIRMAN

Dr. Masud Ahmad Malik
Professor
Deptt. Of Computer Science
Quaid-I-Azam University
Islamabad.

Dr. Masud Malik

DISS
COM
1265

*Dedicated to
those who live for others and thus truly live*

PROJECT BRIEF

Project title : **Cross-platform Version Control System**
Developed by : **Muhammad Ali**
Internal Supervisor : **Madam Mudassira Arshad**
Department of Computer Science
Quaid-I-Azam University, Islamabad
Starting date : **September 2001**
Completion date : **January 2002**
Software used : **Microsoft Visual Studio 6.0**
Operating system used : **Windows 2000 Server**
System used : **Pentium III, 733 MHZ**



ACKNOWLEDGEMENTS

I feel overwhelming gratitude towards the Divine for providing sustenance and assistance to my soul throughout its journey and I am especially grateful for having such a loving family.

It amazes me whenever I think about my father, who started out from very humble beginnings, not just succeeded but provided me with a solid educational foundation. His appreciation and confidence in me has been an abundant source of motivation and I am eternally indebted to him for his instilling a love of books in me. He has been truly a friend and a companion to me. It is impossible to put into words the feelings of love and respect that I have for him.

My mother, who is a symbol of patience and unconditional love, has literally drowned me in her love and care. I have intimately known just one such self-less creature and that was my grandmother.

My sisters, each one of them so unique and special, have provided me with a lot of love and respect. Little Ahmed you are an unlimited source of joy.

This acknowledgement would be incomplete without a mention of my uncles who have had an enormous impact on my life. From Uncle Aziz I have learnt kindness; from Uncle Dayan I have learnt what it is to be a man of principle and discipline. And Uncle Muhib has been my mentor and counselor.

I pay tribute to the Chairman of our department Mr. Masud Malik for his support. I am also thankful to Madam Mudassira Arshad, my project supervisor for her guidance. Hats off to Mr. Nauman Yousuff for his ideas and interest in this project.

Special mention must go to my friends Shah Saud, Muaziz Ali Khattak, Balach Wahid, Abid Khan, Lt. Muhammad Omar, Irfanullah, Bakhtiar Mamoon, Irfan Khalil, Amir Nawas, Shahid-ullah and Abdul Malik. Their sheer presence was an inspiration.

I have also had the pleasure of knowing some terrific seniors who haven't let any opportunity of helping out their juniors astray. People like Nadeem Khan, Syed Jawad-ul-Hassan Bukhari, Majid Ali Chaudry, Nuaman Jamil, Kashif Rasool and Shahid Khan are rare.

I thank you all

Muhammad Ali

ABSTRACT

CVS Server is the server part of a client-server software that provides access to a SourceSafe database over the Internet. It enables users of Windows and non-Windows based operating systems to access and use the SourceSafe database for source and version control purposes.

The CVS Server communicates with the client through HTTP (Hyper Text Transfer Protocol). The medium of communication for the response by the CVS Server as well as for transfer of files is XML (Extensible Markup Language) thus realizing cross-platform access.

The CVS server uses the SourceSafe API to drive the SourceSafe database. It was incorporated in the COM components that were created. These components dealt with different aspects such as those of files, users, projects, conversion from XML to binary, conversion of binary files to XML, sending other XML responses to the client and so on.

The COM components are called from ASP pages which provide communication with the Client. The files and information is sent using HTTP POST request.

PREFACE

This report is concerned with the development process of *cross-platform version control system*; a software that provides the facility of source and version control over the Internet and for different platforms.

The document has been divided into five parts. Each part is comprised of different chapters.

Part I contains the introduction to the version control systems and our own system.

Part II is analysis and contains the requirements and use case diagrams.

Part III is the design and it contains the class diagrams and collaboration diagrams.

Part IV is implementation and it contains the Class Definitions and implementation specific details.

Part V contains the appendices that include XML DTDs, technologies used, process model, required resources, feasibility, glossary and bibliography.

Table of Contents

PART I -- Introduction

CHAPTER 1 -- Version Control Systems	2
1.1 Version control systems:	1
1.2 Microsoft Visual SourceSafe Overview	2
CHAPTER 2 System Overview	7
2.1 Limitations of SourceSafe:	8
2.2 Overcoming the Limitations:	8
2.3 Objectives:	9
2.4 Scope:	10

PART II Analysis

CHAPTER 3 Requirements	12
3.1 Requirement Specifications:	13
3.2 System Functions for the Server:	14
3.3 System Attributes:	18
CHAPTER 4 Use Cases	19
4.1 Use Case Diagram:	20
4.2 Use Cases:	21
4.3 Use Cases in Expanded Format:	22

PART III Design

CHAPTER 5 Class Design	29
5.1 Class Descriptions:	30
CHAPTER 6 Collaboration Diagrams	34
6.1 Collaboration Diagrams:	35

PART IV Implementation

CHAPTER 7 Mapping Design to Code	42
7.1 Creating Class Definitions:	43
7.2 Class Definitions:	43
CHAPTER 8 Implementation Specific Detail	52
8.1 How the system works:	53
8.2 Converting files to and retrieving them from XML:	54

PART IV Testing

CHAPTER 9 Test Cases.....

- 9.1 Server Testing for Client Requests..... 57
- 9.2 Server Testing for Administration Client..... 60

CHAPTER 10 Evaluation and Future Enhancement 62

- 10.1 Evaluation..... 63
- 10.2 Future Enhancement..... 63

PART V Appendices

- Appendix A – Process Model 65
- Appendix B - Required Resources 66
- Appendix C - Feasibility..... 67
- Appendix D - Tools and Technologies..... 69
- Appendix E Document Type Definitions 73
- Appendix F-- Glossary 76
- Appendix F-- Bibliography..... 80

PART I

Introduction

CHAPTER 1

Version Control Systems

1.1 Version control systems:

A version control system keeps a history of the changes made to a set of files. For a developer, that means being able to keep track of all the changes you've made to a program during the entire time you've been developing it. Have you ever lost a day's work to an errant keystroke at the command line? A version control system gives you a safety net.

Version control systems are valuable for anyone as they act as a safety net. But they're usually used by software development teams. Developers working on a team need to be able to coordinate their individual changes; a central version control system allows that.

1.1.1 Code central station

Individual developers who want the safety net of a version control system can run one on their local machines. Development teams, however, need a central server all members can access to serve as the repository for their code. In an office, that's not a problem-- just place the repository on a server on the local network.

1.1.2 Coordinating code

In version control systems, a developer checks out a file, modifies it, then checks it back in. The developer who checks out a file has exclusive rights to modify it. No other developer can check out the file—and only the developer who checked out the file can check in modifications. (There are, of course, ways for administrators to override that.)

1.2 Microsoft Visual SourceSafe Overview

Microsoft VSS helps one manage projects, regardless of the file type (text files, graphics files, binary files, sound files, or video files) by saving them to a database. When there is need to share files between two or more projects, these can be shared quickly and efficiently. When a file is added to VSS, the file is backed up on the database, made available to other people, and changes that have been made to the file are saved so that an old version can be recovered at any time. Members of a team can see the latest version of any file, make changes, and save a new version in the database.

1.2.1 What Makes Visual SourceSafe Unique?

Most source-control systems work well for individual source code files. However, almost all of them fail to establish relationships between files. This can pose a significant problem in an environment, where a single application can consist of multiple executable files and dynamic-link libraries built out of many source files, which in turn may be reused among many other applications. Today it is as important to manage the relationships between source files as it is to protect the contents of the source files themselves.

Microsoft Visual SourceSafe version-control software solves this problem by combining the tasks of project management and source control. By focusing on projects as well as source files, Visual SourceSafe provides an elegant solution to problems not easily solved using standard, file-oriented source-control systems.

1.2.2 Organizing Software Development by Project

To understand the benefits of project-oriented source control, simply compare it to a file-oriented system. A standard version-control system (such as the UNIX utility RCS) is essentially a collection of tools that operate on individual files, controlling file access and updates and comparing previous versions. To operate on a group of files, you write a batch file or specify wildcards on the command line.

Visual SourceSafe stores files in a central database on the network, rather than in an ordinary DOS directory. At the system level, this database appears as a black box. When viewed through Visual SourceSafe, however, the database is seen to contain all the source files and histories organized into project hierarchies.

When a user retrieves a file, Visual SourceSafe marks the file as checked out to by him in its database, then allows the user to make changes to the file on your machine. When the file is checked back in by the user, Visual SourceSafe updates its database.

How is this any different from file-oriented source control? With each change, the Visual SourceSafe database records and tracks project information not available to file-oriented systems. Each time a file is added, modified, shared, moved or deleted from a project, Visual SourceSafe updates both the file and the project's history. One can use project history to simplify these tasks:

- View the status of all the files in a specific project and all subordinate projects before building
- Zoom in on a specific file change that might have caused a bug in a build on a certain date
- Re-create any previous version of an entire application
- Maintain source files that are shared among many applications
- Determine which projects are affected by changing a file that is shared among many applications
- Manage client-specific versions of a general application

Attempting these tasks with file-oriented systems can be an incredible chore a frustrating impediment to software developers. The project-oriented version control in Visual SourceSafe streamlines the development process by making all of these tasks straightforward, as illustrated by the following scenarios.

1.2.3 Preparing for a Build

Suppose one is about to build a major application consisting of many components. Before starting the build, it must be made sure that no one is revising code at the last minute; or, in version-control terms, that no files in the entire system are checked out.

A standard version-control system provides a tool for determining whether a file is checked out. This utility has to be used on every file in every directory that is being used for the build. Batch files and wildcards make the task easier, but in a complicated system, it can still be a big chore.

Visual SourceSafe, like other systems, can determine if a file is checked out. But it can also create a higher-level report: a list of all the checked-out files in a project. This feature becomes even more powerful when used recursively to include all subprojects in the current project. Visual SourceSafe checks every file in every relevant project and generates a list of every checked-out file. It is immediately known whether the build can be proceeded on not.

1.2.4 Pinpointing Regressions

File history reports are available in all version-control systems, including Visual SourceSafe. A file history lists each version of a file, from the most recent to the oldest, with information such as what happened to the file, who modified it, when the changes were made, and what comments were made.

Despite their usefulness, file histories have severe limitations. For instance, suppose a feature that was working correctly last week breaks in this week's build of an application. Obviously, someone introduced this bug recently, but in what file?

To tackle this question with a standard version-control system, a history report would have to be generated on a likely looking file to see if it had changed recently, and then one has to look through the changes. If the bug wasn't found, the process would be repeated with another file and so on. One might go through every file in the system this way and never find the critical change because the change actually added or deleted a file, which standard version-control systems don't track at all.

