ASSIGNMENT 1 REPORT

GROUP 7

CSCI222

CONTENTS

OVERVIEW

This project involves the design and implementation of a file management system. The system needs to allow users to store changes to files for later retrieval while remaining efficient in terms of storage requirements and processing speed. This system aimed to be a solution for arbitrarily large binary files, which is currently efficiently solved by most popular source code control versioning systems like Git.

The program's main functionalities are storing and managing files in persistent storage, as well as compressing and decompression of those files, keeping track of additions and removals and implementing a functional graphical user interface, the project uses a MySQL database as its backbone for data storage.

GROUP

Role	**Assignee**
Manager (1)	Nicholas Morgan
Lead Designer (1)	Josh Coleman
Lead Implementer (1)	Philip Edwards
Designer (*)	many
Data Persistence Specialist (1)	Thomas Nixon
Systems integration and systems test (*)	Ivana Ozakovic, Philip Edwards
Documentation (*)	many
Implementer (*)	Philip Edwards, Thomas Nixon, Ivana Ozakovic, Josh Coleman
Document Backup Maintainer (1)	Thomas Nixon

PROGRAM PRESENTATION

LIST OF FEATURES

1. Adding files to persistent storage

The system stores a copy of the file and its details in the persistent storage with a comment supplied by the user. The program compresses the given file/data to minimize storage space.

2. Save modified version of the file

The program will detect if any changes are made in the current file and if a user attempts to save changes, a user will be asked to supply a comment to go with the new version of the current file. Only changes that were modified since the previous version archived will be saved in order to use the storage space efficiently.

3. Display summary details of versions in storage for selected file

In case a user selects the file that is already in the storage, all versions of the file will be displayed in the table view with summary details of each version, which includes version number, date and time when version was created, and its size.

4. Retrieve chosen version of the file

If the user wishes to retrieve a specific version of the file, the program retrieves the chosen version in the directory specified by the user along with the filename.

5. Discard old

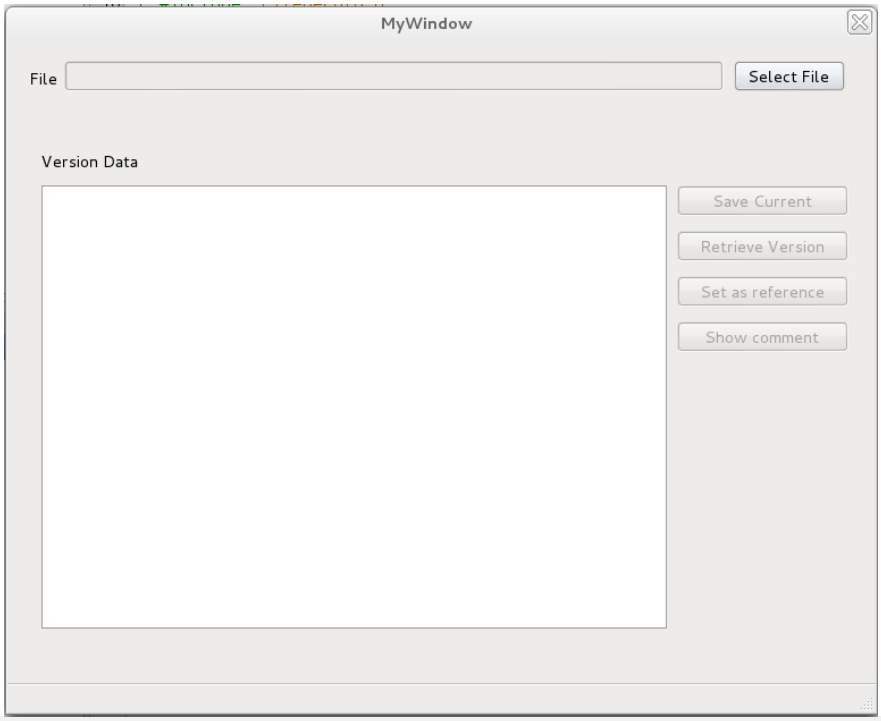
The program allows the user to set a “reference version”, which is used to purge all versions prior to the selected reference version.

6. Show comment associated with the version

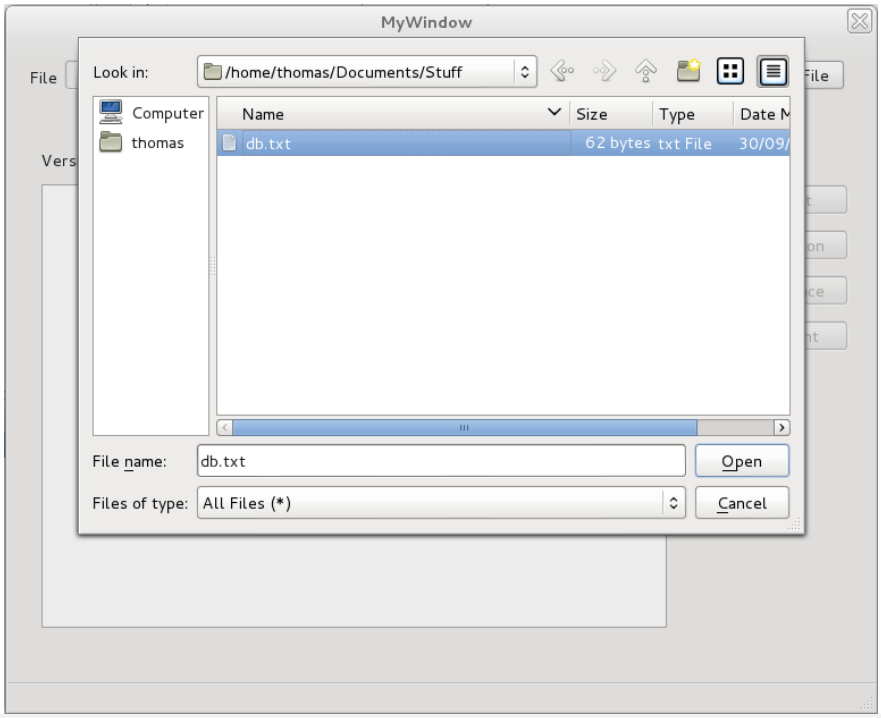
The comment related to the selected version of the file from the table view will be displayed by clicking on the “Show Comment” button.

USAGE

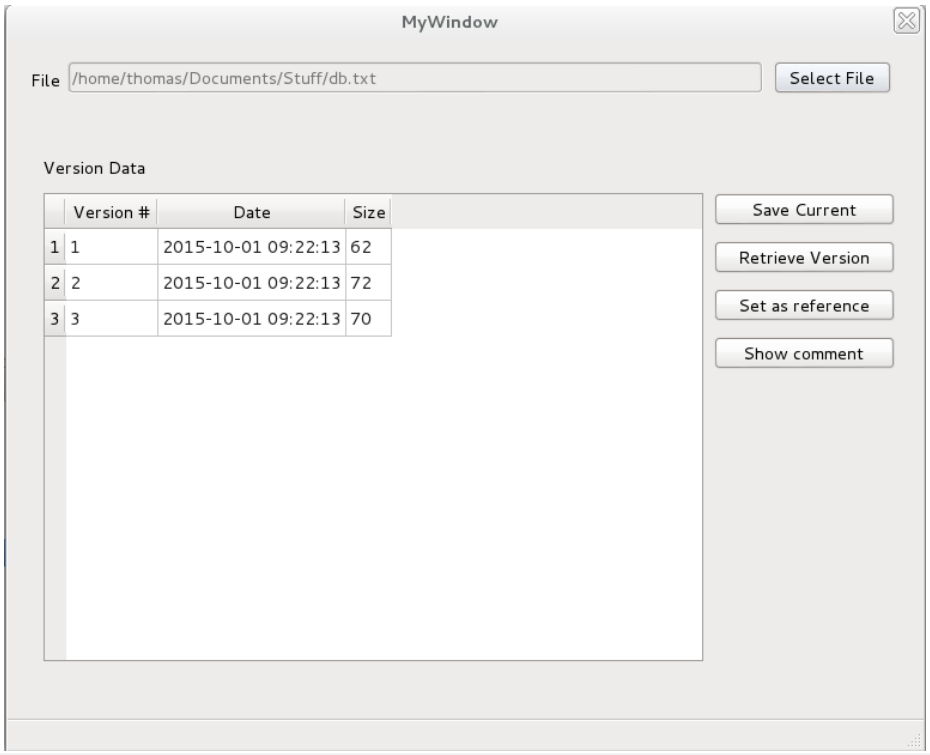
The initial main window gives user an option to select file with “Select File” button.



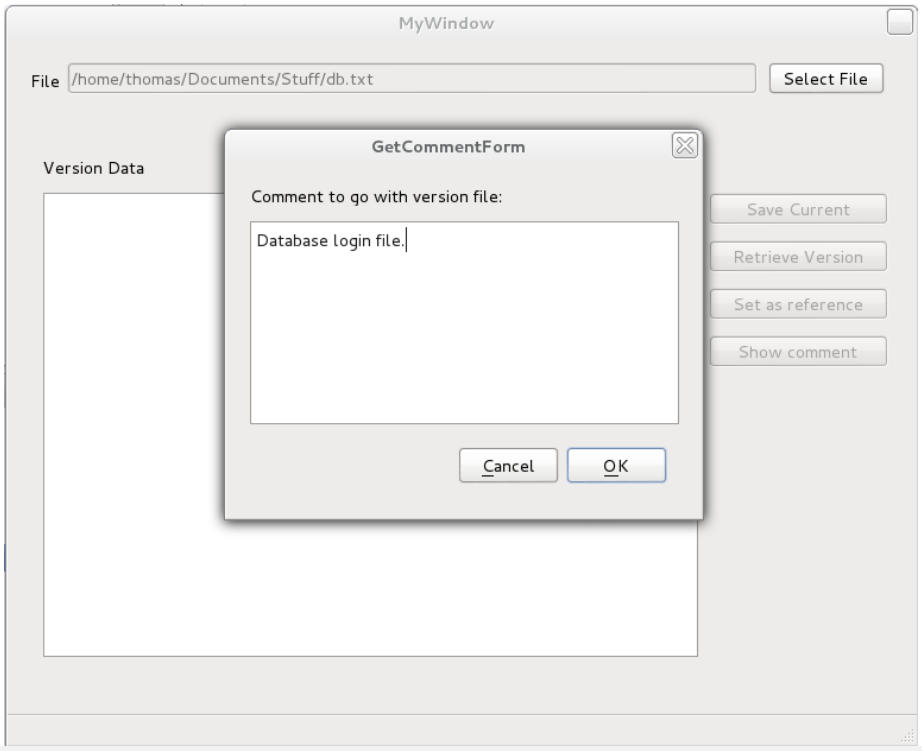
After clicking on “Select File”, the choose file window will appear for the user to select a file.



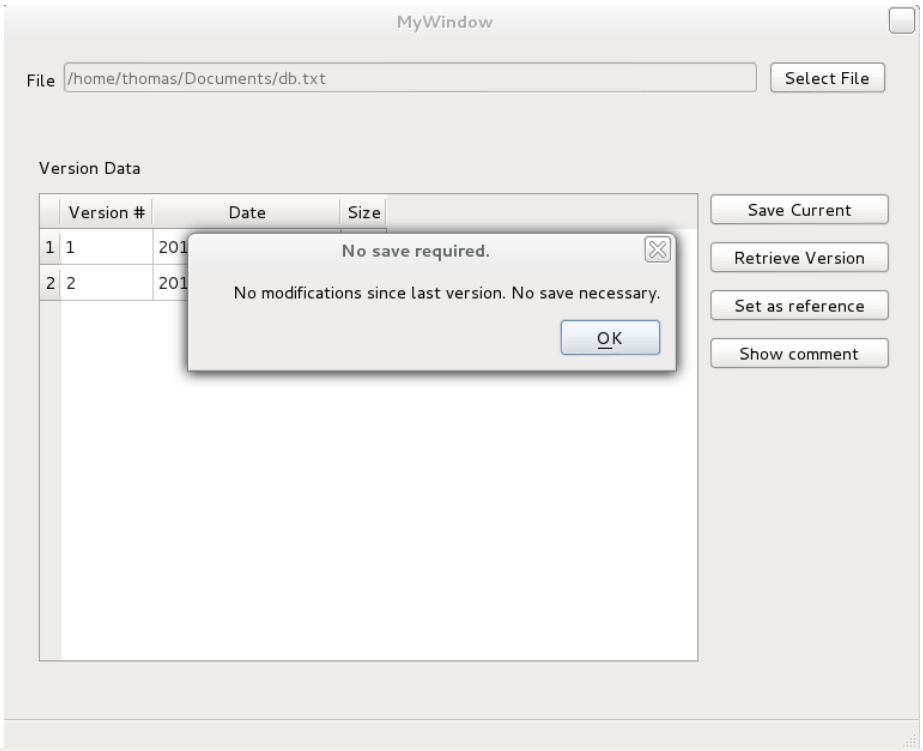
If the file is already saved in the persistent storage, stored versions of this file will be shown in the table.



