VERSION CONTROL SYSTEM IMPLEMENTED IN PHP

RATHAPOL KONKAEW

MASTER OF SCIENCE

IN COMPUTER SCIENCE

MAE FAH LUANG UNIVERSITY

2007

VERSION CONTROL SYSTEM IMPLEMENTED IN PHP

RATHAPOL KONKAEW

AN INDEPENDENT STUDY SUBMITTED TO

MAE FAH LUANG UNIVERSITY IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN COMPUTER SCIENCE

MAE FAH LUANG UNIVERSITY

2007

VERSION CONTROL SYSTEM IMPLEMENTED IN PHP

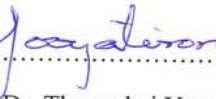
RATHAPOL KONKAEW

THIS INDEPENDENT STUDY HAS BEEN APPROVED
TO BE A PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
IN COMPUTER SCIENCE
2007

EXAMINING COMMITTEE

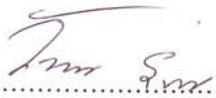
.....CHAIRPERSON

(Assoc. Dr. Wanchai Rivepiboon)



.....MEMBER

(Lecturer Dr. Thongchai Yooyativong)



.....MEMBER

(Lecturer Vittayasak Rujivorakul)

.....MEMBER

(Lecturer Somsamorn Srisangwan)

.....MEMBER

(Lecturer Dr. Punnarumol Temdee)

ACKNOWLEDGEMENT

I sincerely thank School of Information Technology, Mae Fah Luang University, especially Ms. Somsamorn Srisangwan for advising me. I thank Technology team in Chanwanich Co., Ltd. for testing system and giving comments. I thank Mr. Somphong Intana, a friend of mine in the company, for sharing web hosting for use in the project. And last, I thank Chanwanich Co., Ltd. for allowing me work on the project at the office.

Rathapol Konkaew

Independent Study Title Version Control System Implemented in PHP

Author Mr. Rathapol Konkaew

Degree Master of Science (Computer Science)

Supervisory Committee

Assoc. Dr. Wanchai Rivepiboon	Chairperson
Dr. Thongchai Yooyativong	Member
Lecturer Vittayasak Rujivorakul	Member
Lecturer Somsamorn Srisangwan	Member
Dr. Punnarumol Temdee	Member

ABSTRACT

This project is an attempt to implement a low cost version control system in PHP which can run on low cost web server platforms like Linux. Subversion, the popular existing system, is studied for its functionalities and techniques. The project implements basic version control functionalities, such as versioning, and concurrency handling. It uses MySQL, the database management system, to store and manage repository data. It provides a client application to interface with the system.

Delta compression is applied to reduce disk space of the storage. Content of a newly committed revision is stored as full text while content of the previous revision is replaced with a delta of the new content against the previous content. The size of the storage is O(N), where N is a number of revisions. Time taken to reproduce content of a particular revision is O(N – M), where N is a number of revisions and M is the requested revision number. As a result, content of the head revision takes O(1) to be reproduced. The system has been deployed and tested on a Linux web server. For normal use, the system performs reasonably. The limitations of the system are numbers of revisions, bandwidth, and database disk space in the server.

Keyword: Version Control System / PHP

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Cont.)

LIST OF FIGURES (Cont.)

CHAPTER I

INTRODUCTION

1.1 Background

Version control systems support teamwork by collecting documents in a centralized repository and controlling the document version [13]. Each team member checks out document files from the system. He modifies the files in his system. Once he has checks in the files back to the system, the controlling version is increased. Some systems also have a functionality to support editing the same file by multiple users.

Most version control systems have their own server implementation [11]. Some of them have a functionality to work with web HTTP protocol. This allows users to work on the systems via the Internet [12]. Because they have their own server implementation, only specific servers can run the systems. For example, SourceForge.net runs CVS (a version control system) server for open source projects. To run proprietary/private projects with supporting a version control system over the Internet, hiring a server supporting such systems is required. This kind of system is usually expensive.

This project's target is to implement a version control system which runs on general online web servers. It is planed to be written in PHP. MySQL is used for its database management system. PHP code can run on multiple platforms [15]. Servers running PHP are usually cheaper than those running a version control system. A client program will be written to manage working copies on a personal computer.

1.2 Objectives of the study

1.2.1 To support teamwork by centralizing documents on the Internet.

1.2.2 To implement a version control system running on low-cost web hosting services.

1.3 Scope of the study

1.3.1 Provide a web page to view file structures in the file repository.

1.3.2 Provide a web page to view version-controlled text documents.

1.3.3 Provide a client program to manage working copies on user computers.

1.3.4 The software can manage user permission.

1.3.5 The software can compare a document between different versions.

1.3.6 The software can merge document files together.

1.3.7 Support text documents only.

1.4 Expected Benefits

1.4.1 Documents are centralized on a repository.

1.4.2 Documents are version controlled. All document versions exist in the system. This encourages team members to modify/remove data, because they can be reverted.

1.4.3 The system can be deployed on low-cost servers.

1.4.4 Backing up the repository is done by backing up the database.

CHAPTER II

LITERATURE REVIEW

This project studies functionalities and versioning models of various popular version control systems such as CVS and Subversion (SVN). ASVCS, the web-based version control system project, is studied since the project shares some similar approaches. For instance, it attempts to implement such system entirely in PHP.

The replacement system will be implemented in PHP with MySQL database management system. Diff algorithm and its applications are studied for text compression algorithm used in the system. Some third party libraries and tools are studied to help reducing development time.

2.1 ASVCS Project

A Simple Version Control System (or ASVCS) is a web-based version control system written in PHP [2]. To install the system, a user downloads PHP source code from the vendor's website then uploads to his server and initializes MySQL database.

The user's version-controlled files are stored in a server file system. Initially, particular files are uploaded by using SSH or FTP protocols. Consequently, the user registers those files to the system. Then, the files are version-controlled. Later on, when the controlled file content is changed and uploaded, then the user tells the system to create a new revision.

The system creates a new revision by detecting changes of the version-controlled files and recording the changes into the database. The system uses the changes stored in the database and the current file content stored in the file system together to reproduce the content of the

previous revision.

An algorithm used for detecting changes is Diff, the algorithm for comparing between two sequences of strings. The system utilizes Diff algorithm functionality of Text_Diff library. It can render output in a number of formats including the unified format. The library depends on Pearweb package, which may not be installed on some platforms.

The limitation of the project is that it is suitable for only one user. Because files are transferred to the repository via external protocols, it is possible that multiple users modify the same file and may accidentally overwrite other's content.

The cost of using this system is low. It can be deployed on a web server supporting PHP and MySQL.

This project will use the similar concept of file storing by storing changes between revisions to reduce disk space, but this project will also store full content to the database for maintenance reason. The system of this project needs to handle file transferring to handle concurrency. The server application of this project is also implemented in PHP to deploy it on low cost web servers.

2.2 CVS

Concurrent Version System (CVS) is a free open source version control system written in C developed by Dick Grune [3]. Its repository model is client-server. Supported platforms include Windows, Unix-like, and Mac OS X. Supported protocols include pserver and SSH.

The architecture of CVS is client/server. The server side organizes repositories and their revisions in a file system. The client side enables users to work with local revisions in their local machines. Copy-Modify-Merge solution is used as versioning model. All users' modified copies are merged together into the final reversion.

The general usage of the system is the followings. Initially, a user creates a working directory, and then invokes *checkout* command. A special hidden directory named “CVS” is then

created. It contains the head revision's information and file content based on the revision. The file content is also copied to his working directory. The user modifies file content in the working directory and invokes *commit* command to submit his changes to the server to create a new revision. He can invoke *update* command to receive the latest revision and merge file content in the working directory with the revision.

To run CVS server, the CVS server application must be installed as a service (on Windows) or component (on Unix). There are free CVS hosting services, such as sourceforge.net, for open source projects. However, CVS hosting service cost for closed-source / private-use projects is high.

The project will take the concept of client/server, copy-modify-merge model, and the usage as initial ideas. We can adapt the idea of storing revision on users' machines in special directories. We can reduce cost by implementing a PHP version control service that runs on web server rather than running such special version control service.

2.3 Subversion (SVN)

Subversion (SVN) is a free version control system written in C developed by CollabNet Inc [9]. Its repository model is client-server. Supported platforms include Windows, Unix-like, and Mac OS X. Supported protocols include SSH, HTTP and SSL, and svnserve (its proprietary protocol). A database used in the system can be Berkeley DB or file system. Its versioning model is copy-modify-merge.

Subversion is designed as a replacement of CVS. It mimics and extends CVS's functionalities and user interface. Its user interface for the client application is compatible with CVS. For instance, it has the similar commands / subcommands to CVS.

Subversion compressed data by using delification or delta compression. When a new revision is committed, it compresses the previous revision as a delta against the new revision.

With Subversion, a user can commit several changes of any files/directories as one revision. This feature is useful for a project dealing with multiple programming source code,

which changes of one file may affect others.

Subversion system supports various Internet protocols. It can be used in a local network with free-of-charge. There are free Subversion hosting services, such as sourceforge.net and trigis.org, for open source projects. However, Subversion hosting service cost for closed-source / private projects is high.

Our project will take the idea of committing several files as one revision. Our client application will have similar commands / subcommands for user experience reason.

2.4 HTTP Protocol

HTTP Protocol is a communication protocol layered on top of the TCP protocol with the default port of 80. Its applications include web contents transferring and web services.

HTTP standard is developed by World Wide Web Consortium (W3C). The standard defines request and response message formats. HTTP request message consists of a request line, headers, an empty line, and an optional body. The request line contains a method to perform on the identified resource. There are eight defined methods; such as HEAD, GET, POST, PUT, DELETE, TRACE, OPTION, and CONNECT.

HTTP protocol is usually not good for transferring huge binary data since its encoding method (**url-encoding**) works well only for a small amount of text data.

Our project will utilize HTTP protocol for communication between the server application and the client application. It will use GET and POST methods for requesting and transferring content to the server. The GET method is that arguments are passed by appending pairs of argument=value to the requested URL. This method is good for transferring small amount of data or requesting because the length of the URL is limited. The POST method is used for transferring larger data. The data reside in the protocol's content section allowing larger data to transfer.

2.5 PHP

PHP is a free-software script language which can run on popular web servers, such as Apache and Microsoft IIS. PHP provides a range of functions for creating dynamic web pages, manipulating files, connecting to databases, sending e-mail, etc.

Most web hosting service providers offer Linux web hosting service as a low cost solution. Usually PHP and MySQL are included in the package.

PHP is a feature-rich language coming with ranges of libraries. With string manipulation functions, built-in array/associative array, and database connection, it is possible to create such system. The limitation of PHP is its processing speed to parse and process the scripts. Writing less scripts and use its built-in functions can improve the processing speed. Usually, one built-in function works on many operations in a lower level.

Our project will implement the server in PHP because of its portability and cost as well as its language features and libraries which are possible to develop the server side application. Because of the limitation, the algorithm working with text data is per-line operation which is less expensive than per-character operation. The project will use built-in PHP functions as much as possible to improve its processing speed.

2.6 MySQL

MySQL is a free open-source relational database management system written in C/C++ licensed under GPL and proprietary license [7]. It can run on many platforms, such as Linux, Mac, Windows, etc. MySQL supports pluggable multiple storage engines, which are effective in different applications. It supports multi-threaded and multi-user.

MySQL is free, fast, and reliable. It is continuously developed and is used in many open-source and commercial projects and won many awards.

Our project will use MySQL as database management system since many web

hosting services bundle MySQL in many low cost packages. We will use SQL language to communicate with the database system.

2.7 Difference Algorithm

A paper, “An O(ND) Difference Algorithm and Its Variations” by Eugene W. Myers, presents an algorithm for finding the longest common subsequence of two sequences A and B and the shortest edit script for transforming based on an intuitive edit graph formalism [1].

The algorithm is designed for comparing two text files. The project applies the algorithm to find minimal changes between revisions and stores only the changes to keep the storage size minimum.

The algorithm is implemented for text comparison applications, such as diff in GNU's diffutils package and other Wiki web applications.

Our project will implement this algorithm for text comparison by porting it from Java source code and apply the same test suite to ensure that both versions produce the same result.

2.8 Unified Format

The Unified Format is a text format generated from the product of the difference algorithm. Its content can be read and understandable by human [14].

Unified Format content is produced based on a result of the difference algorithm. Modification lines and some context lines are grouped in a chunk with starting line numbers of both the old file and the new file and numbers of removed lines and inserted lines.

Our project will store differences of two revisions in this format. The file size produced in this format is not smallest possible, but it is maintainable.

2.9 Versioning Models

Versioning models are used to solve the concurrency problem which is that the system shares data to two or more users and prevents them to overwrite other's changes in a repository. There are two solutions which are usually used in most version control systems; lock-modify-unlock and copy-modify-merge [10].

With lock-modify-unlock solution, a user modifying a file requests for file locking to prevent other to modify it at the same time. The lock is released when he finishes modifying the file. This solution may cause an administrative problem when a user locks files and forgets to release the locks.

With copy-modify-merge solution, the system allows two or more users modify the same file independently at the same time. Eventually all the changes are merged into a final revision. The problem may cause when the user changes overlap other users' changes but the system can report the situation to the user to review the overlap and manually solve the problem.

Our project will use copy-modify-merge solution to allow multiple users modify the same file. When the user updates revision in his machine, the system merges the latest content in the repository with his working copies.

2.10 Libcurl

Libcurl is a cross-platform library for network communication [5]. The supported protocols include FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS, and FILE. Libcurl is written in C licensed under MTI/X derivate license. It supports various platforms, such as Windows, OS/2, Solaris, Unix-like, Mac OS X, etc.

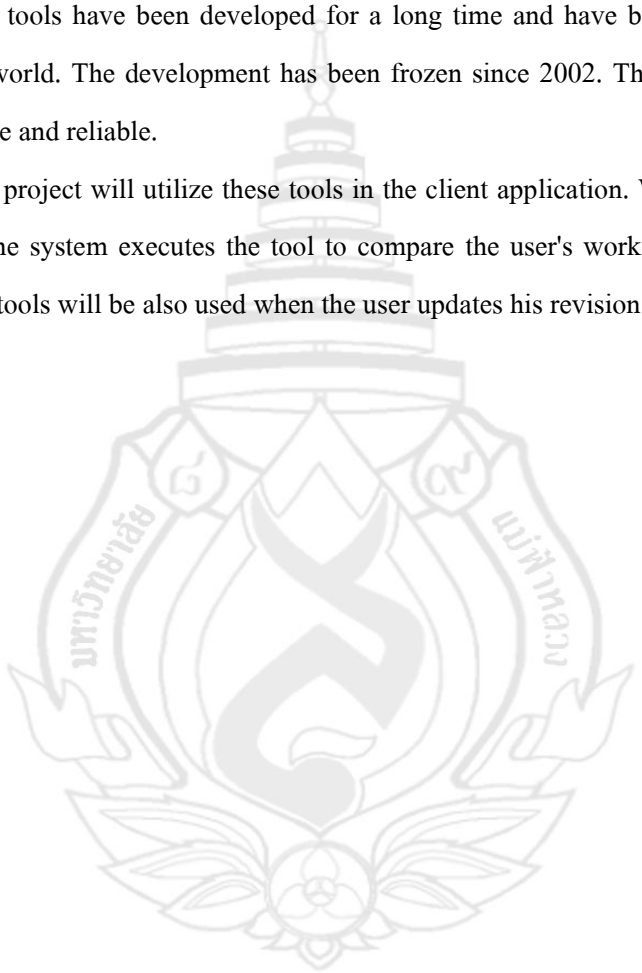
The project will use this library in the client-side application to communicate with the server-side application via HTTP protocol.

2.11 GNU Diffutils

GNU Diffutils is a collection of file comparing / merging tools [4]. The tools are developed based on the difference algorithm. It can be used to compare files as well as directory structures. It can produce a number of output formats including the unified format.

The tools have been developed for a long time and have been tested by amount of users over the world. The development has been frozen since 2002. Therefore, the tools can be considered stable and reliable.

Out project will utilize these tools in the client application. When a user invokes *diff* subcommand, the system executes the tool to compare the user's working file and a file in the repository. The tools will be also used when the user updates his revision in his machine.



CHAPTER III

FEASIBILITY STUDY

3.1 Introduction

Version control systems are systems that support team members working together in software development field. The Internet is used to resolve a problem of distance collaboration. There are many version control system product titles available. Some of them provide commercial license. Some of them are free allowing people to download and use them. Examples of free popular systems are CVS and Subversion (SVN).

CVS SVN and most other version control systems are designed as client/server system. Typically the server application of the system has to be installed as a service or a component of an operating system. Many version control system hosting services charge expensive.

There are projects available on the Internet implementing web-based version control systems. Example projects are ASVCS and Web-based Wikis. These web-based systems typically do not handle concurrency.

The project's approach is to develop a client/server version control system. A server application of the system is responsible for managing documents and handling concurrency of document submission. A client application of the system is responsible for managing documents on user machines and providing control to the users. HTTP protocol is used for client/server communication.

The server application of the system is developed in PHP. MySQL is used as database management system. Delta compression is used to reduce document spaces. The text comparison

algorithm is ported from Java source code available on the Internet [6]. The same test set is applied to ensure accuracy. The Unified format is used to represent result of the algorithm. The PHP server application can be deployed on most low-cost web hosting services.

PHP is a popular programming script language. It is typically used to develop dynamic web pages. It provides ranges of functions as well as interfaces to connect with several database management systems including MySQL, the cross platform database management system. Its scripts can run on many platforms, such as Windows, Mac, Linux and other Unix-like systems. HTTP protocol is supported by the language.

The client application of the system is developed in C/C++ with additional libraries and third party tools. It is a simple console application allowing users to interface with the system. The additional libraries and the third party tools are libcurl and GNU diffutils. Libcurl is used for HTTP handling. GNU diffutils provides tools for comparing and merging text documents.

3.2 Problem statement

Most version control hosting services typically charge expensive. For the lower cost services, they limit numbers of users or they permit only for open source projects.

3.3 Related research and projects

This project studies some basic functionality of existing popular systems. Such systems are the followings.

3.3.1 CVS - CVS is a cross platform version control system. It is widely used by

many open-source projects. It supports pserver and ssh protocols.

3.3.2 Subversion (SVN) - SVN is a cross platform version control system.

There is an existing project implementing a version control system in PHP. The project can do version control tasks and can be deployed on low cost web hosting services. It lacks of multi-user and concurrency handling.

3.3.3 ASVCS – A Simple Version Control System - The ASVCS project implements a simple version control functionalities entirely in PHP.

This project attempts to implement a version control system to run on low cost platforms while it has some basic functionality similar to the popular systems. Such functionalities are auto-version-numbering, concurrency handling, multi-user support, and version traceability.

3.4 Requirement specifications for the new system

The followings are requirements for the system:

3.4.1 Use the HTTP web protocol for client-server communication.

3.4.2 The server application can be installed into low-cost web hosting services, such as Linux web hosting.

3.4.3 The system can be used for private/closed-source projects.

3.4.4 The system supports concurrency.

3.4.5 An operating system for the server application which can run PHP and MySQL

3.4.6 WindowsXP for the client application

3.4.7 A web application service, such as IIS or Apache

3.4.8 PHP version 4 or higher

3.4.9 MySQL version 4 or higher

3.4.10 GCC – GNU Compiler Collection

3.4.11 Libcurl

3.4.12 GNU diffutils

3.4.13 A server computer

3.4.14 A client computer

3.4.15 Internet connection

3.5 Implementation techniques

The following implementation techniques will be applied:

3.5.1 The server application is written in PHP – PHP scripts can run on most Linux web hosting services. Moreover, Linux web hosting usually costs lower than Windows or other web hosting.

3.5.2 All data are stored in a database – this makes easy for maintenance.

The client application uses existing proven tools, such as GNU Diffutils and Libcurl, to reduce development time.

3.5.3 The difference algorithm is ported from a Java source code published in the Internet under GPL license. As a result, this project must derive the license. Provide the same test inputs as the original source.

3.5.4 The product of the difference algorithm is stored in the unified format – this makes it small and readable by human.

3.6 Deliverables

When the project is finished, the followings will be delivered:

3.6.1 The server web application

3.6.2 The client application

3.7 Implementation plan

The implementation plan starts with gathering information about version control systems and requirements. Some version control systems are selected to study and analyze. After that, the system will be designed and implemented. Finally, several tests will be applied. See Table 3.1.