With Visual SourceSafe, a report is generated on the project itself. For instance, Visual SourceSafe might report that COMMON.BAS was just modified; before that, OPENALL.FRM was changed; before that, FILESUPP.BAS was added to the project; and so on. Visual SourceSafe collates the changes that you would otherwise have to sort through manually, enabling the viewing of the order of changes over the past week.

1.2.5 Re-Creating Previous Project Versions

By tracking project history, Visual SourceSafe allows to quickly re-create previous versions of an entire application. This ability can help in resolving bugs reported from previous shipped versions, to make sure they were fixed in the version under development.

For instance, suppose a client reports a printing problem in version 2.03 of an application. That version of the application may have included version 10 of one file, version 15 of another, and so on - but that's not a concern. By requesting the specific version of the project

from Visual SourceSafe, a complete, local copy of the application's source files used to build version 2.03 can be restored.

To restore the proper source files with a standard version-control system, one would have to either archive the sources of each released version of the application separately, or track the specific file versions for each release.

Either way, restoring the correct source files for a previous build becomes a tedious, manual process.

1.2.6 Maintaining Reusable Code

Most applications are developed around a common base of core code. These files typically are used over and over again in many applications and usually evolve over time, receiving bug fixes, performance improvements and new features. The benefits of reusing existing code are enormous, but so are the headaches when it comes to managing the organizational issues. One has to remember which applications are using each file and propagate every change to all the appropriate places. This task is only a minor annoyance when five applications reuse one file, but it gets quite complicated when 20 applications mix and match 50 reusable files. A standard version-control system can't help at all with this problem. Visual SourceSafe can automate it completely, however, because one file can exist simultaneously in many projects. Within its database, Visual SourceSafe stores each file only once. Each project that contains a file has a pointer to the location of the file in the database. All versions of the file are available to each project, and a project can freeze the version of a file to avoid introducing errors while another development team works on the reused code. To cite a common example, suppose there is a single source file that contains many procedures for printing reports. In Visual SourceSafe, each application that needs to print reports would share the file. If a bug is found, the file is updated from any project and the change is instantly propagated to every project that shares the file. Then Visual SourceSafe can report on which projects share the file, so it is known that which applications are affected and may need to be rebuilt.

1.2.7 Creating Client-Specific Versions

Another common source-control problem concerns clients who want applications customized to meet their specific needs. Essentially, there maybe many applications that share almost all of the same source files. Using standard source-control tools, tracking changes and keeping

builds straight can take more time than the programming. With Visual SourceSafe, a project can be created for each new client, indicating which files are shared and which files are unique. When working on a project, client-specific changes stay with the current project, and changes to shared files are propagated to all client versions.

CHAPTER 2

System Overview

2.1 Limitations of SourceSafe:

The major problems with Visual SourceSafe (VSS) are:

- It cannot be used over the Internet. It can only be used over a local network.
- It is platform specific. It can only be used with Windows 95, Windows 98, Windows NT, Windows 2000 operating systems.

2.2 Overcoming the Limitations:

This system overcomes these limitations by enabling distributed, cross-platform development teams to have fast and secure access to centralized SourceSafe databases via an Internet connection. The product is a client/server application with a GUI client similar to SourceSafe Explorer.

The System consists of two parts: a Server and a Client. The Server is installed on the machine with access to the file system on which the SourceSafe (VSS) database resides. Once installed there, the Server provides access to SourceSafe operations to Clients that connect using an Internet connection. The Server can be installed on a Windows NT/2000 platform. It runs in the background with no user interface of any kind.

The Client is a GUI application that closely resembles the SourceSafe Explorer user interface. SourceSafe users should find the Client to be familiar and easy to use. Users can perform most SourceSafe operations in a similar fashion.

The Client can connect to any Server by simply specifying the location of that server. The Client and Server can communicate over any standard TCP/IP connection, including a PPP connection over a modem, an ISDN line, or an Ethernet connection on the same LAN as the Server.

2.3 Objectives:

This system will solve the limitations in VSS by:

- Providing SourceSafe operations to clients from any remote location over Internet.
- Providing client that is platform-independent. Hence the SourceSafe operations could be accessed from operating systems other than the Windows family of operating systems.

2.4 Scope:

The cross-platform Version Control System (VCS) will be consisting of two parts: a client part and a server part.

2.4.1 VCS Server

This server communicates with Visual SourceSafe (VSS) to fulfill requests those are sent by VCS client. VCS server keeps track of users, user authentication; perform all the operation relating to VSS and other similar things.

Client communicates with server that performs tasks (automation) ranging from establishing connections between the client and SourceSafe Server to actual transfer of files and all the other tasks.

PART II

Analysis

CHAPTER 3

Requirements

3.1 Requirement Specifications:

Correct and thorough requirements specifications is essential to a successful project. Requirements are a description of needs or desires for a product. The primary goal of the requirements phase is to identify and document what is really needed, in a form that clearly communicates to the client and to development team members. The challenge is to define the requirements unambiguously, so that the risks are identified and there are no surprises when the product is finally delivered.

The requirements contain the following:

- system functions
- system attributes

The overview statement and goals are self-explanatory. System functions are what a system is supposed to do; they are also called the functional requirements. Functions should be categorized in order to prioritize them and identify those that might otherwise be taken for granted (but which consume time and resources). The categories that have been included are evident and hidden.

¹*Evident category:* Those functions are included in this category that a user is aware of.

Hidden category: This category includes those functions that are not visible to the user like underlying technical services.

¹ See [Craig98] page 42,43

3.2 System Functions for the Server:

3.2.1 SourceSafe Database Functions:

Ref #	Function	Category
R1.1	Users must login with username and password.	Evident
R1.2	Server must open the SourceSafe database using the password and username sent to it by the client, retrieve the information in the database.	Hidden

3.2.2 Communication Functions:

Ref #	Function	Category
R2.1	Communication mechanisms between the client and the server must be provided.	Hidden
R2.2	Retrieve encoded file from the client and reconvert it to its original form.	Hidden
R2.3	Retrieve the filename and its path in the SourceSafe database from the client.	Hidden
R2.4	Retrieve new filename from the client.	Hidden
R2.5	Retrieve label for a file from the client.	Hidden
R2.6	Retrieve the request for the operation to be performed from the client.	Hidden
R2.7	Retrieve the parent project path and project name from the client.	Hidden
R2.8	Send the current information about the SourceSafe database to the client upon client's request in an appropriate format.	Evident
R2.9	Convert the file that was checked out or gotten from the SourceSafe database to an appropriate format and send it to the	Evident

	client that requested the operation.	
R2.10	Retrieve new username for the admin from the client.	Hidden
R2.11	Retrieve username for the admin from the client.	Hidden
R2.12	Retrieve projects rights for a user from the client for admin.	Hidden
R2.13	Retrieve a user's password from the client for admin.	Hidden
R2.14	In case of errors, send error information to the client.	Evident
R2.15	Send the details about projects and files in the SourceSafe database to the client.	Evident
R2.16	Retrieve username and password from the client.	Hidden

3.2.3 Source Control Functions:

Ref #	Function	Category
R3.1	Based upon the filename, the request, the file and the parent project path in the SourceSafe database retrieved from the client, add the file to SourceSafe database in the appropriate project.	Evident
R3.2	Based upon the filename, the request, the file and the parent project path in the SourceSafe database retrieved from the client; check in the file to SourceSafe database in the appropriate project.	Evident
R3.3	Based upon the filename or project name, the request and its path in the SourceSafe database retrieved from the client, delete the file or project.	Evident
R3.4	Based upon the filename or project name, the request and its path in the SourceSafe database retrieved from the client, label the file or project.	Evident
R3.5	Based upon the filename, the request, its path in the SourceSafe	Evident

	database retrieved from the client undo-checkout the file.	
R3.6	Based upon the filename, the request, its path in the SourceSafe database retrieved from the client checkout the file.	Evident
R3.7	Based upon the filename, the request, its path in the SourceSafe database retrieved from the client get the file.	Evident
R3.8	Based upon the project name, the path of its parent project in the SourceSafe database and the request from the client, create a new project in the specified project.	Evident
R3.9	Based upon the file name, its version, its path in the SourceSafe database and the request from the client, get an older version of that file.	Evident
R3.10	Based upon the file name, its version, its path in the SourceSafe database and the request from the client, roll-back to an older version of that file.	Evident
R3.11	Based upon the file name, its version, its path in the SourceSafe database and the request from the client, show older versions of that file.	Evident

3.2.4 Administrative Functions:

Ref #	Function	Category
R4.1	Based upon the username and request retrieved from client, add a new user to the SourceSafe database.	Evident
R4.2	Based upon the username and request retrieved from client, delete a user from the SourceSafe database.	Evident
R4.3	Based upon the new username of a user and request retrieved from the client, change the user's name in the SourceSafe database.	Evident

R4.4	Based upon the username of a user and request retrieved from the client, change the user's rights in the SourceSafe's database.	Evident
R4.5	Based upon the username of a user, the new password and request retrieved from the client change the user's password.	Evident
R4.6	Change the access rights of a user based upon the username, project and request from the client.	Evident
R4.7	Copy user access rights to the selected user based upon the user names and request from the client.	Evident