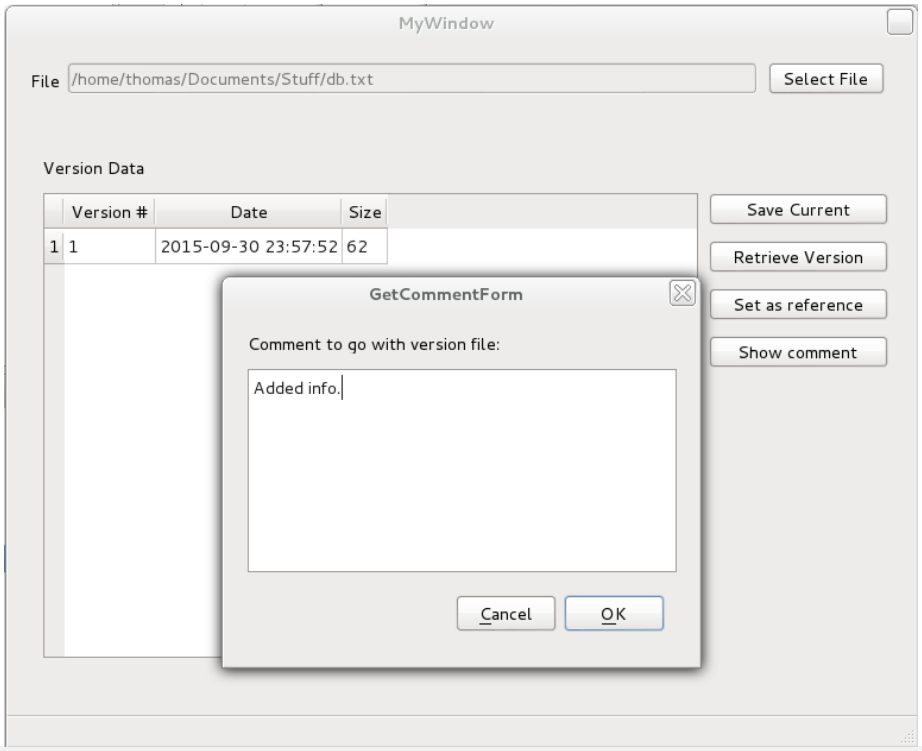
Otherwise, the program will create the initial version in the database and ask user for a comment to go with the initial file version.



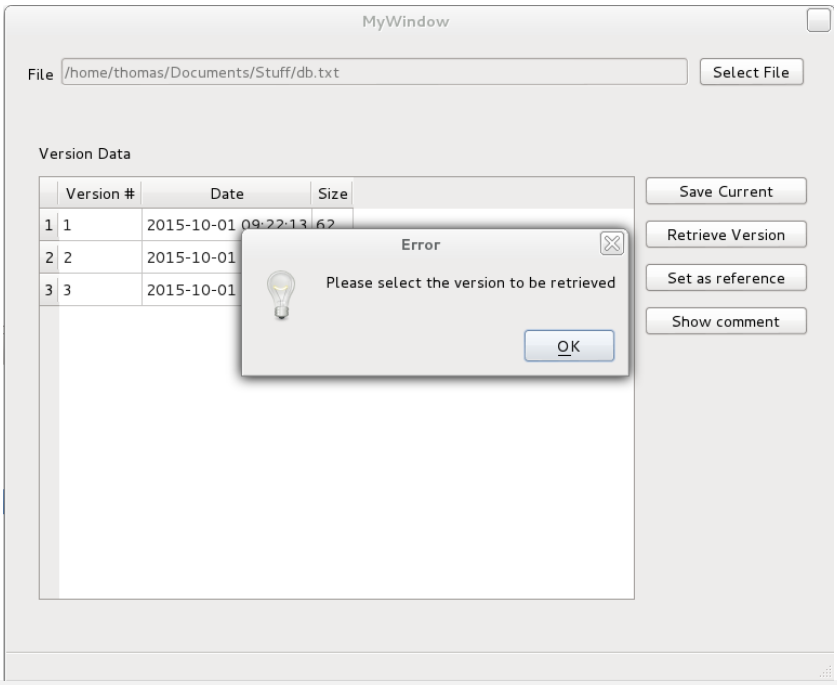
If the user attempts to save the current version of the file that has not been changed, the message box will show up to inform the user that there is no need for saving the current version.



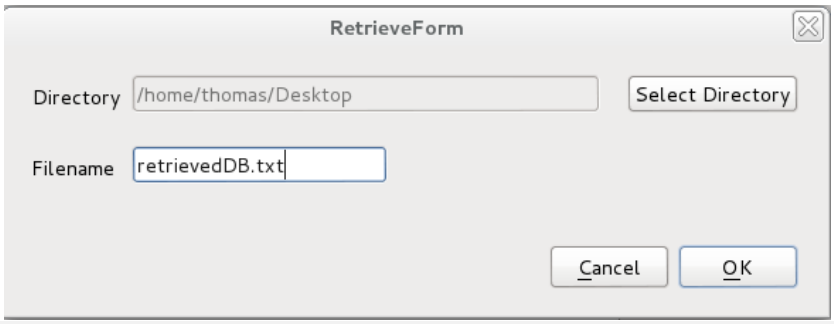
In the case a change in the current version was detected, the user will be asked to supply a comment for the new version and the new version of the file will be displayed in the table.



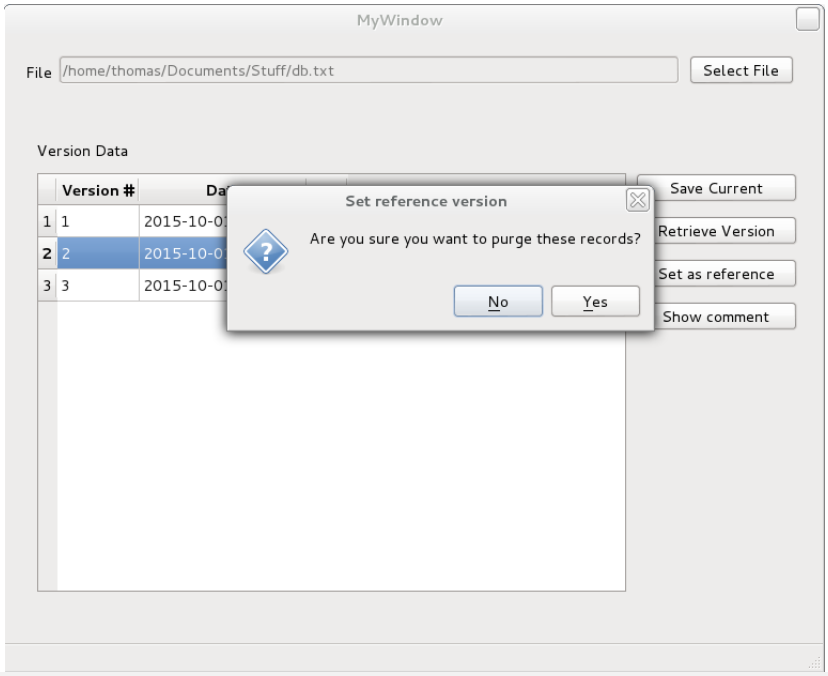
The retrieve version button provides the user the ability to retrieve the version of the file that is selected in the table view. If the file version to be retrieved was not selected in the table, user will be informed.



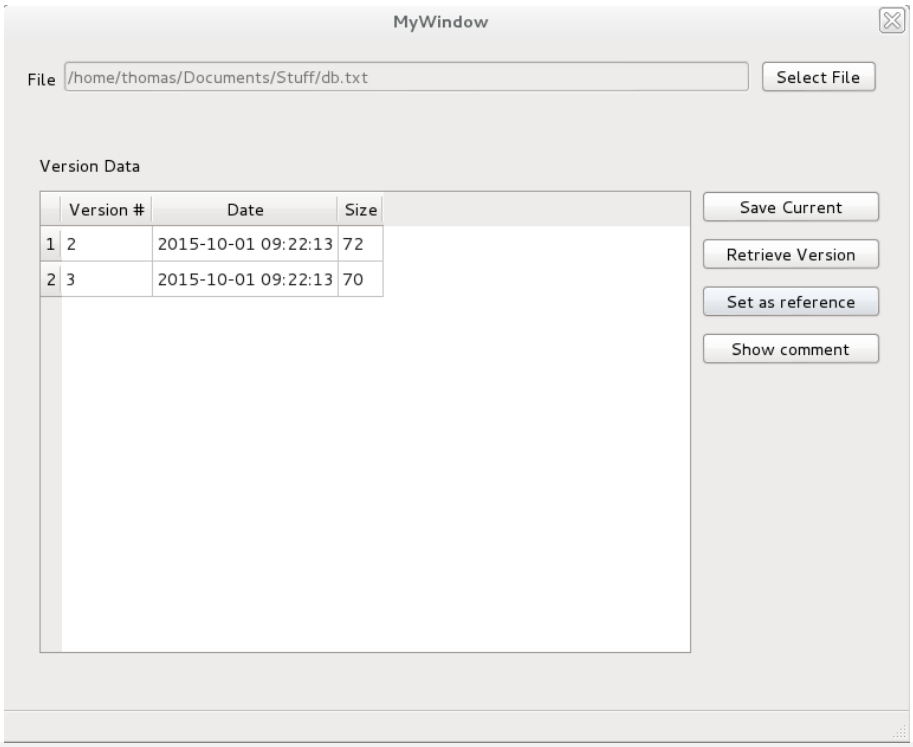
Otherwise, the retrieve form dialog will show up asking user to specify the directory where the retrieved version will be saved, along with the filename.



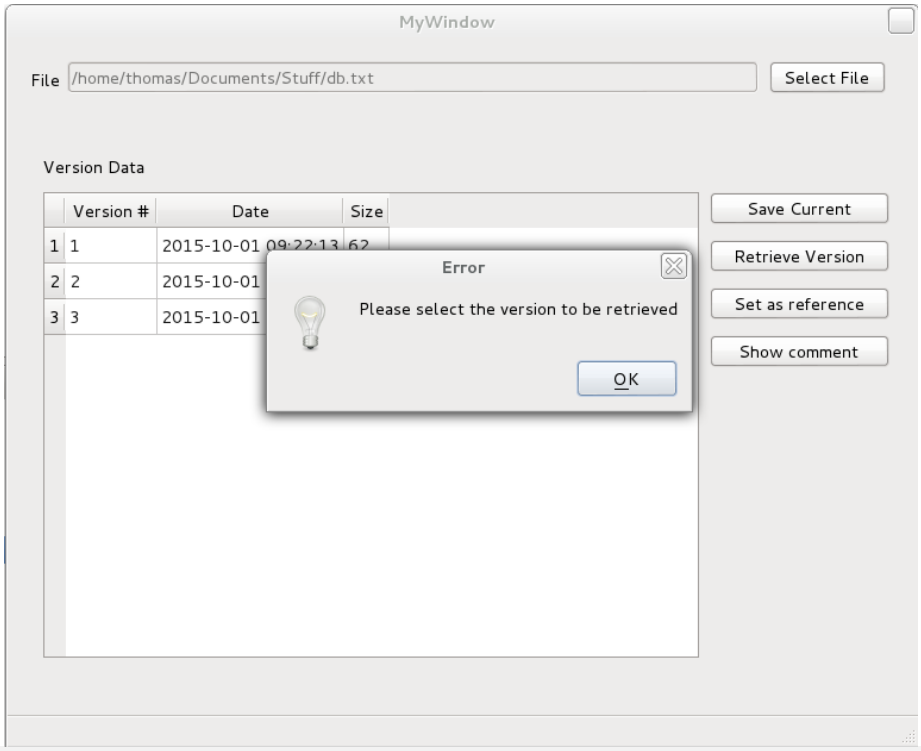
Set as reference option allows user to select version that will be used as a reference to delete all previous versions of the file that are not needed anymore. The program will ask for confirmation to proceed with deletion.