Table 3.1 Implementation Plan

No	Task	Duration
1	Gathering Information / Requirements	2 days
2	Studying existing systems	2 days
3	Analysis / Design	5 days
	System Analysis (1day) System Design (1day) Database Design (1day) User Interface Design (1day) Selecting Software / Tools (1day)	
4	Implement	10 days
	Server Application Implementation (4 days) Web Pages (2 days) Client Application Implementation (3 days) Maintenance Tools Implementation (1 day)	
5	Test	12 days
	Algorithm Test (2 days) Unit Test (5 days) Integration Test (5 days)	
		32 days

CHAPTER IV

SYSTEM ANALYSIS AND DESIGN

4.1 Introduction

This chapter covers analysis of an existing system and requirements for the new system as well as a design of the new system.

An existing system analyzed is Subversion. The study covers its architecture, usage, versioning model, and data storage and compression techniques. The study can be used as knowledge base for developing the new system.

User requirements of the new system are that the version control system can be deployed on low cost platforms like Linux while it does basic version control functionalities, and works reasonably.

The design of the system covers use case, system architecture, network, system tier, business logic, database, operation, algorithm, and user interface. Several diagrams are included.

4.2 Analysis of the existing system

This project mainly studies Subversion, which is an improvement of CVS, another popular system. The study covers its architecture, usage, versioning model, and data storage and compression techniques.

4.2.1 Architecture

The system architecture of Subversion is client/server. It supports various application protocols including SSH, HTTP and SSL, and svnserve (its proprietary protocol). Its server application and client its application can be installed in the same machine or separate. The system supports two types of storage managements: Berkeley DB and native file system. A client library is available for developing enhancement of the client application.

4.2.2 Usage

The system provides client applications for interfacing with users and administrators. The applications are command-line applications providing commands similar to CVS, another popular system.

The basic usage of the system is checking in / checking out contents with repositories. A user starts using repository with *checkout* command. Then, the head revision is copied to his local machine. After modifications have been done, he can execute *commit* command to check in the changes. Then, the system creates a new revision. At a particular time, he can execute *update* to update his copy with the latest revision submitted by other users.

There are other third party tools for enhancement. TortoiseSVN is a tool integrated with Windows shell to improve visualization and user experience. Subversion commands are attached into context menu in Windows Explorer. Icons of files and folders in version control directories show status of the objects. Other kind of tools is to make the server visualized by developing a web portal to the system. Users do not need to have the client application. They can browse and view repositories via a web browser.

4.2.3 Versioning Model

The versioning model used in the system is copy-modify-merge. When a user commits modifications, the system creates a new revision. Consequently, another user can not commit his modifications right after the previous commit. He needs to update his work with content of the latest revision. Once the work has been updated, he can review the merged content, modify it, and commit it.

4.2.4 Storage

Subversion provides storage options to manage repositories: Berkeley DB or native file system. Initially, the system was designed to use Berkeley DB for taking advantage of the DBMS's functionalities, such as transaction support, support for cursors, hot backups, logging facilities, etc. The DBMS is not available on all platforms and its database file is not compatible across platforms. Later, the file system storage was developed.

Deltification is used for compressing data in repositories [8]. The system stores delta chunks rather than full texts to keep the space size minimum. When a new revision is created, the previous revisions are computed as a delta against the new revision. Therefore, the size of the storage is O(N), where N is a number of revisions. Time taken to reproduce content of a particular revision is O(N – M), where N is a number of revisions and M is the requested revision number. As a result, content of the head revision takes O(1) to be reproduced.

4.3 User requirement analysis

This system is designed for developer teams who have limited budget requiring a low-cost version control system for their closed projects.

4.3.1 Personal / private version control system

A user can install his/her own system into his/her own server. The server side implemented in PHP can be installed on most of personal / shared / rented servers if they can run the PHP script and have MySQL database installed. Additional services or extensions are not required.

4.3.2 Low cost

The server side is a PHP web application. It can run on Linux servers which are usually lower cost than Windows servers or others.

4.3.3 Accessible via HTTP protocol

The HTTP protocol is used for network communication for web sites / web services on the Internet and the Intranet. The HTTP port is generally open while other ports are blocked by firewalls or routers.

4.3.4 Minimum space

Delta compression is used to reduce data size stored in the database.

4.3.5 Backup / Restore system

Because all revisions are stored in the database, a user can backup entire data in the database to backup all revisions.

4.4 System design

4.4.1 Use Case

There are two groups of users in the system: administrator and repository user. The administrators are responsible for managing repositories and users as well as doing backing up and restoring repository information. The repository users have rights to check out, modify, and check in documents. They are able to browse documents in a repository, navigate through different revisions, and track document changes. See Figure 4.1 for the use case diagram.

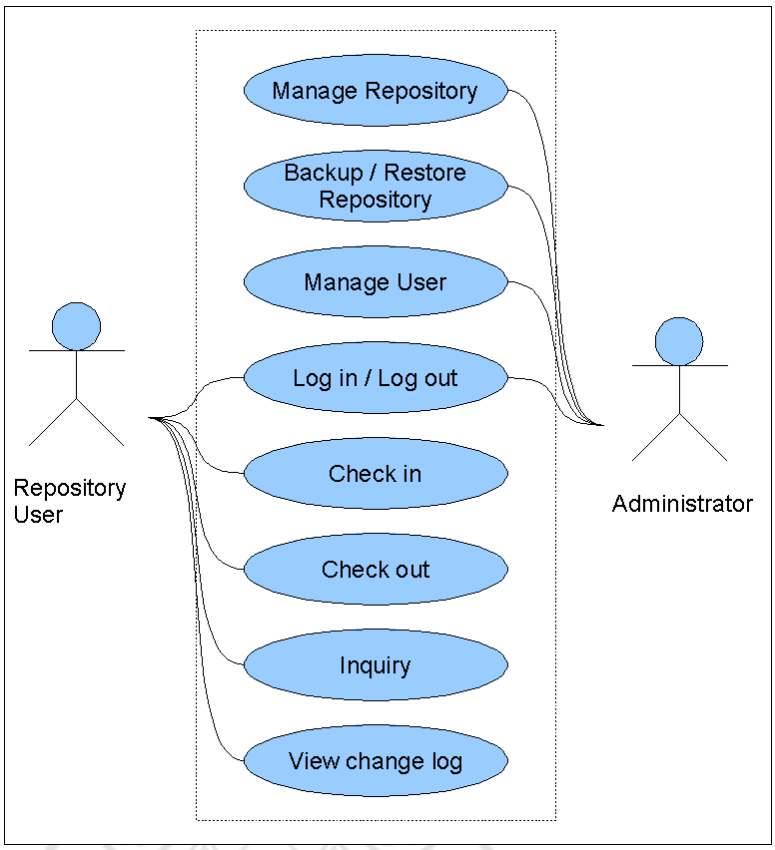


Figure 4.1 Use Case Diagram

4.4.2 Architecture and Network Design

The system consists of two parts; server and client. The server part manages and stores submitted contents into the database. The client part is used to connect to the server part and manage working copies on local machines. The server and the client applications connect each other via the HTTP protocol.

In the server part, the management code is written in PHP which can run on various platforms. MySQL is used as a repository database.

The client part is a simple application written in C/C++. It provides an interface for the user to interact with the server. It can be used to compare working copies and revisions checked out from the server.

The server runs on a web hosting service. It communicates multiple users via HTTP protocol. The web server and the database server may or may not physically be in the same machine (See Figure 4.2).

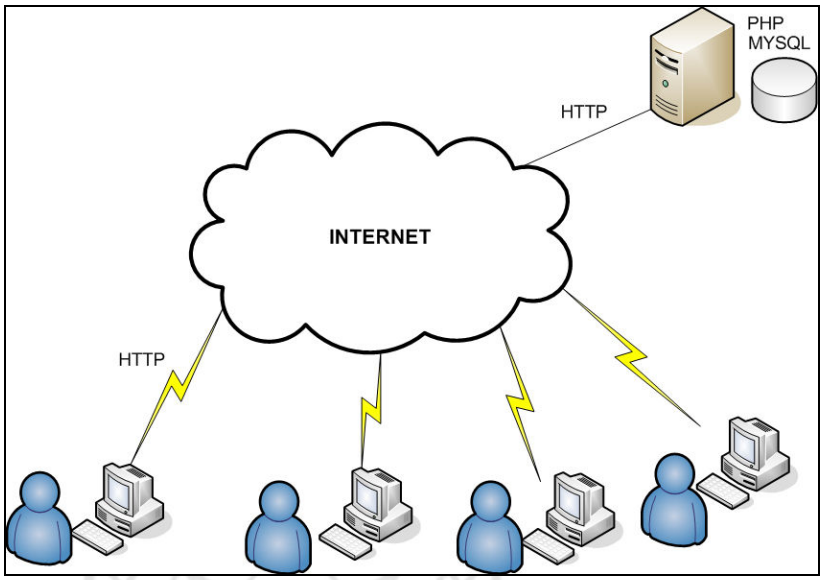


Figure 4.2 Network Diagram

4.4.3 Tier Design

The system is separated into four tiers; UI, Web Interface, Business Logic, and Data (See Figure 4.3).

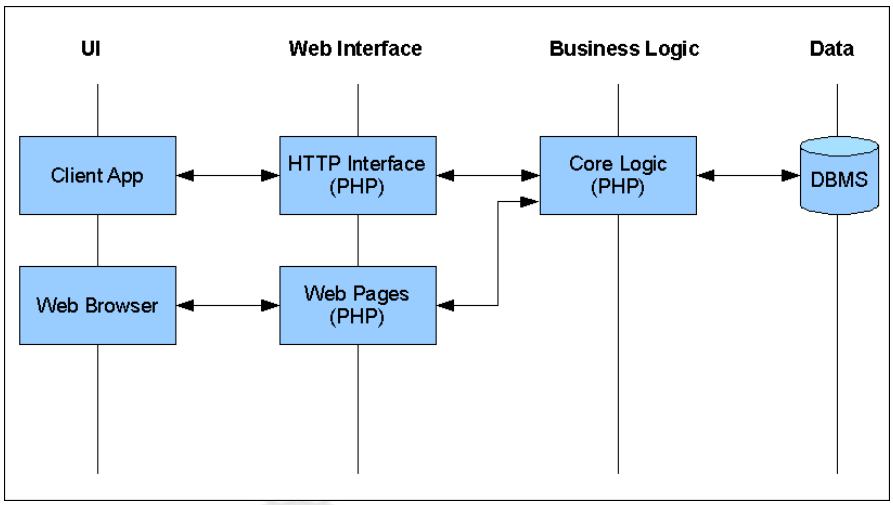


Figure 4.3 Tier Design

1. UI Tier

UI Tier interacts with users directly. This tier consists of two applications; the client application and a web browser to view web pages to manage/administrate repositories. UI Tier communicates with Web Interface Tier via HTTP protocol.

2. Web Interface Tier

Web Interface Tier consists of two applications; a HTTP Interface to interface with the client application and web application to interface with users via web browser. Web Interface Tier communicates with Business Logic Tier via PHP function call.

3. Business Logic Tier

Business Logic Tier does version control system functions. It connects directly to MySQL databases.

4. Data Tier

MySQL is used to store repository, user, and other data.

4.4.4 Business Logic

There are two major business logics in the system: Check In and Check Out.

1. Check In

Once a user has created/modified documents, he commits them to a repository. The system firstly checks whether the documents have modified from the latest revision. If so, all documents are submitted to the server then a new revision is committed. Otherwise, he fails to commit his contents and the system notifies him to update his copies.

For each document submission, the system find differences between the document and the current head revision then add the differences in to a temporary database table. Once all documents have been submitted, the server moves the data from the temporary table to the working table.

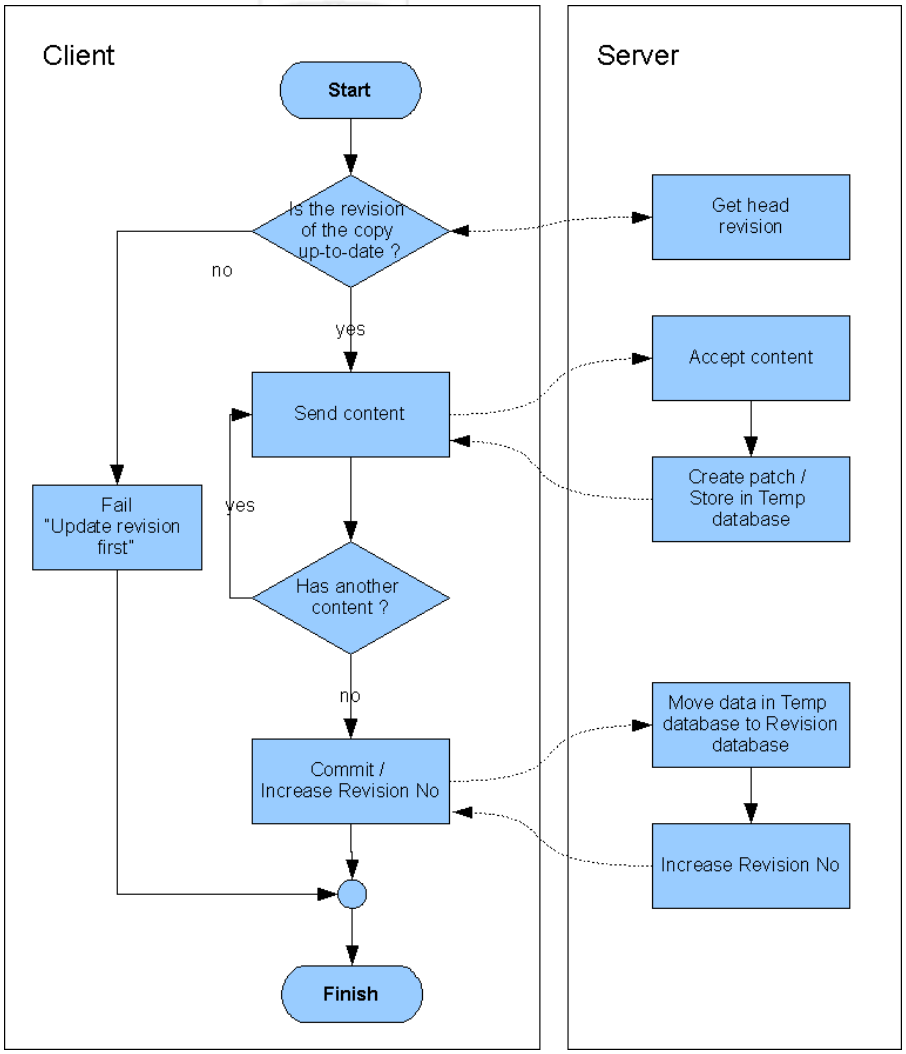


Figure 4.4 Check In

2. Check out

When check out, the client application requests / retrieves latest revision file list from the server. Then, it iterates each file in the list. For each file, it requests/retrieves file content from the server and updates the content with the local copy. Once all documents have been updated, the revision number in the user's machine is changed to the latest revision number.

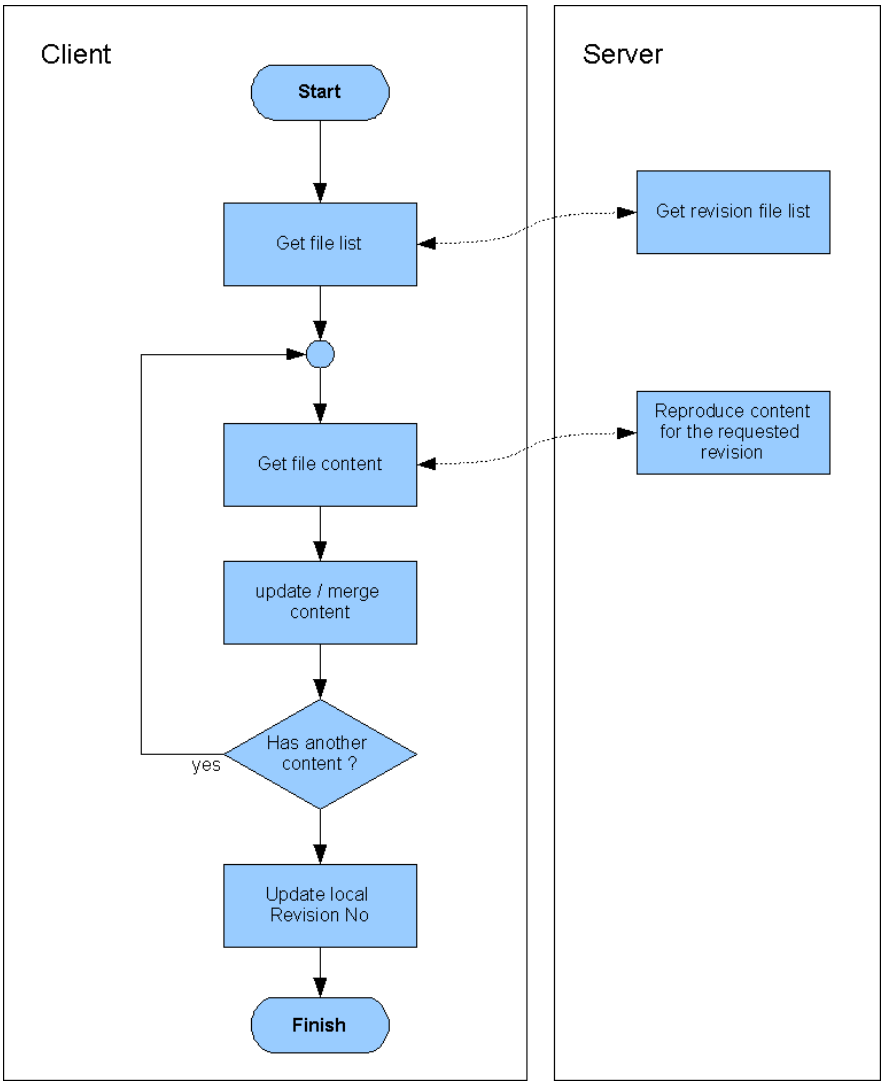


Figure 4.5 Check Out

4.4.5 Operation

This section describes major operations by sequence diagrams showing interaction between modules from the client application through the repository database. The operations include update, check out, commit, add, and delete.

1. Update

The update operation is used for updating copies in a local machine. Files of the requested revision are downloaded from the repository and merged with the local copies. See Figure 4.6.

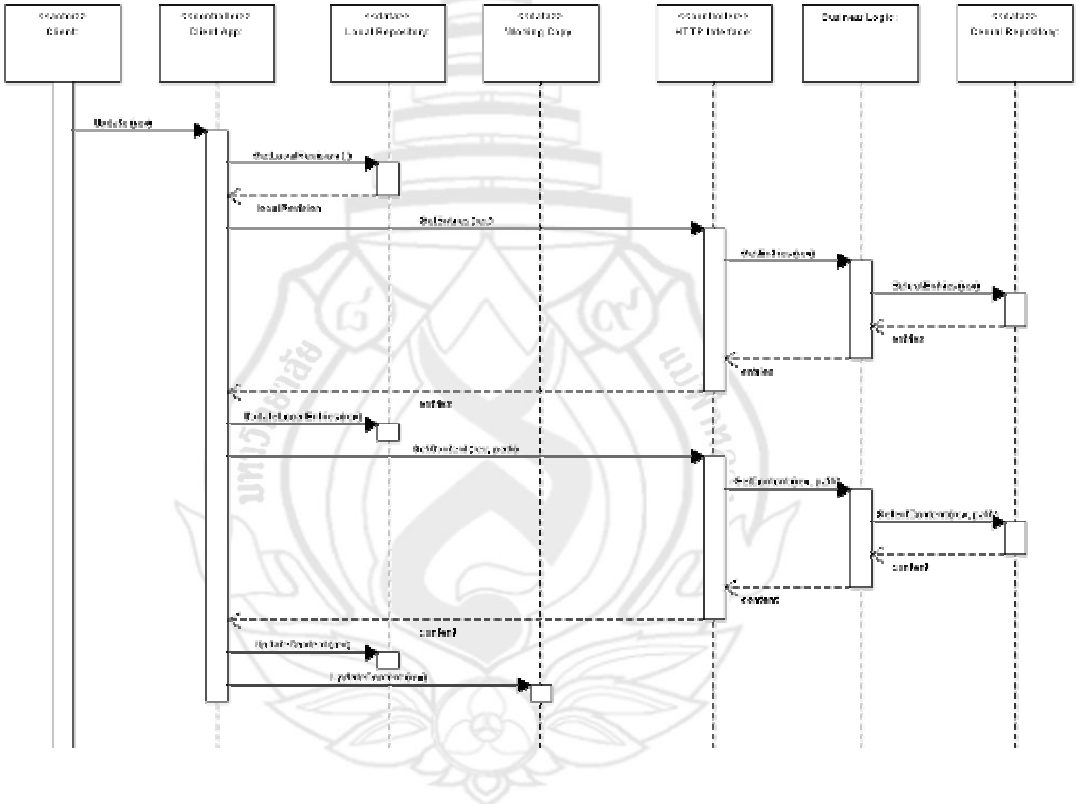


Figure 4.6 Update Sequence Diagram

2. Check Out

The check out operation is used when a user starts working with the system on his local machine. The system creates special directories for storing local repository meta-data and executes the update operation to download files. See Figure 4.7.