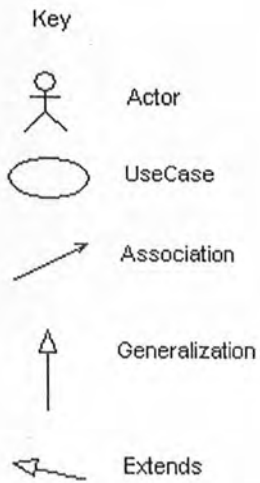
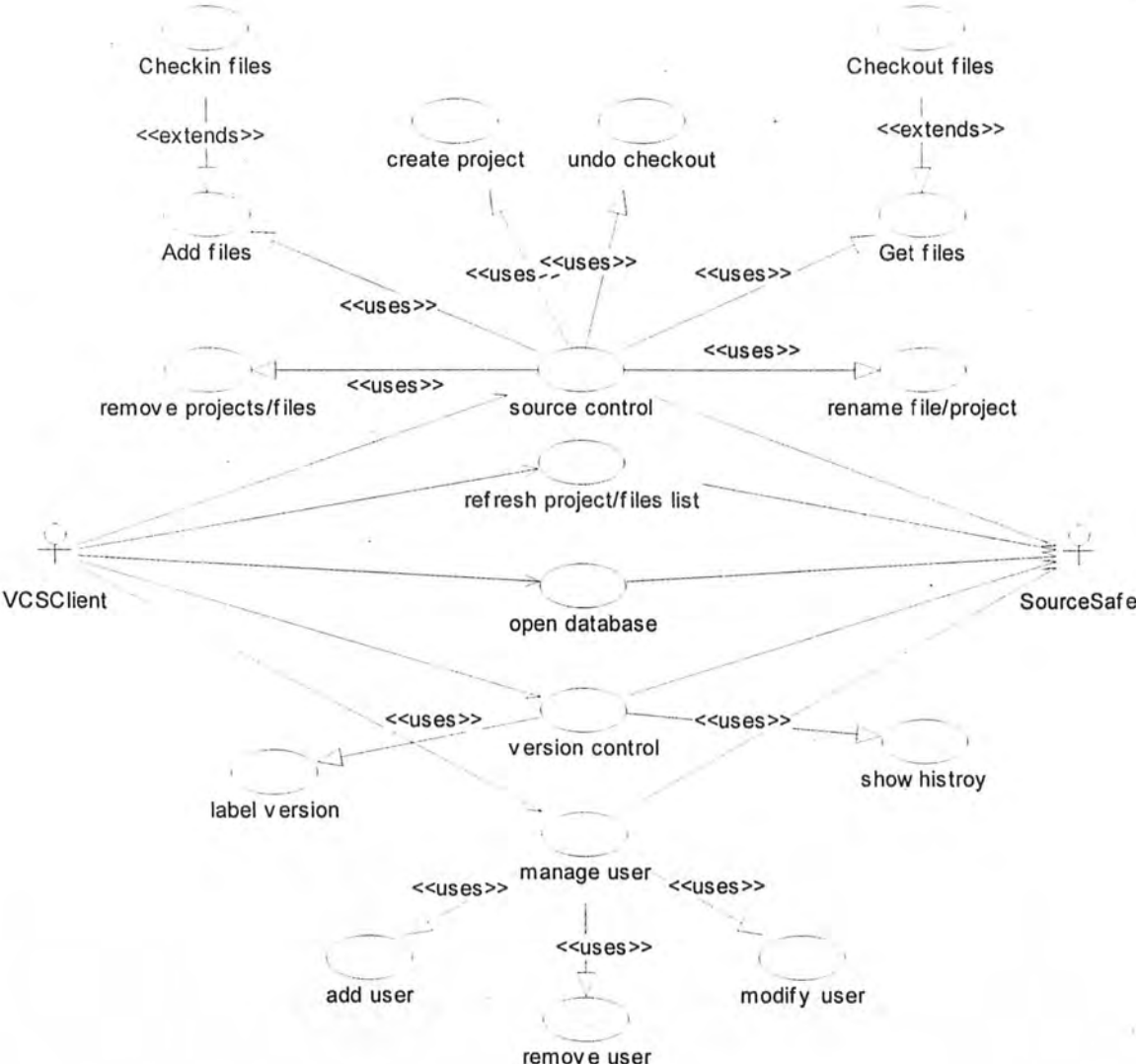
3.3 System Attributes:

- Access over the Internet: Product has to be developed such that it could be accessed over the Internet.
- Final Date: Product has to be developed by the end of February
- Extensibility: Product should be developed in such a way that it could be easily made web-based in future

CHAPTER 4

Use Cases

4.1 Use Case Diagram:



4.2 Use Cases:

An excellent way to understanding the requirements of a system is the creation of use cases. Use cases are dependent on having at least partial understanding of the requirements of the system, ideally expressed in a requirements specifications document.

A use case is a narrative document that describes the sequence of events of an actor (an external agent) using a system to complete a process [Jacobson92]. These are the cases of using a system and not requirements or functional specifications, but they illustrate and imply requirements in the descriptions of the processes

4.2.1 Use Case Formats:

There are two types of use case formats. A high level format and an expanded use case format. Expanded use case formats are useful in order to obtain a deeper understanding of the processes and requirements. Expanded use cases are often done in a “conversational” style between the actors and the system [Wirfs-Brock93].

The *typical course of events* is the heart of the expanded format and this is where the expanded format differs from the high level format. The *typical course of events* describes in detail the conversation of interaction between the actors and the system. The *alternative course of events*, describes important alternatives or exceptions that may arise with respect to the typical course [Craig98].

Following are the important use cases in the expanded format:

4.3 Use Cases in Expanded Format:

Use case: **Open Database**

Actors: user client/admin client, SourceSafe

Purpose: Read SS repository and send this information to user client/admin client

Overview: Client sends user name, password and requests opening the database from the system. System accesses the SourceSafe database and opens it if authorized. It then reads the database and sends information in the database to client in an appropriate format.

Cross Ref: *Functions:* R1.1, R1.2, R2.1, R2.6, R2.8, R2.16, R2.14

Typical Course of Events:

Actor Action	System Response
1. Client opens connection with the system.	
2. Client sends username and password to the server.	3. System invokes the SourceSafe database with the received user name and password.
4. SourceSafe returns the result of the operation.	5. System then reads all the information of files and projects and returns it to the client in an appropriate format.
6. Client retrieves all the information.	
7. Client Closes the connection.	

Alternative Courses:

- Line 4: Invalid username or password entered. Indicate error and stop.

Use case: **Create New Project**

Actors: user client/admin client, SourceSafe

Purpose: Add a new project to the SourceSafe database

Overview: Client sends the parent project path, the new project name and the request to the system which then accesses the SourceSafe database and makes the new entry in it.

Cross Ref: *Functions:* R2.1, R2.6, R2.7, R2.14, R3.8

Typical Course of Events:

Actor Action	System Response
1. Client opens connection with the system.	
2. Client sends parent project path, the new project name and request to the system.	3. System accesses the SourceSafe database and adds the new entry into it.
4. SourceSafe returns the result of the operation.	5. System sends the indication to the client of success or failure.
6. Client retrieves all the information.	
7. Client Closes the connection.	

Use case: **Check In File**

Actors: user client/admin client, SourceSafe

Purpose: Check In a file to the SourceSafe database after having made changes to it.

Overview: Client sends the file name, its path, the file in an encoded form to the system. The system decodes the file and saves it. It then checks in the file to SourceSafe database.

Cross Ref: *Functions:* R2.1, R2.2, R2.3, R2.14, R3.2

Typical Course of Events:

Actor Action	System Response
1. Client opens connection with the system.	
2. Client sends the file, its name and path. The file is in encoded form.	3. System retrieves this information, decodes the file to its original form and saves it. System then accesses the source safe and Checks In the file.
4. SourceSafe returns the result of the operation.	5. System sends the indication to the client of success or failure.
6. Client retrieves all the information.	
7. Client Closes the connection.	

Use case: **Check Out File**

Actors: user client/admin client, SourceSafe

Purpose: Check Out a file from the SourceSafe database for making changes to it.

Overview: Client sends the file name, its path and the request to the system. The system checks out the file. Encodes the file in an appropriate format and then sends it to the client.

Cross Ref: *Functions*: R2.3, R2.6, R2.9, R2.14, R3.6

Typical Course of Events:

Actor Action	System Response
1. Client opens connection with the system.	
2. Client sends the request, file name and path to the system.	3. System retrieves this information and Checks Out the file from the SourceSafe database.
4. SourceSafe returns the result of the operation.	5. System sends the indication to the client of success or failure.
7. Client retrieves all the information.	6. System encodes the checked out file to a form appropriate for transmission and sends the file.
8. Client Closes the connection.	

Alternative Course of action:

Line 6: In case of error do not send the file.

Use case: **Manage User**

Actors: admin client, SourceSafe

Purpose: Adds a user, deletes, modifies username or password and sets rights.

Overview: Admin client sends the relevant information about the user and the request to the system. The system based upon the request adds user, deletes user or modifies the user.

Cross Ref: *Functions*: R2.1, R2.10, R2.11, R2.13, R2.14, R4.1, R4.2, R4.3, R4.4, R4.5

Typical Course of Events:

Actor Action	System Response
1. Admin opens connection with the system.	
2. Admin sends the information about the user such as user's name, database and new password.	3. System retrieves this information and adds deletes or modifies the user in the SourceSafe
4. SourceSafe returns the result of the operation.	5. System sends the indication to the client of success or failure.
6. Admin retrieves all the information.	
7. Admin Closes the connection.	

Use case: **Manage Access Rights**

Actors: admin client, SourceSafe

Purpose: Change access rights of a user by project or copy the access rights of one user to another user's.

Overview: Admin client sends the relevant information about the user and the request to the system. The system based upon the request copies the rights of one user to another's.

Cross Ref: *Functions*: R2.1, R2.11, R2.12, R2.14, R4.6, R4.7

Typical Course of Events:

Actor Action	System Response
1. Client opens connection with the system.	
2. Client sends the information about the user such as usernames, and their rights.	3. System retrieves this information and copies the rights of one user to another's in the SourceSafe database.
4. SourceSafe returns the result of the operation.	5. System sends the indication to the client of success or failure.
6. Client retrieves all the information.	
7. Client Closes the connection.	

PART III

Design

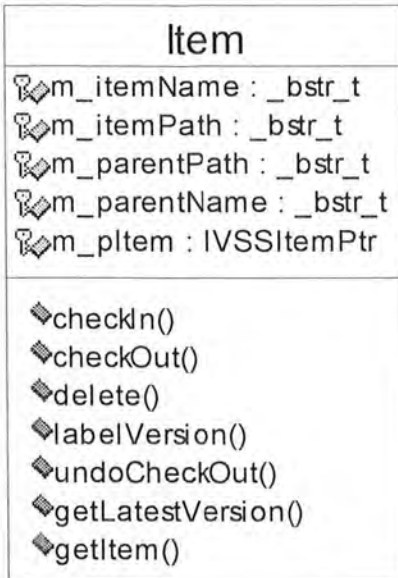
CHAPTER 5

Class Design

5.1 Class Descriptions:

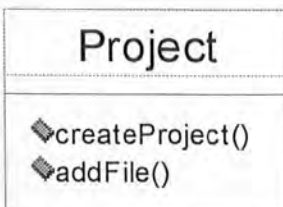
5.1.1 Item:

Purpose: To provide source control facilities for a project or file.



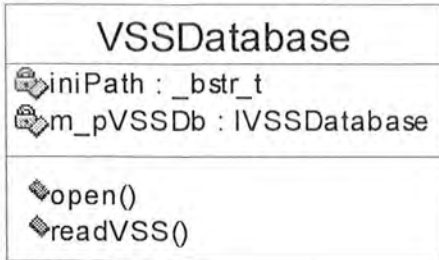
5.1.2 Project:

Purpose: To provide source control facilities that are related to project alone.