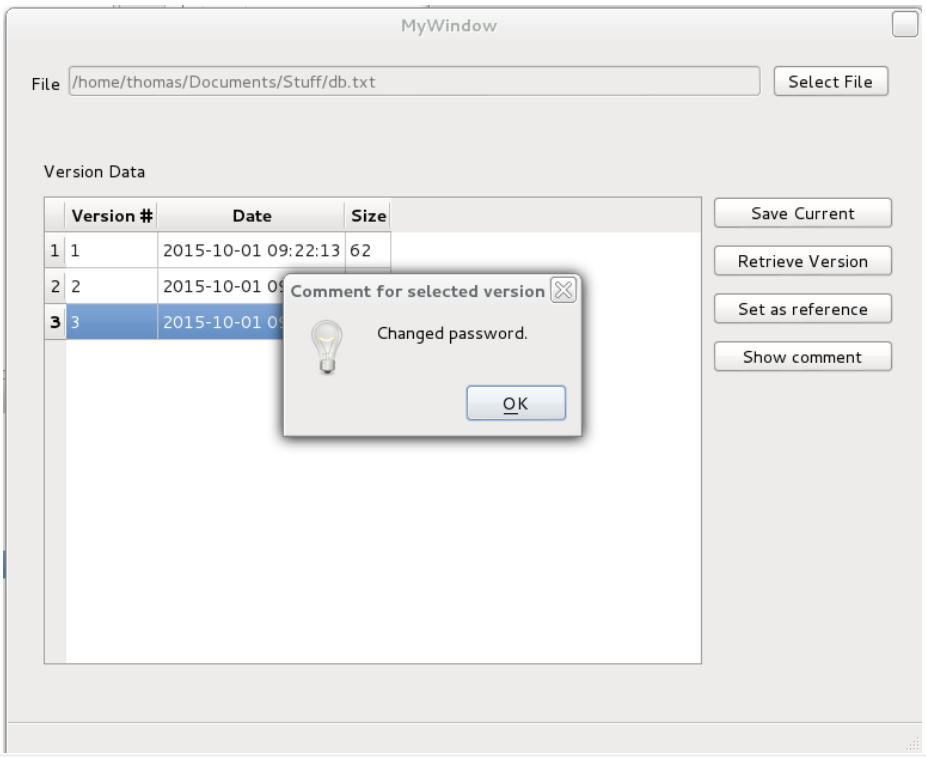
After a purge has completed, the program will then show only the available versions



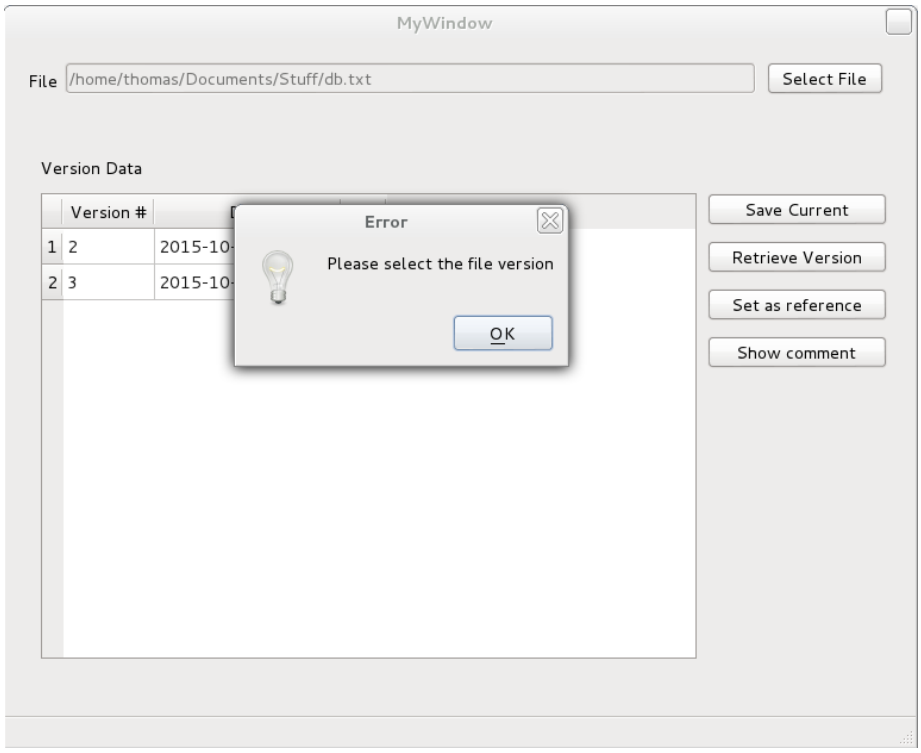
When attempting to retrieve a version when the reference version is not selected in the table, the program will notify the user.



If a user wishes to view the comment related to specific version, the show comment button will invoke the information dialog with the comment of the selected version.



In case the user did not select a version, the program will notify the user.



FUNCTIONAL REQUIREMENTS

The program specification had 9 functional requirements that needed to be met.

ID	TITLE	PRIORITY	STATUS
1	Create Initial Archive	1	Completed
2	Detect Changes	2	Completed
3	Save Modified Version	3	Completed
4	Display a summary of versions in storage	4	Completed
5	Retrieve Chosen Version	5	Completed
6	Show comment associated with version	6	Completed
7	Use Compression	7	Completed
8	Store incremental changes	8	Completed
9	Discard Old Versions	9	Completed

DELIVERABLES OF MEMBERS

THOMAS NIXON – TN941

- Researched the benefits of using MySQL
- Designed new database layout
- Created database diagrams
- Wrote SQL code for database
- Created the SQL statements and C++ code required for the program
- Created initial document in Google Docs for outlining responsibilities and requirements
- Defined layout of classes and member functions and their interaction (With Phil Edwards)
- Implemented VersionRecord member function code
- Moved documents from Google Docs to Git
- Wrote code to handle compression of files and implemented it in required functions

IVANA OZAKOVIC – IO447

- Created GUI main window and dialogs in QtBuilder.
- Created all the GUI functionality in collaboration with Josh.
- Took GUI screenshots for the report.
- Created Program Presentation section for the report.
- Created GUI Implementation and Planning section for the report.
- Declared data members and functions in the VersionRecord.h.
- Created VersionRecord.cpp file and set up stubs for declared functions.
- CPPUnit Tests

PHILIP EDWARDS – PME446

- Created GitHub suppository for project
- Created initial class files
- Created the Netbeans project
- Wrote coding standard document
- Fixed indentation issues across Netbeans project
- Moved database connection to a static function in a class
- Add dependency for MySQL to Netbeans project
- Fixed header guard bug
- Added murmur hash to implementation
- Implemented FileRecord
- Implemented FileArchiver
- Implemented RetrieveVersionRecord in VersionRecord
- Fixed Bugs VersionRecord
- Created Wiki for Git
- Created TODO document
- Wrote Git Primer for other members to refer to
- Defined layout of classes and member functions and their interaction (With Thomas Nixon)
- Created a modified version of murmur hash function to read from file.
- Wrote test code for functions in FileArchiver & FileRecord
- Added logging to program
- Made some modifications to database
- Created MySQL database server setup for other group members to use

JOSH COLEMAN – JJC224

- Created all functionality for FileLib.
- Created all CppUnit tests for FileLib.
- Took images of FileLib code (both header/stubs and source/definitions).
- Took images of FileLib CppUnit tests.
- Created the table model for the table view in the GUI.
- Worked on all GUI table/button functionality (in collaboration with Ivana).
- Report on construction phase iterations

NICHOLAS MORGAN – NRM154

- Meeting Reports
    - Timekeeping and note taking
- Documentation
    - Elements and unit testing procedure
    - Supporting code samples and listings
    - Version Management – initial
    - Detailed meeting report – revision
    - Bug log/Testing sample section
- Project collation

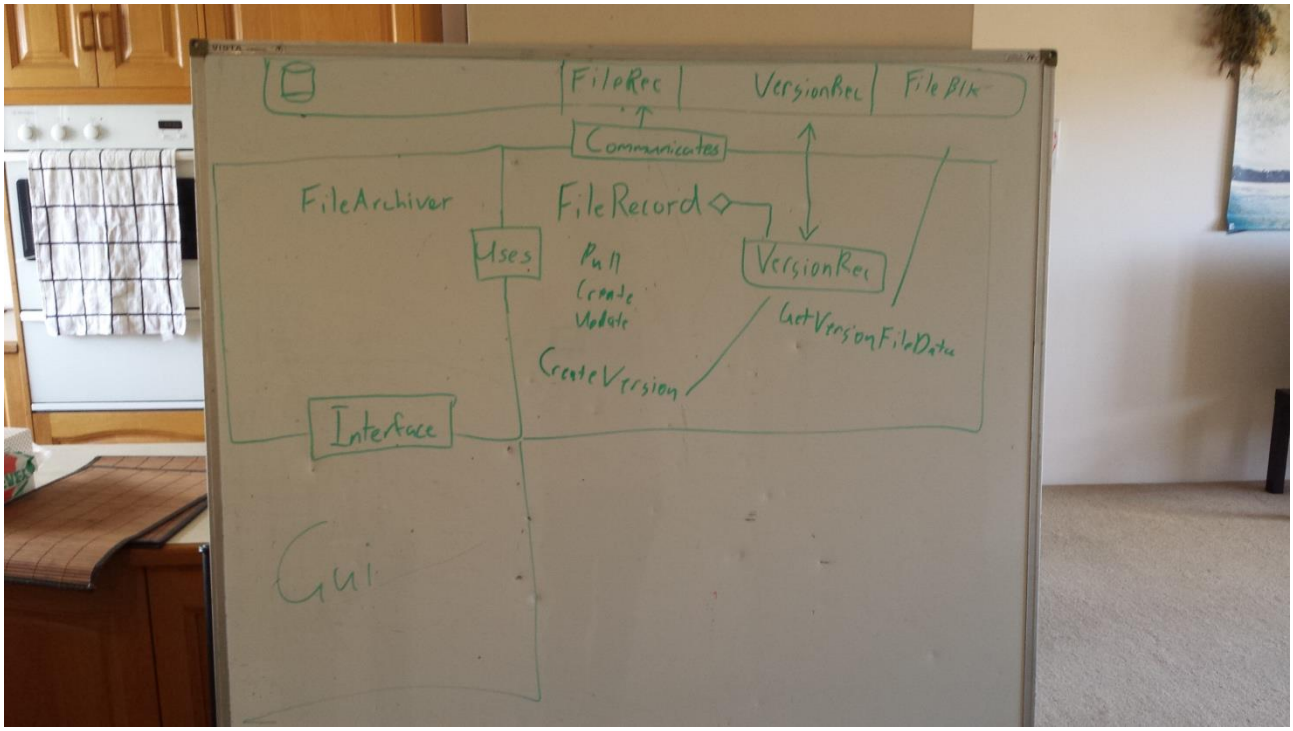
PHILLIP MIHAJLOVSKI – PM976

- Use Case and Sequence Diagrams
- Documentation – Version Management
- Documentation - Diagrams