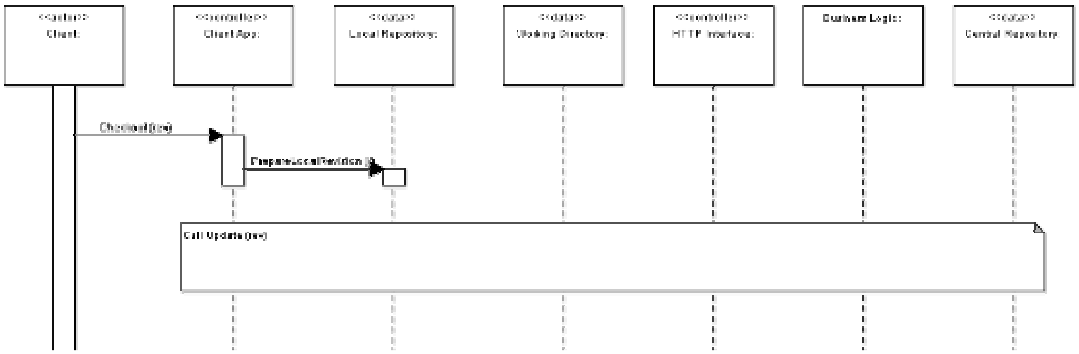


Figure 4.7 Check Out Diagram

3. Commit

The commit operation is used to check in changes into the repository and create a new revision. The system sends all changes to the repository server. Once they have been successfully sent, the system creates a new revision. See Figure 4.8.

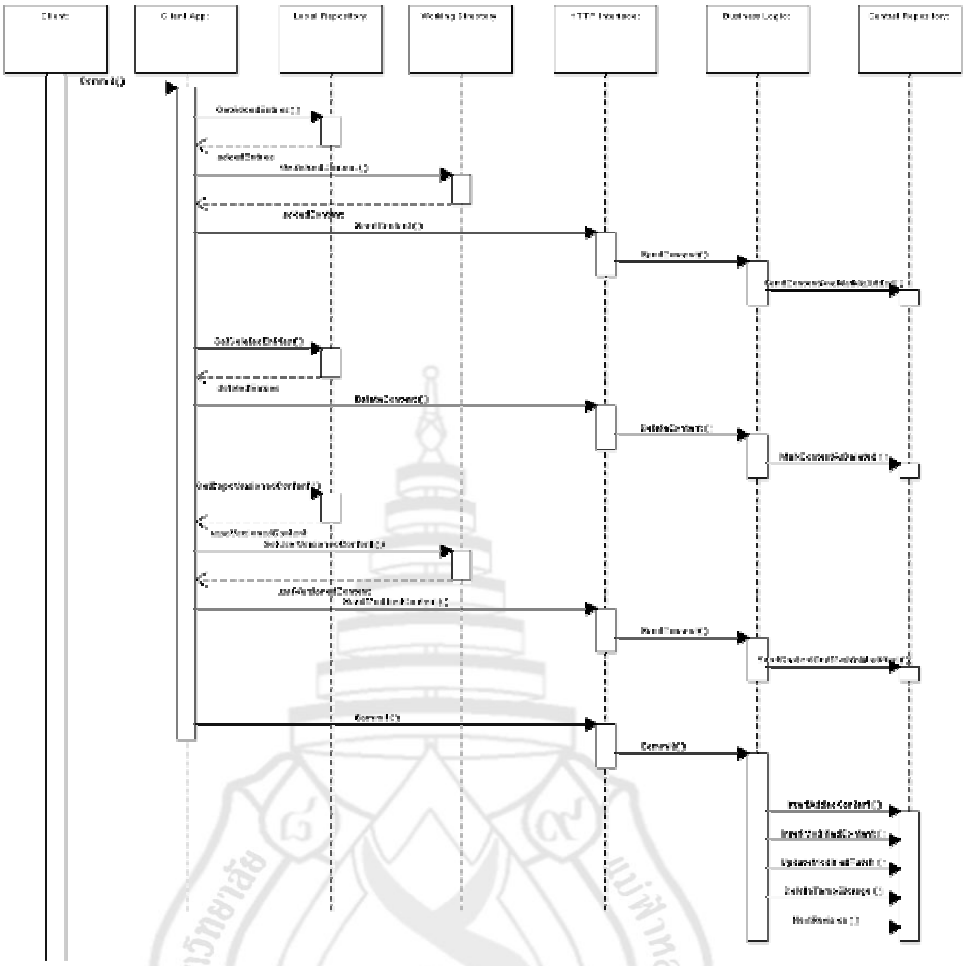


Figure 4.8 Commit Sequence Diagram

4. Add

The add operation is used to add non-version-controlled files into the repository. The files are not actually added until the commit operation is executed. See Figure 4.9.

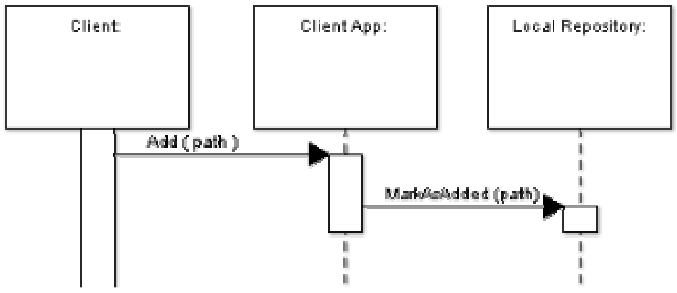


Figure 4.9 Add Sequence Diagram

5. Delete

The delete operation is used to remove version-controlled files from the repository. The files are not actually removed until the commit operation is executed. See Figure 4.10.

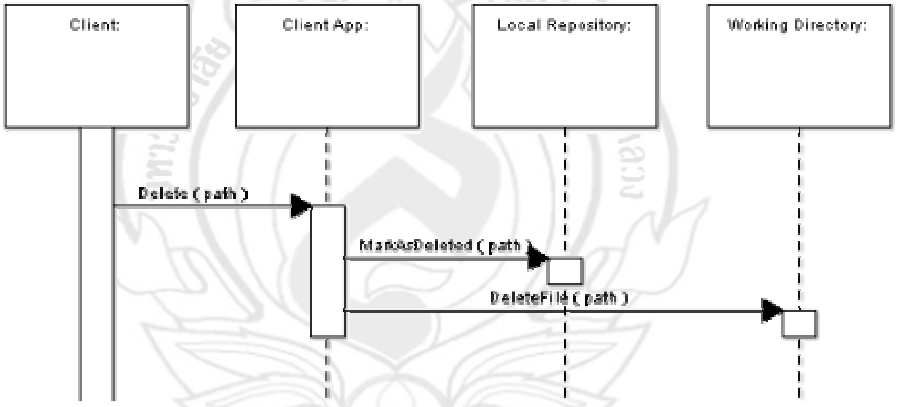


Figure 4.10 Delete Sequence Diagram

4.4.6 HTTP Interface

HTTP interface contains PHP script files to interact with the client application. Each file is responsible for a specific task. All HTTP interface files link to core logic with PHP function calls (See Figure 4.11).

Each HTTP Interface file handles different HTTP methods depending on an input data size. To handle a small amount of data, the GET method is used. To handle a large amount of data, the POST method is used. See Table 4.1.

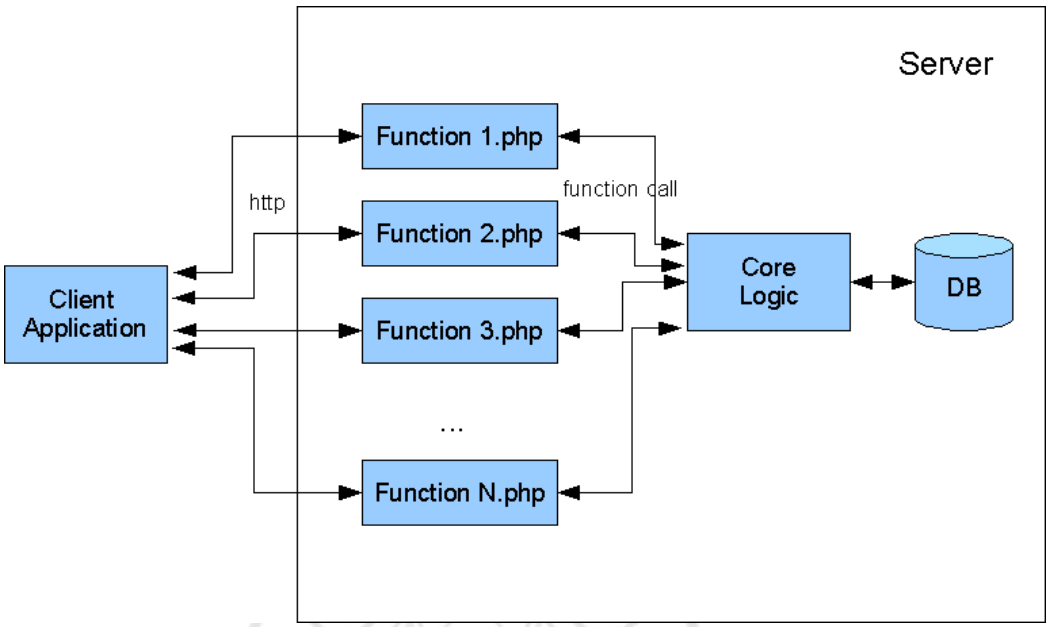


Figure 4.11 HTTP Interface Diagram

Table 4.1 HTTP Handler Design

PHP File Name	Description	Method	Input	Output
getheadrevision.php	Get head revision of the specified repository	GET	Repository ID	Head revision no
listfiles.php	List all files in the specified revision	GET	Repository ID, Revision	A list of file entries
sendcontent.php	Send content to server	POST	Repository ID, Revision, Path, Content, ContentType, User ID	-
deletecontent.php	Mark a file in the specified repository as deleted	GET	Repository ID, Revision, Path, User ID	-
commit.php	Commit as a new revision	GET	Repository ID, User ID	-
filecontent.php	Read file content of the specified path	GET	Repository ID, Revision, Path	File content

4.4.7 Algorithm

This section describes some algorithms used in two major operations: Check In and Check out.

To check in a modification, the system invokes several commands of *Send Content* and *Send Delete Tag*. The command's messages are sent to the server. Consequently, the server generates a transaction for each command message stored in a temp database table (Temp storage). When all messages have been successfully sent, the system executes *Commit* command to actually update the actual revision database table (Permanent storage). See Figure 4.12.

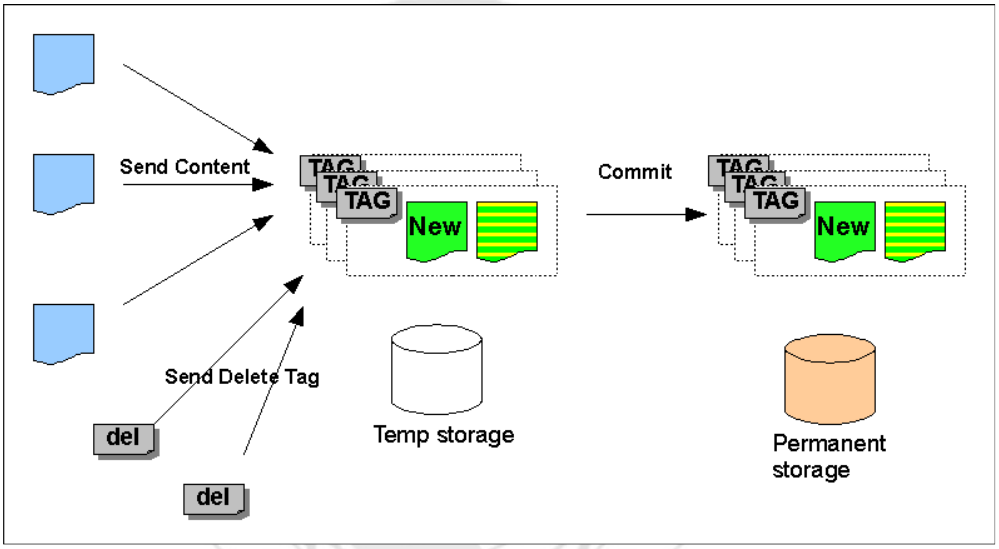


Figure 4.12 Check In Sequence Diagram

1. Send Content

Send Content algorithm is used when the client application is about to send new or modified content to the server application. The server application starts with accepting the new content. It then checks whether the file is new or existing. If the file is a new file, the content is stored in the temp storage as “add” transaction. Otherwise, the content is compared to the existing file's content generating deltified content. The deltified content is used for replacing the existing

file's content when *Commit* command is executed. Both the new content and the deltified content are stored in the temp storage as “mod” transaction. See Figure 4.13.

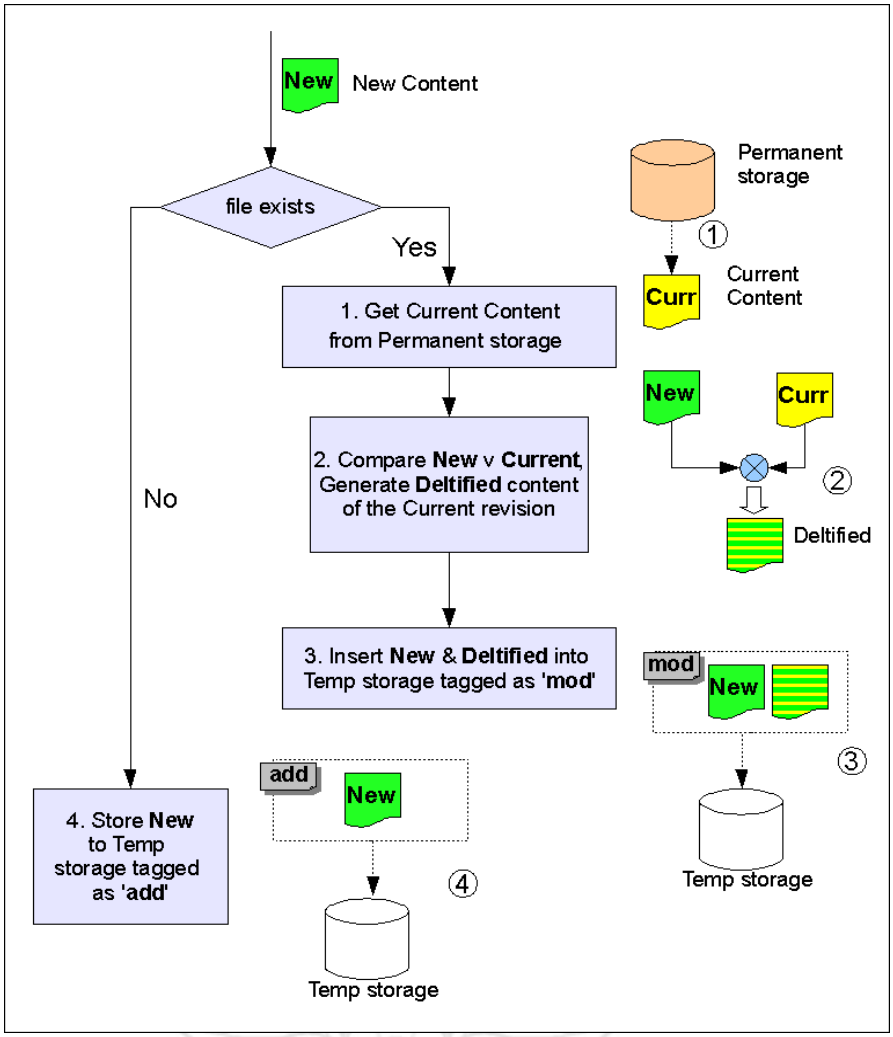


Figure 4.13 Send Content Algorithm (Server)

2. Delete Content

Send Content algorithm is used when the client application is about to delete file in a repository. The algorithm generates a “del” transaction and stores it in the temp storage. See Figure 4.14.

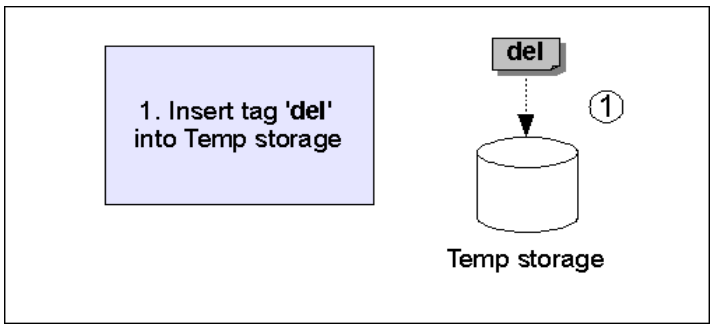


Figure 4.14 Delete Content Algorithm

3. Commit

Commit algorithm is used for actually modifying a repository and creating a new revision. The algorithm starts with reading transactions in the temp storage. Consequently, it processes each transaction depending on how it is marked: “add”, “mod”, or “del”. If it is marked as “add” or “del”, the algorithm inserts the transaction with its content into the permanent storage as a new revision. Otherwise, the “mod” transaction and its content are inserted into the permanent storage as a new revision while its deltified content is replaced to its previous revision. See Figure 4.15. As a result, the latest revision always stores full text while previous revisions store deltified contents (See Figure 4.16).

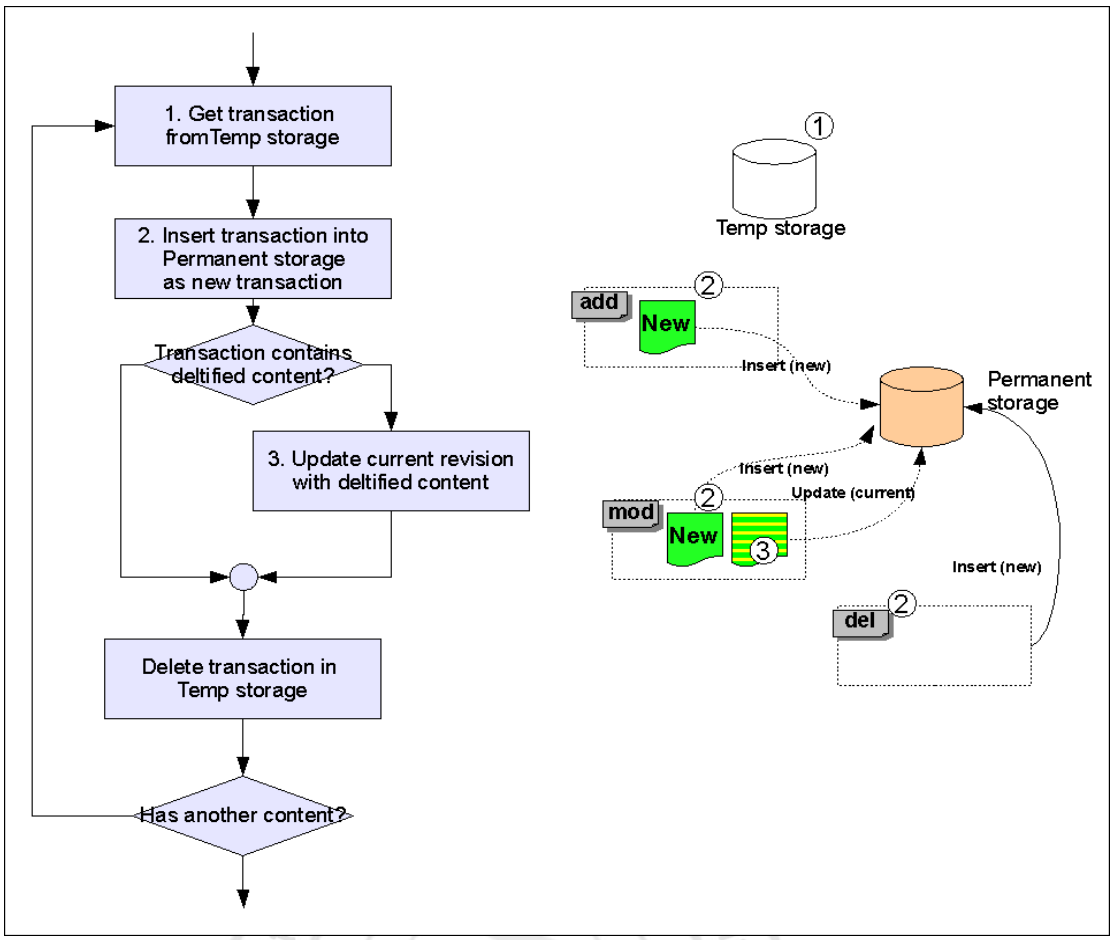


Figure 4.15 Commit Algorithm (server)

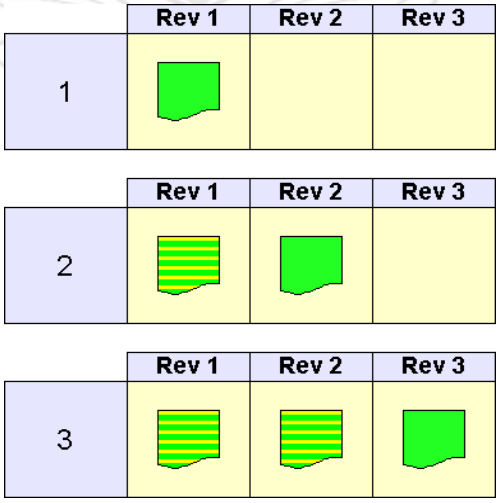


Figure 4.16 Revisions Stored in Repository

4. Check out

Check out operation is requested when the client application needs content at a particular revision. Once the operation has been requested, the server application reproduces (or renders) content at the requested revision. See Figure 4.17.