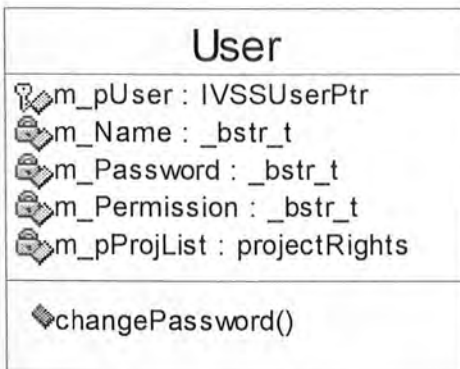
5.1.3 VSSDatabase:

Purpose: To provide access to and retrieval of information from the SourceSafe database.



5.1.4 User:

Purpose: To provide for user related functions.



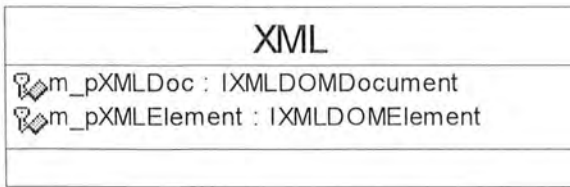
5.1.5 Administrator:

Purpose: To provide for user related functions.



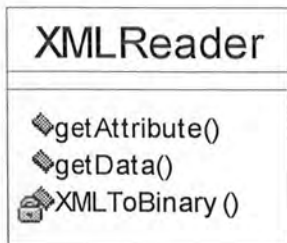
5.1.6 XML:

Purpose: To provide a base class for XMLReader and XMLWriter classes.



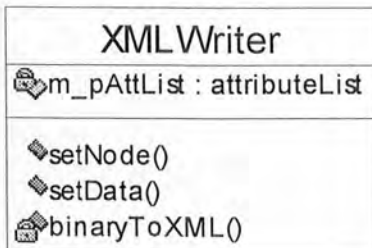
5.1.7 XMLReader:

Purpose: To provide facility for parsing XML and converting XML data to binary form.

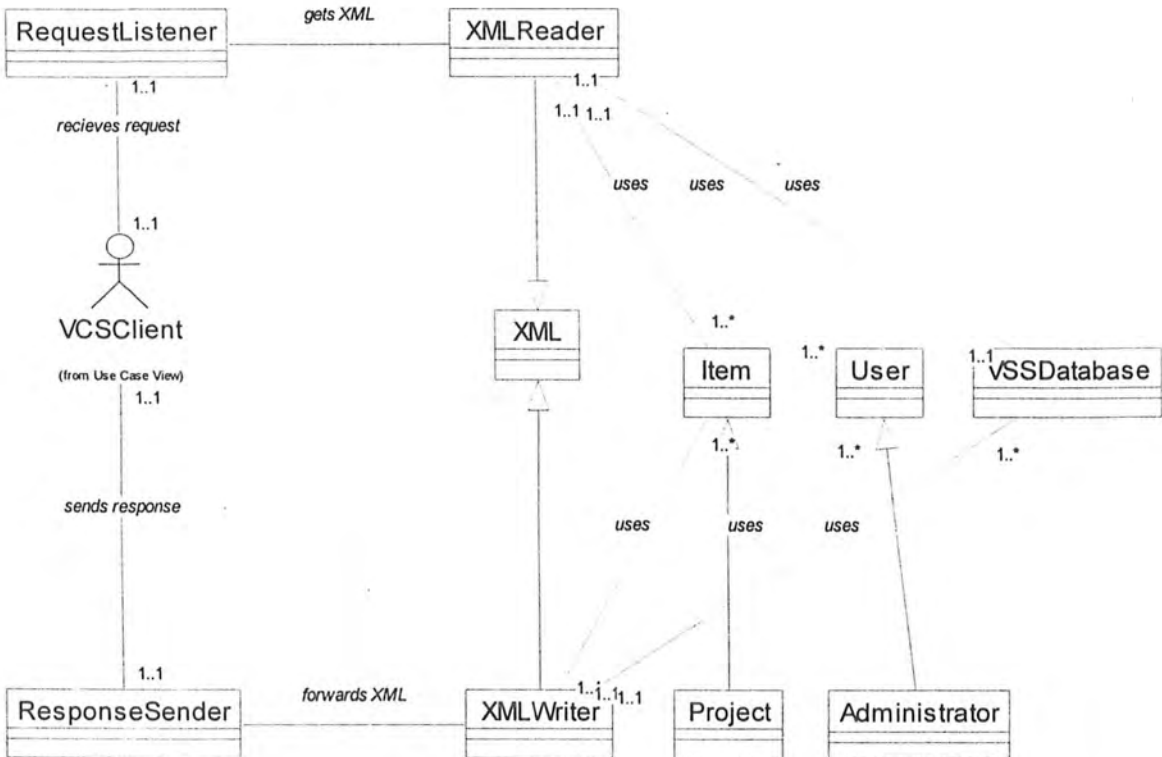







5.1.8 XMLWriter:

Purpose: To provide facility for generating XML and converting binary data to XML.



5.2 Class Relationship Diagram:



- Key
-  Class
 -  Actor
 -  Association
 -  Generalization
 -  Aggregation

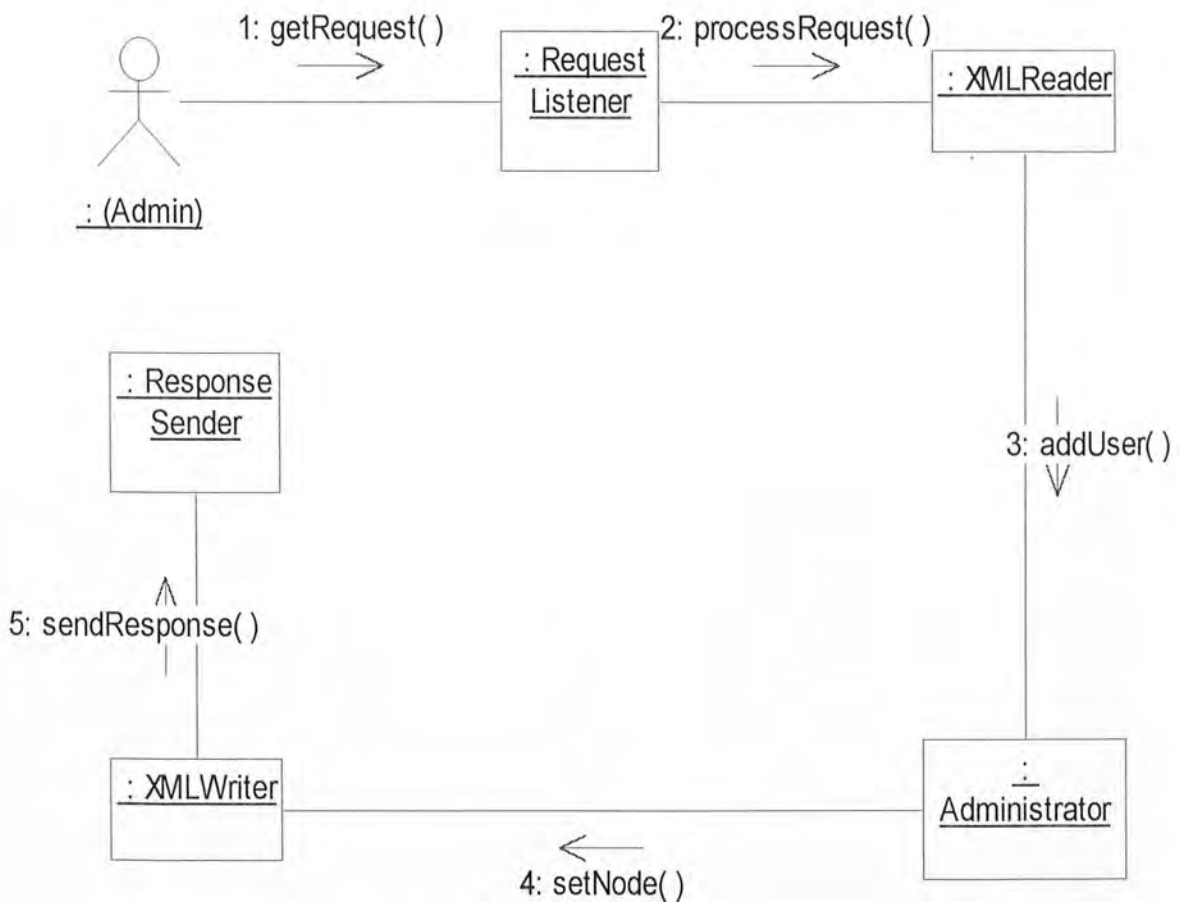
CHAPTER 6

Collaboration Diagrams

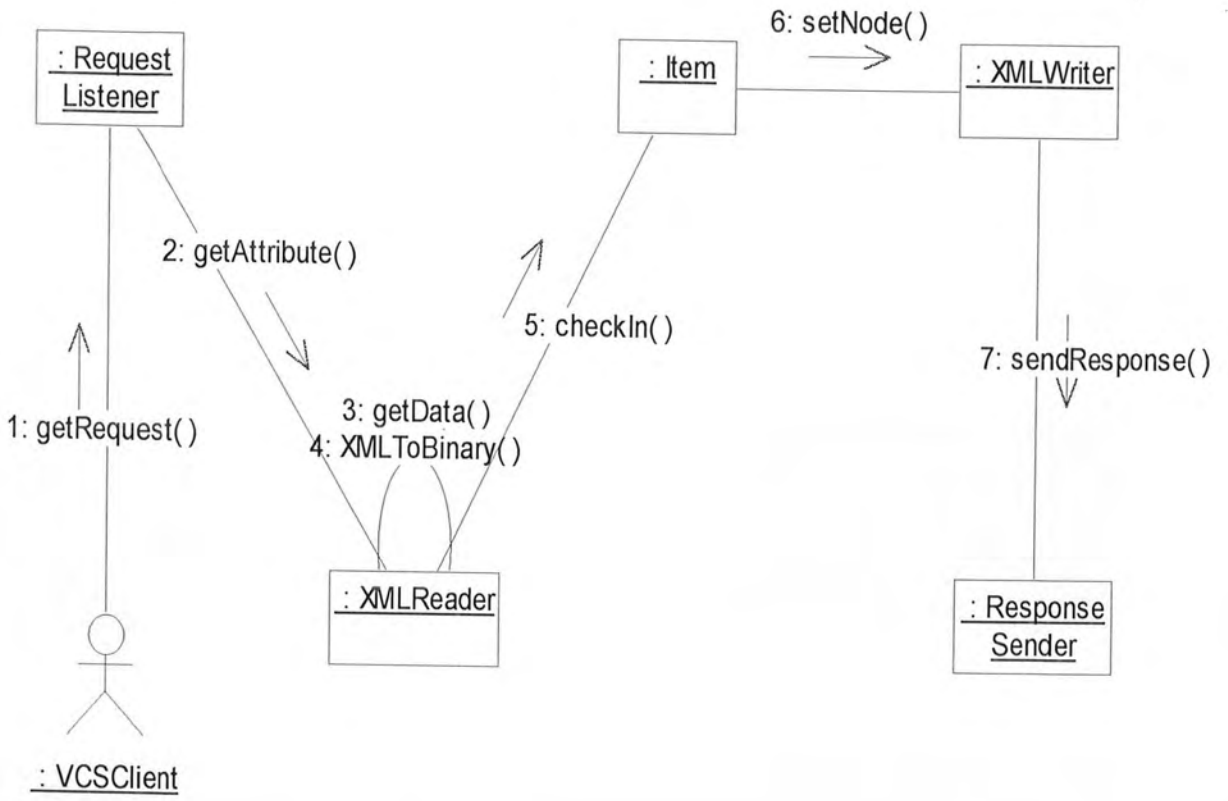
6.1 Collaboration Diagrams:

