Annexure - 1

TABLE OF CONTENTS

Page No.

Annexure - 2

CHAPTER 1
INTRODUCTION

1.1 GENERAL

Software development is a challenge faced by most developers when developing medium or large sized project. Software is usually developed as a team work and even when developers are working in the same office, careful collaboration of work is of utmost importance.

The foremost importance when developing a piece of software is managing the code. If hundreds of developers are working on the same code, they need to be streamlined and taken care that no two developer's work is in conflict with each other. In short we need Source Control Management. This source control management software should allow concurrent access while reading and exclusive access while making changes.

Source control management is not the only thing to be taken care of. To err is human and developers too fall in this category. All software has bugs and they need to be tracked. These bugs may be due to poor testing or due to shortcoming of the code. Logical errors also go un-noticed. If this software uses networking using TCP/IP, then care has to be taken for network

failures. Some of the bugs may not be a bug but a wish-list. Normal users may file bugs which are actually features instead of bugs, which need to be addressed carefully.

1.2 SOURCE CONTROL MANAGEMENT

Source Control Management can be of two types – Centralized and Distributed. The former is most widely used SCM, whereas distributed is used for extremely large projects such as Linux Kernel project where thousand or even more developers may be working at the same time. Distributed SCM has its own share of problems and at the same time very useful when concurrent reads and writes need to be supported.

In this project we shall concentrate only on Centralized source control management since it is most widely used. In centralized SCM, each developer checks out the project from the central code repository. The checked-out (downloaded) code is called the working copy of the project. Developers make changes to the code and then commit it to the same code repository. Committing the changes makes the local changes saved on the central repository. Only the changes of the file are saved rather than keeping redundant copies of the same file.

SCM has support for Access control lists. Not all developers can commit the changes they make. Anonymous check-out though discouraged is also supported, though they need to be enabled. A part-time contributor can make changes to the code and submit a patch to the developers, if they don't have access to make changes to the central repositories.

1.3 BUG TRACKER

The Bug Tracker is one of the most interactive module of SOFTWARE DEVELOPMENT PLATFORM .Every project which is under development contains many undesired properties, faulty methods, erroneous codes E.T.C. These exceptional cases has to be dealt properly in order create a reliable and fully functional project.

Bug Tracker is a kind of forum where people can report any kind of errors they found on the registered projects. Bug Tracker works as a centralized list for all the reported bugs. Developer can allow or restrict normal users, other developers to try out their pieces of codes. And these people can report all the bugs they found. This reduces the burden on developers. It also increases efficiency of the project by distributing error identifying process.

Along with bug reporting, Bug Tracker can also enlist valuable suggestions for future developments. Bug Tracker is highly interactive as it works like a problem solving forum. It supports threaded forum like structure, where a discussion is initiated by each and every reported bug or feature. The administrator and developers can decide when a problem is solved. They can close a discussion if they think the appropriate solution is provided for the discussion starter. The Bug Tracker supports assignment of priority to each bug or feature request.

1.4 ACCESS CONTROL LISTS

Every person who registers on this Software Development Platform has not to be given equal access to the resources since a permissions hierarchy needs to be set up. Closed list of

developers, users need to be set up for actively developing a project and be free from any outside interference. Developers may want to keep some features or operations reserved to a fixed set of users to make sure that they are not misused. Access Control helps fine-grain the security policies and it makes sure that the system is not abused.

Access Control Lists is implemented as a table where each row represents a role of the user accessing it and the columns have the operations. The values are represented either in bool to indicate absence or presence (YES/NO) or integers to indicate the priority if it's not boolean. Role as opposed to contrary belief is not the role such as administrators or moderators. Each role is assigned using a userid and project with he is attached. A user may have more than one role depending on the project. This means that user can be a moderator in one project and an admin in another and a normal developer in another. In all such cases he has three roles, one in each project. The normal definition of roles can now be inferred from the Access Control Lists.

The benefits of having access control is immense! There is no hard coding and user's permissions can be controlled at a very fine grain level. If a user tries to abuse his rights of commit, he does not need to be downgraded to lower level permissions, just the commit rights can be revoked without even touching his other rights.

In this project Access Control Lists is used extensively for source control management module. This module is most prone to abuses from errant users. The permissions are fine grained to such a fine level that even the listing of files can be blocked in some cases.