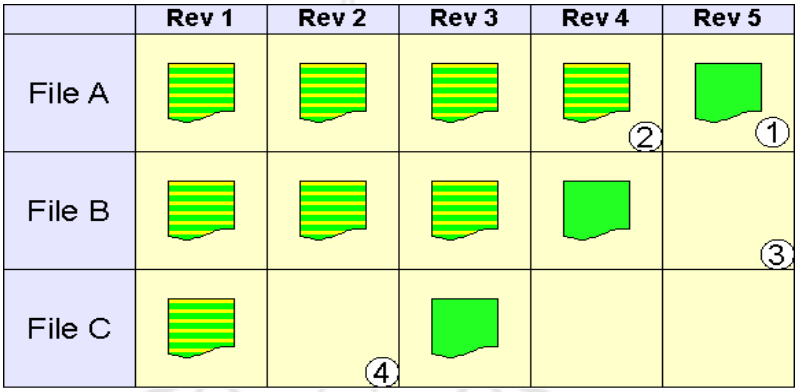


Figure 4.17 Reproducing Content at Particular Revisions

As can be seen in Figure 4.17, contents at some revisions may be stored as full text, deltified, or not stored at all.

In case of the requested transaction contains full text or deltified content as in (1) and (2), the algorithm starts with reading the full text as its current content. It then repeatedly steps back to the requested revision. While stepping back, it reads each delta and uses it to modify the current content. When the looping is finished, the current content is the final result. In case (1), the requested revision is 5 as well as the recently-added transaction. Thus, no looping and modification occur. Only full text is read. In case (2), the requested revision is 4 and the recently added transaction is 5. The algorithm reads full text from revision 5 and loops back through revision 4 to reproduce the final content.

In case of the requested transaction does not contain full text or deltified content as in (3) and (4), the algorithm performs similar to the previous algorithm, except that it loops

from the transaction containing full text through the previously-modified transaction. In case (3), it reads the full text from revision 4. In case (4), it starts looping from revision 3 though revision 1.

4.4.8 Database Design

This project is primarily designed to work with MySQL, the relational database management system.

In the database design, there are five major entities: Repository, User, Revision, Revision Detail, and Modification Session (See Figure 4.18 for ER diagram). Revision and Revision Detail entities together support submitting multiple modifications as one commit. Revision entity contains one record per one commit while Revision Detail entities contain multiple modification data.

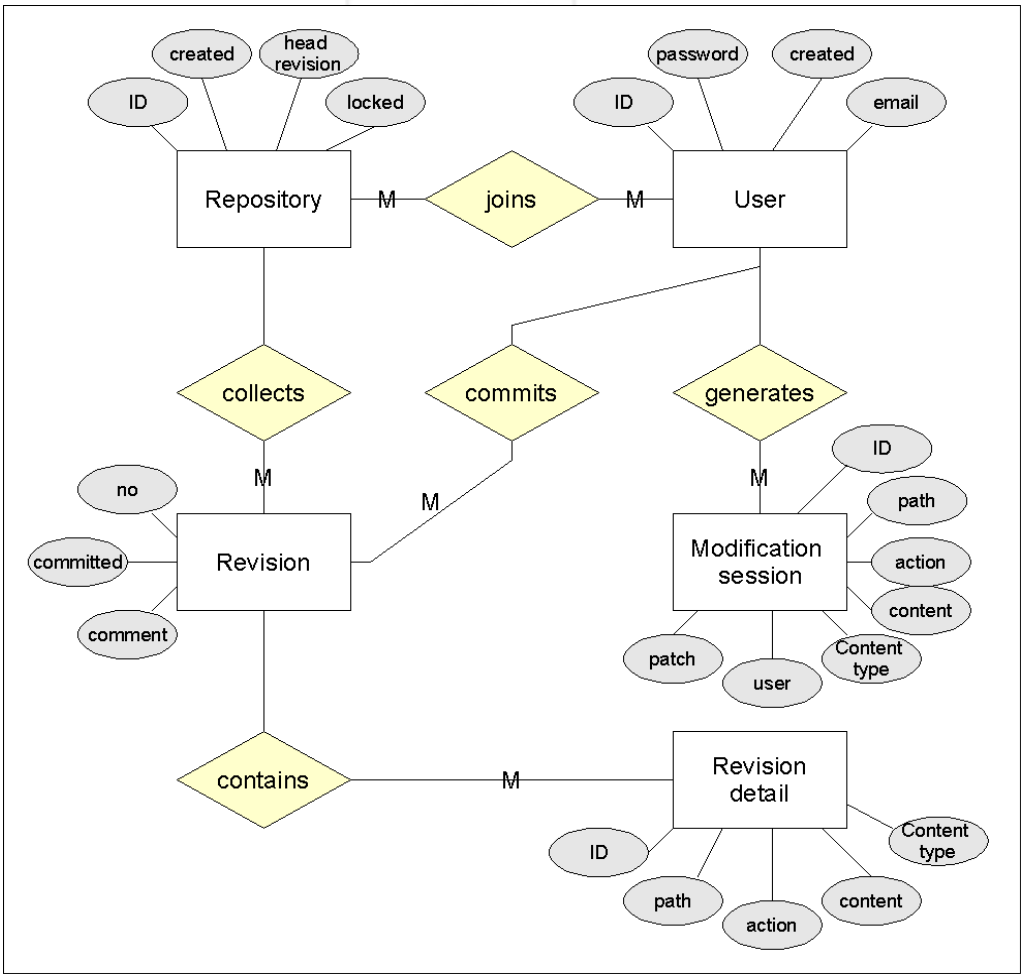


Figure 4.18 Database Design, ER Diagram

User entity stores user information including user ID, password, email, and a date when a user is created. One user can join multiple repositories or none. User records are managed by system administrators.

Repository entity stores repository information including ID, head revision, lock status, and a date when a repository is created. One repository can serve multiple users. Repository records are managed by system administrators.

Revision entity stores master revision information including revision number, a date when a revision is committed, and a comment. One revision contains multiple revision details.

Revision Detail entity stores detailed revision information including ID, path, content and its type, and action.

Table 4.2 Database Permission Design

Entity	Administrator			User		
	Insert	Update	Delete	Insert	Update	Delete
User	✓	✓	✓			
Repository	✓	✓	✓		✓	
Revision			✓	✓		
Revision Detail			✓	✓	✓	
Modification Session				✓		✓

4.4.9 User Interface Design

In this project, user interface consists of two applications; the client application and the web application.

1. The client application

The client application is a console application. The application prompts for a user command (See Figure 4.19). The user types a name of command and press Enter button to commit the command (See Figure 4.20). Then, the system processes and displays output on the screen. When the processing is completed, the application prompts for the next command (See Figure 4.21).

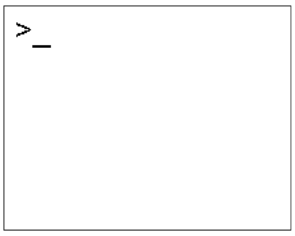


Figure 4.19 Console Application 1

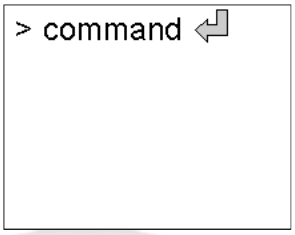


Figure 4.20 Console Application 2

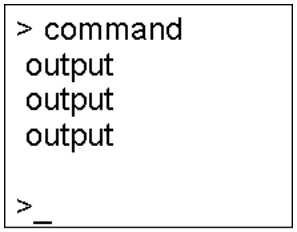


Figure 4.21 Console Application 3

Table 4.3 Command Design

Command	Description	Input parameters	Output
login	Log in to the system	User ID, Password	Authentication result
Logout	Log out of the system	-	-
List	list all files in the current directory	-	List of file entries
Add	add a file into the repository	File name to add	-
Delete	delete file in the repository	File name to delete	-
Update	update working copies with the latest revision	Revision to update to (Optional)	-
Commit	commit changes into the repository	-	-
Checkout	create a version controlled environment in the client machine	Server Address, User ID, Password, Repository ID	-

2. Web Application

Web Application enables an administrator manages repositories including users

in the system. The authorized user can view his repository information, such as revision files, revisions comparison, revision log, etc.

Web pages include;

1) Log In Page – A user logs in to the system.
2) User's Account Page – This page displays user's information and summarized repository information.
3) Browse Repository Page – This page displays files in repository. A user can navigates each revision.
4) Revision Comparison Page – This page displays differences between two revisions.
5) View Log Page – This page displays committed revision log.
6) Repository Management Page – An administrator can create/remove repositories as well as add/remove users in the repositories.
7) User Management Page – An administrator can create/modify/delete users in the system.

3. Web Page Layout

All web pages have the same layout for consistent look (See Figure 4.22). At the top of every page, there is a panel showing log in status denoted as (1). The panel displays a welcome message to current user name logging in to the system. At the left of the page, there is a panel used for page navigation denoted as (2). The panel contains links to other pages. Page content denoted as (3) is displayed at the center of the page. Each web page has different page content depending on the page subject.

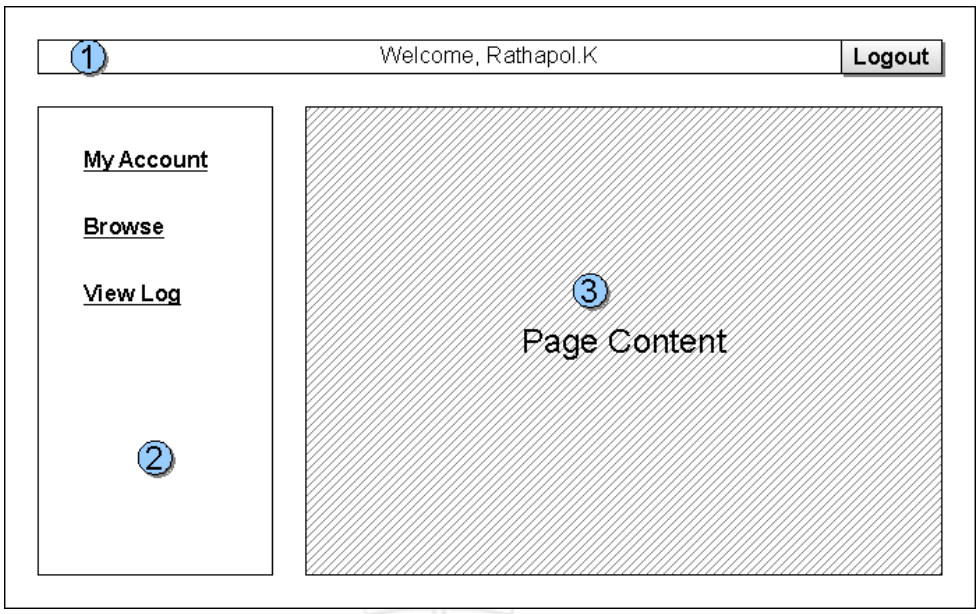


Figure 4.22 Web Page Layout

4. Log In Status Panel

Login status panel shows a welcome message and the name of user logging in. The panel provides Logout button to allow the user to logout of the system. When the user presses the button, the system clears the current user's session data and redirects to Log In page (See Figure 4.23). Note, the [user_name] will be replaced with the user name who has logged in.

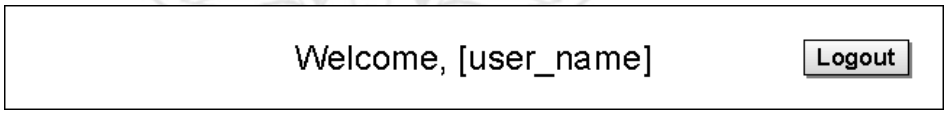


Figure 4.23 Log In Status Panel

5. Page Navigation Panel

Page navigation panel contains links to My Account page, Browse page, and View Log page (See Figure 4.24). In the administrator's application, page navigation panel contains links to Create User page, Accounts page, Create Repository page and Repositories page (See Figure 4.25).

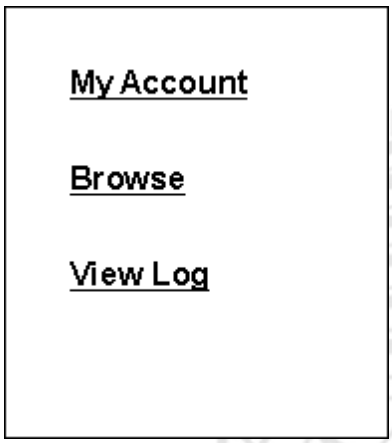


Figure 4.241 Page Navigation Panel

Figure 4.25 Page Navigation Panel (for Administrator)

6. Web Pages for Repository Users

1) Log In Page

Log In page is used for verifying users getting access to the system. The page contains a login form composed of user (1), password (2), and repository (3) text boxes and Log In (4) and Reset (5) buttons (See Figure 4.26). User text box enables the user to input his/her user name. Password text box enables the user to input his/her password. The input characters in the text box are masked, usually by a series of asterisks on most systems. Repository text box enables the user to specify a repository to work with. Log In button is for submitting and verifying the inputs. Reset button is used to clear data in the text boxes. When the inputs are verified, the system redirects to My Account page. Otherwise, the system keeps displaying this page showing an error message.

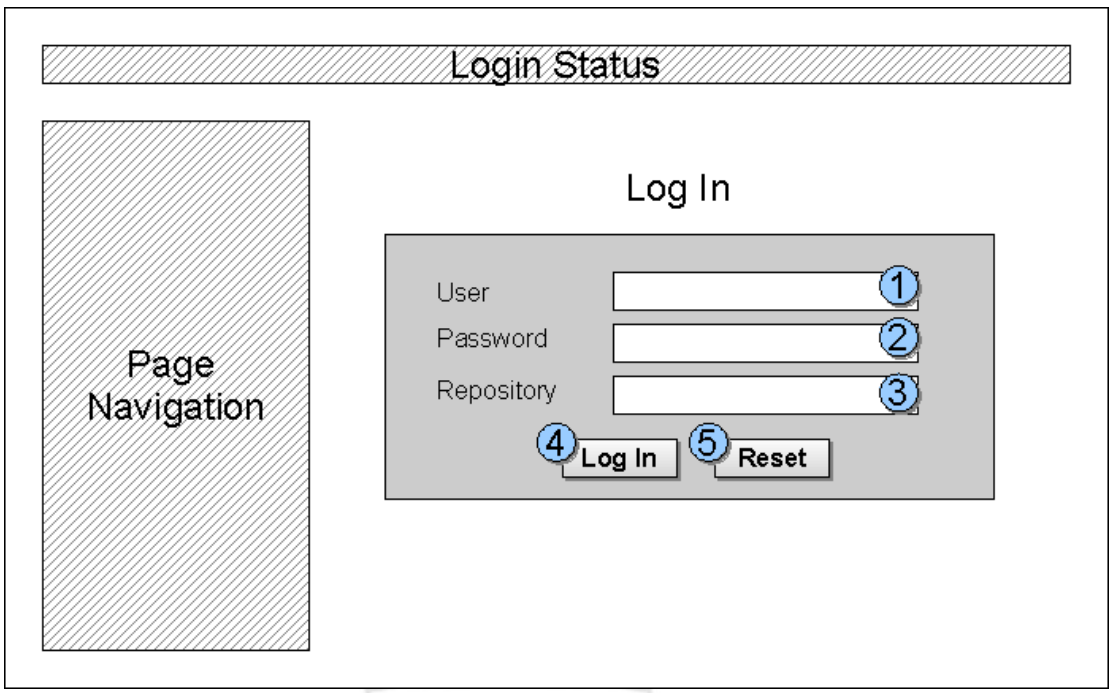


Figure 4.26 Log In Page Design.

2) My Account Page

My Account page shows up once a user has logged in (See Figure 4.27). This page displays User Profile panel (1), Head Revision Information panel (2), and Head Revision Log panel (3).

User Profile panel displays basic information of the registered user. This project provides only basic information, such as user name and user e-mail. It can be expanded later if required.

Head Revision Information panel displays information of latest-submitted revision. The information includes the active repository, the latest revision number, date/time when the revision has been committed, a user name of who has committed, and an additional comment from the committing user.

Head Revision Log panel displays a change log of the latest revision (See

Figure 4.28). The change log is displayed as a table of change items. The first column represents change action: add, modify, or delete. The second column represents files affected. The last column represents the content type of the affected file.

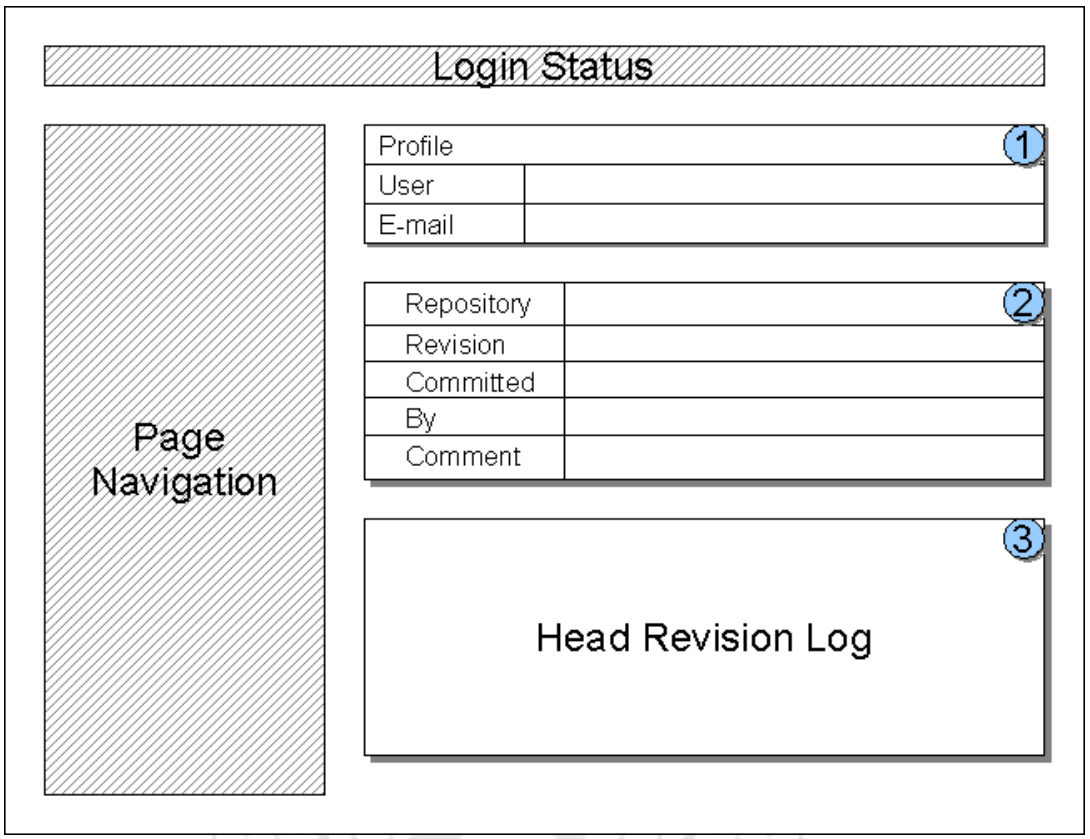


Figure 4.27 My Account Page Design

	File	Content Type
Add	File 1	Text/Plain
Add	File 2	Text/Plain
Modify	File 3	Text/Plain
Modify	File 4	Text/Plain
Delete	File 5	Text/Plain

Figure 4.28 Head Revision Log Panel Design

3) Browse Page

Browse page enables a user to view files in the specified repository. This page is composed of revision navigation panel (1), revision information panel (2), revision entry list panel (3), and revision log panel (4) (See Figure 4.29).