Collaboration diagrams illustrate object interactions in a graph or network format. They illustrate the message interactions between classes in a class model. Following are the more important collaboration diagrams:

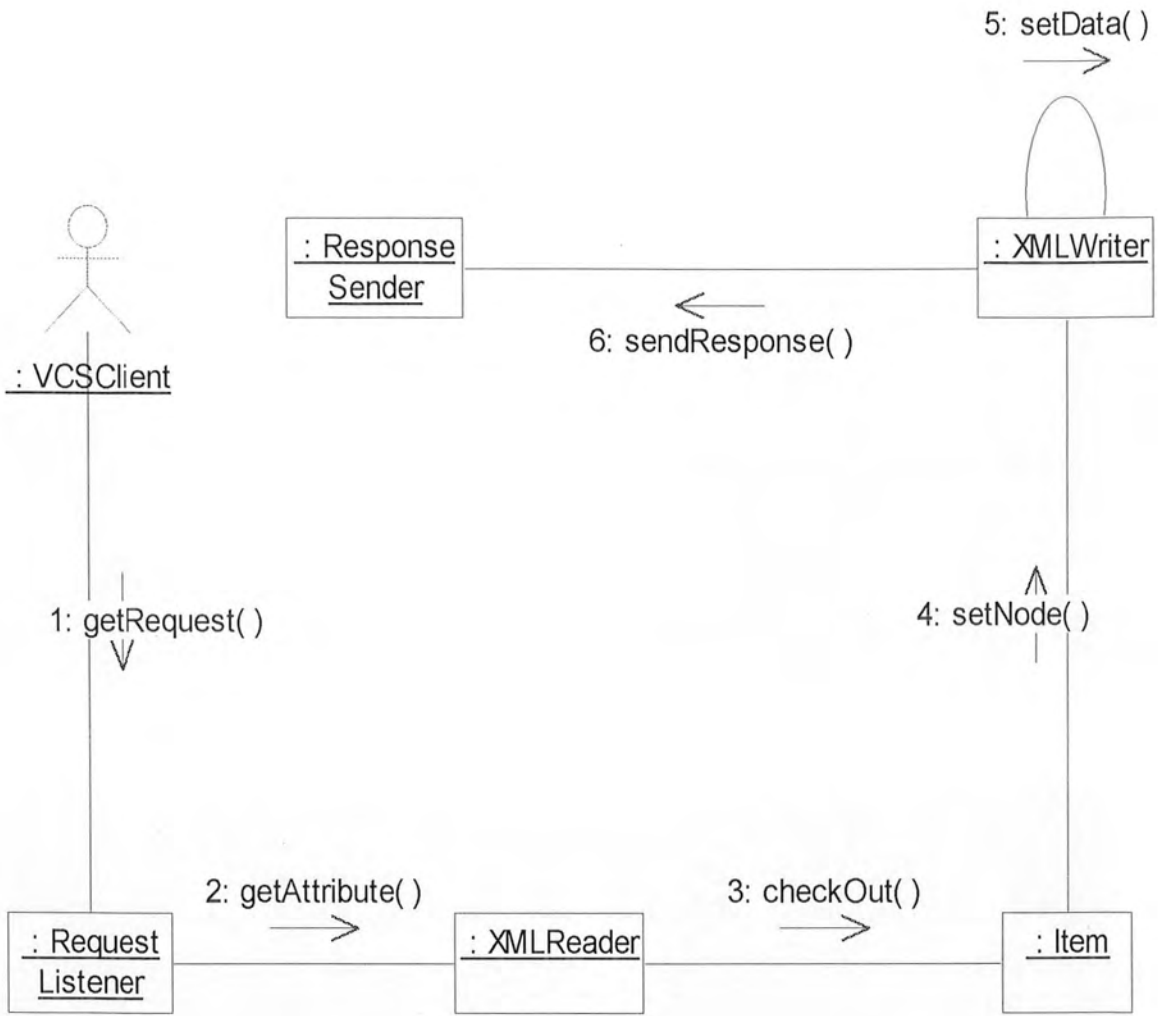
6.1.1 Add a user



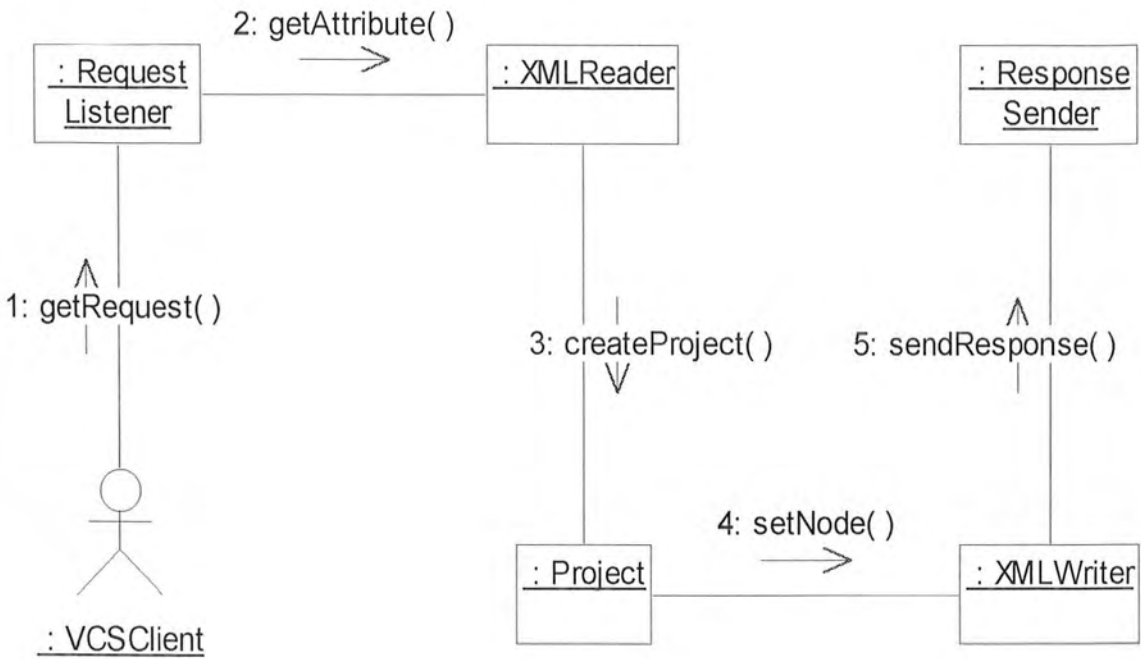
6.1.2 Check in a file



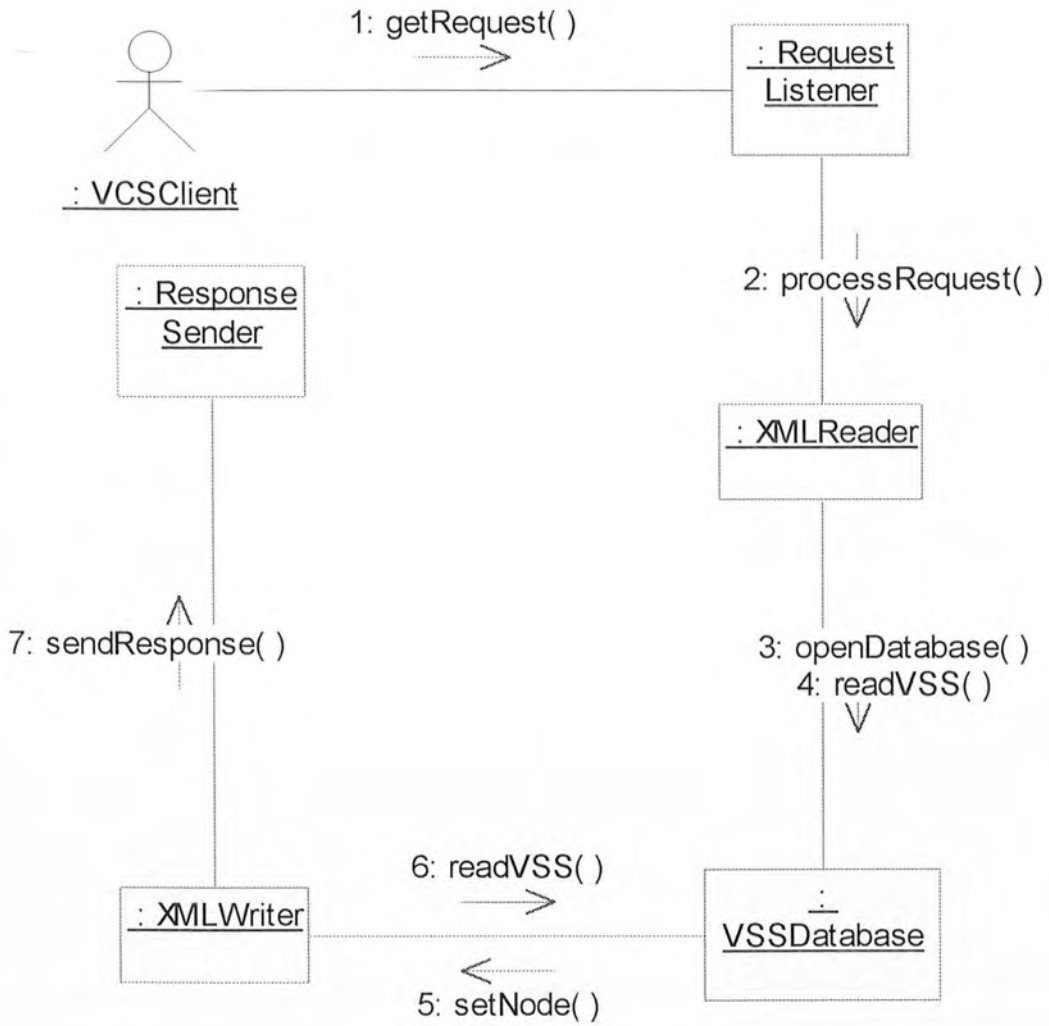
6.1.3 Check out a file



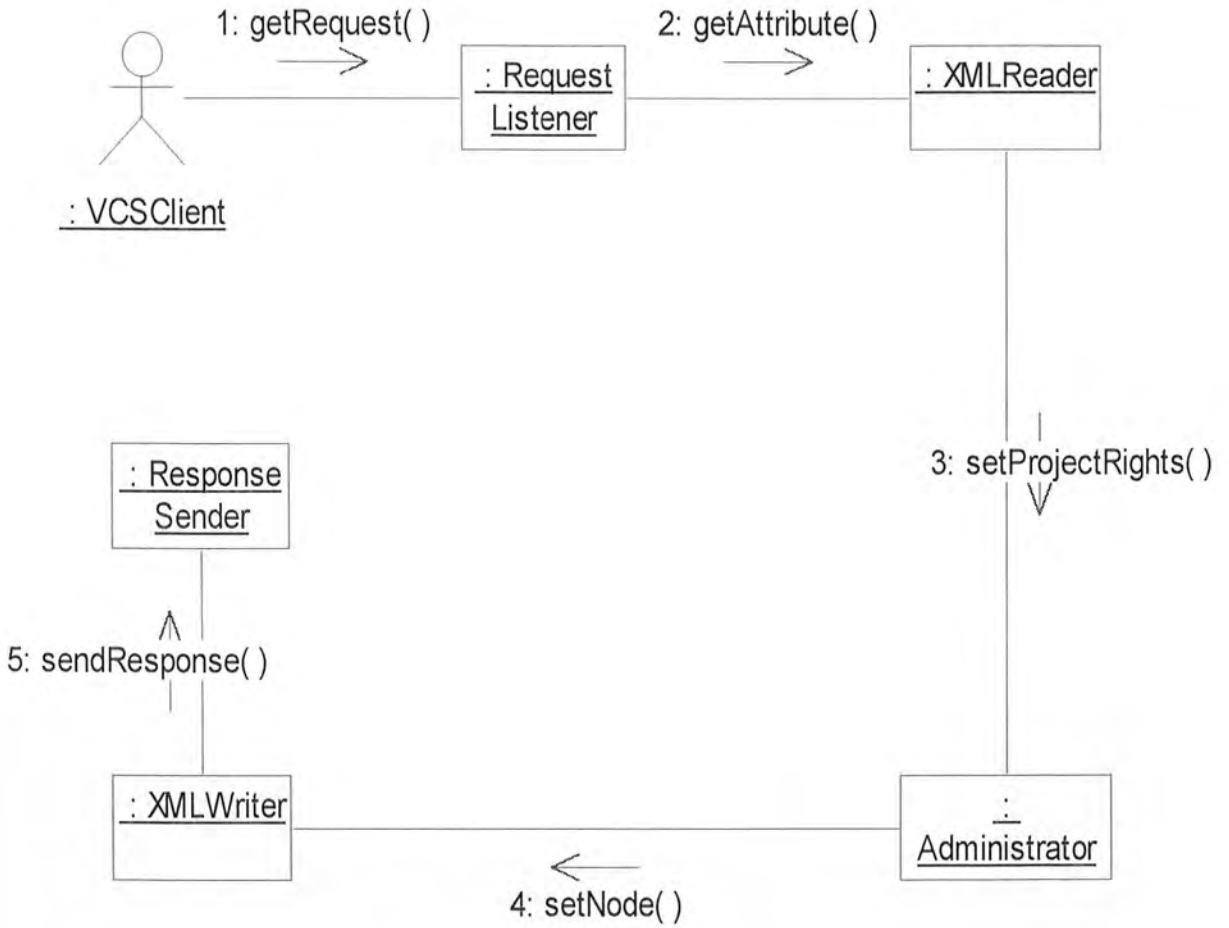
6.1.4 Create a new project



6.1.5 Open Database



6.1.6 Set a user's rights by project



PART IV

Implementation

CHAPTER 7

Mapping Design to Code

7.1 Creating Class Definitions:

Implementation in an object-oriented programming language such as C++ requires source code for:

- Class definitions
- Method definitions

At the very least, class diagrams depict the class name, superclasses, method signatures, and simple attributes of a class. This is sufficient to create a basic class definition in an object-oriented language. These class definitions in C++ are included below.

7.2 Class Definitions:

7.2.1 ItemClass:

```
#ifndef Item_h
#define Item_h 1

#include "XMLReader.h"
#include "XMLWriter.h"

class Item {
public:
    Item();
    ~Item();

    bool checkIn ();
    bool checkOut ();
    bool destroy();
    bool labelVersion ();
    bool undoCheckOut ();
    bool getLatestVersion ();
    bool getItem ();
```

```
//Get and Set Operations for XMLWriter and XMLReader
```

```
const XMLWriter * get_the_XMLWriter () const;
```

```
void set_the_XMLWriter (XMLWriter * value);
```

```
const XMLReader * get_the_XMLReader () const;
```

```
void set_the_XMLReader (XMLReader * value);
```

```
protected:
```

```
// Get and Set Operations for Class Attributes
```

```
const _bstr_t get_m_itemName () const;
```

```
void set_m_itemName (_bstr_t value);
```

```
const _bstr_t get_m_itemPath () const;
```

```
void set_m_itemPath (_bstr_t value);
```

```
const _bstr_t get_m_parentPath () const;
```

```
void set_m_parentPath (_bstr_t value);
```

```
const _bstr_t get_m_parentName () const;
```

```
void set_m_parentName (_bstr_t value);
```

```
const void get_m_pItem () const;
```

```
void set_m_pItem (void value);
```

```
private:
```

```
_bstr_t m_itemName;
```

```
_bstr_t m_itemPath;
```

```
_bstr_t m_parentPath;
```

```
_bstr_t m_parentName;
```

```
IVSSItem *m_pItem;
```

```
XMLWriter *the_XMLWriter;
```

```
XMLReader *the_XMLReader;
```

```
};  
#endif
```

7.2.2 Project Class:

```
#ifndef Project_h  
#define Project_h 1  
#include "Item.h"  
class Project : public Item{  
public:  
Project();  
~Project();  
Boolean createProject ();  
Boolean addFile (); };  
#endif
```

7.2.3 User Class:

```
#ifndef User_h  
#define User_h 1  
#include "XMLReader.h"  
#include "XMLWriter.h"  
class User {  
public:  
User();  
~User();  
bool changePassword ();  
// Get and Set Operations
```



```

const XMLWriter * get_the_XMLWriter () const;
void set_the_XMLWriter (XMLWriter * value);
const XMLReader * get_the_XMLReader () const;
void set_the_XMLReader (XMLReader * value);

protected:
// Get and Set Operations for Class Attributes
const IVSSUserPtr get_m_pUser () const;
void set_m_pUser (IVSSUserPtr value);

private:
// Get and Set Operations for Class Attributes
const _bstr_t get_m_Name () const;
void set_m_Name (_bstr_t value);