1.5 COMPUTER NETWORKING

Computer Networking is an integral part of this project. TCP/IP Protocol Suite is the most widely used Protocol suite in use these days. This project takes the most out of the heavily and exhaustively documented protocol. BSD sockets as used for low level communication between the two hosts. TCP protocol is used unless and otherwise stated.

The organization of the source control management is that of a client-server. A source control daemon runs at the server which accepts the connection from the client. The daemon is multithreaded, so multiple clients can connect to the server at the same time. There is no clashes at the server for wait conditions and care has been taken to avoid race-conditions. Mutual exclusive lock on any resource is avoided unless required. MySQL Database is used to store data which locks tables row-wise if it has to update a row.

A lot of hand-shaking is used for passing data from one host to another. The client needs to authenticate itself in each and every request. Needless to say, the server is stateless, which

means that the session is not saved at the server. Each connection request is fresh and carries no resemblance from the earlier one.

File transfer is accomplished by opening the file in read only binary mode, reading the entire contents and then sending it over the network. The daemon reads the binary data in a variable, opens a file in write only binary mode and writes the contents back to the file. This method is part of tftp specification which is trivial file transfer protocol. Since fully-fledged features of tftp isn't required, only one specification is used for accomplishing the requirements.

CHAPTER 2

SOFTWARE REQUIREMENTS & SPECIFICATIONS

- **2.1 MODULES**

- The modules are as follows

Module Name	Usage
ADD	Adding a file
DELETE	Deleting a file
COMMIT	Synchronizing (client to server)
CHECKOUT	Downloading files from server
LS	Listing out all files
UPDATE	Synchronizing (server to client)
REGISTER	Adding new user
LOGIN AGENT	Authenticate a user
BUG REPORTING	Reporting a bug
COMMENT POSTING	Replying to messages
REPORT GENERATOR	Generating a brief summary.

2.1.1 ADD

Description: Adds a new file to the local working copy.

Input: Filename and path.

Processing: Added to local sqlite table called ‘pending_changes’ and the tag is set to ‘A’.

Output: If the operation is successful user will receive an acknowledgement from central repository. Otherwise, user will receive a negative acknowledgement.

Condition: User must have read permission on the file to be added, and the file must exist.

2.1.2 DELETE

Description: Deletes a file from local working copy.

Input: Filename and its path.

Processing: Added to local sqlite table called ‘pending_changes’ and the tag is set to ‘D’.

Output: If the operation is successful user will receive an acknowledgement from central repository. Otherwise, user will receive a negative acknowledgement.

Condition: User must have write permission on the file to be deleted, and the file must exist.

2.1.3 COMMIT

Description: This module synchronizes local working copies of files with files stored in central repository.

Input: No Input.

Processing: Compares all the checksums and timestamps of local files with that of the central repository. If the checksums are different for any file, then it checks the timestamps. If local timestamp is greater than that of central repository’s copy, then local file is copied to the central repository and new checksum and timestamps are added.

Output: Acknowledgement from server that the files have been received and client can update local database entries and timestamp.

Condition: At least one entry must exist in the pending changes table.

2.1.4 CHECKOUT

Description: Downloads all the contents from a specified branch of central repository to the client. So that it can be used as a working copy.

Input: Repository name, branch (optional), Authentication and the local location where the files are to be copied.

Processing: The client sends all the information to central repository, and the repository sends all the requested files one after another along with its type and relative position.

Output: To have an updated local copy.

Condition: User must have checkout rights, and must provide proper authentication.

2.1.5 LS

Description: Lists all the files under current working directory in local working copy.

Input: None

Processing: Checks the local database for all the files and lists out all the files of current folder in local working copy.

Output: List of files in current working directory

Condition: None

2.1.6 UPDATE

Description: This module synchronizes files stored in central repository with local working copies.

Input: None.

Processing: Compares all the checksums and timestamps of local files with that of the central repository. If the checksums are different for any file, then it checks the timestamps. If central timestamp is greater than that of local copy, then central repository's file is copied to the local working directory and new checksum and timestamps are added.

Output: Fully updated local working directory.

Condition: None.

2.1.7 REGISTER

Description: Allows a new user to be added.

Input: User informations like user id, name, password, email.

Processing: Data are collected from the php based front end module and sent to central User table. If the user id is not present is not present in database, then registration will be granted. Else, user will be prompted to choose another user id.

Output: On successful completion, user will be granted authentication.

Condition: User name must be unique.

2.1.8 LOGIN AGENT

Description: This module carries out the authentication process.

Input: User Name or email and password.