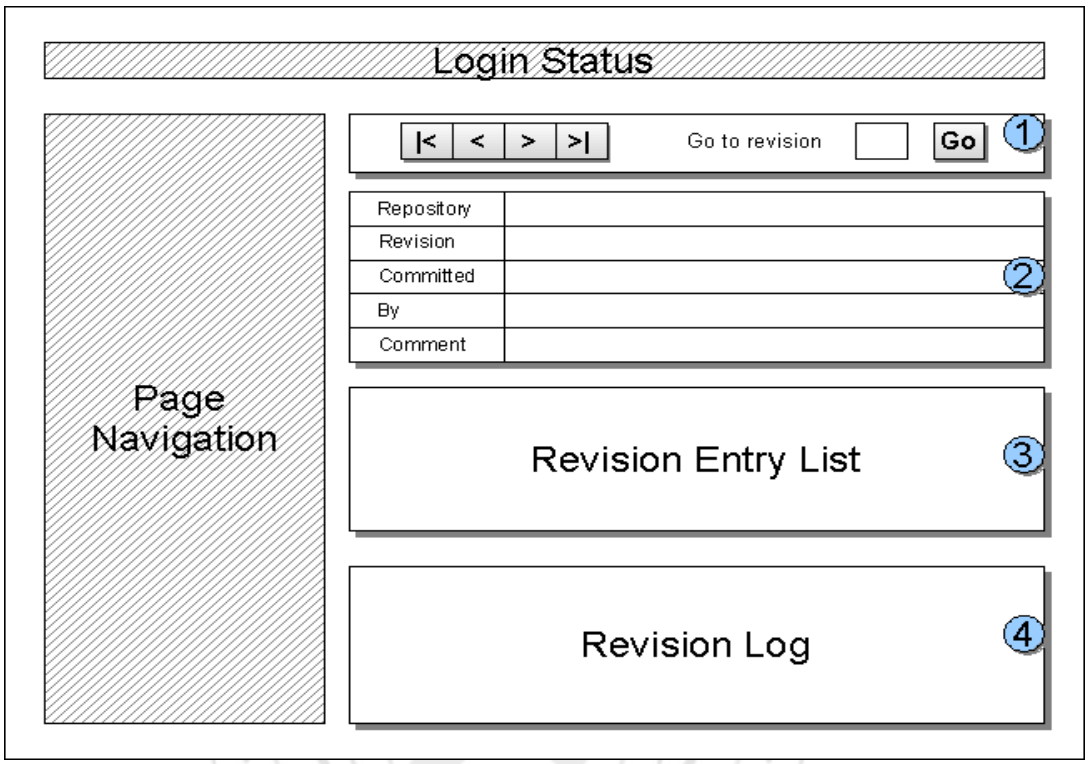


Figure 4.292 Browse Page Design

Revision navigation panel is located at the top of the page content area. It enables the user to navigate through each revision (See Figure 4.30). The user can navigate to the first revision, the previous revision, the next revision and the latest revision by pressing First Revision button (1), Previous Revision button (2), Next Revision button (3), and Latest Revision button (4), respectively. Go button (6) enables the user to switch to a revision specified in the nearby text box (5).

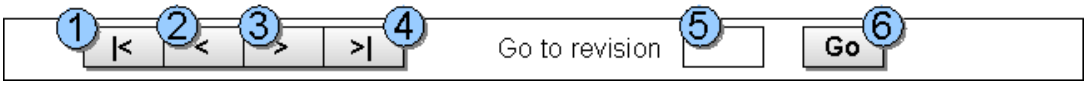


Figure 4.30 Revision Navigation Panel Design

The revision entry list panel is used to display file entries of the selected revision. It shows file names and content types. It provides a compare button enabling a user to compare the selected file against the same file of different revision. See Figure 4.31.

	File	Content Type
◉	File 1	Text/Plain
○	File 2	Text/Plain
○	File 3	Text/Plain
Compare to revision	Compare	

Figure 4.313 Revision Entry List Panel Design

The revision log panel is used to display modification list affecting to the selected revision. For each log entry, the first column shows an action type. The next column shows an affected file name. The last column shows the content type of the file. See Figure 4.32.

	File	Content Type
Add	File 1	Text/Plain
Add	File 2	Text/Plain
Modify	File 3	Text/Plain
Modify	File 4	Text/Plain
Delete	File 5	Text/Plain

Figure 4.32 Revision Log Panel Design

4) View Log Page

The view log page lists all revision log in one page ordered by revision number (See Figure 4.33). Each revision log contains master revision data (1) and detailed data (2). The master data includes repository ID, revision number, commit date, committing user, and a comment by the user. The detailed data is displayed similar to the revision log panel.

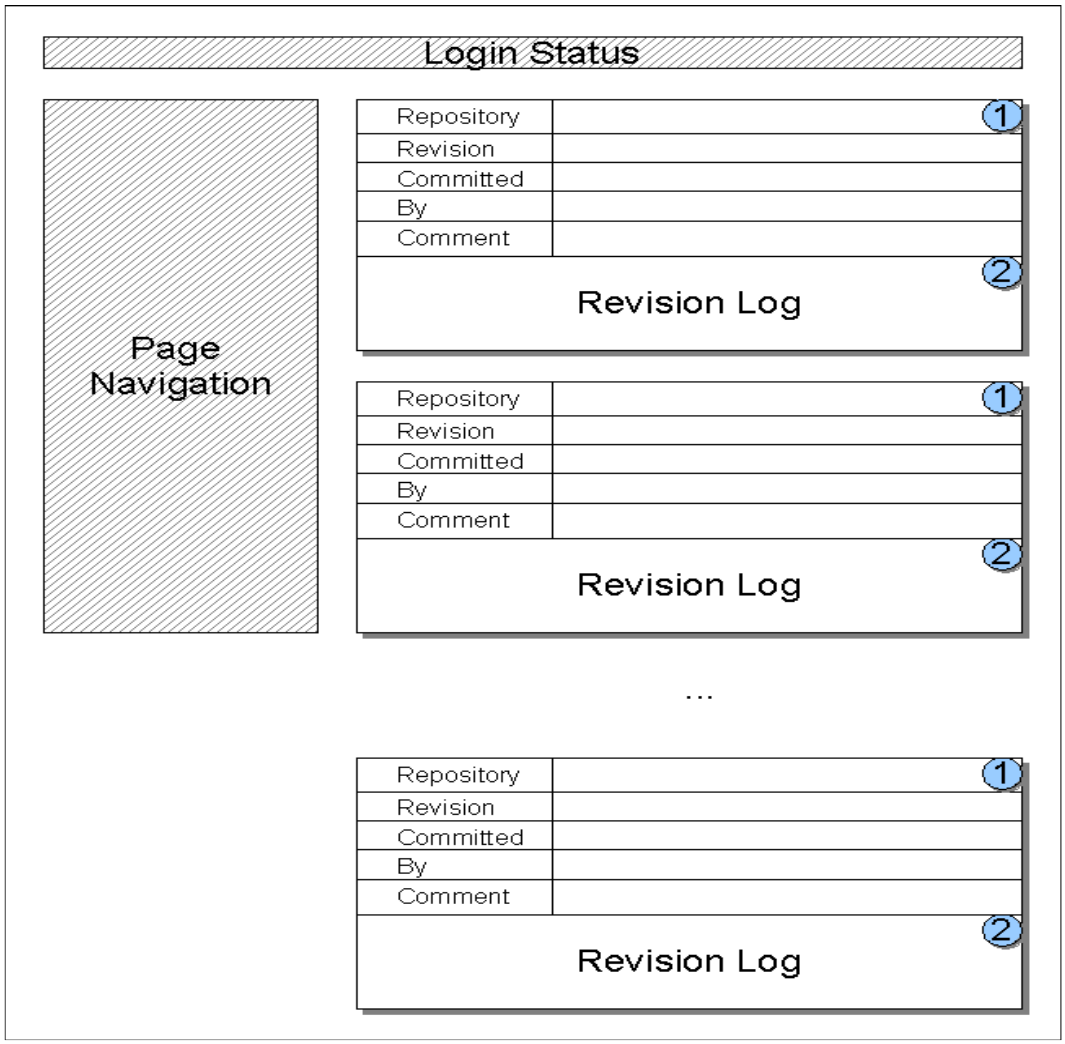


Figure 4.33 View Log Page Design

5) Create User Page

The administrator's Create User page is use to create a new user in the system. The required fields are user ID, password, password confirmation, and e-mail. Create button is used to submit data in the form. See Figure 4.34.

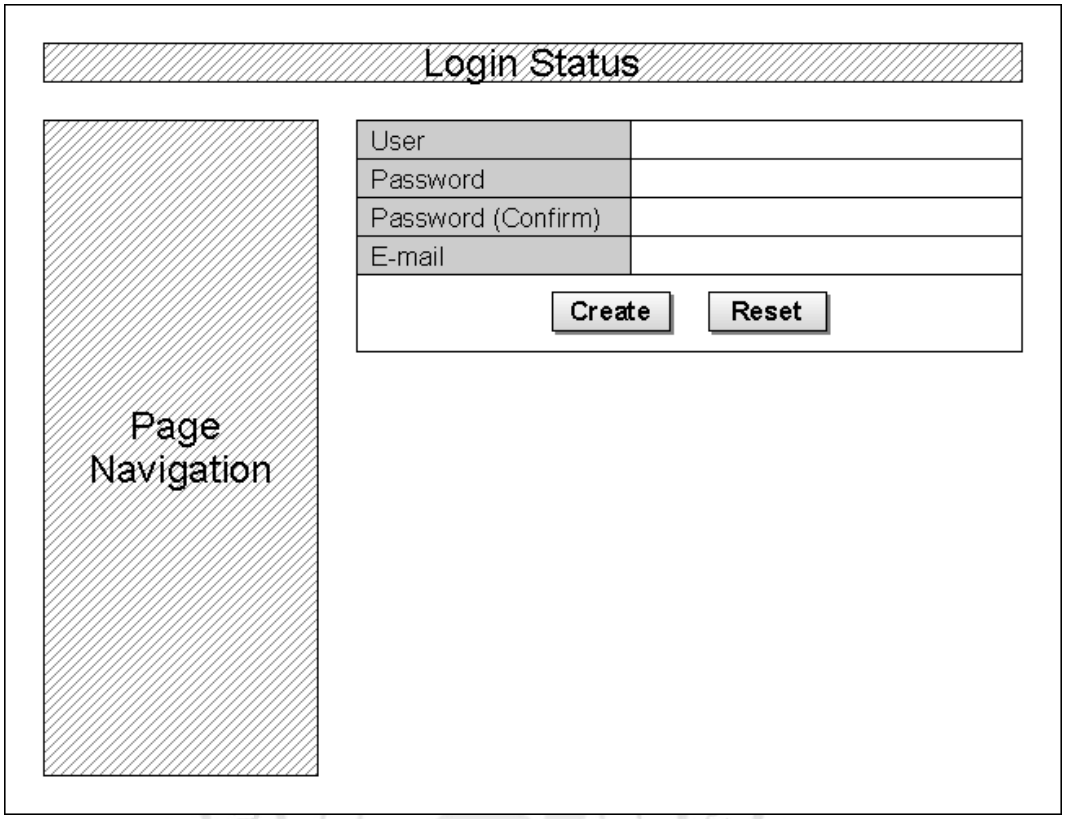


Figure 4.34 Create User Page Design

6) Account List Page Design

The Account Management page is used for displaying all users created. An administrator can select user names and delete them with in this page. See Figure 4.35.

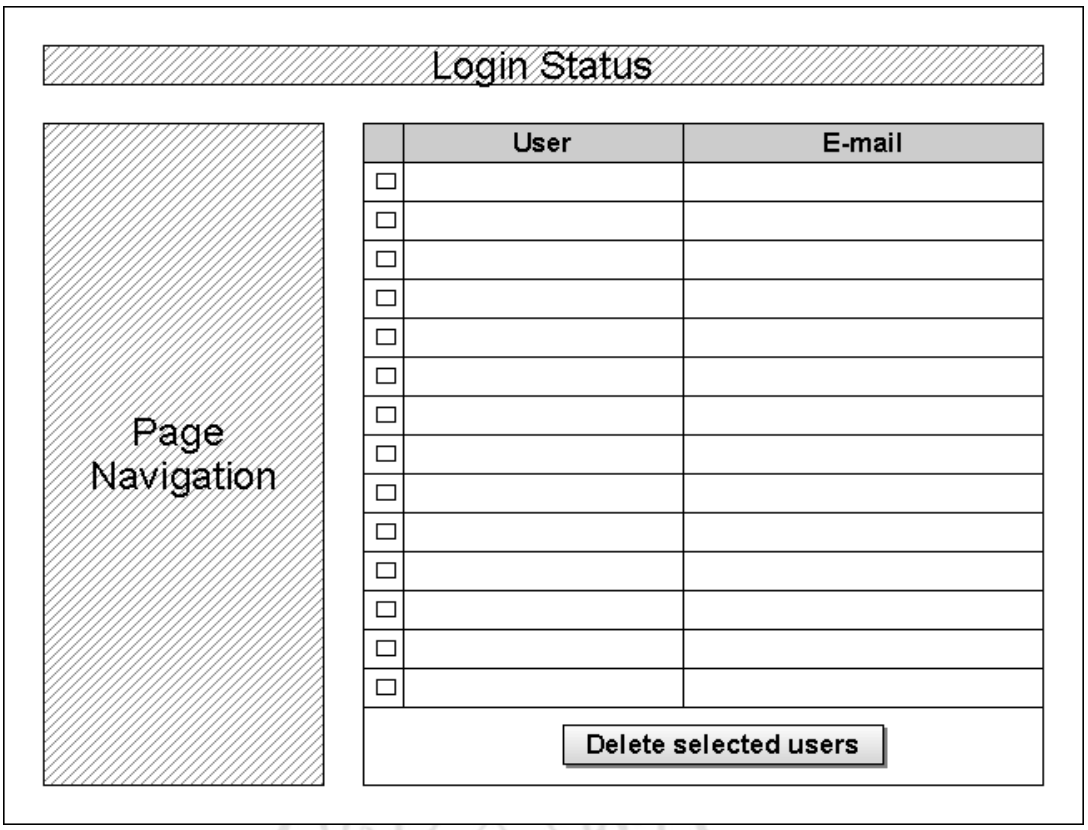


Figure 4.35 Account Management Page Design

7) Create New Repository Page

The Create New Repository page is used to create a new repository in the system. An administrator creates a repository by inputting a repository ID and an optional comment. See Figure 4.36.

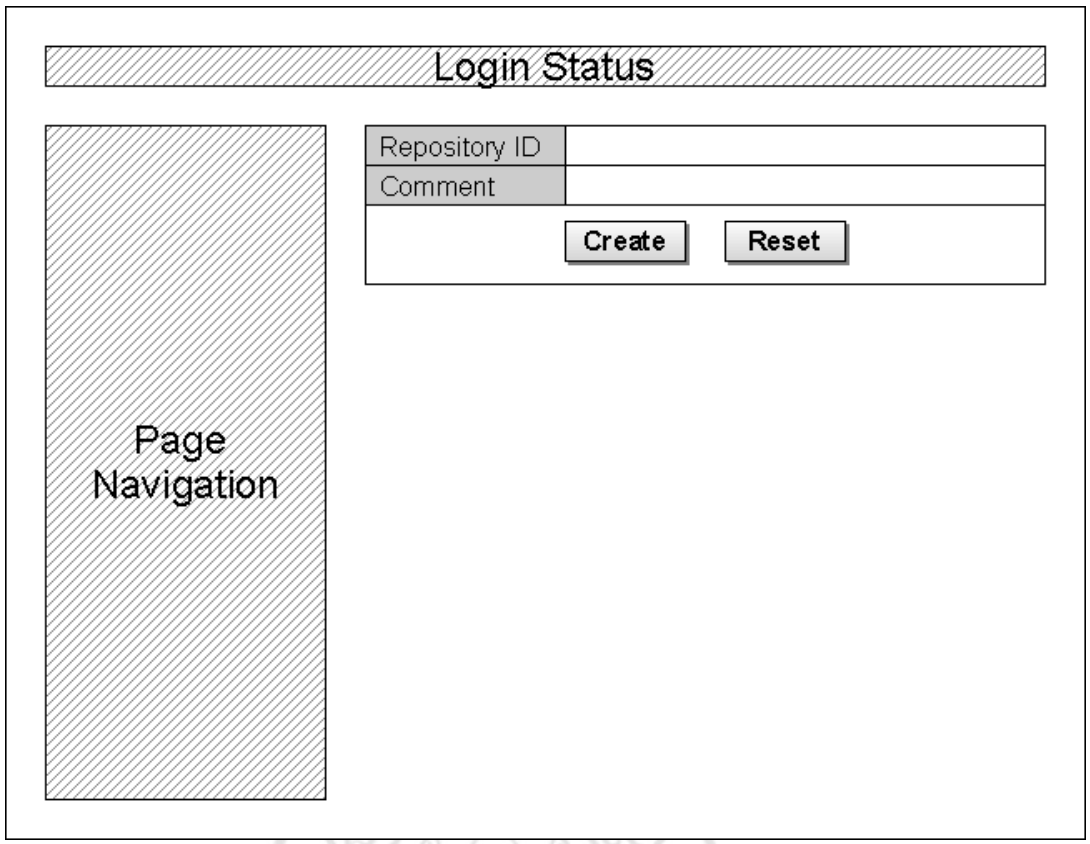


Figure 4.36 Create New Repository Page Design

8) Repository Management Page

The repository management page is used to manage repositories and users in the repositories. All repositories are listed. Each repository entry provides buttons to add users, remove users, and delete the repository. See Figure 4.37.

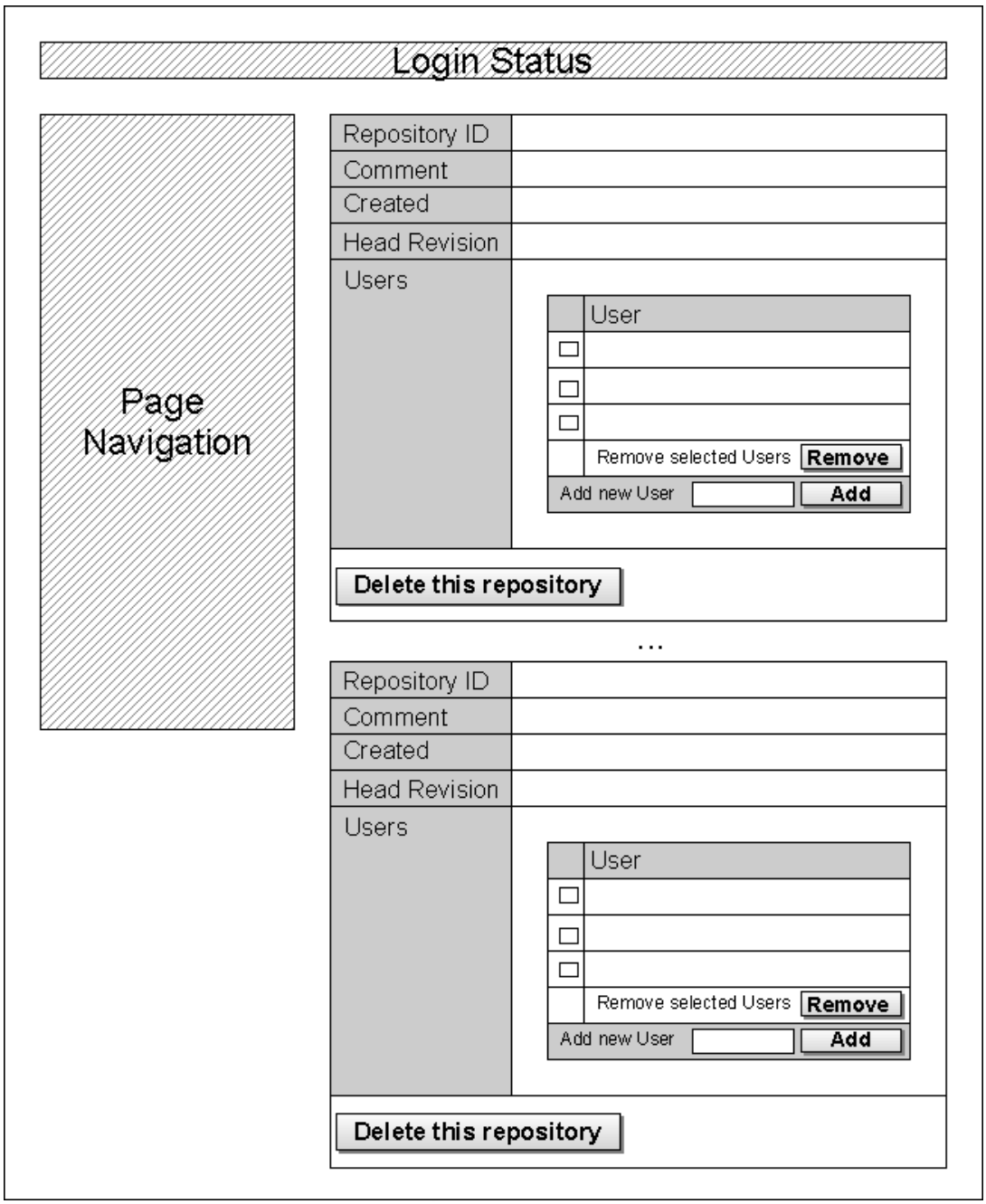


Figure 4.37 Repository Management Page Design

CHAPTER V

SYSTEM FUNCTIONALITY

5.1 Introduction

This chapter covers a description of architecture of the system implemented and its functionalities. It also covers test plans and test results of the system.