const _bstr_t get_m_Password () const;
void set_m_Password (_bstr_t value);
const _bstr_t get_m_Permission () const;
void set_m_Permission (_bstr_t value);
const projectRights get_m_pProjList () const;
void set_m_pProjList (projectRights value);
// Data Members for Class Attributes
IVSSUserPtr m_pUser;
_bstr_t m_Name;
_bstr_t m_Password;
_bstr_t m_Permission;
projectRights m_pProjList;
XMLWriter *the_XMLWriter;

```

```
XMLReader *the_XMLReader;
};
#endif
```

7.2.4 Administrator Class:

```
#ifndef Administrator_h
#define Administrator_h 1
#include "User.h"
class Administrator : public User {
public:
Administrator();
~Administrator();
bool addUser ();
bool deleteUser ();
bool editUser ();
bool setProjectRights ();
};
#endif
```

7.2.5 VSSDatabase Class:

```
#ifndef VSSDatabase_h
#define VSSDatabase_h 1
#include "XMLReader.h"
#include "XMLWriter.h"
class VSSDatabase{
public:
VSSDatabase();
```

```

~VSSDatabase();

Boolean open ();

void readVSS ();

// Get and Set Operations

const XMLWriter * get_the_XMLWriter () const;

void set_the_XMLWriter (XMLWriter * value);

const XMLReader * get_the_XMLReader () const;

void set_the_XMLReader (XMLReader * value);

private:

// Get and Set Operations for Class Attributes

const _bstr_t get_iniPath () const;

void set_iniPath (_bstr_t value);

const IVSSDatabase get_m_pVSSDb () const;

void set_m_pVSSDb (IVSSDatabase value);

private:

// Data Members for Class Attributes

_bstr_t iniPath;

IVSSDatabase m_pVSSDb;

XMLWriter *the_XMLWriter;

XMLReader *the_XMLReader;

};

#endif

```

7.2.6 XML Class:

```
#ifndef XML_h
#define XML_h 1

class XML {

public:

virtual XML();

virtual ~XML();

// Get and Set Operations for Associations

const RequestListener * get_the_RequestListener () const;

void set_the_RequestListener (RequestListener * value);

protected:

// Get and Set Operations for Class Attributes

const IXMLDOMDocument get_m_pXMLDoc () const;

void set_m_pXMLDoc (IXMLDOMDocument value);

const IXMLDOMEElement get_m_pXMLElement () const; void set_m_pXMLElement
(XMLDOMEElement value);

private:

// Data Members for Class Attributes

IXMLDOMDocument m_pXMLDoc;

IXMLDOMEElement m_pXMLElement;

};

#endif
```

7.2.7 XMLReader Class:

```
#ifndef XMLReader_h
#define XMLReader_h 1

#include "XML.h"

class XMLReader : public XML {

public:
XMLReader();
~XMLReader();

Boolean getAttribute ();

Boolean getData ();

private:
Boolean XMLToBinary ();

};

#endif
```

7.2.8 XMLWriter Class:

```
#ifndef XMLWriter_h
#define XMLWriter_h 1
#include "XML.h"
class XMLWriter : public XML{
public:
XMLWriter();
~XMLWriter();
Boolean setNode ();
Boolean setData ();
protected:
bool binaryToXML ();
const attributeList get_m_pAttList () const;
void set_m_pAttList (attributeList value);
private:
attributeList m_pAttList;
};
#endif
```

CHAPTER 8

Implementation Specific Detail

8.1 How the system works:

We know that the complete system is formed of a client and a server. The SourceSafe database resides at the machine where the server is installed. The client closely resembles the Visual SourceSafe.

8.1.1 Request is made:

What the client does is to send the requests to the server. These requests come in the form of XML². For instance, for the login request, the client in-fact opens an ASP page named *login.asp* with an HTTP POST request to send the XML information to the server.