Processing: Data are collected from the php based front end module and sent to central User table. If the user id and password combination is present in database, then authentication will be granted. Else, user will be prompted to try again.

Output: User will be granted privileges.

Condition: Proper identity must be provided.

2.1.9 BUG REPORTING

Description: This module allows user to report about bugs.

Input: Bug type, title, description, priority, project name, attachments (optional)

Processing: All the collected data are added to tickets table.

Output: User will receive a bug id on successful submission.

Condition: User must have privileges to report bugs.

2.1.10 COMMENT POSTING

Description: This module allows user to comment on reported bugs or other thread starters.

Input: Title, description, attachments (optional)

Processing: All the collected data are added to tickets table.

Output: Reply will be added in the appropriate position.

Condition: User must have privileges to comment.

2.1.11 REPORT GENERATOR

Description: This module generates various reports to sum up current status of projects.

Input: it takes inputs from almost all the tables. Mainly tickets, projects and user tables are used.

Processing: All the collected data are processed by graph generating component.

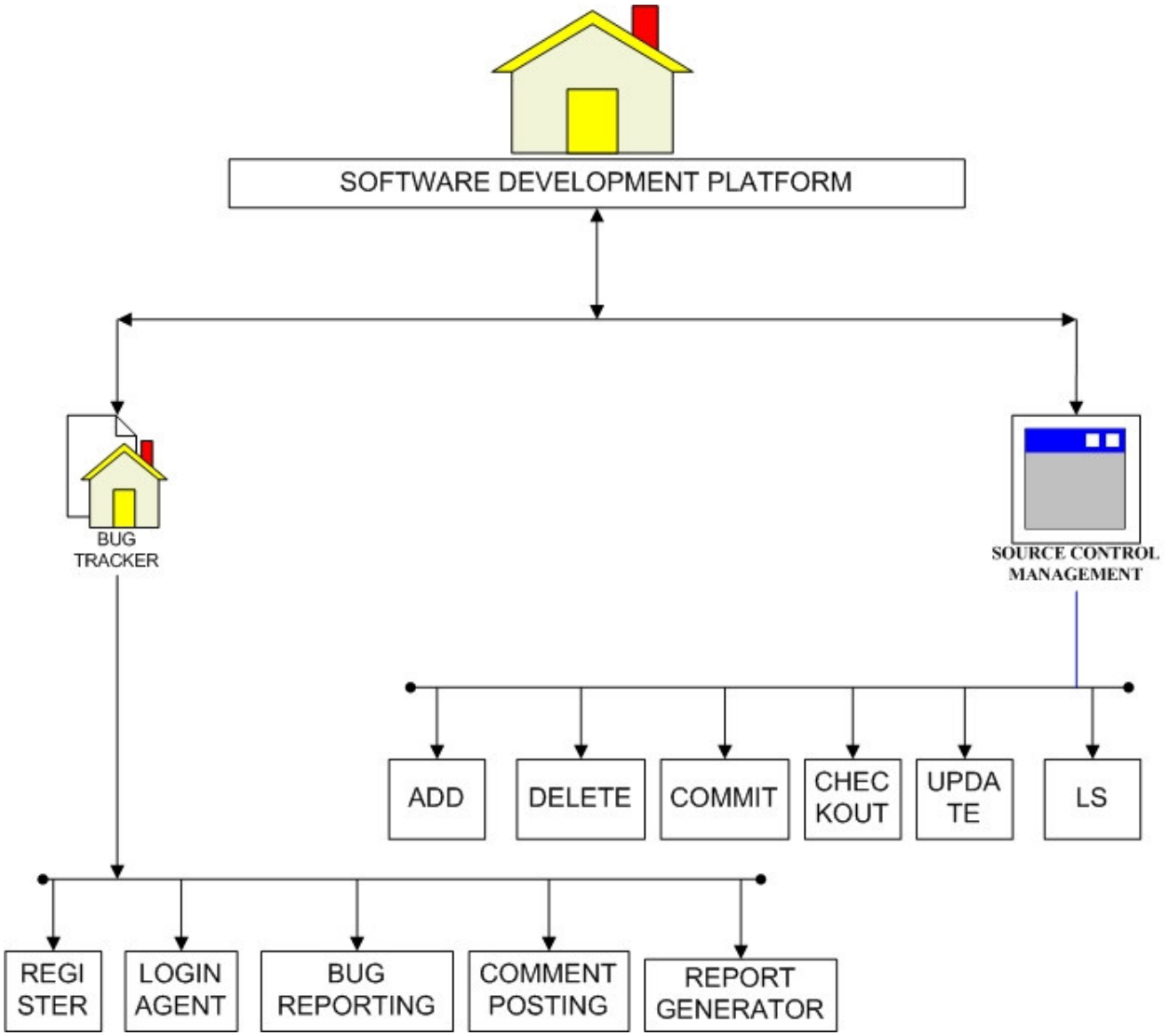
Output: Graphs are created.