System architecture of the system is client/server. Server and client machines communicate together via the HTTP protocol. The server application manages central repositories and does version control tasks, such as versioning and supporting concurrency. It is implemented as PHP scripts running on web server applications like apache or IIS. Repository data are stored in MySQL database. Delta compression is used to reduce content size. The client application provides user interface to work with the repository server as well as manages local copies.

This project applies several tests on the system. The tests include algorithm test, performance test, functional test, and user satisfaction test. Servers used for the tests are both local server and public web hosting server.

After applying all tests, one test case for Diff algorithm fails. The algorithm is ported from Java source code. Both ports generate the same results including the result from the failing test. The performance test result shows that the system can work reasonably in normal use. The system will be slow down if content of the earlier revision is requested when there are amounts of revisions committed. This may cause timeout on the client.

5.2 System architecture

System architecture of the system is client/server. The server application manages central repositories and does version control tasks, such as versioning and supporting concurrency. MySQL is used to store repository information and content. The client application provides user interface to work with the repository server as well as manages local copies. The server and client applications communicate together via the HTTP protocol. See Figure 5.1.

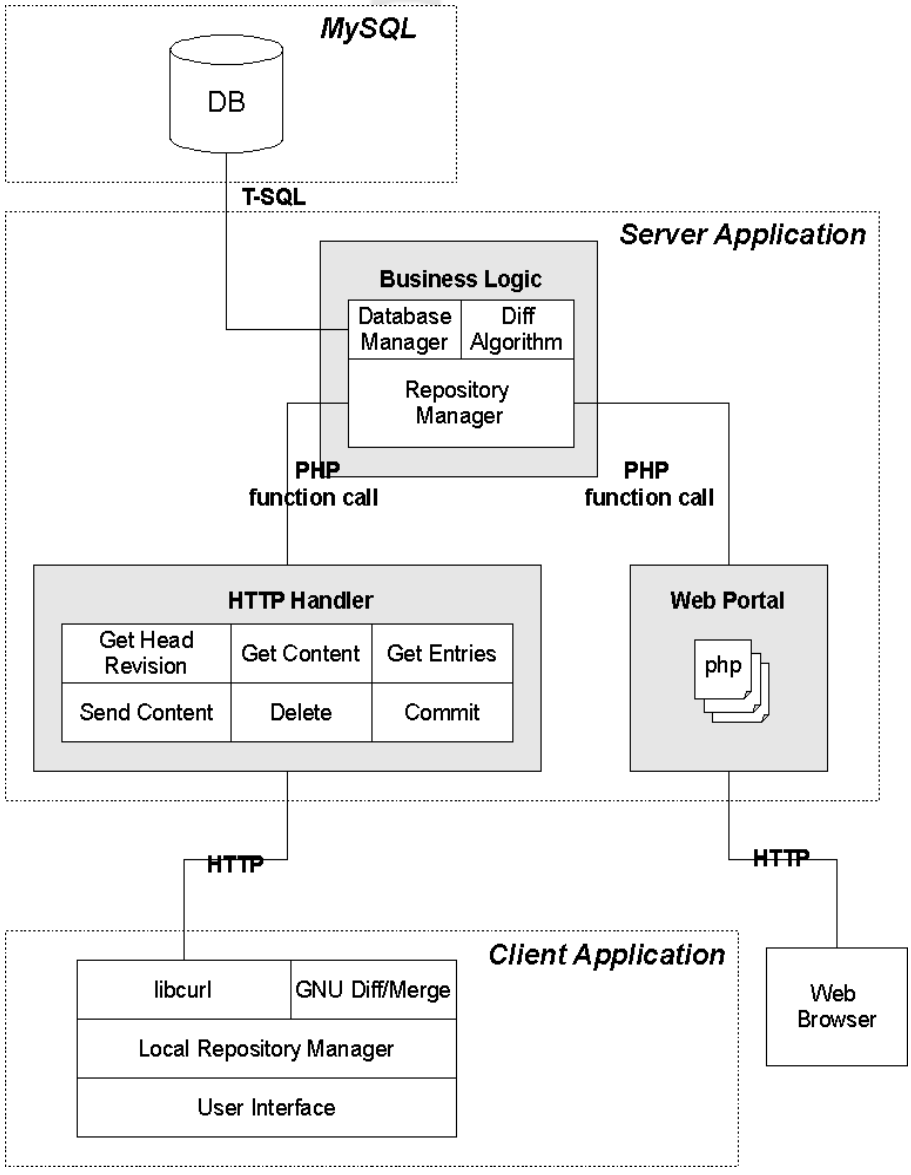


Figure 5.1 System Architecture

5.2.1 Server Architecture

The server application manages central repositories and does version control tasks, such as versioning and supporting concurrency. It is implemented as PHP scripts running on web server applications like apache or IIS. It provides web pages for monitoring and administrating. Repository data are stored in MySQL database. Delta compression is used to reduce content size. See Figure 5.2.

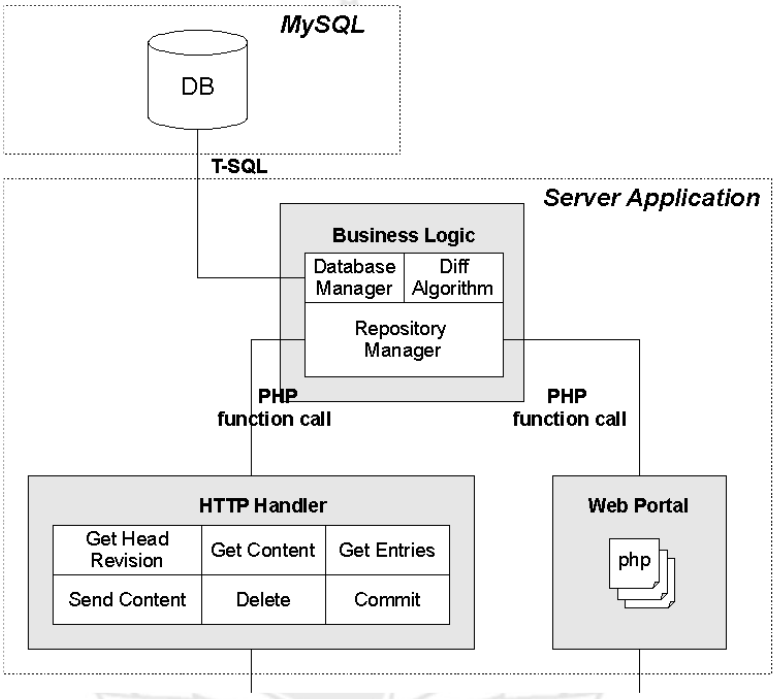


Figure 5.2 Server Architecture

The server application is composed of three modules: Core Module (business logic), HTTP Handler, and Web Portal.

1. Core Module

Core Module does all version control tasks including managing repositories and handling concurrency. It is composed of three submodules: Repository Manager, Database Manager, and Diff Algorithm.

Repository Manager is a central control of the core module. It contains all version control logics. It links directly to the other two submodules to utilize their functionalities.

Database Manager provides interfaces to work with MySQL. This submodule is designed as a single point of control for accessing the database.

Diff Algorithm provides functions for comparing text files as well as merging a text file with a patch. It is used to generate a delta (result of the comparing algorithm) rendered in unified format.

2. HTTP Handler

HTTP handler is used as an interface with the client application. It is composed of eight PHP files. Each file handles a different request. The request is passed to Core Module via PHP function to be processed. See Table 5.1.

Table 5.1 HTTP Handler Files

No	File name	Description
1	getheadrevision.php	Returns head revision number of the requested repository
2	login.php	A user logs in for working with the repository
3	permission.php	Returns a permission of the logged in user
4	listfiles.php	Lists all file entires of the requested revision
5	filecontent.php	Returns content of a file of the requested revision
6	sendcontent.php	Accepts content from the client application
7	deletecontent.php	Deletes file content in the requested repository
8	commit.php	Commits changes and creates a new revision

5.2.2 Web Portal

Web Portal contains several PHP web pages for monitoring and administrating the

system via web browsers. There are two sets of web portals: Repository User Web Portal and Administrator Web Portal.

1. Repository User Web Portal

Web portal provided for repository users is composed of four main web pages: Log In Page, User Account Page, Browse Page, and View Log Page.

Log In Page is used for verifying users. The page provides a form containing text boxes for inputting user name and password and a combo box for inputting repository to work with. See Figure 5.3.

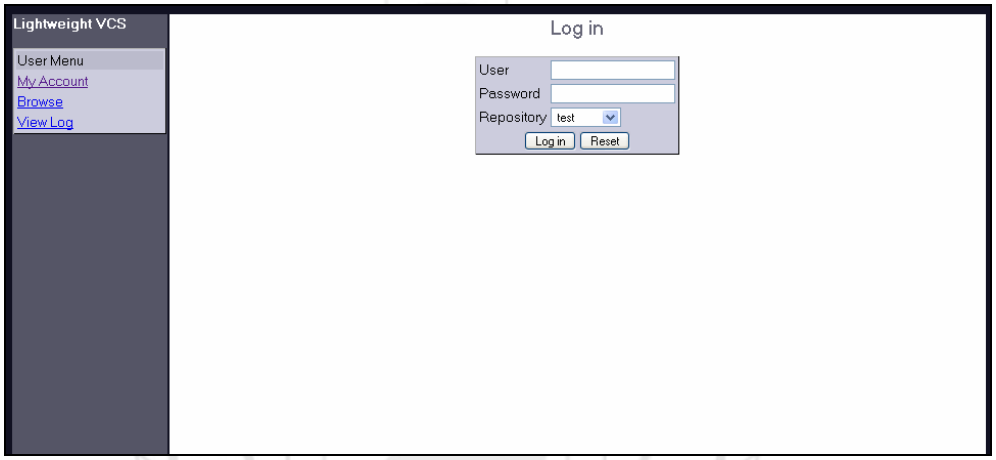


Figure 5.3 Log In Page

User Account Page is used for viewing current information of the user and summarized information of the repository such as head revision information and latest modifications. See Figure 5.4.

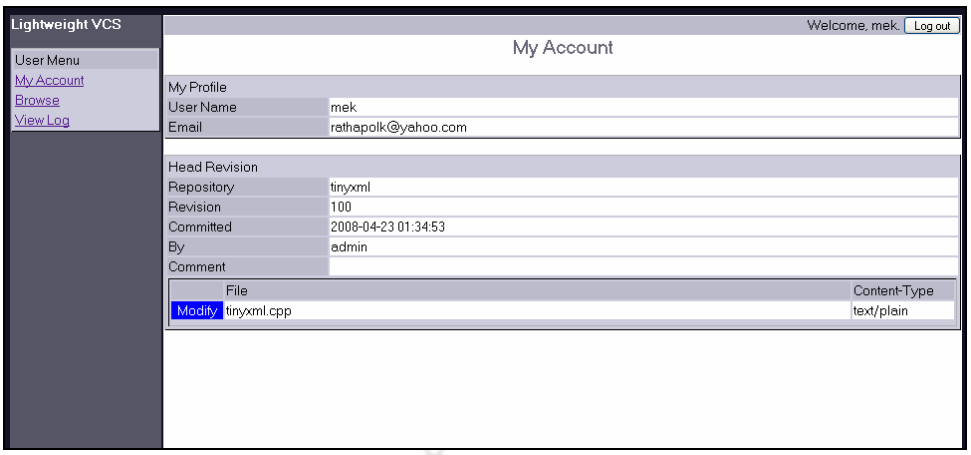


Figure 5.4 User Account Page

Browse Page is used for browsing files of every revision and modification log related to the revision (See Figure 5.5). It contains a navigation panel for switching to any revision. File content can be displayed by clicking at a file name. Once the file name has been click, a new window is populated displaying the full text content of the file of the current revision. Each entry has a content link used for viewing actual content in the database. This is useful for technical analysis and maintenance.

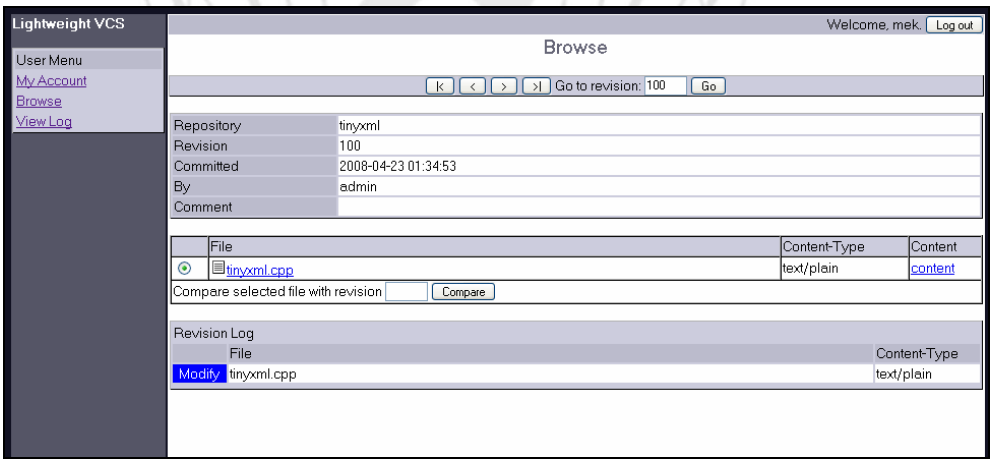


Figure 5.5 Browse Page

Two different revisions of a file can be compared by checking a radio button of associated with a desired file, inputting a desired revision nearby the Compare button and clicking the button. Consequently, a new window is populated displaying a comparison result. Added lines are highlighted in green. Deleted lines are highlighted in read. See Figure 5.6.

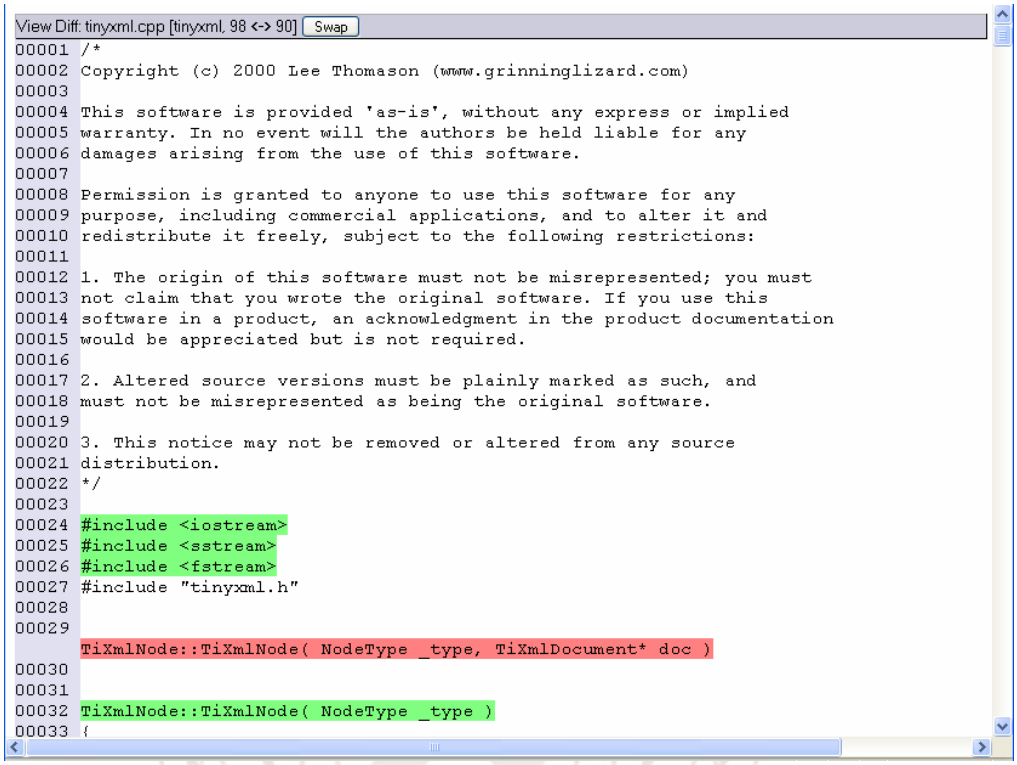


Figure 5.6 Comparison Result Page

View Log Page is used to view modifications of every revision. Revisions displayed are sorted by revision number. The latest revision is in the first order. See Figure 5.7.

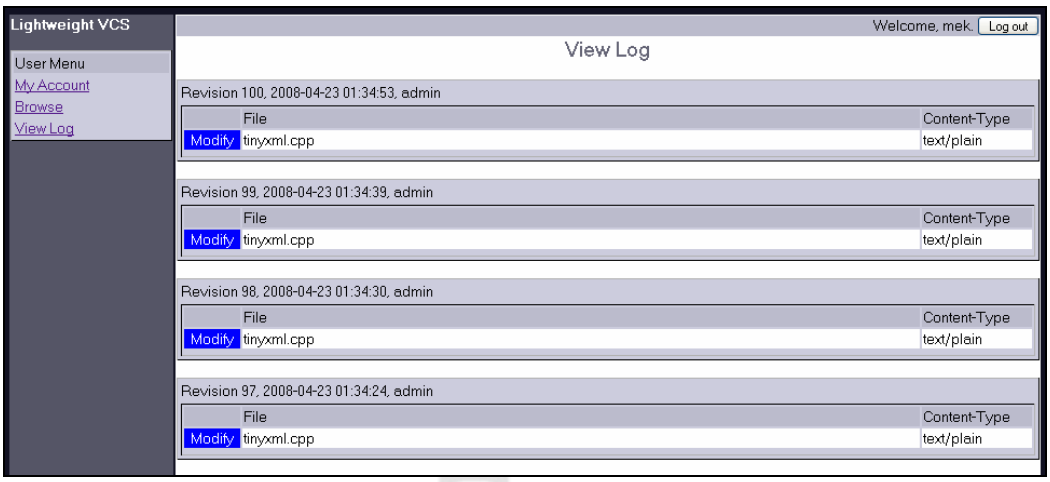


Figure 5.7 view Log Page

2. Administrator Web Portal

Web portal provided for repository administrators is composed of four main web pages: Create User Page, User Management Page, Create Repository Page, and Repository Management Page.

Create User Page is used for creating a new user in the system (See Figure 5.8).

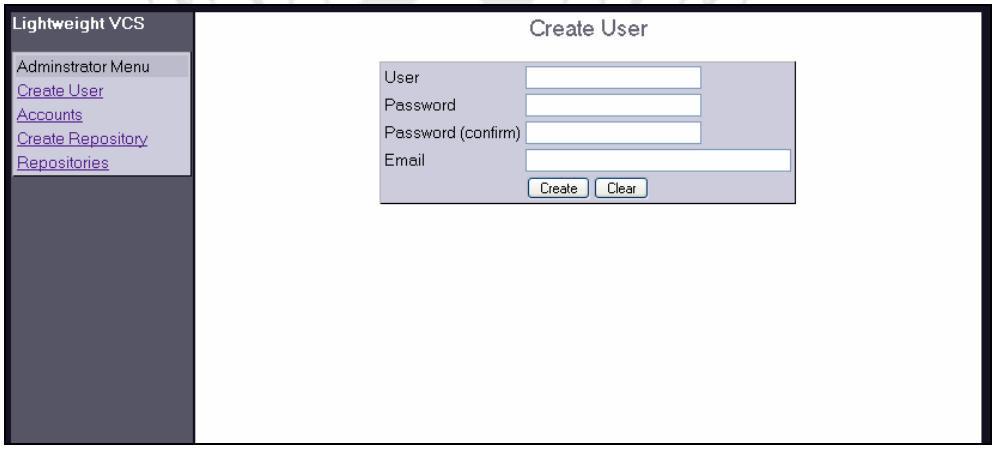


Figure 5.8 Create User Page

User Management Page is used to view and delete users in the system (See Figure 5.9).