8.1.2 ASP page is opened:

As the result of the POST request, the ASP page uses the BinaryRead method to retrieve data sent to the server and stores it in a Safe Array of bytes.

8.1.3 COM component is called:

The COM component is called from within the ASP page. This component is an in-server component that retrieves the Safe Array of bytes from the ASP page and converts it to XML. It then proceeds to extract the relevant information, such as the user name and password in this case. This information is then used for accessing the SourceSafe database by automation. The result of the operation is either success or failure so this information is converted to XML and sent to the ASP page.

8.1.4 ASP page writes response:

The ASP page writes the XML response to the client and closes.

Thus the system works as above for different requests.

² Appendix D contains the DTDs for different XML files that have been used.

8.2 Converting files to and retrieving them from XML:

The file transfer between the client and the server also takes place in XML. For instance, in case of checking out a file, the file is read from the secondary storage in binary format. This is then converted to base64 encoded form and attached to a node in the XML file. Hence binary data is passed along in XML as well.

When a file is being added or checked in, then the base64 encoded information (which contains the file) is extracted, decoded and the file is saved in its original form on the secondary storage.

PART IV

Testing

CHAPTER 9

Test Cases

2.1 Server Testing for Client Requests

Test Case	Expected Test Results	Actual Results	Requirements
Login with valid username and password	XML file is generated for the VSS Repository and is sent to the client.	Same as expected	R1.1 R1.2
Login with invalid username	XML file is generated with the error of "invalid user name".	Same as expected	R1.1 R2.14
Login with correct username but invalid password	XML file is generated with the error of "invalid password".	Same as expected	R1.1 R2.14
Checking out a file from a project	File is successfully checked out, converted to XML and sent to the client.	Same as expected	R3.6 R2.2
Checking in a file to a project	XML file is retrieved from the client. It is converted into the original file and checked in successfully.	Same as expected	R3.2 R2.9
Add a file to the project when user has add access rights.	XML file is received from the client, converted to the original file and is successfully added to the SourceSafe repository.	Same as expected	R3.1 R2.9
Add a file when user doesn't have add access rights	XML file is generated with the error of "access denied" and is sent to the client successfully.	Same as expected	R3.1 R2.9
Add a file to the project when user has read-only access to VSS	XML file is generated with the error of "access denied" and is sent to the client successfully.	Access is denied but error is not sent to client	R3.1 R2.14

Create a project	XML received from the client is parsed, relevant information is extracted and project is successfully added	Same as expected	R3.8 R2.15
Create a project when user does not have Add access rights	XML file is generated with the error of "access denied" and is sent to the client successfully.	Access is denied but error is not sent to client	R3.8 R2.14
Create a project when user has Read-only access to VSS	XML file is generated with the error of "access denied" and is sent to the client successfully.	Access is denied but error is not sent to client	R3.8 R2.14
Deleting a file/project	XML received from the client is parsed, relevant information is extracted and file or project is successfully deleted.	Same as expected	R3.3 R2.15
Deleting a file when user does not have add access rights	XML file is generated with the error of "access denied" and is sent to the client successfully.	Access is denied but error is not sent to client	R3.3 R2.14
Deleting a file if user has Read-only access to VSS	XML file is generated with the error of "access denied" and is sent to the client successfully.	Same as expected	R3.3 R2.14
Deleting a project when user does not have Add access rights	XML file is generated with the error of "access denied" and is sent to the client successfully.	Same as expected	R3.3 R2.14
Deleting a project when user has Read-only access to VSS	XML file is generated with the error of "access denied" and is sent to the client successfully.	Access is denied but error is not sent to client	R3.3 R2.14

Labeling a file/project	XML received from the client is parsed, relevant information is extracted and file or project is successfully labeled.	Same as expected	R3.5 R2.15 R2.5
Get latest version of a file	Latest version of file is retrieved from the SourceSafe database, converted to XML and sent to the client successfully.	Same as expected	R3.7 R2.3
Get latest version of a file which is checked out	Latest version of file is retrieved from the SourceSafe database, converted to XML and sent to the client successfully.	Same as expected	R3.7 R2.3
Get latest version of a file when user does not have Read access rights	XML file is generated with the error of "access denied" and is sent to the client successfully.	Access is denied but error is not sent to client	R3.7 R2.3 R2.14
Get history of a file	After reception and parsing of the Client request, a history of the file is retrieved from the SourceSafe database. It is converted to XML and sent to the client.	Same as expected	R3.11 R2.3
Get history of a file which is checked out	Same as "Get history of a file"	Same as expected	R3.11 R2.3
Get earlier version of a file	After reception and parsing of the Client request the earlier version of a file is gotten. It is converted to XML and sent to the client.	Same as expected	R3.9 R2.3

9.2 Server Testing for Administration Client

Test Case	Evaluation	Actual Results	Requirements
Adding a user	User is successfully added to user list	Same as expected	R4.1 R2.10
Adding a user with incorrect verify password	XML file is generated with the error of "access denied" and is sent to the client successfully.	Same as expected	R4.1 R2.14
Adding a user with assigning Read-only access	User is successfully assigned Read-only access rights	Same as expected	R4.1
Changing user password	User password is successfully changed	Same as expected	R4.5
Changing user password with wrong password	Error message is sent to the client in XML.	Error message is not sent	R4.5 R2.14
Changing admin password with incorrect current password	Error message is sent to the client in XML.	Same as expected	R4.5
Deleting a VSS user	User is successfully deleted form user list	Same as expected	R4.2 R2.11
Changing rights of a user from Read-only to Read-Write	User Read-only access rights are successfully changed to Read-Write	Same as expected	R4.4 R2.11 R2.12
Changing rights of a user from Read-Write to Read-only	User Read-Write access rights are successfully changed to Read-only	Same as expected	R4.4 R2.11 R2.12

Assigning project rights to a user	Users rights successfully assigned	Rights were not successfully assigned	R4.6 R2.11
Copying project rights of a user to another	Copying project rights successful	Same as expected	R4.7 R2.12

CHAPTER 10

Evaluation and Future Enhancement

10.1 Evaluation

The system is providing SourceSafe operations to clients from any remote location over a LAN network using HTTP.

We have tested the Client on both Linux and Windows and it is working properly. Hence the SourceSafe operations can access from operating systems other than the Windows family of operating systems.

The system fulfills almost all the requirements accept assignment of user rights.

GUI is very much similar to that of Visual SourceSafe Explorer except that Administrator and Client Explorer are merged for Admin.

The server part has also been installed and tested on Windows 2000 Server. IIS has to be configured before the server can be used. The server has been accessed from both Linux and Windows and works properly and according to the requirements.

10.2 Future Enhancement

In future following enhancements are easily possible

- The system could be made entirely web-based so that all these features could be available from within a web-browser
- The server could be converted to a web service using SOAP
- Data encryption could be used for better security.
- Projects or files sharing could be added
- Multiple check out could be permitted
- Files merging could be added
- Recursive SourceSafe operation on projects could be added
- Fast searching for files and text in files could be added

PART V

Appendices

Appendix A

Process model

The model that we have chosen for our project is the spiral model. It is an evolutionary process model that couples the iterative nature of prototyping with the controlled and systematic aspects of linear sequential model.

Software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

The model is suitable for our system as it allows developers to work independently. Further, in the beginning, the understanding of the system is low. As such, once the beginning spiral for concept development is complete, understanding of the whole system will increase. Requirements will become clarified; the knowledge of what's technically possible will also increase. The project will evolve as the process progresses thus early reaction to risks at each evolutionary level will also be possible. The software development process will also be manageable due to the development of the project in stages.

Appendix B

Required Resources

Developer End:

The minimum hardware required for the development of this system is two Intel Pentium class processor based computers. One computer, a Pentium-II, with at least 64 MB RAM is required for the server while another computer at least Pentium-I 133 MHz would act as a client machine. These computers would have to be inter-connected using network-cards to simulate a TCP-IP network.

The software required for the server will include a Windows 2000 server along with the installation of VC++ (including VSS as VCS uses the VSS APIs) and Internet Information Services. At the client side the software required are Windows 2000 professional, Windows 98/95 and Linux as these are the operating systems on which the client side of the system would be tested. Apart from these JDK 1.3 is also required.

Server End:

A Windows 2000 Advanced Server (IIS configured and online) and Visual Source Safe server installation is needed.

Appendix C

Feasibility

Operating System Consideration:

For Server Operating System I considered Windows 2000 Server and Windows NT. I couldn't consider other operating systems such as Linux as the SourceSafe APIs that I have used come with Visual SourceSafe available in Visual Studio only installable on a Windows Operating System.

I selected Windows 2000 Server as it is the current industry standard as well as contains improved features over Windows NT. In addition to this it has built-in support for COM+.

Windows 2000 Server includes improved network, application, and Web services. It provides increased reliability and scalability, lowers the cost of computing with powerful, flexible management services, and provides the best foundation for running business applications.

Economic Feasibility:

The project is economical feasible as the specified technology is easily available. Our project would be geared towards software companies. Hence, most would already have Developer Studio installed along with which SourceSafe comes.

Further software companies already have a LAN setup in their offices. So a server would always be present. Nowadays, few of the companies would be without a dedicated Internet connection. Hence there is no need for any extra hardware for this project and it can be integrated within the existing setup of the organization.

Technical Feasibility:

The technology needed to develop the system is available. The hardware setup of a single server and a client interconnected is also present. So is the software such as, VSS, ASP, win2000, Java etc.

Any organization that already uses any component of the Developer Studio could easily use this system.

Operational Feasibility:

The system would be invaluable for developers who travel a lot as well as for multiple- site development teams. Without such a setup it would be impossible for multiple developers to work on a project outside the LAN (intranet) setup of the organization. With this system developers could access and share the SourceSafe database from any geographical location hence making it an invaluable system to use especially for telecommuters and multiple-site development teams. Further developers could access the SourceSafe database from their homes.

Appendix D

Tools and Technologies

Windows 2000:

In Windows 2000, the ASP built-in objects add methods that provide developers with enhanced program control, including improved server-side processing and error handling. Windows 2000 also includes version 5 of the script engine (VBScript.dll and JScript.dll), which provide more extensibility for ASP pages.

Internet Information Services 5.0:

Internet Information Services (IIS) 5.0 sports improvements for administration and application development. Building on the foundation of its predecessor, IIS 5.0 takes performance, reliability, and scalability to the next level. In addition to an engine overhaul for IIS, updates to core technologies improve the overall Active Server Pages experience.