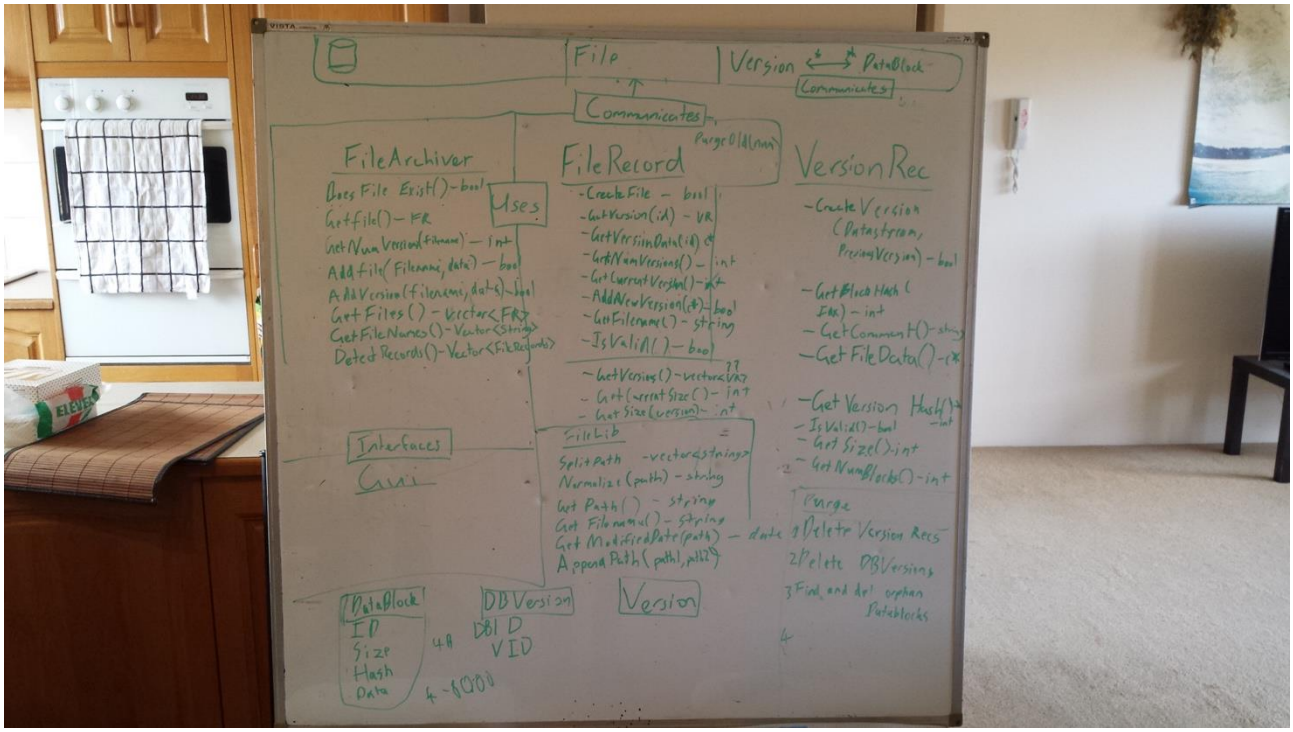
DESIGN & IMPLEMENTATION

FILEARCHIVER

To understand the program better we wrote the main functions on a white board



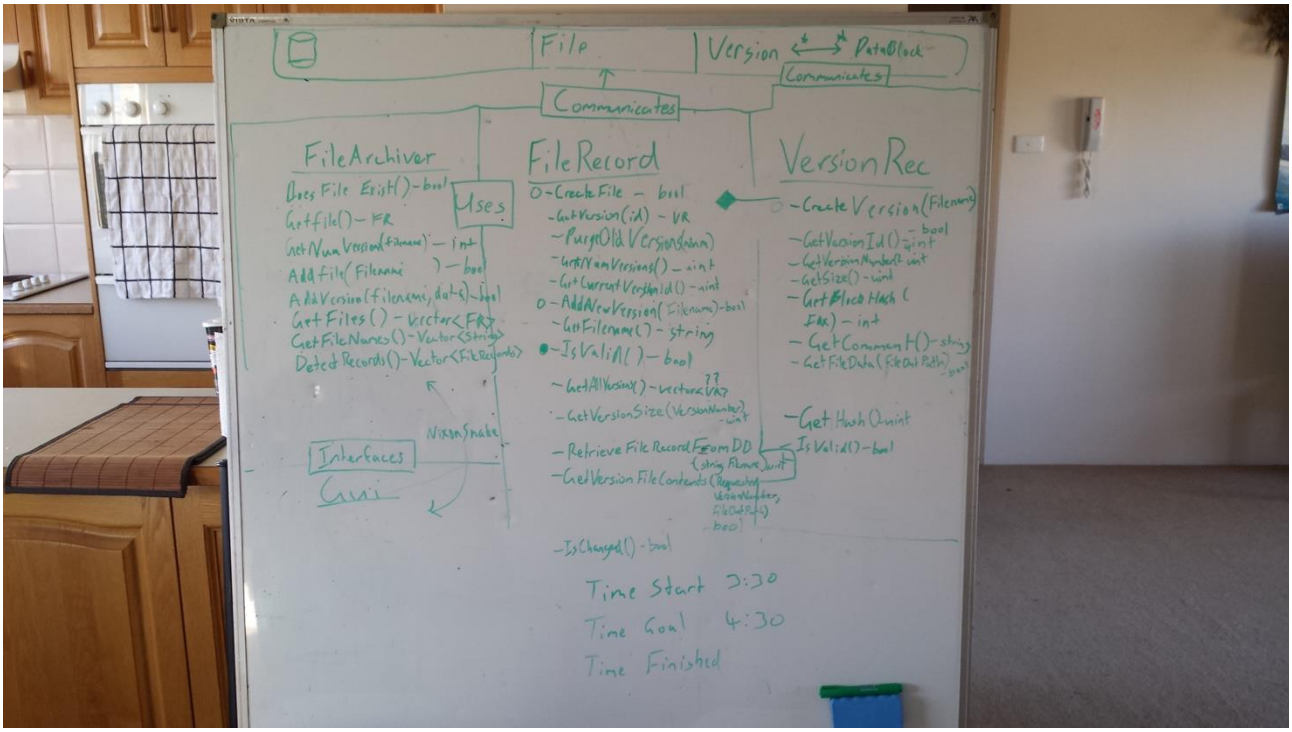
This gave a good starting point. From here we began to flesh it out and understand what functions were needed and how to communicate between the different components.



Expansion from the starting point, designed to provide, at a glance an overview of the structure of the

entire program, to provide insight into program flow and to highlight any area's where improvements could be made.

The further we got the better we understand what we were working towards. We were able to trim excess functions and remove anything that was doubled up.



All the class files were designed from this initial layout. From here we decided to get some of the core functionality going.

DATABASE CONNECTION

Our program uses a single database connection to reduce load on the database if multiple clients are connecting to the same version database. We have a global function which manages the database connection and passes a pointer to any part of the program that requires access to the database.

BACKEND DESIGN

The design of the backend was a collaborative effort primarily between the Lead Implementer and the Data Persistence Specialist with input from the implementers of the GUI. During this design phase the database schema, the backend API, and the associated classes were designed. This enabled all incumbents to ensure that the correct data was stored, that there was a useful interface to the backend, and that all functional requirements were met.

CREATING A FILERECORD

FileRecord is a class that should have a direct correlation with an entry in the database. It provides a function IsValid() which can be used to check that the FileRecord is safe to use. If this is true we can us it to pull information from the database or to commit new versions.

FileRecord provides easy access to get a record from the database through the constructor, as well as providing an interface for creating a new FileRecord in the database.

CREATING A VERSIONRECORD

A VersionRecord is created through a valid FileRecord object. A new version record should never be created in other places in the program. Getting a reference to an existing file record can be accomplished with the VersionRecord constructor in a similar way to using filerecord, however filerecord also provides functions for retrieving a VersionRecord.

STORING A FILE IN THE DATABASE.

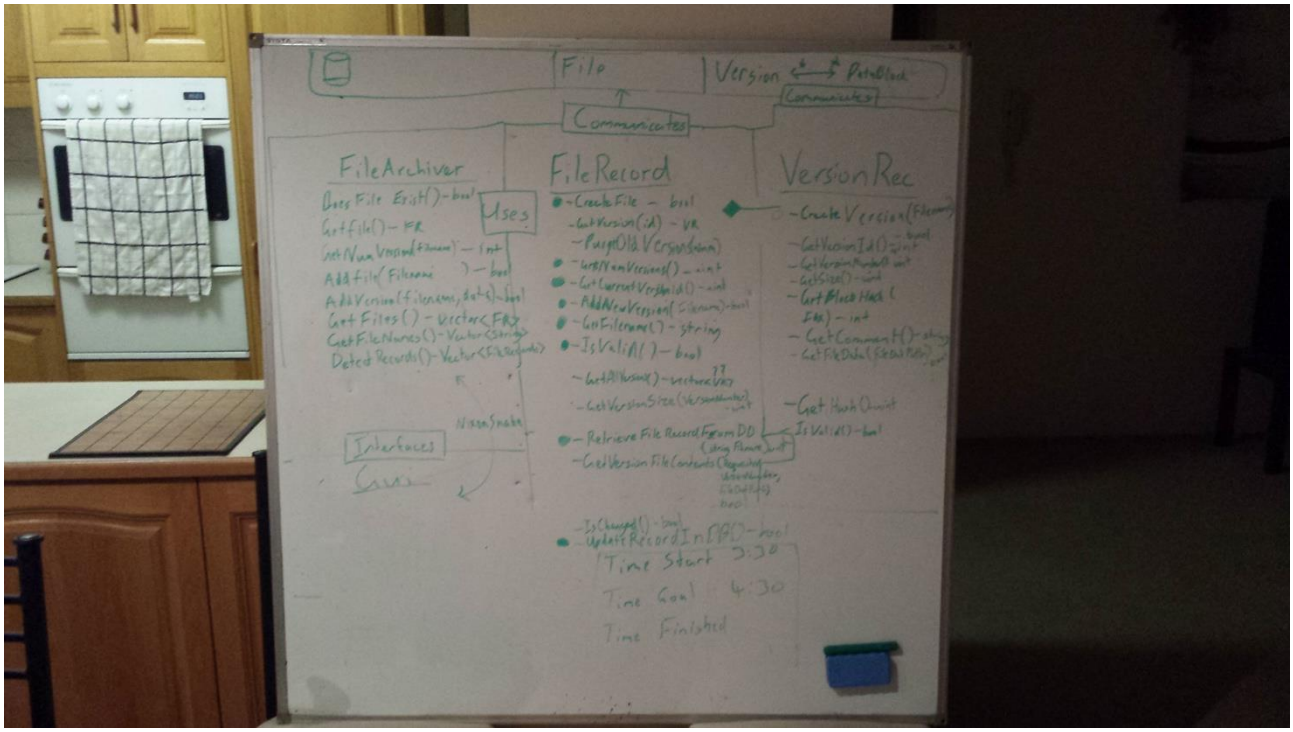
Files are stored in the database when a new VersionRecord is created through FileRecord. The VersionRecord class handles creation of the VersionRecord database entry. It also handles checking for duplicate blocks and creation of VtoB records which associate blocks with a version record.

RETRIEVING A FILE FROM THE DATABASE.

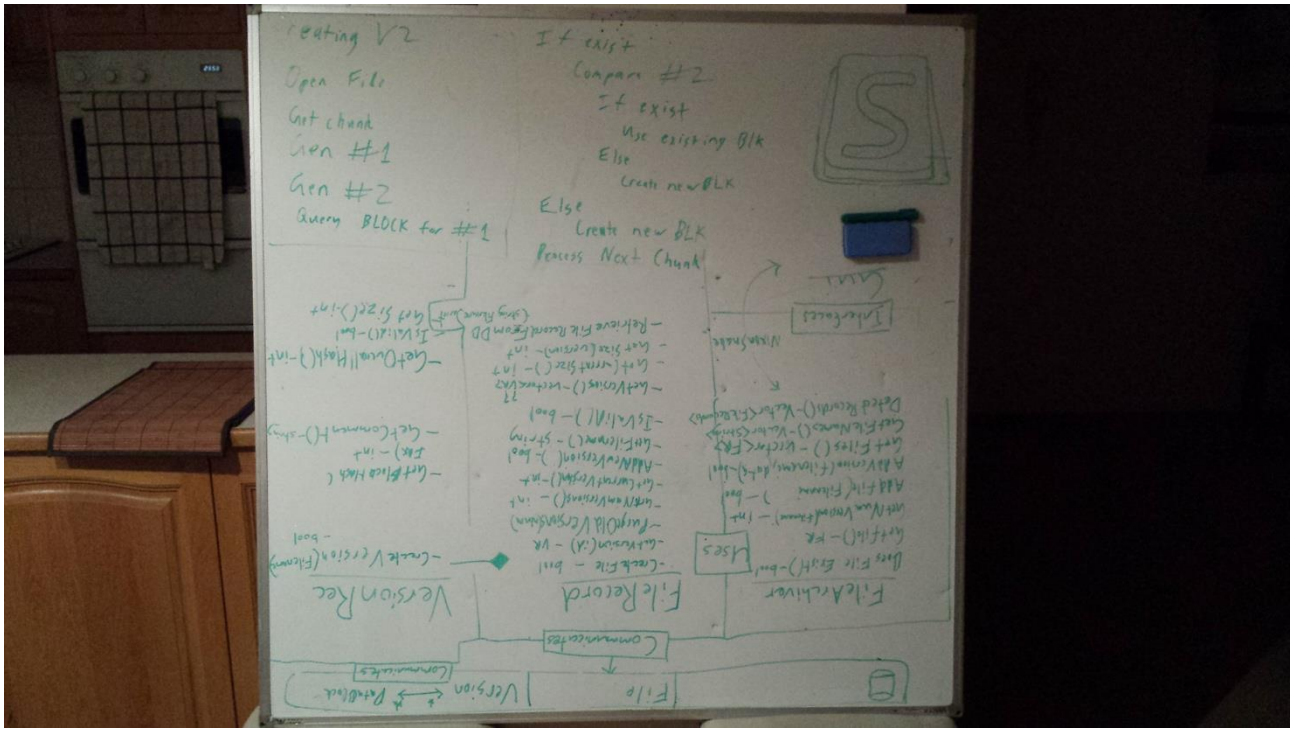
Retrieving a file from the database is accomplished through a valid VersionRecord instance. It will retrieve a list of VtoBs and then gather all required blocks and store them in a file on the database, before finally decompressing the file to the final destination.