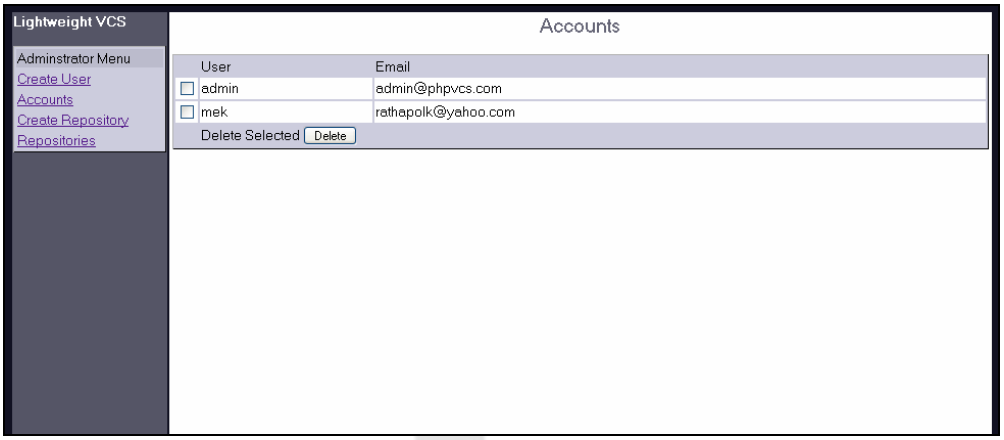


Figure 5.9 User Management Page

Create Repository Page is used to create a new repository in the system (See Figure 5.10).

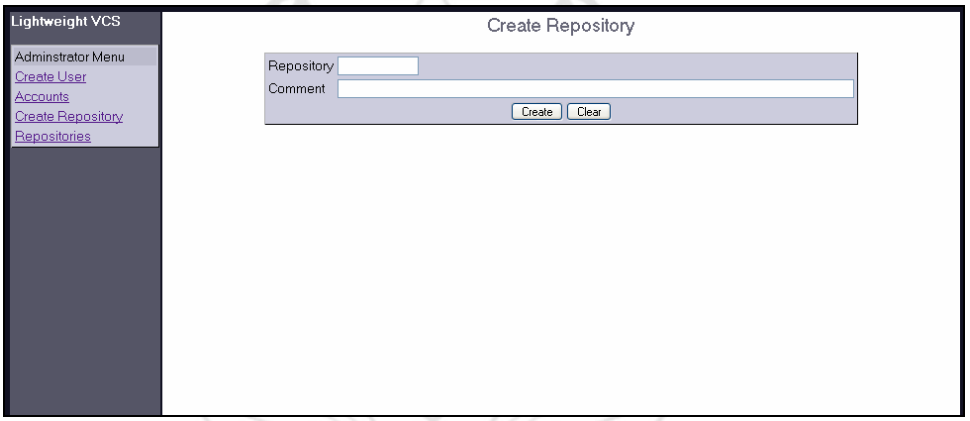


Figure 5.10 Create Repository Page

Repository Management Page is used to view and delete all repositories. It is also used to assign users to repositories or remove them from repositories (See Figure 5.11).

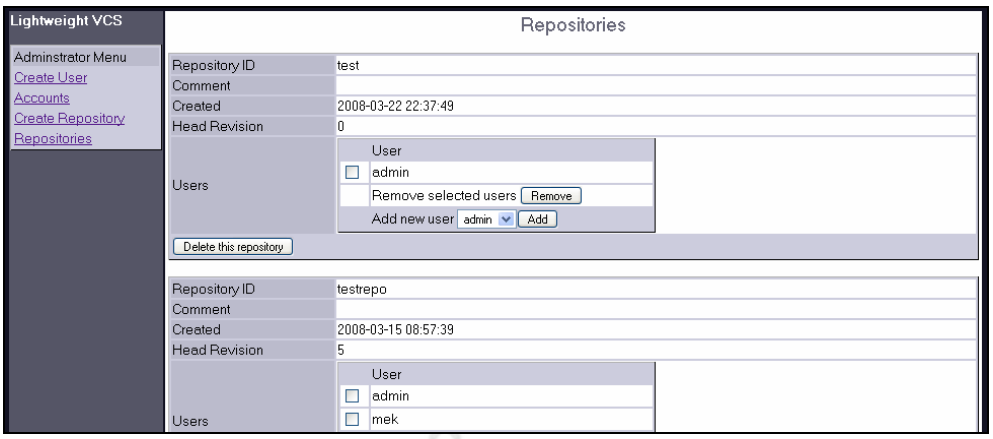


Figure 5.11 Repository Management Page

5.2.3 Client Architecture

The client application provides user interface to work with the repository server as well as manages local copies. The application is composed of two modules: User Interface and Local Repository Manager. It uses third party libraries and tools. The major library is libcurl. The major tool is Diffutils. See Figure 5.12.

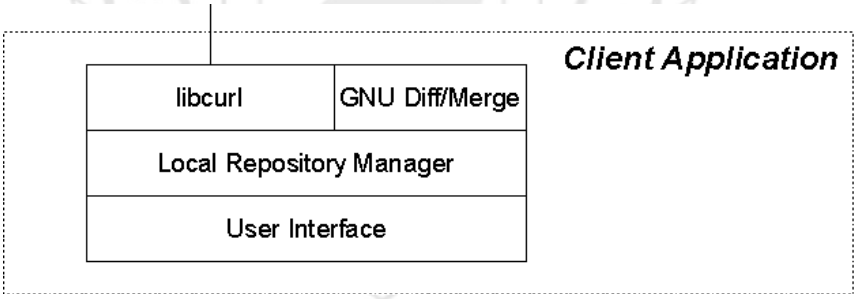


Figure 5.12 Client Architecture

1. User Interface

User Interface module accepts commands from users and calls Local Repository Manager to process the commands.

2. Local Repository Manager

Local Repository Manager is responsible for creating and updating local repositories in a local machine. It uploads content to and downloads content from a repository by using libcurl, the library for the HTTP protocol.

GNU Diffutils is a third party tool. It is used for comparing text files and merging a patch with a text file. Comparison results of the tool can be rendered in various formats. This system uses the unified format.

5.3 Test plan

This project plans to apply four major tests: algorithm test, performance test, functional test, and user satisfaction test.

5.3.1 Algorithm Test

The algorithm test is a test for individual algorithms or units. There are tests for two major algorithms: diff algorithm and content reproduction algorithm.

1. Diff Algorithm

Diff algorithm is a major algorithm used for the delta compression. The implemented algorithm is ported from Java source code. Thus, the same test set will be applied to make sure that both ports work identically.

The test set contains eight test cases. The Java port does not pass the test case number 8. The author stated that he was not sure whether it was an algorithm artifact or the case was incorrect.

2. Content Reproduction Algorithm

A test set for the content reproduction algorithm is designed as a File/Revision matrix. There are three files in a test repository. Contents are modified and committed multiple

times. The final revision is revision 5. Therefore, all contents at every revision are read back and compared to expected results (See Figure 5.13).

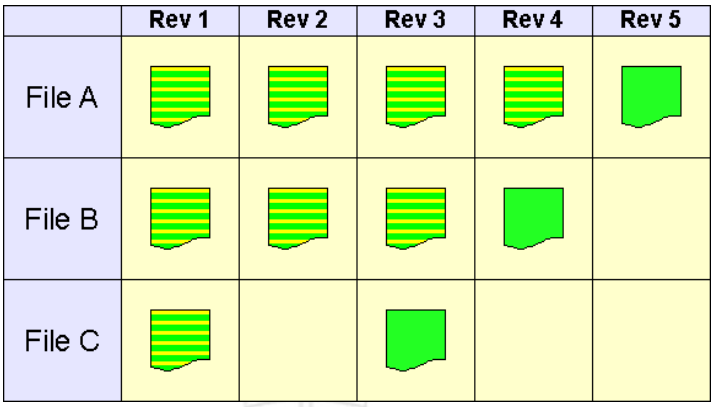


Figure 5.13 Test Set for Content Reproduction Algorithm

5.3.2 Performance Test

This performance test will be applied to the revision rendering function which is most frequently used in the system.

This project selects source code of TinyXML project for the test. A selected file is tinyxml.cpp, which contains about 1700 lines of code. The file is size about 32 KB. The file has been in development since December 2000. Until August 2007, there are more than one hundred committed revisions. One hundred revisions are downloaded and inputted into the project's system. Each time taken in reproducing a revision is recorded. Note, the time is a processing time in the server. Network communication time is not included.

5.3.3 Functional Test

This functional test will be used to check whether the system works correctly or not. Various test cases will be applied. They are grouped into test sets simulating different / independent situations.

Test set A contains various test cases which will be used for testing various check-in and check-out operations. An observer needs to look inside the database tables. This test requires

one user. See Table 5.2.

Table 5.2 Test Set A, Test Check In / Check Out Operations

No	Description	Prerequisite	Expected	Result
1	Commit after adding one new file, known as *File A*.	-	An “add” transaction with full text content of *File A* is inserted into the permanent storage as revision 1.	
2	Commit a modification of *File A*.	Case 1	A “mod” transaction with full text content of *File A* is inserted into the permanent storage as revision 2. The content of the transaction of revision 1 is replaced with deltified content.	
3	Commit after adding another new file, known as *File B*.	Case 2	An “add” transaction with full text content of *File B* is inserted into the permanent storage as revision 3.	
4	Commit a modification of *File A*.	Case 3	A “mod” transaction with full text content of *File A* is inserted into the permanent storage as revision 4. The content of the transaction of revision 2 is replaced with deltified content.	
5	Commit a deletion of *File B*.	Case 4	A “del” transaction with empty content is inserted into the permanent storage as revision 5.	

Table 5.2 Test Set A, Test Check In / Check Out Operations (cont.)

No	Description	Prerequisite	Expected	Result
6	Review a file list of revision 1	Case 5	A file list contains the following entries: *File A*	
7	Review a file list of revision 2	Case 5	A file list contains the following entries: *File A*	
8	Review a file list of revision 3	Case 5	A file list contains the following entries: *File A, File B*	
9	Review a file list of revision 4	Case 5	A file list contains the following entries: *File A, File B*	
10	Review a file list of revision 5	Case 5	A file list contains the following entries: *File A*	
11	Review file contents of revision 1 though revision 5	Case 5	Every file contains correct content.	
12	Update to revision 1 through revision 5 and review every revision.	Case 5	A file list of each revision is correct. Every version controlled file contains correct content.	

Test set B contains various test cases which will be used for testing versioning model when multiple users are working on the same repository. See Table 5.3.

Table 5.3 Test Set B, Test Functionality of Versioning Model

No	Description	Prerequisite	Expected	Result
13	*User A* commits a new file, known as *File C*.	-	A new revision is successfully created as revision 1.	
14	*User B* commits a new file, known as *File D*.	Case 13	Fails to commit.	
15	*User B* updates local repository.	Case 14	Local repository is now revision 1.	
16	*User B* commits *File D*.	Case 15	A new revision is successfully created as revision 2.	
17	*User A* commits a new file, known as *File E*.	Case 16	Fails to commit.	
18	*User A* updates local repository.	Case 17	Local repository is now revision 2.	
19	*User A* commits *File E*.	Case 18	A new revision is successfully created as revision 3.	
20	Review revision 1	Case 18	A file list contains the following entries: *File C* Committing user is *User A*.	
21	Review revision 2	Case 18	A file list contains the following entries: *File C, File D* Committing user is *User B*.	
22	Review revision 3	Case 18	A file list contains the following entries: *File C, File D, File E* Committing user is *User A*.	

Test set C contains various test cases which will be used for testing versioning model when multiple users are working on the same repository trying to modify content in the same file. See Table 5.4.

Table 5.4 Test Set C, Test Functionality of Versioning Model (Modifying the Same File)

No	Description	Prerequisite	Expected	Result
23	*User A* commits a new file, known as *File F*.	-	Revision 1 is created.	
24	*User B* updates local repository.	Case 23	Local repository is updated to revision 1.	
25	*User B* modifies *File F* and commits the change.	Case 24	Revision 2 is created.	
26	*User A* modifies *File F* and commits the change.	Case 25	Fails to commit.	
27	*User A* updates local repository.	Case 26	Local repository is updated to revision 2. *File F* is merged with with the change.	
28	*User A* commits *File F*.	Case 27	Revision 3 is created.	

5.3.4 User Satisfaction Test

The user satisfaction test will be applied to some users who have experience with existing version control systems like Subversion. A survey method used is interview. The server application will be installed in a web hosting service. Each user will have a chance to work as a repository user and an administrator.

5.4 Test results

5.4.1 Algorithm Test

1. Diff Algorithm

As can be seen in Table 5.5, both the Java port and the PHP port produce the same results. The first seven cases pass. The last case fails. See Appendix B for test inputs.

Table 5.5 Diff Algorithm Test Result

Test Case	Java	PHP
Case 1	Passed	Passed
Case 2	Passed	Passed
Case 3	Passed	Passed
Case 4	Passed	Passed
Case 5	Passed	Passed
Case 6	Passed	Passed
Case 7	Passed	Passed
Case 8	Failed	Failed

2. Content Reproduction Algorithm

Table 5.6 Content Reproduction Algorithm Test Result

	Revision 1	Revision 2	Revision 3	Revision 4	Revision 5
File A	Passed	Passed	Passed	Passed	Passed
File B	Passed	Passed	Passed	Passed	Passed
File C	Passed	Passed	Passed	Passed	Passed

5.4.2 Performance Test

One hundred revisions of tinyxml.cpp are inputted into the project's system. Each time taken in reproducing a revision is recorded (See Appendix C for detailed result). A graph is plotted from the result set (See Figure 5.14). The x-axis represents revision number. The y-axis represents time to reproduce content of each revision in seconds.

A platform used to test was ASUS A6000KM Notebook, Turion MT32 CPU, 512MB RAM. The server application ran on WindowsXP OS. The web application server was Apache 2.2. The PHP version was 5.0. MySQL server version was 4.1. The database engine used was InnoDB.

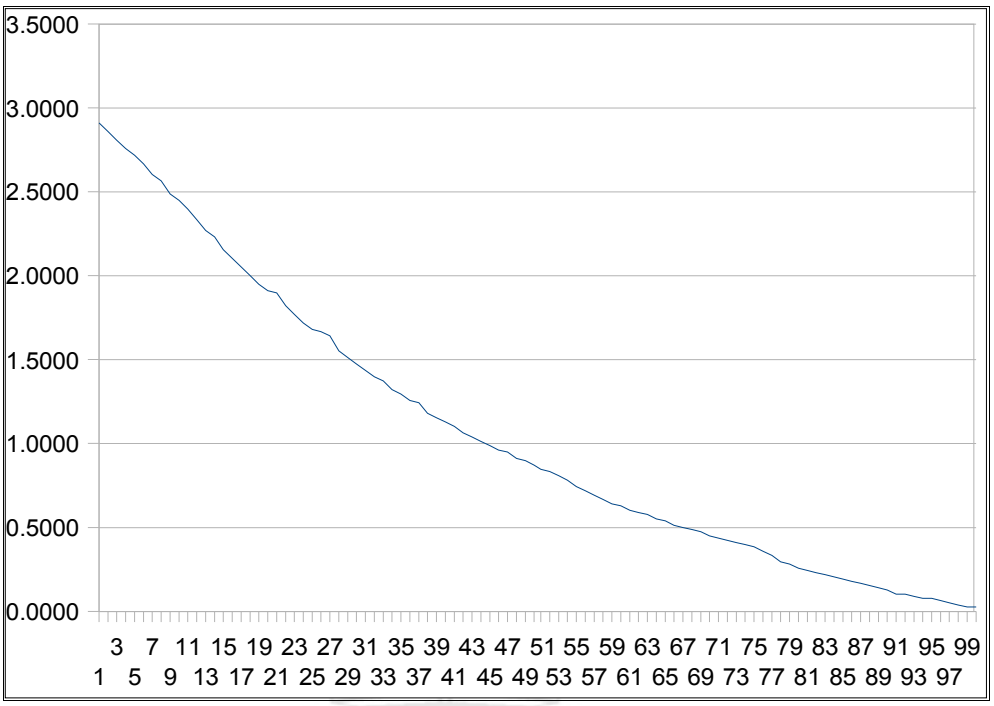


Figure 5.14 Performance Test Graph

As can be seen in the graph, the content of the latest revision takes least time to reproduce. The older revisions take longer time to reproduce. The content of the first revision takes almost 3 seconds to reproduce. The graph looks linear.

The limitation of the system can be predicted by reading the graph. If the requested revision is old enough, the processing time will take more time than a web browser's connection time out value. As a result, the content can not be displayed on the browser.

5.4.3 User Satisfaction Test Result

The testers of the system were software developers at Chanwanich Company. They had been experienced in the software development field at least three years. Subversion was used regularly in the company.

Most testers agreed that the system has basic version control functionalities and its speed was acceptable. The system was not considered a replacement but it could be an alternative if the system would support binary files and provide friendly interface like TortoiseSVN.

The system may be used for projects which team members join together remotely.

Examples are a project working with out source or an off-site project.

5.4.4 Functional Test Result

The test result of Test Set A is shown in Table 5.7.

Table 5.7 Test Result A, Test Check In / Check Out Operations

No	Description	Prerequisite	Expected	Result
1	Commit after adding one new file, known as *File A*.	-	An “add” transaction with full text content of *File A* is inserted into the permanent storage as revision 1.	Passed
2	Commit a modification of *File A*.	Case 1	A “mod” transaction with full text content of *File A* is inserted into the permanent storage as revision 2. The content of the transaction of revision 1 is replaced with deltified content.	Passed
3	Commit after adding another new file, known as *File B*.	Case 2	An “add” transaction with full text content of *File B* is inserted into the permanent storage as revision 3.	Passed
4	Commit a modification of *File A*.	Case 3	A “mod” transaction with full text content of *File A* is inserted into the permanent storage as revision 4. The content of the transaction of revision 2 is replaced with deltified content.	Passed
5	Commit a deletion of *File B*.	Case 4	A “del” transaction with empty content is inserted into the permanent storage as revision 5.	Passed

Table 5.7 Test Result A, Test Check In / Check Out Operations (cont.)

No	Description	Prerequisite	Expected	Result
6	Review a file list of revision 1	Case 5	A file list contains the following entries: *File A*	Passed
7	Review a file list of revision 2	Case 5	A file list contains the following entries: *File A*	Passed
8	Review a file list of revision 3	Case 5	A file list contains the following entries: *File A, File B*	Passed
9	Review a file list of revision 4	Case 5	A file list contains the following entries: *File , File B*	Passed
10	Review a file list of revision 5	Case 5	A file list contains the following entries: *File A*	Passed
11	Review file contents of revision 1 though revision 5	Case 5	Every file contains correct content.	Passed
12	Update to revision 1 through revision 5 and review every revision.	Case 5	A file list of each revision is correct. Every version controlled file contains correct content.	Passed

The test result of Test Set B is shown in Table 5.8.

Table 5.8 Test Result B, Test Functionality of Versioning Model

No	Description	Prerequisite	Expected	Result
13	*User A* commits a new file, known as *File C*.	-	A new revision is successfully created as revision 1.	Passed
14	*User B* commits a new file, known as *File D*.	Case 13	Fails to commit.	Passed
15	*User B* updates local repository.	Case 14	Local repository is now revision 1.	Passed
16	*User B* commits *File D*.	Case 15	A new revision is successfully created as revision 2.	Passed
17	*User A* commits a new file, known as *File E*.	Case 16	Fails to commit.	Passed
18	*User A* updates local repository.	Case 17	Local repository is now revision 2.	Passed
19	*User A* commits *File E*.	Case 18	A new revision is successfully created as revision 3.	Passed
20	Review revision 1	Case 18	A file list contains the following entries: *File C* Committing user is *User A*.	Passed
21	Review revision 2	Case 18	A file list contains the following entries: *File C, File D* Committing user is *User B*.	Passed
22	Review revision 3	Case 18	A file list contains the following entries: *File C, File D, File E* Committing user is *User A*.	Passed

The test result of Test Set C is shown in Table 5.9.

Table 5.9 Test Result C, Test Functionality of Versioning Model (Modifying the Same File)

No	Description	Prerequisite	Expected	Result
23	*User A* commits a new file, known as *File F*.	-	Revision 1 is created.	Passed
24	*User B* updates local repository.	Case 23	Local repository is updated to revision 1.	Passed
25	*User B* modifies *File F* and commits the change.	Case 24	Revision 2 is created.	Passed
26	*User A* modifies *File F* and commits the change.	Case 25	Fails to commit.	Passed
27	*User A* updates local repository.	Case 26	Local repository is updated to revision 2. *File F* is merged with with the change.	Passed
28	*User A* commits *File F*.	Case 27	Revision 3 is created.	Passed

In conclusions, most of the applied tests pass. There is a failing test case in the algorithm test. The test case is a rare case. After applying one hundred revisions in the performance test, contents of committed revisions were examined and found that they were correct. Therefore, the probability to reproduce the failing test case in practical work is less than 1%. Most features requested were support for binary files and directories and a shell- integrated interface like TortoiseSVN.