ASP:

Microsoft® Active Server Pages is a server-side scripting technology that can be used to create dynamic and interactive Web applications. An ASP page is an HTML page that contains server-side scripts that are processed by the Web server before being sent to the user's browser. You can combine ASP with Extensible Markup Language (XML), Component Object Model (COM), and Hypertext Markup Language (HTML) to create powerful interactive Web sites.

Server-side scripts run when a browser requests an .asp file from the Web server. ASP is called by the Web server, which processes the requested file from top to bottom and executes any script commands. It then formats a standard Web page and sends it to the browser.

It is possible to extend your ASP scripts using COM components and XML. COM extends your scripting capabilities by providing a compact, reusable, and secure means of gaining access to information. You can call components from any script or programming language

that supports Automation. XML is a meta-markup language that provides a format to describe structured data by using a set of tags.

COM+ and COM Components:

COM+ currently encompasses two areas of functionality: a fundamental programming architecture for building software components (which was first defined by the original COM specification), plus an integrated suite of component services with an associated run-time environment for building sophisticated software components. Components that execute within this environment are called configured components. Components that do not take advantage of this environment are called un-configured components; they play by the standard rules of COM. While one can use the COM+ programming model without the component services and run-time model, much of the power of COM+ is realized when these two parts are used together.

Components provide the following benefits to an ASP application:

- Encapsulation of functionality and hiding of implementation details
- Reusability (including reuse by different client applications)
- Protection of intellectual property
- Scalability (by allowing you to distribute your application across machines)
- Configuration and deployment flexibility
- Performance (especially when early binding is a significant factor)
- Access to the system, such as Win32 API calls or any other low-level features of programming languages
- Strong typing (Visual Basic® Scripting Edition [VBScript] is weakly typed, and JScript® isn't much better)
- Separation of business logic from the user interface, or separation of the Web designer from the developer.

HTTP:

This is the Internet protocol used by World Wide Web browsers and servers to exchange information. The protocol makes it possible for a user to use a client program to enter a URL (or click a hyperlink) and retrieve text, graphics, sound, and other digital information from a Web server. URLs of files on Web servers begin with `http://`.

There are two ways to send information to the server using ASP. One is the GET request and the other is POST. GET request has many limitations. We have used the POST request.

A POST request contains entity data that is written to the server. The URI requested specifies the parent entity of the written data. For instance, you might POST an HTML file into the URI for a directory, or you might POST a database record into the URI for the database itself. This request actually contains entity data, so a number of headers describing the content apply here that you won't normally see in a request. In particular, the content-length header is required for POST request.

A server may react in many different ways to a POST request depending on the application involved. Although it's easy to think of a POST request as a write, a successful POST doesn't need to store any persistent information on the server at all. It can just be a way of sending information to the server.

XML:

Extensible Markup Language is the universal language for data on the Web. It gives developers the power to deliver structured data from a wide variety of applications to the desktop for local computation and presentation. XML allows the creation of unique data formats for specific applications. It is also an ideal format for server-to-server transfer of structured data.

XML compresses extremely well due to the repetitive nature of the tags used to describe the structure of the data. It is worth noting that compression is standard for HTTP 1.1 servers and clients, and XML automatically benefits from this.

XML also solves the problem of transferring data between different platforms. XML adds a new, intermediate level of abstraction between the data source on one hand and the user interface on the other. This layer lets you access cross-platform data from any system that does XML. Since the data is completely separate from the user interface, you can perform client-side processing before displaying the data.

Appendix E

Document Type Definition (DTD)

Document Type Definition (DTD) for items list

```
<?xml version = "1.0"?>
<!DOCTYPE VSSProjects[
  <!ELEMENT VSSProjects (PROJECT, FILE)*>
  <!ELEMENT PROJECT (PROJECT, FILE)*>
  <!ATTLIST PROJECT
    Name CDATA #REQUIRED
    Label CDATA #REQUIRED
    vssVersion CDATA #REQUIRED
    IsCheckedOut (true|false) #REQUIRED
    User CDATA #REQUIRED
    DateTime CDATA #REQUIRED
  >
  <!ATTLIST FILE
    Name CDATA #REQUIRED
    Label CDATA #REQUIRED
    Version CDATA #REQUIRED
    User CDATA #REQUIRED
    IsCheckedOut (true|false) #REQUIRED
    DateTime CDATA #REQUIRED
    Size CDATA #REQUIRED
  >
]>
```

Document Type Definition (DTD) for request

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE VSSRequest[
  <!ELEMENT VSSRequest (User, VSSItem)>
  <!ELEMENT VSSItem (Data)>
  <!ELEMENT Data (#PCDATA)>
  <!ATTLIST VSSRequest
    Type CDATA #REQUIRED
  >
  <!ATTLIST User
    name CDATA #REQUIRED
    password CDATA #REQUIRED
  >
  <!ATTLIST VSSItem
    name CDATA #REQUIRED
    path CDATA #REQUIRED
  >
]>
```

Document Type Definition (DTD) for operation result

```
<?xml version = "1.0">
<?DOCTYPE operation[
  <!ELEMENT operation >
  <!ATTLIST operation
    result (success|failure) #REQUIRED
  >
]>
```

Document Type Definition (DTD) for user list

```
<?xml version = "1.0">
<?DOCTYPE VSSUsers[
    <!ELEMENT VSSUsers (User)*>
    <!ELEMENT User (Project)*>
    <!ATTLIST User
        Name CDATA #REQUIRED
        Password CDATA #REQUIRED
        Rights (ReadOnly|ReadWrite) #REQUIRED
    >
    <!ATTLIST Project
        Name CDATA #REQUIRED
        CheckedoutAccess (true|false) #REQUIRED
        AddAccess (true|false) #REQUIRED
        ReadAccess (true|false) #REQUIRED
        DestoryAccess (true|false) #REQUIRED
    >
]>
```

Appendix F

Glossary

A

Access rights

Levels of permission users are granted by the SourceSafe administrator to use the SourceSafe. The levels of access rights are Read, Check Out, Add, and Destroy.

C

Checked-in file

File stored in the SourceSafe and unavailable for modification.

Checked-out file

File that has been reserved for work by a user. Users check out files to make changes to them. In the default configuration, SourceSafe allows only one user at a time to check out a file. Checking out a file copies its latest version into the user's working folder.

Current project

Project selected in the project pane of the VCS Explorer window.

Current version

Version of a file most recently stored in the SourceSafe. The current version has the highest version number of a file in SourceSafe.

D

Delete command

Permanently removes deleted files and projects from the SourceSafe. Once destroyed, the items cannot be recovered.

F

File list

List of files in the current project, which can be found in the file pane of the VCS Explorer window

H

History

Record of changes to a file since it was initially added to SourceSafe. With the file history, you can return to any point in the file's history and recover the file as it existed at that point.

L

Label

User-defined name you can attach to a specific version number of a file or project.

Log on

Process of entering and verifying a user's name and password to access the SourceSafe

P

Parent project

The project a file or subproject exists in. For example, the parent of the file `$/Project/Abc.txt` is `$/Project` and the parent of the project `$/Project` is the root (`$/`).

Password

Text string used as security to verify the identity of a user. A user password is often required to use the SourceSafe.

Project

Group of related files, typically all the files required to develop a software component. Files can be grouped within a project to create subprojects. Projects can be defined in any way meaningful to the user(s). For example, one project per version, or one project per language. Projects tend to be organized in the same way as file directories.

Project list

List of all the projects available in the SourceSafe; the project list is found in the left pane of the VCS Explorer window.

R

Read-only file

File marked as read-only in its file attributes. Such a file can be viewed in an appropriate text editor, but cannot be modified. VCS marks the file as read-only when you use the Check In and Get Latest Version commands.

Root project

The highest-level project with the name `/` in the project list. All projects in a SourceSafe are subprojects of the root project.

S

Security

SourceSafe has two levels of security: default security and project security. Default security provides two access rights: read/write and read-only. When project security is enabled, four access rights are available per user, per project: Read, Check Out, Add, and Destroy. Each succeeding right includes all rights preceding it. The Destroy access right provides unlimited access and is equivalent to Read/Write rights under default security.

Source code control

The management of a file's change history and the file's relation to a larger grouping of related files known as a project. Source code control is a vital part of the efficient development of software applications. SourceSafe is a project-oriented source code control.

Subproject

Project within a parent project.

U

User list

List of users who can use the SourceSafe. The list is maintained by the SourceSafe administrator and displayed in VCS Administrator's main window.

Username

Unique identifying string for a given user. Used for logging on.

V

Version control

SourceSafe maintains multiple versions of a file, including a record of the changes to the file from version to version.

Version number

Number that indicates the number of revisions a file has undergone since it was added to VSS. This number is displayed in the History dialog box. Version numbers are always whole numbers.

Version tracking

Record keeping process of tracking a file's history from the initial version to the current version. Changes to a file are tracked as part of this process.

VCS Administrator Client

The client for the administrator with an additional menu for administrative tasks.

VCS Server

The ASP pages that reside on the IIS along with the registered COM components.

VCS User Client

The client for the normal users without the additional menu for administrative tasks.

VCS User

The developer who uses VCS Client Explorer to control his source and perform common VSS operations.

W**Working folder**

Specified folder on a user's local computer used to store files when they are checked out of the SourceSafe. A user makes changes to files in the working folder and then checks the modified files back into the SourceSafe for version tracking.

Bibliography

1. [Craig98] Craig Larman. 1998, *Applying UML and Pattern*, PRENTICE HALL.
2. [Rumbaugh00] James Rumbaugh, Ivar Jacobson, Grady Booch. 2000, *The Unified Modeling Language Reference Manual*, ADDINSON-WESLEY.
3. [Boggs99] Wendy Boggs, Michael Boogs. 1999, *UML with Rational Rose*, SYBEX
4. [Sommerville96] Ian Sommerville. 1996, *Software Engineering*, ADDINSON-WESLEY
5. [Pressman97] Roger S. Pressman. 1997, *Software Engineering A Practitioner's Approach*, McGRAW-HILL
6. [Chris99] Chris Ullman, John Kauffman, David Syssman. 1999, *Beginning ASP 3.0*, WROX PRESS Ltd
7. [Mueller00] John Paul Mueller. 2000, *Windows 2000, COM+, MTS & MSMQ Programming Bible*, IDG Books Worldwide Inc.
8. [Jacobson92] Jacobson, I. 1992, *Object-Oriented Software Engineering: A Use Case Driven Approach*, ADDINSON-WESLEY.
9. [Wirfs-Brock93] Wirfs-Brock, R. 1993, Designing Scenarios: Making the Case for a Use Case Framework. *Smalltalk Report* Nov-Dec 1993. SIGS PUBLICATIONS.
10. [Ash00] Ash Rofail, Yasser Shohoud. 2000, *Mastering COM and COM+*, SYBEX.
11. The MSDN Library.