STORING MULTIPLE VERSIONS.

Multiple versions are handled easily by associating a VersionRecord with a FileRecord in the database. Actual file data is broken up into blocks of 4 or 8kb which are verified as unique. A block will never be stored in the database twice. A version then has 1 or more VtoB entries associated with it, which link a block with the VersionRecord in the correct order.

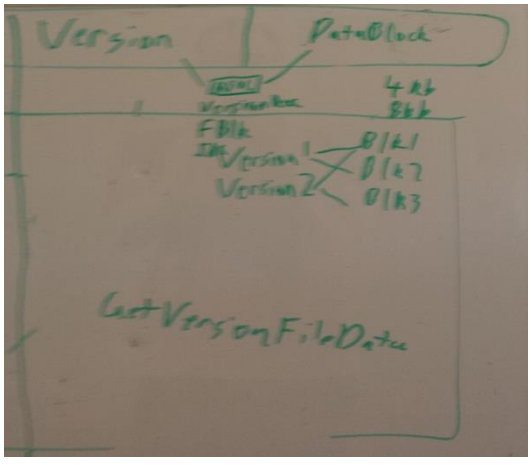


The whiteboard was also used to help us come up with logic in some of the functions. That way two people could look at it and make sure it made sense before implementing.

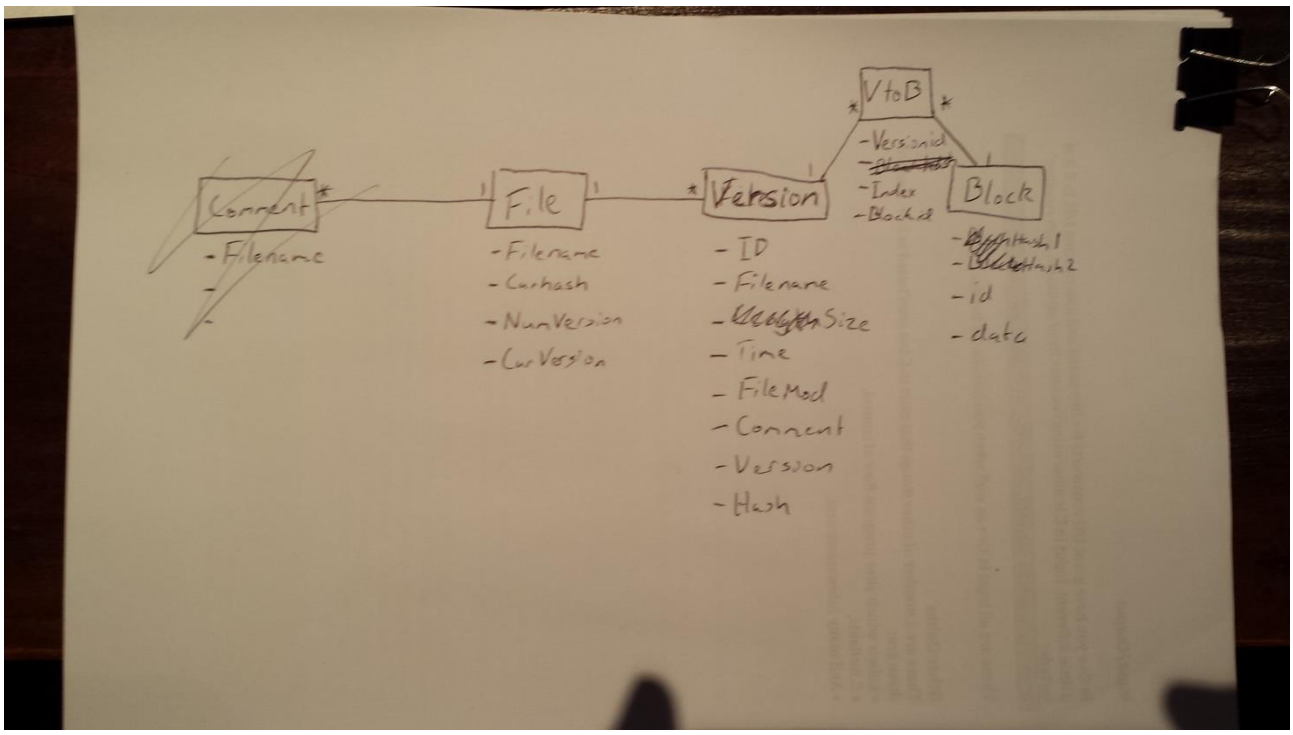


DATABASE

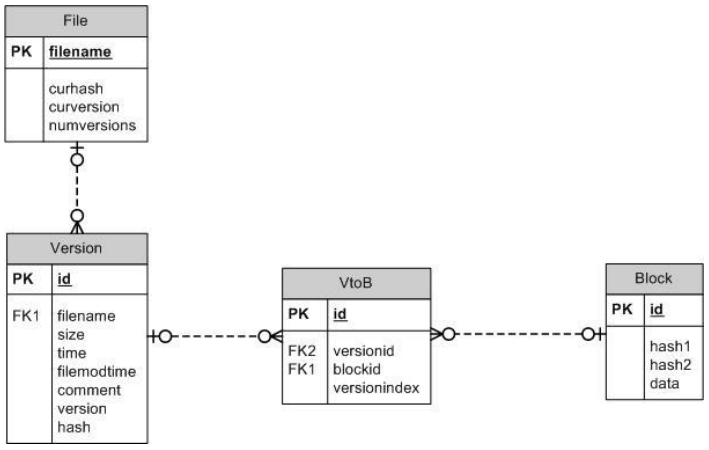
When looking at how to store the data of the file we decided to break the file down into blocks. These blocks would be linked to a version with the intermediate table VtoB which would keep track of the blocks required for a version file and the appropriate index. This allowed for a many to many relationships between Version and Block so that we could store blocks that were the same under different versions without needing to duplicate data.



After deciding on how to store the files we redesigned the database to better reflect what we wanted to achieve. Redundant tables and fields were removed. Names of tables were modified to better reflect the data within them and reduce confusion.



Finally, an ERD was created to show how the database worked. This was useful if anyone needed to refer to the database or see where data was stored in the database. The relationship between the tables is also shown. You can see that there are two hashes in the Block table. This was done reduce the number of collisions possible. This is very important in the Block table as a collision would destroy the integrity of our version files. When a hash1 is the same as a block that is stored it generates a second hash (hash2) which has a different seed. We found this a suitable solution. Below you can see the ERD and the first version of the database creation code in SQL.



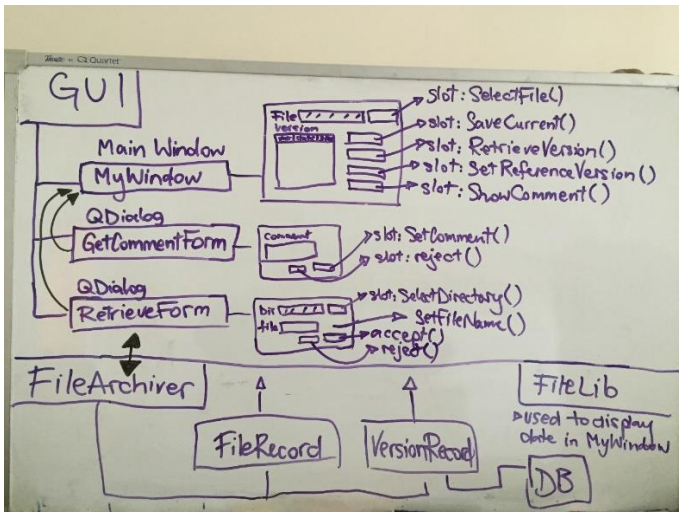
db.txt db.sql

```
create table File (
    filename varchar(767) not null,
    curhash int(11) unsigned,
    curversion int(11) unsigned,
    numversions int(11) unsigned,
    primary key (filename)
);

create table Version (
    id int(11) unsigned not null auto_increment,
    filename varchar(767),
    size int(11) unsigned,
    time int(11) unsigned,
    filemodtime int(11) unsigned,
    comment mediumtext,
    version int(11) unsigned,
    hash int(11) unsigned,
    primary key (id),
    foreign key (filename) references File(filename)
);


create table Block (
    id int(11) unsigned not null auto_increment,
    hash1 int(11) unsigned,
    hash2 int(11) unsigned,
    data mediumblob,
    primary key (id),
    index(hash1),
    index(hash2)
);


create table VtoB (
    id int(11) not null auto_increment,
    versionid int(11) unsigned,
    blockid int(11) unsigned,
    versionindex int(11) unsigned,
    primary key (id),
    foreign key (versionid) references Version(id),
    foreign key (blockid) references Block(id)
);

alter table Version auto_increment = 1;
alter table Block auto_increment = 1;
alter table VtoB auto_increment = 1;


delete from VtoB;
delete from Block;
delete from Version;
delete from File;


drop table VtoB;
drop table Block;
drop table Version;
drop table File;
```

GUI DESIGN PROCESS

In the process of getting the GUI started, the initial task was to create the main window of the program and its dialogs using Qt Designer, which generated all necessary files for the GUI elements in the NetBeans project.

Next thing on the agenda was to set up all needed slot functions and connect them to the elements of the GUI. After a discussion with other implementers to understand how different components of the program will interact with each other, diagram was drawn on the whiteboard to provide an overview of which elements will contain each functionality, and how it will interact with rest of the program.



The following image is the example of how the diagram shown was used to start off the MyWindow class.

```
21 MyWindow::MyWindow() {
22    widget.setupUi(this);
23
24    widget.saveCurrentBttn->setEnabled(false);
25    widget.retrieveVersionBttn->setEnabled(false);
26    widget.setReferenceBttn->setEnabled(false);
27    widget.showCommentBttn->setEnabled(false);
28
29    //connect SelectFile() with selectFileBttn
30    connect(widget.selectFileBttn, SIGNAL(clicked()), this, SLOT(SelectFile()));
31    //connect saveCurrentBttn with SaveCurrent()
32    connect(widget.saveCurrentBttn, SIGNAL(clicked()), this, SLOT(SaveCurrent()));
33    //connect RetrieveVersion() with retrieveVersionBttn which will open RetrieveForm
34    connect(widget.retrieveVersionBttn, SIGNAL(clicked()), this, SLOT(RetrieveVersion()));
35
36    //connect setReferenceBttn with SetReferenceVersion() to delete unnecessary file versions
37    connect(widget.setReferenceBttn, SIGNAL(clicked()), this, SLOT(SetReferenceVersion()));
38    //connect ShowComment() with showCommentBttn
39    connect(widget.showCommentBttn, SIGNAL(clicked()), this, SLOT(ShowComment()));
40 }
```

From the above whiteboard diagram it can be seen that the GUI will be communicating mostly with FileArchiver class, but also using FileRecord and VersionRecord class through FileArchiver.

This was a helpful way to start implementing the functionality of the slot functions. First slot function needed to be defined was SelectFile(), as all other functionality of the program is based on the file that the user has selected to initiate the file or retrieve version of the chosen file from the persistent storage.

As the backend code was initially tested separately from the GUI functionality, the implementation with the GUI was pretty straight forward, as the test code provided insight of the usage of different class objects and how they interact.