Condition: None.

CHAPTER 3

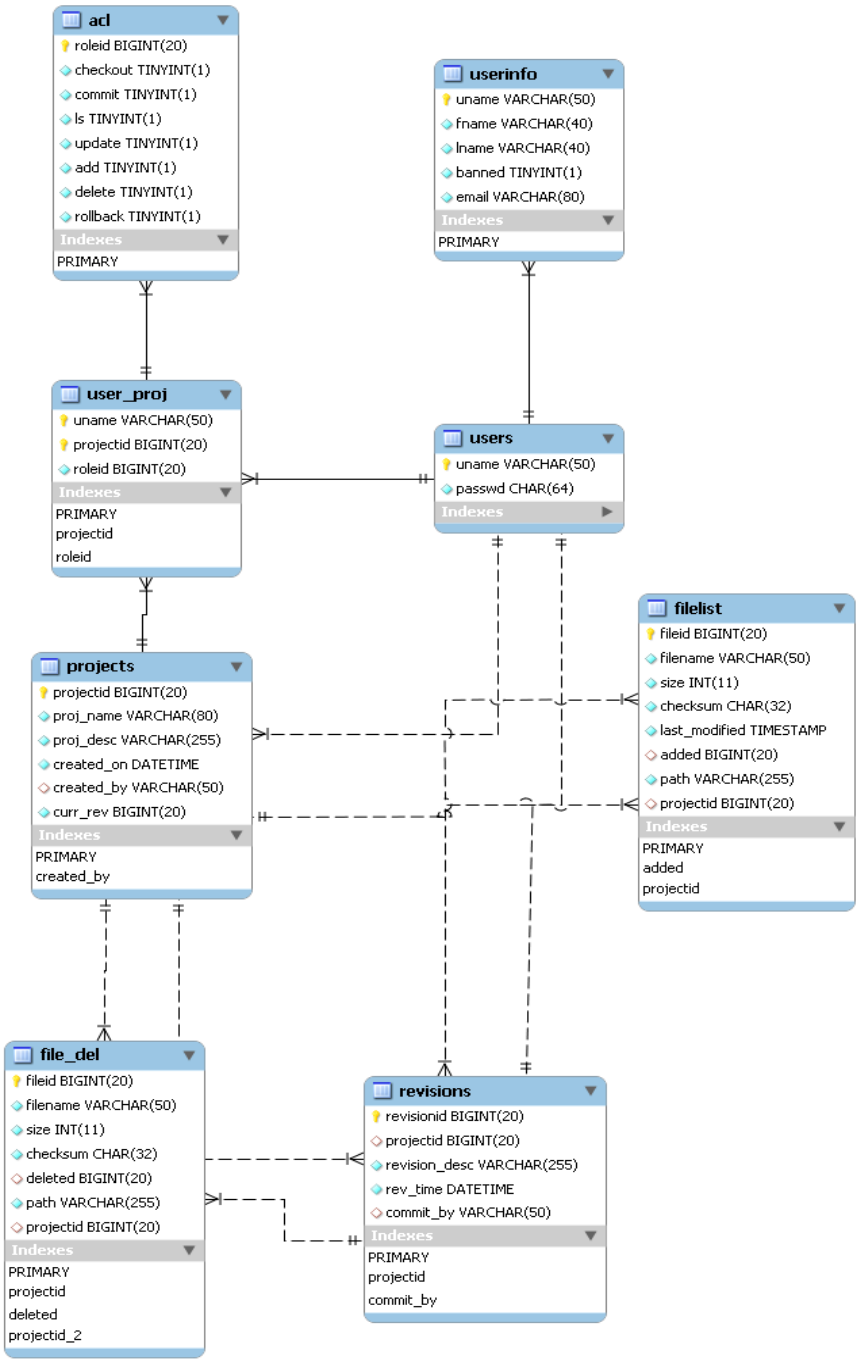
SYSTEM LEVEL DESIGN

3.1 SOFTWARE ARCHITECTURE

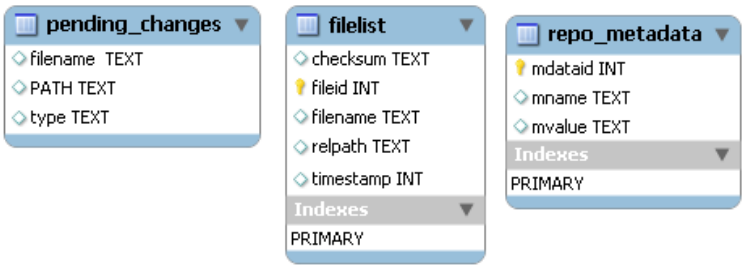


3.2 DATABASE SCHEMA

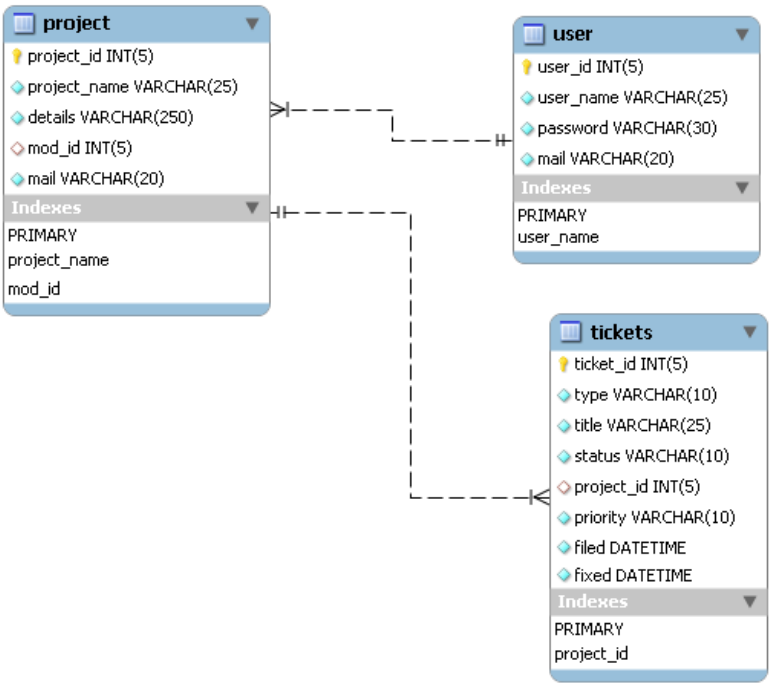
3.2.1 REVCONTROL DATABASE



3.2.2 CLIENT DATABASE



3.2.3 BUG TRACKER DATABASE



CHAPTER 4

Annexure -3

REFERENCES

1. **Tim Converse, Joyce Park, and Clark Morgan** (2004) PHP5 and MySQL Bible, (Wiley) , USA

2. **David Sklar** (2004) -Learning PHP 5, O' Reilly , USA

3. **Mike Owens** (2006)- The Definitive Guide to SQLite, Apress, USA

4. **Peter C. Norton, Alex Samuel, Dave Aitel, and Eric Foster-Johnson** (2005) - Beginning Python (Programmer to Programmer) , Wiley, USA

5. **Mark Lutz** (2006)- Programming Python , O Reilly, USA

6. **Alex Martelli** (2006)- Python in a Nutshell, Second Edition, O Reilly , USA

7. **Paul DuBois** (2007)- MySQL Cookbook , O Reilly , USA

8. **Russell Dyer** (2008)- MySQL in a Nutshell , O Reilly , USA

9. **Sterling Hughes and Andrei Zmievski** (2006)- PHP Developer's Cookbook (2nd Edition) (Developer's Library) , O Reilly , USA

10. **W. Richard Stevens** (1998)-UNIX Network Programming, Volume 2: Interprocess Communications (2nd Edition) (The Unix Networking Reference Series , Vol 2) , Pearson Education Asia, USA

11. **John Goerzen** (2004)- Foundations of Python Network Programming , Apress, USA

12. **Alex Martelli, Anna Ravenscroft, and David Ascher** (2005)- Python Cookbook, O Reilly , USA

13. **Peter Saint-Andre , Kevin Smith, and Remko Tronçon** (2009)- XMPP: The Definitive Guide , O Reilly , USA

14. **Thomas W. Christopher** (2001)- Python Programming Patterns, O Reilly , USA