CHAPTER VI

SUMMARY AND SUGGESSTIONS

6.1 Introduction

This chapter covers project summary of the system attempting to implement a version control system for low cost servers. During development and testing, there were some problems encountered. Most of the problems encountered are limitations of the software used and technical problems. Most of the time, a search engine web site was used as a primary tool.

6.2 Project Summary

This project builds a version control system for low cost servers. The server application is implemented in PHP. It uses MySQL to store repository data. The data stored are compressed by using delta compression. The compression result is rendered in the unified format. The client application is implemented in C/C++. Third party libraries and tools are used to reduce development time.

The system works well with text files. It lacks of handling binary files and folders. It supports only ASCII encoding. Unsupported files do not crash the system but the system will produce incorrect results when they are checked out.

In normal use, the system performs well both speed and disk space consuming. The system will be slow down in case of checking out older revisions. The content size will be larger

if two revisions of a file are completely different.

In practical, a modern software project does not contain only text files. It may also contain binary files, like icon and image, and many levels of directories. The system needs to be improved in order to work with this kind of software project.

6.3 Problems encountered and solutions

The followings are problems encountered during development:

6.3.1 Standard C/C++ does not support directories. It also lacks of file querying. This problem is solved by using Win32 API. But, this makes the client application platform dependent. There are other cross-platform libraries for dealing with file system such as Boost, the collection of free peer-reviewed portable C++ libraries.

6.3.2 The system supports only ASCII text files. It was not initially designed to support other encodings at the design state. It can be solved by modifying all functions regarding text manipulating. This issue was not solved because it was realized after the implementation phase had finished.

6.3.3 The web application was initially written in PHP version 5 but most web hosting services provide PHP version 4. Major difference of the two versions is encapsulation used in class. Version 5 supports modifiers like public, private, and protected while version 4 does not. This problem is solved by removing all the modifiers and losing encapsulation. To prevent this problem, the development environment should be as close as to the production environment.

6.3.4 This project had been initially for a proof-of-concept but user satisfaction test was required at the later time. The project could not be used in production because it lacks of binary file and directory supports. To solve the problem, the system must be implemented to support those files but this is beyond the scope.

6.4 Suggestions for further development

The system is just a beginning of developing a version control system. There is still room for improvement. The following suggestions may be used for further development.

6.4.1 The project functionalities can be improved by adding support for binary files, directories and other encoding than ASCII.

6.4.2 The system should decide whether a delta or full text is stored in the database. This can reduce disk space in case of contents of two revisions are completely different – the delta produced is larger than full text.

6.4.3 A result of reproducing content of a particular revision can be cached and saved in the file system. If the system finds the associated cache file then it uses the file. If it does not then it reproduces content and saves a new cache file. This can improve speed when the same result is required.

6.4.4 The client application can be improved by providing better user interface. Graphical user interface may be used to improve user experience.

The web applications can be improved by providing better information and control. AJAX may be applied to make the web pages more friendly and responsive.

REFERENCES

1. Myers EW. An O(ND) Difference Algorithm and Its Variation. [Online] 1986 [cited 2008 Apr 30]; Available from: URL: http://xmailserverg/diff2.pdf
2. ASVCS. A Simple Version Control System. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://asvcs.com/
3. CVS. CVS - Concurrent Versions System. [Online] 2008 [cite 2008 Apr 30]; Available from: URL:http://www.nongnu.org/cvs/
4. GNU. Diffutils. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://www.gnu.org/software/diffutils/diffutils.html
5. Haxx. libcurl. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://curl.haxx.se/
6. Incava. Difference algorithm for Java. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://www.incava.org/projects/java/java-diff/index.html
7. MySQL AB. MySQL. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://www.mysql.com/
8. Subversion. Deltification. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://svnbook.red-bean.com/
9. Subversion. Version Control with Subversion. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://svnbook.red-bean.com/
10. Subversion. Versioning Model. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://svnbook.red-bean.com/
11. Wikipedia. Comparison of revision control software. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
12. Wikipedia. Hypertext Transfer Protocol. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://en.wikipedia.org/wiki/HTTP

13. Wikipedia. Revision Control. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://en.wikipedia.org/wiki/Version_control
14. Wikipedia. Unified Format. [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://en.wikipedia.org/wiki/Diff
15. Zend. PHP Hypertext Preprocessor. . [Online] 2008 [cite 2008 Apr 30]; Available from: URL: http://www.php.net/

APPENDIX A

TECHNICAL TERMS AND ABBREVIATIONS

Commit

An operation to apply several changes at once

Diff

An algorithm to compare two text files outputting a change set

Deltify / Deltification

An algorithm to store data defined and used by Subversion

Head Revision

The latest revision

HTTP

Hypertext Transfer Protocol, a protocol for transferring files on the World Wide Web

Repository

A centralized database for storing documents

Revision

A version of a file representing a number of changes committed

APPENDIX B

ALGORITHM TEST INPUTS

Table A.1 Algorithm Test Inputs

No	A	B	Expected
1	"a", "b", "c", "e", "h", "j", "l", "m", "n", "p"	"b", "c", "d","e","f","j","k", "l", "m", "r", "s", "t"	Difference(0, 0, 0, -1), Difference(3, -1, 2, 2), Difference(4, 4, 4, 4), Difference(6, -1, 6, 6), Difference(8, 9, 9, 11)
2	"a", "b", "c", "d"	"c", "d"	Difference(0, 1, 0, -1)
3	"a", "b", "c", "d", "x", "y", "z"	"c", "d"	Difference(0, 1, 0, -1), Difference(4, 6, 2, -1)
4	"a", "b", "c", "d", "e"	"a", "x", "y", "b", "c", "j", "e"	Difference(1, -1, 1, 2), Difference(3, 3, 5, 5)
5	"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l"	"a", "b", "p", "q", "r", "s", "t", "c", "d", "e", "f", "g", "h", "i", "j", "u", "l"	Difference(2, -1, 2, 6), Difference(10, 10, 15, 15)
6	"a", "a", "a", "a", "b", "b", "b", "a", "a", "a", "a", "b", "b", "b", "a", "a", "a", "a", "b", "b", "b", "a", "a", "a", "a", "b", "b", "b"	"a", "a", "a", "a", "b", "b", "b", "a", "b", "b", "b", "a", "a", "a", "a"	Difference(8, 10, 8, -1), Difference(18, 27, 15, -1)
7	"A", "B", "C", "D", "E", "F", "G", "A", "H", "I", "J", "D", "K", "L", "C", "G", "M", "H", "N", "J", "I", "K", "O", "C", "G", "M", "P", "Q", "J", "R", "K", "S", "C", "C", "F", "G", "D", "T", "N", "G", "M", "U", "V", "J", "Q", "K", "W", "C", "G", "M", "X", "C", "V", "K", "Y", "C", "G", "G", "A", "Z", "AA", "J", "C", "Z", "G", "V", "K",	"A", "B", "C", "JJ", "G", "A", "II", "KK", "A", "B", "C", "D", "E", "F", "G", "A", "H", "I", "J", "D", "K", "L", "C", "G", "M", "H", "N", "J", "I", "K", "O", "C", "G", "M", "P", "Q", "J", "R", "K", "S", "C", "C", "F", "G", "D", "T", "N", "G", "M", "U", "V", "J", "Q", "K", "W", "C", "G", "M", "X", "C", "V", "K", "Y",	Difference(3, -1, 3, 10), Difference(88, -1, 96, 96)

	"BB", "C", "G", "M", "CC", "DD", "J", "EE", "K", "FF", "C", "AA", "G", "M", "GG", "K", "HH", "C", "DD", "G", "M", "II", "II", "II"	"C", "G", "G", "A", "Z", "AA", "J", "C", "Z", "G", "V", "K", "BB", "C", "G", "M", "CC", "DD", "J", "EE", "K", "FF", "C", "AA", "G", "M", "GG", "K", "HH", "C", "DD", "G", "M", "II", "II", "II", "II"	
8	"{", "ZipEntry", "e", "=", "entry", ";", "if", "(", "e", "!=", "null", ")", "{", "switch", "(", "e", ".", "method", ")", "{", "case", "DEFLATED", ":", "if", "(", "(", "e", ".", "flag", "&", "8", ")", "==", "0", ")", "{", "if", "(", "e", ".", "size", "!=", "def", ".", "getTotalIn", "(", ")", ")", "{", "throw", "new", "ZipException", "(", "\"invalid entry size (expected \"", "+", "e", ".", "size", "+", "\" but got \"", "+", "def", ".", "getTotalIn", "(", ")", "+", "\" bytes)\"", ")", ";", "}", "if", "(", "e", ".", "csize", "!=", "def", ".", "getTotalOut", "(", ")", ")", "{", "throw", "new", "ZipException", "(", "\"invalid entry compressed size (expected \"", "+", "e", ".", "csize", "+", "\" but got \"", "+", "def", ".", "getTotalOut", "(", ")", "+", "\" bytes)\"", ")", ";", "}", "if", "(", "e", ".", "crc", "!=", "crc", ".", "getValue", "(", ")", ")", "{", "throw", "new", "ZipException", "(", "\"invalid entry CRC-32 (expected 0x\"", "+", "Long", ".", "toHexString", "(", "e", ".", "crc", ")", "+", "\" but got 0x\"", "+", "Long", ".", "toHexString", "(", "crc", ".", "getValue", "(", ")", ")", "+",	"{", "ZipEntry", "e", "=", "entry", ";", "if", "(", "e", "!=", "null", ")", "{", "switch", "(", "e", ".", "method", ")", "{", "case", "DEFLATED", ":", "if", "(", "(", "e", ".", "flag", "&", "8", ")", "==", "0", ")", "{", "if", "(", "e", ".", "size", "!=", "def", ".", "getBytesRead", "(", ")", ")", "{", "throw", "new", "ZipException", "(", "\"invalid entry size (expected \"", "+", "e", ".", "size", "+", "\" but got \"", "+", "def", ".", "getBytesRead", "(", ")", "+", "\" bytes)\"", ")", ";", "}", "if", "(", "e", ".", "csize", "!=", "def", ".", "getBytesWritten", "(", ")", ")", "{", "throw", "new", "ZipException", "(", "\"invalid entry compressed size (expected \"", "+", "e", ".", "csize", "+", "\" but got \"", "+", "def", ".", "getBytesWritten", "(", ")", "+", "\" bytes)\"", ")", ";", "}", "if", "(", "e", ".", "crc", "!=", "crc", ".", "getValue", "(", ")", ")", "{", "throw", "new", "ZipException", "(", "\"invalid entry CRC-32 (expected 0x\"", "+", "Long", ".", "toHexString", "(", "e", ".", "crc", ")", "+", "\" but got 0x\"", "+",	Difference(3, -1, 3, 10), Difference(88, -1, 96, 96)

	"\")\"", ")", ";", "}", "}", "else", "{", "e", ".", "size", "=", "def", ".", "getTotalIn", "(", ")", ";", "e", ".", "csize", "=", "def", ".", "getTotalOut", "(", ")", ";", "e", ".", "crc", "=", "crc", ".", "getValue", "(", ")", ";", "writeEXT", "(", "e", ")", ";", "}", "def", ".", "reset", "(", ")", ";", "written", "+=", "e", ".", "csiz e",";", "break", ";", "}", "}", "}"	"Long", ".", "toHexString", "(", "crc", ".", "getValue", "(", ")", ")", "+", "\")\"", ")", ";", "}", "}", "else", "{", "e", ".", "size", "=", "def", ".", "getBytesRead", "(", ")", ";", "e", ".", "csize", "=", "def", ".", "getBytesWritten", "(", ")", ";", "e", ".", "crc" , "=", "crc", ".", "getValue", "(", ")", ";", "writeEXT", "(", "e", ")", ";", "}", "def", ".", "reset", "(", ")", ";", "written", "+=", "e", ".", "csize", ";", "break", ";", "}", "}", "}"	

APPENDIX C

PERFORMANCE TEST DETAILED RESULT

Revision	Time (Sec)					
	1	2	3	Min	Max	Average
1	2.9018	2.9175	2.9191	2.9018	2.9191	2.9128
2	2.8599	2.8521	2.8515	2.8515	2.8599	2.8545
3	2.8008	2.8119	2.8023	2.8008	2.8119	2.8050
4	2.7648	2.7651	2.7484	2.7484	2.7651	2.7595
5	2.7334	2.6999	2.7012	2.6999	2.7334	2.7115
6	2.6563	2.6474	2.6780	2.6474	2.6780	2.6606
7	2.5902	2.5999	2.6022	2.5902	2.6022	2.5975
8	2.5925	2.5395	2.5463	2.5395	2.5925	2.5595
9	2.4783	2.4974	2.4958	2.4783	2.4974	2.4905
10	2.4503	2.4410	2.4368	2.4368	2.4503	2.4427
11	2.3828	2.3899	2.4272	2.3828	2.4272	2.3999
12	2.3449	2.3255	2.3205	2.3205	2.3449	2.3303
13	2.2621	2.2580	2.2691	2.2580	2.2691	2.2631
14	2.2371	2.2334	2.2024	2.2024	2.2371	2.2243
15	2.1467	2.1720	2.1494	2.1467	2.1720	2.1560
16	2.1044	2.0971	2.1010	2.0971	2.1044	2.1008
17	2.0575	2.0513	2.0525	2.0513	2.0575	2.0538
18	2.0001	1.9974	2.0039	1.9974	2.0039	2.0005
19	1.9474	1.9491	1.9513	1.9474	1.9513	1.9493
20	1.9063	1.9002	1.9173	1.9002	1.9173	1.9079
21	1.8858	1.8823	1.9046	1.8823	1.9046	1.8909
22	1.8126	1.8069	1.8190	1.8069	1.8190	1.8128
23	1.7672	1.7685	1.7629	1.7629	1.7685	1.7662
24	1.7155	1.7168	1.7209	1.7155	1.7209	1.7177
25	1.6721	1.6693	1.6995	1.6693	1.6995	1.6803

Revision	Time (Sec)					
	1	2	3	Min	Max	Average
26	1.6569	1.6317	1.6979	1.6317	1.6979	1.6622
27	1.6221	1.6352	1.6467	1.6221	1.6467	1.6347
28	1.5487	1.5599	1.5414	1.5414	1.5599	1.5500
29	1.5061	1.5116	1.5297	1.5061	1.5297	1.5158
30	1.4907	1.4645	1.4707	1.4645	1.4907	1.4753
31	1.4297	1.4355	1.4323	1.4297	1.4355	1.4325
32	1.4078	1.3930	1.3986	1.3930	1.4078	1.3998
33	1.3930	1.3599	1.3602	1.3599	1.3930	1.3711
34	1.3231	1.3230	1.3255	1.3230	1.3255	1.3239
35	1.2879	1.2948	1.3012	1.2879	1.3012	1.2947
36	1.2565	1.2534	1.2562	1.2534	1.2565	1.2553
37	1.2367	1.2234	1.2528	1.2234	1.2528	1.2376
38	1.1822	1.1879	1.1801	1.1801	1.1879	1.1834
39	1.1537	1.1498	1.1562	1.1498	1.1562	1.1532
40	1.1315	1.1213	1.1203	1.1203	1.1315	1.1244
41	1.1009	1.0923	1.0949	1.0923	1.1009	1.0960
42	1.0662	1.0654	1.0585	1.0585	1.0662	1.0633
43	1.0329	1.0287	1.0318	1.0287	1.0329	1.0311
44	1.0057	1.0050	1.0055	1.0050	1.0057	1.0054
45	0.9880	0.9888	0.9881	0.9880	0.9888	0.9883
46	0.9784	0.9540	0.9605	0.9540	0.9784	0.9643
47	0.9431	0.9425	0.9575	0.9425	0.9575	0.9477
48	0.9107	0.9166	0.9118	0.9107	0.9166	0.9131
49	0.8988	0.8931	0.8896	0.8896	0.8988	0.8939
50	0.8687	0.8648	0.8698	0.8648	0.8698	0.8678

Revision	Time (Sec)					
	1	2	3	Min	Max	Average
51	0.8442	0.8452	0.8503	0.8442	0.8503	0.8466
52	0.8191	0.8345	0.8212	0.8191	0.8345	0.8249
53	0.7973	0.8013	0.8012	0.7973	0.8013	0.7999
54	0.8001	0.7662	0.7759	0.7662	0.8001	0.7807
55	0.7417	0.7400	0.7421	0.7400	0.7421	0.7412
56	0.7234	0.7116	0.7194	0.7116	0.7234	0.7181
57	0.7009	0.6939	0.6895	0.6895	0.7009	0.6947
58	0.6648	0.6675	0.6632	0.6632	0.6675	0.6652
59	0.6389	0.6427	0.6428	0.6389	0.6428	0.6415
60	0.6178	0.6213	0.6221	0.6178	0.6221	0.6204
61	0.6035	0.5977	0.6114	0.5977	0.6114	0.6042
62	0.5841	0.5881	0.5821	0.5821	0.5881	0.5848
63	0.5710	0.5700	0.5671	0.5671	0.5710	0.5694
64	0.5542	0.5493	0.5494	0.5493	0.5542	0.5510
65	0.5301	0.5362	0.5292	0.5292	0.5362	0.5318
66	0.5152	0.5095	0.5150	0.5095	0.5152	0.5133
67	0.4973	0.4945	0.4974	0.4945	0.4974	0.4964
68	0.4859	0.4811	0.4818	0.4811	0.4859	0.4830
69	0.4615	0.4689	0.4662	0.4615	0.4689	0.4655
70	0.4446	0.4551	0.4533	0.4446	0.4551	0.4510
71	0.4374	0.4367	0.4365	0.4365	0.4374	0.4368
72	0.4225	0.4235	0.4199	0.4199	0.4235	0.4220
73	0.4065	0.4015	0.4067	0.4015	0.4067	0.4049
74	0.4257	0.3838	0.3913	0.3838	0.4257	0.4003
75	0.3780	0.3988	0.3722	0.3722	0.3988	0.3830

Revision	Time (Sec)					
	1	2	3	4	5	6
76	0.3492	0.3519	0.3540	0.3492	0.3540	0.3517
77	0.3274	0.3284	0.3232	0.3232	0.3284	0.3263
78	0.3006	0.2936	0.2980	0.2936	0.3006	0.2974
79	0.2760	0.2693	0.2760	0.2693	0.2760	0.2738
80	0.2595	0.2559	0.2556	0.2556	0.2595	0.2570
81	0.2419	0.2401	0.2403	0.2401	0.2419	0.2408
82	0.2250	0.2299	0.2239	0.2239	0.2299	0.2263
83	0.2147	0.2045	0.2126	0.2045	0.2147	0.2106
84	0.2000	0.1987	0.1995	0.1987	0.2000	0.1994
85	0.1856	0.1805	0.1849	0.1805	0.1856	0.1837
86	0.1736	0.1772	0.1701	0.1701	0.1772	0.1736
87	0.1554	0.1616	0.1563	0.1554	0.1616	0.1577
88	0.1462	0.1491	0.1459	0.1459	0.1491	0.1471
89	0.1356	0.1297	0.1297	0.1297	0.1356	0.1317
90	0.1206	0.1187	0.1176	0.1176	0.1206	0.1190
91	0.1042	0.1076	0.1051	0.1042	0.1076	0.1056
92	0.0899	0.0965	0.0949	0.0899	0.0965	0.0938
93	0.0873	0.0860	0.0861	0.0860	0.0873	0.0865
94	0.0780	0.0811	0.0731	0.0731	0.0811	0.0774
95	0.0689	0.0706	0.0670	0.0670	0.0706	0.0688
96	0.0617	0.0642	0.0548	0.0548	0.0642	0.0602
97	0.0500	0.0456	0.0525	0.0456	0.0525	0.0494
98	0.0335	0.0332	0.0340	0.0332	0.0340	0.0336
99	0.0248	0.0245	0.0295	0.0245	0.0295	0.0263
100	0.0422	0.0122	0.0118	0.0118	0.0422	0.0221

APPENDIX D

INTERVIEW SCRIPT

The following questions were used for interview.

Personal Questions:

- Gender
- Experience working with computers
- Experience in software development
- Experience working with existing version control systems

Questions relating to the project:

- Does the system work reasonably?
- Is the speed acceptable?
- Can the system be a replacement to existing systems?
- What features are missing and required?
- Do you plan to use the system?

CURRICULUM VITAE

NAME	Mr. Rathapol Konkaew
DATE OF BIRTH	April 18, 1983
EDUCATION	
Bachelor Degree	Computer Engineering Mahidol University 2005
WORK EXPERIENCE	2005-Present Software Engineer Chanwanich Co., Ltd.