The most time was spent on the decision which model will be used to display retrieved file versions in the table view. In order to find an easier and more concise solution than writing our own QAbstractTableModel class, we have researched the abilities of QtCore classes and decided to use QAbstractItemModel class. QAbstractItemModel supplied all the functionality required for our program.

The following screenshot shows the usage of table view and QAbstractItemModel to retrieve information for a selected version of the file by wrapping the RetrieveVersionDataForFile function in MyWindow class.

```
QStandardItemModel *myModel = new QStandardItemModel(fileRec.GetNumberOfVersions(), 3, this);
myModel->clear();

myModel->setHorizontalHeaderItem(0, new QStandardItem(QString("Version #")));
myModel->setHorizontalHeaderItem(1, new QStandardItem(QString("Date")));
myModel->setHorizontalHeaderItem(2, new QStandardItem(QString("Size")));

vector<VersionRecord> versionRecs = fileRec.GetAllVersions();
unsigned int currentRow = 0;

for(vector<VersionRecord>::iterator it = versionRecs.begin(); it != versionRecs.end(); ++it)
{
   myModel->setItem(currentRow, 0, new QStandardItem(QString(boost::lexical_cast<string>(it->GetVersionNumber()).c_str())));
   myModel->setItem(currentRow, 1, new QStandardItem(QString(FileLib::GetFormattedModificationDate(fileRec.GetFilename()).c_str())));
   myModel->setItem(currentRow, 2, new QStandardItem(QString(boost::lexical_cast<string>(it->GetSize()).c_str())));

   ++currentRow;
}

widget.tableView->setModel(myModel);
widget.tableView->resizeColumnsToContents();
widget.tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
widget.tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
widget.tableView->setSelectionMode(QAbstractItemView::SingleSelection);
```

After all functionality was implemented, we focused on making sure that the control flow and error-handling logic of the program solves each issue where users can make erroneous interactions through the GUI.

Each possible user interaction case was considered and we made sure to inform users how to achieve what was intended by supplying information in form of different kinds of message dialogs.

The screenshot below shows how this was implemented in the case where the user tries to retrieve a version of a file, but has not selected the version to be retrieved from the display in the table view.

```
if(!(indexes.size() > 0))
{
    QMessageBox msgBox(QMessageBox::Information, "Error",
                "Please select the version to be retrieved", QMessageBox::Ok, 0);

    msgBox.exec();
    return;

}
```

Finally, after GUI was completely functional and there were no bugs, we made sure everyone was happy with the final product.

DESIGN DIAGRAMS

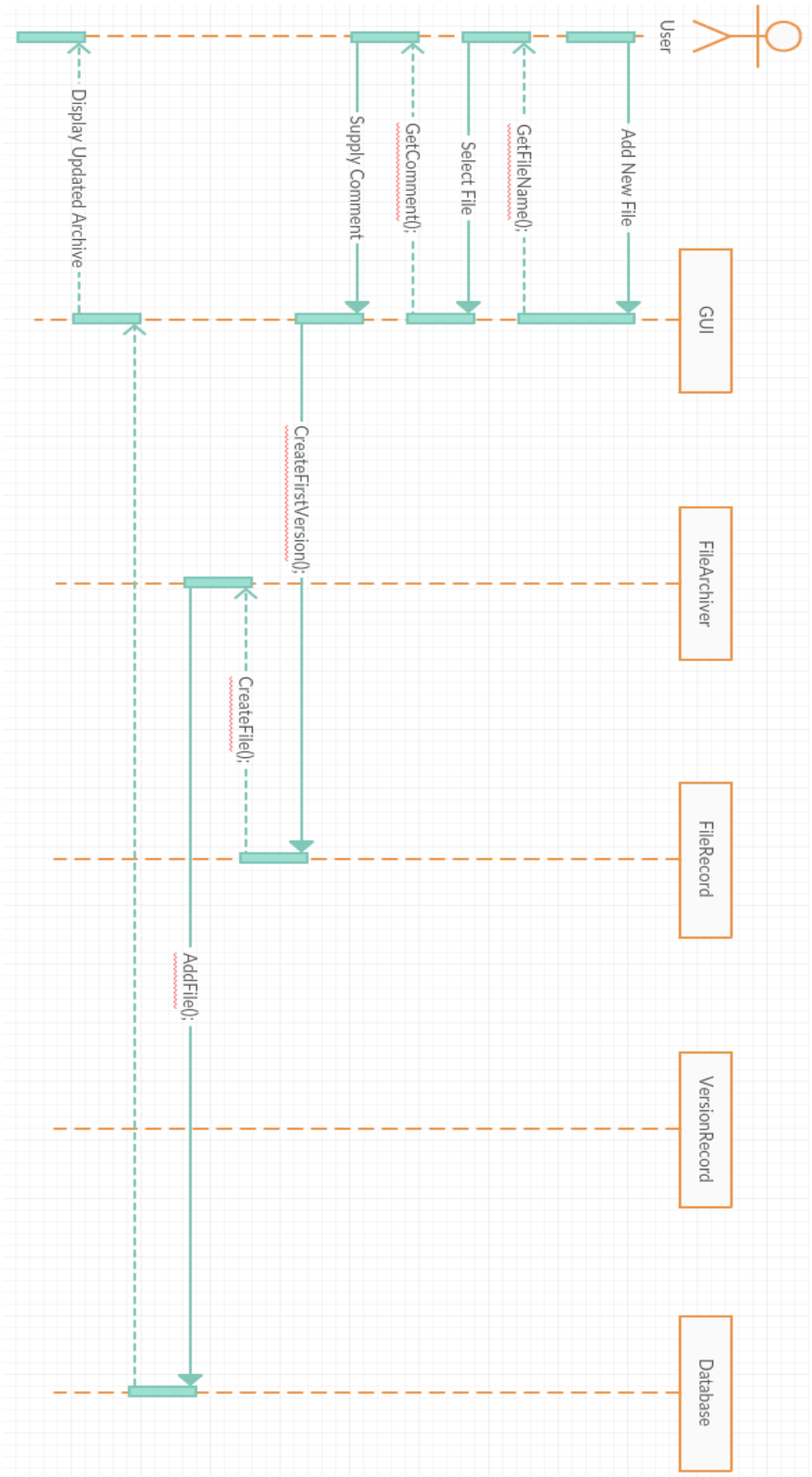
USE CASE

This use case includes the use cases U1 through to U5.

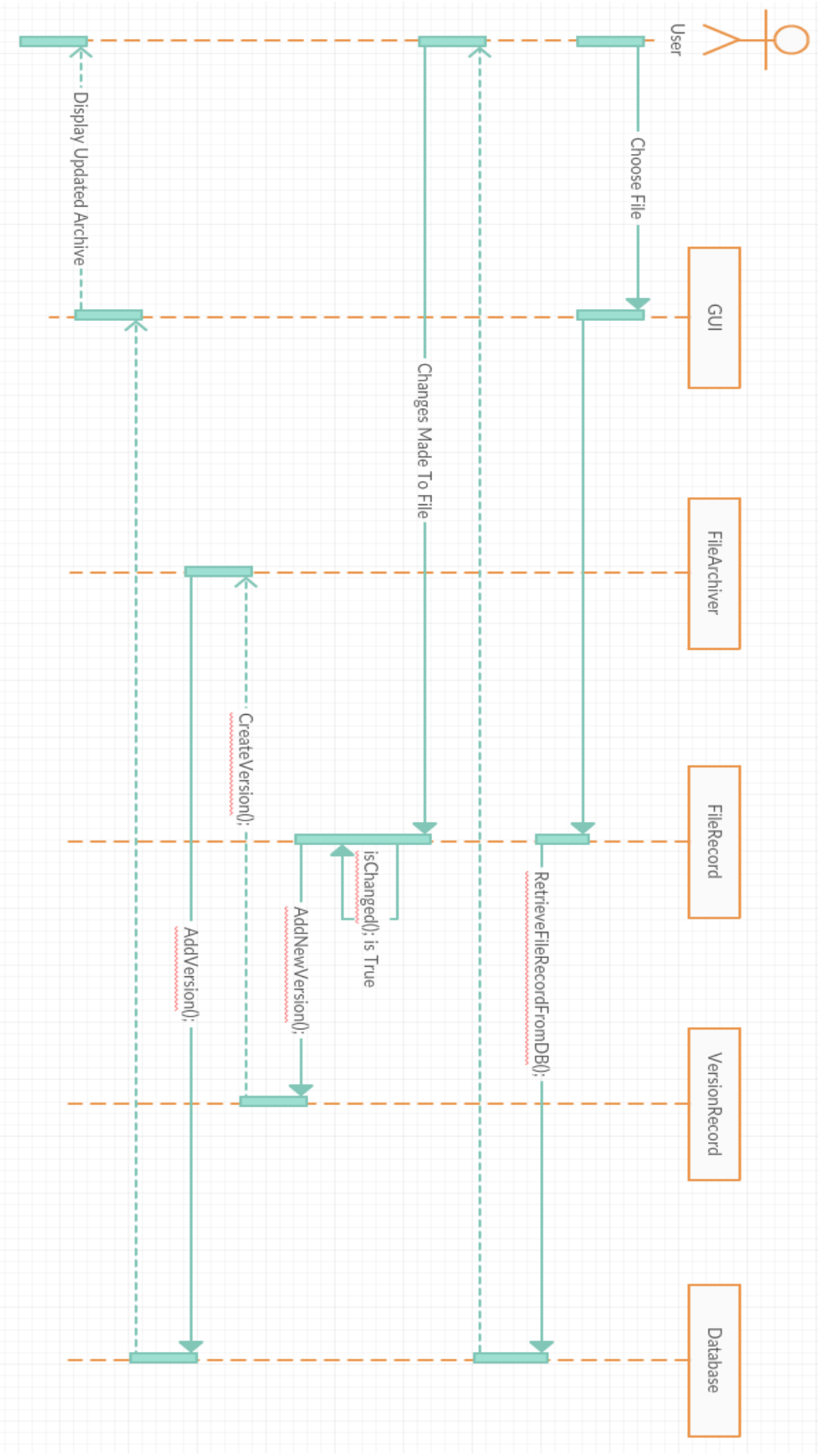
SEQUENCE DIAGRAMS

The sequence diagrams will highlight how the system uses its classes to work together.

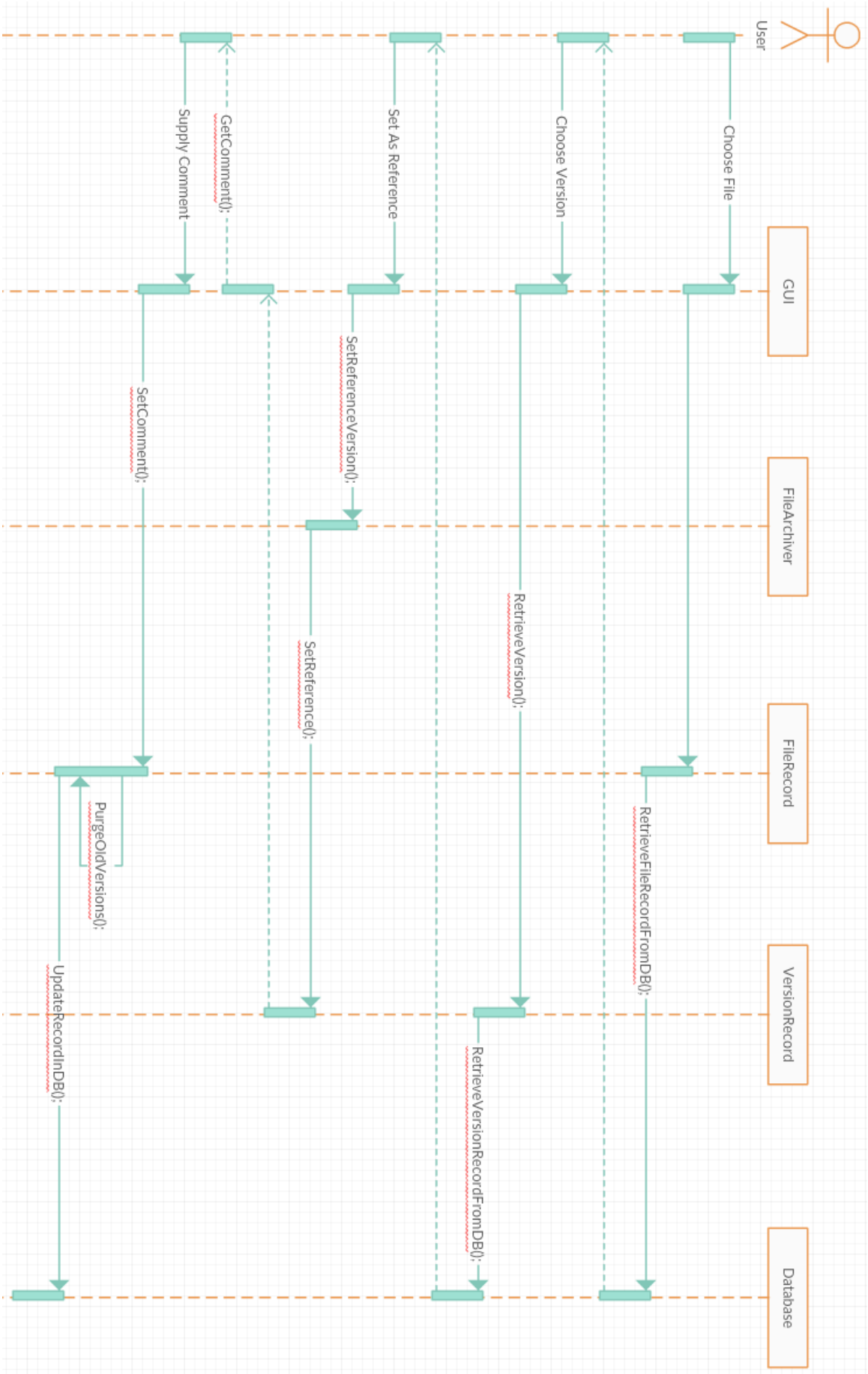
ADDING THE INITIAL FILE TO THE FILEARCHIVER SYSTEM.



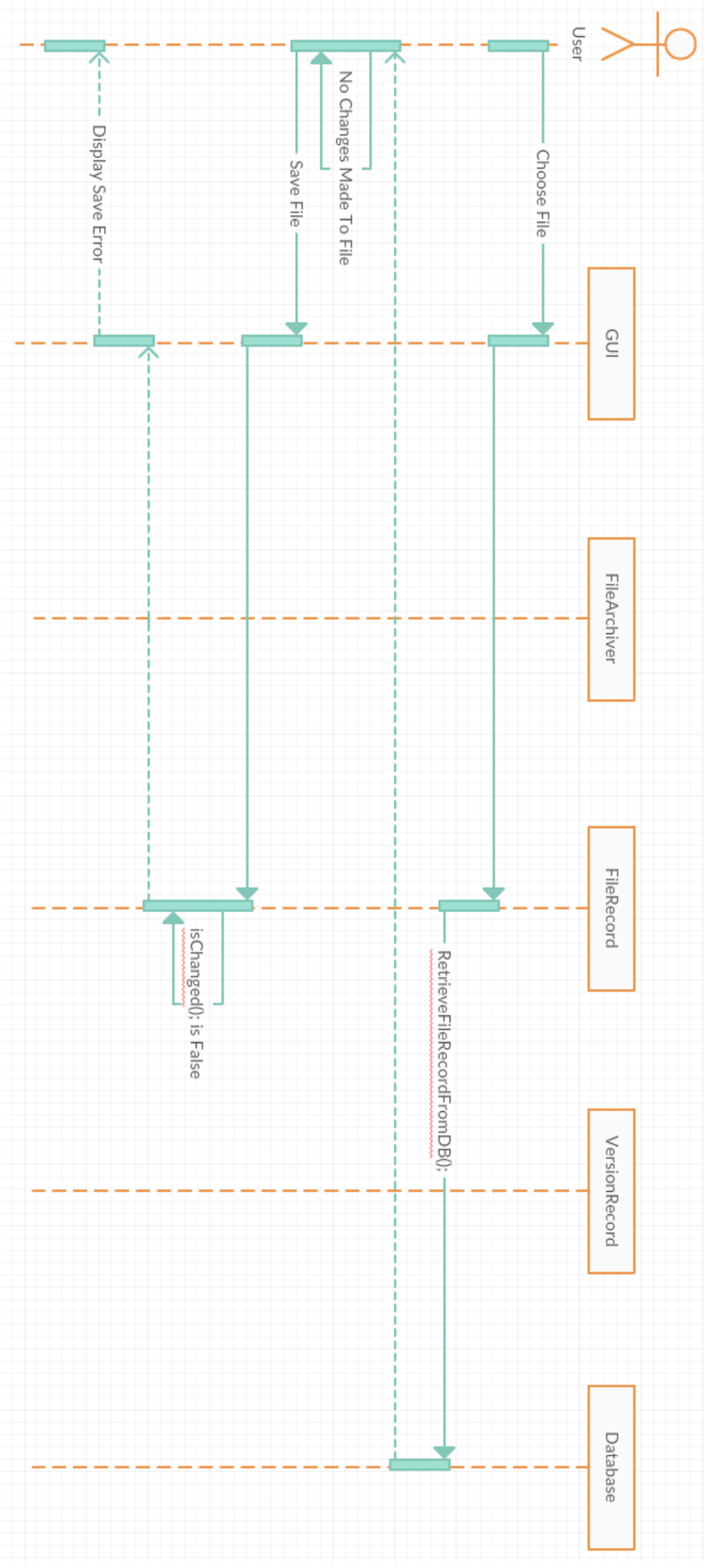
UPDATING A CURRENT FILE IN THE SYSTEM.



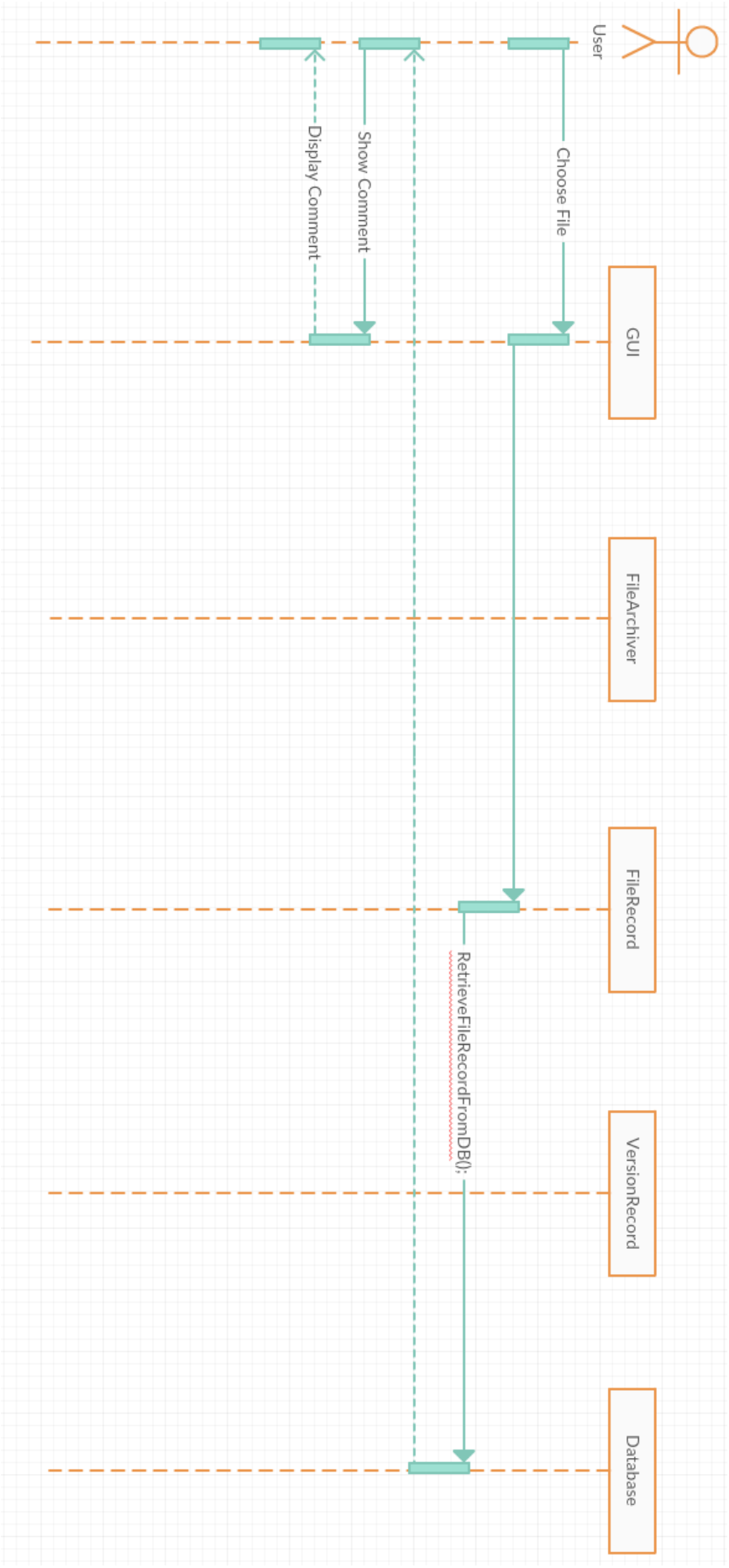
RETRIEVING A CURRENT VERSION AND SETTING AS REFERENCE



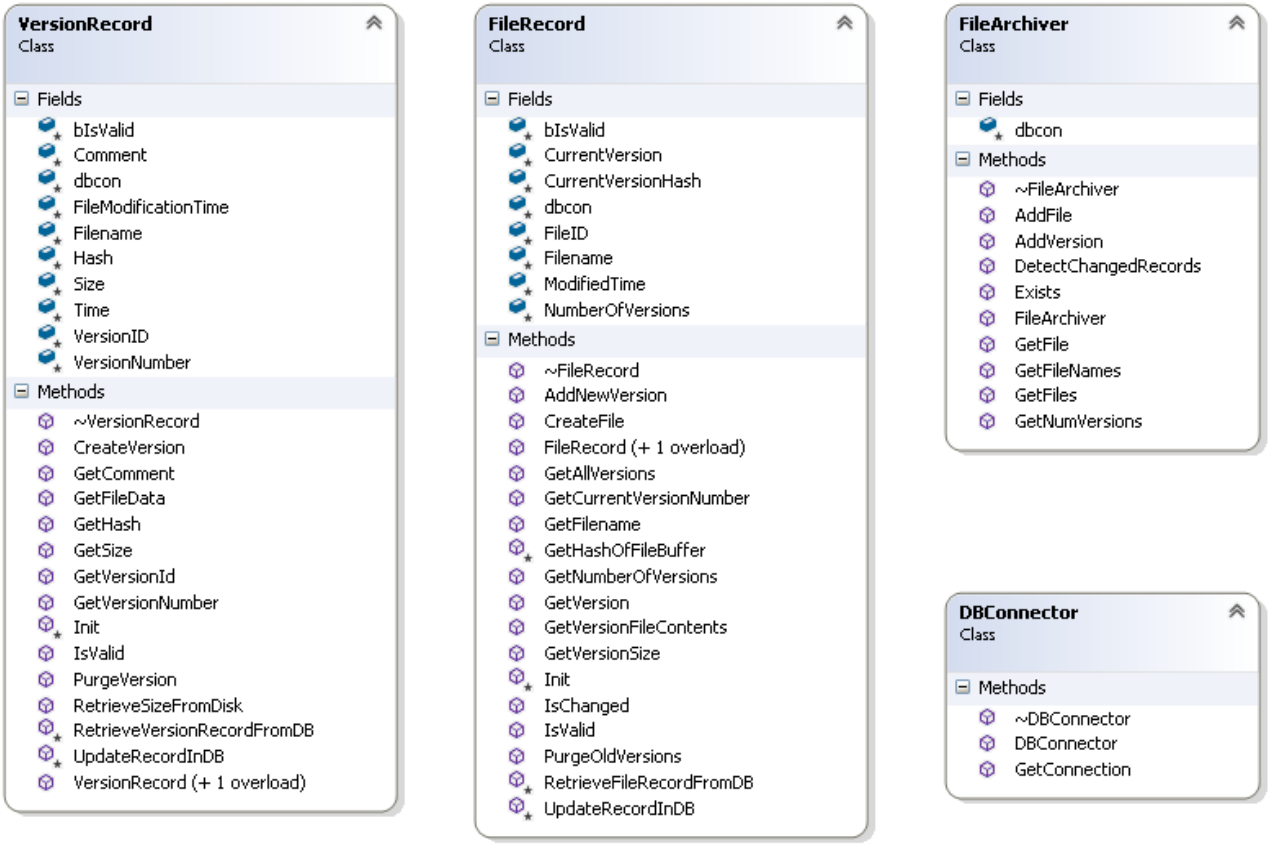
ATTEMPTING TO SAVE WITHOUT HAVING MADE CHANGES



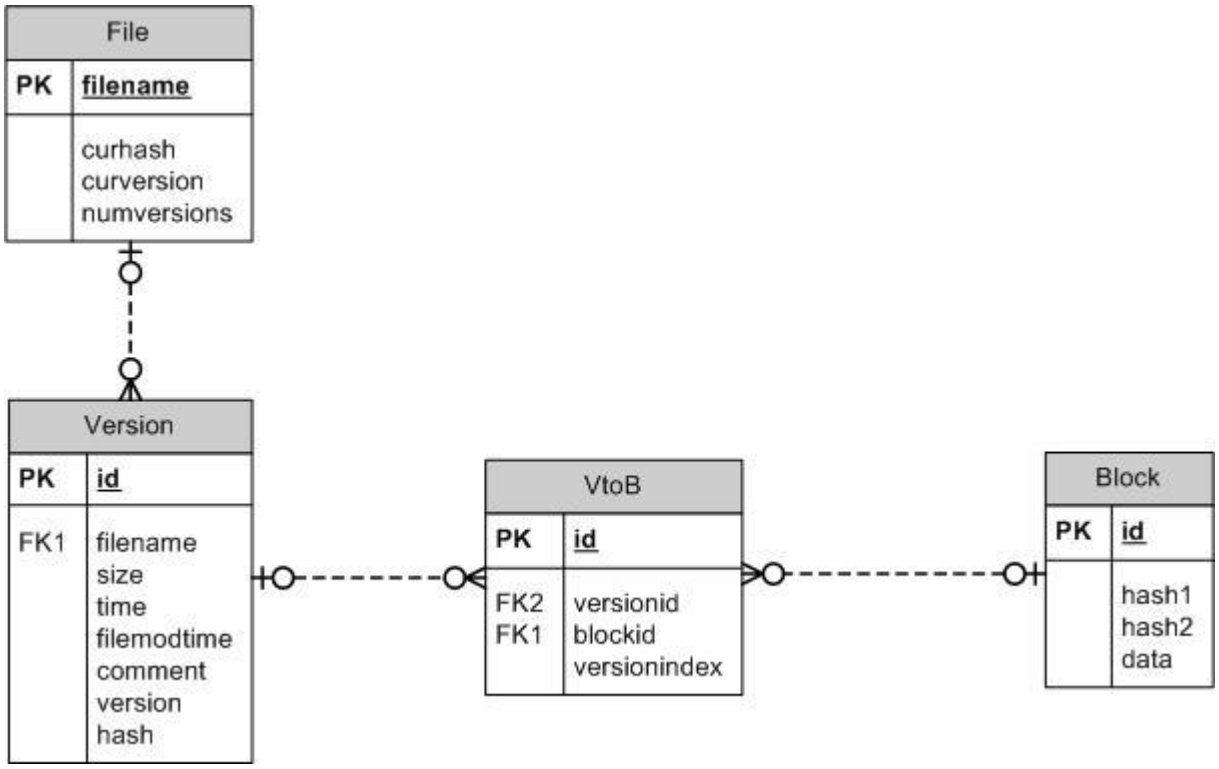
VIEWING THE COMMENT OF A FILE



BACKEND CLASS DIAGRAM



DATABASE DESIGN – ENTITY RELATIONSHIP DIAGRAM



DESIGN CHANGES

Our final database structure and specifics varied from the provided design and diagrams within the project documentation, the above image shows that final structure. Throughout the design process of the project we identified several key areas where we felt the design could be improved to allow the program functionality to easier to implement and reduce possibilities of corruption of data. These changes all had the ultimate goal of simplifying the design and implementation process and to ensure we were creating a high quality, functional program by the end of the project.

DESIGN ITERATION BREAKDOWN

ITERATION 1

12/09/15 – 17/09/15:

Once the members met for the first time, a thorough overview of the program design was collaborated on a whiteboard, producing the UML diagrams. After breaking down the design, Thomas then created the database and populated it with dummy data, and performed the necessary checks to ensure the integrity of the database.

While the database was being configured for the project, each member was assigned to write the skeleton of the components previously identified in the overview:

- Ivana: created the stubs for the VersionRecord class.
- Josh: created the stubs for the FileRecord class and began to work on the FileLib class so that it was ready for use in a range of classes.
- Phil E: created the stubs for the FileArchiver class.

The members went home later this day and were assigned tasks to continue on with. Members collaborated over Skype, keeping up to date with one another.

- Thomas completed the code responsible for connecting the FileArchiver class to the database.

- Phil E completed the base functionality of the FileRec and FileArchiver classes. At this stage, the backend of the program was able to handle the creation of the blob data given a file and retrieve it back to some local file system, building off of FileRec, which stores the actual data that FileArchiver passes to/from the database.

- Ivana began to create the base GUI, while Josh was working on FileLib and CPP unit tests for it.

- On completion of FileLib and its tests, Josh and Ivana collaborated to complete the base GUI, which on completion used the functionality produced by Thomas and Phil. The GUI was tested completely functional in terms of displaying what was in the database.

- Phil M and Nicholas collaborated to create the diagrams that were outlined during the initial meeting (where the UML diagrams were drawn up as a group). The integrity of each diagram was verified as a group post-production.

ITERATION 2

18/09/15 – 21/09/15:

During the implementation of the version control (VersionRecord and related components) in regards to the design of the code and the database, we came to the realisation that we can design it more concisely. So, the database design was refactored in the sense that redundant functionality was omitted.

- Thomas created an ERD of the new design. Josh completed FileLib at this point, but some of the functionality was omitted in source files relying on it in the end, as better implementations became apparent.

- Thomas and Phil E worked on refactoring the definitions of VersionRecord.

- Ivana and Josh worked on creating and connecting functionality that wraps the new logic of VersionRecord, which was completed.

- Nicholas worked on compiling notes and screenshots taken by the other members and himself.

- Phil M worked on creating the sequence diagrams of the new design.

21/09/15 – 27/09/15:

At this stage, most of the project has been completed. There are some important aspects of the version control to be implemented, but overall, the project is close to completion.

- Thomas created functionality responsible for file/data compression, which also includes the side-functionality of temporary storage being created during the compression/decompression phase of the interaction between the client and the server/database.

- Ivana and Josh completed the functional requirements of the GUI. All buttons slot into their related functions, and all probable user cases have been identified and handled thoroughly. Most dialogs notifying users of error have been completed.

- Phil E and Josh collaborated to write CPP unit tests for the functional requirements of the backend. Completed and tested thoroughly.

  Josh noticed some logic errors associated with version purging, and notified Thomas to fix. It was fixed in a matter of minutes.

Phil E noticed a troublesome segmentation fault in a case where you close the window without doing anything when blackbox testing the GUI, so Josh and Ivana debugged to find a fault in the destructor of the MyWindow class that allowed deletion of a form that hadn't been opened. This was fixed promptly.

- Ivana applied final dialogs for user-error cases in the form of message dialogs. The GUI is now complete in all aspects, apart from one core functionality (old version purging). The stub and button have been interconnected as required, encapsulating pseudo-code of the functionality in VersionRecord responsible for the purging.

- Thomas finalized his part of the iteration by cleaning up code related to compression (CompressionUtils) and also fixed it to use gzip instead of zip.

- Josh, Phil E, and Ivana went through and did a final round of testing while also cleaning up useless comments and redundant code.

- The group was notified over Skype of the progress made: almost complete.

- Nicholas and Phil M are compiling what is required for the report, and begin writing it up.

ITERATION 3

The project is practically complete: final touching up and functional testing to finish off, along with the addition of the final component for the project overall, which is also core functionality of the GUI. That is, MyWindow::SetReferenceVersion (use of VersionRecs::PurgeOldVersions in conjunction with Qt components) has to be completed. Finally, the report needs to be worked on.

27/09/15 – 29/09/15:

- Thomas identified an insertion issue with the database. This was fixed promptly. Thomas then finalised the functionality responsible for purging a given number of old version.

- Josh and Ivana wrapped the purging functionality that Thomas wrote up to work within the GUI and its components.

- Phil E requested that Josh adds functionality to FileLib (as he is responsible for the class) for use in the backend CPP unit tests and main source file of the program to setup and destroy temporary directories used for the compression/decompression phase when interacting with the database. Phil E wrote up the prototype to outline what it should do, and Josh implemented it into FileLib as needed.

- The group as a whole was notified over Skype of the completion of the design and code aspect of the project. The group has confirmed the integrity of the overall project, and the report can begin to be written.

- Nicholas and Phil M have working on the report during this time.

- All other members have now been allocated sub-tasks for the report: collaborating to finalise

ELEMENT LIST

1. File record functionality – (Full functionality Completed during iteration 2)
    a. Create file records – (Completed iteration 1)
    b. Update file records – (Completed iteration 2)
    c. Show version of files – (Completed iteration 2)
    d. Remove versions - (Completed iteration 2)
    e. Add new versions of files – (Completed iteration 2)
    f. Show size information – (Completed iteration 1)
    g. Check if a version has changed –(Completed iteration 1)
2. Version record functionality
    a. Retrieve version record from database (Completed iteration 1)
    b. Update Version information from a database (Completed iteration 2)
    c. Create new version records (Completed iteration 1)
    d. Show file size information (Completed iteration 1)

    e. Remove a version from the database (Completed iteration 2)
3. Gui Functionality (full functionality with data integration completed iteration 3)
    a. Provide graphical interface for workings of the program (Gui base completed at iteration 1)
4. File Archiver (Full functionality achieved during iteration 3)
    a. Check for already existing files (Completed during iteration 2)
    b. Get version info (completed at iteration 2)
    c. Insert a new file into database (Completed at iteration 2)
    d. Add versions to files (completed at iteration 2)
    e. Retrieve files and information from database (completed iteration 2 – at iteration 1-local file storage)
    f. Compressing and uncompressing files from the database (Completed at iteration 2)

UNIT TESTING PROCEDURES

Throughout the project various Unit testing functions were utilized to verify correct functionality of areas of the project.

A test for each bit of functionality was designed and utilized before the final commit to the repo and the item marked as completed on the internal TODO list maintained

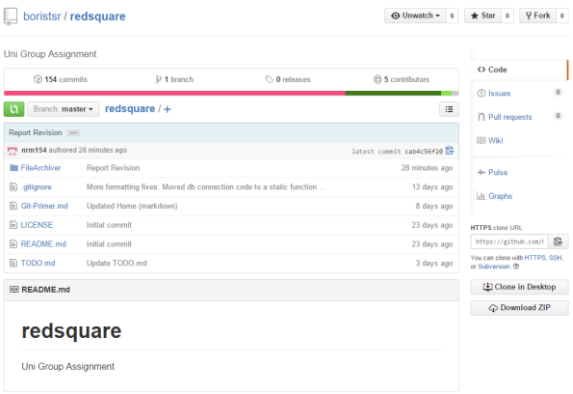
The additional testing functionality created by members was:

1. Testing of functionality when appending a path
2. Testing of functionality when querying a file name
3. Testing for proper hashing of files
4. Testing for accuracy of files modify dates
5. Testing for accuracy when querying the path of a file
6. Testing of successful file compression/decompression
7. Testing for successful commit and retrieve operations
8. Testing for successful removal operations
9. Testing for correct retrieval and accuracy of returned file information

VERSION CONTROL

For version management we have decided to utilise GitHub, a web based Git repository hosting service. We have chosen GitHub due to most members of the group having particular familiarity with it and it's easy to use desktop application.

This is the main screen of the repository we used on the website.



SETUP

GitHub makes setting up a local repository simple.

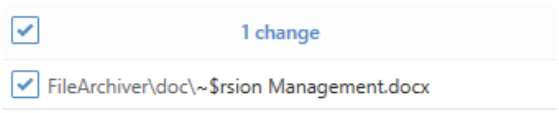
Firstly everybody in the group created a GitHub account, Phil E then created the repository ‘redsquare’ and added each group member to it for proper version control, after this the group is free to pull from the master branch and push/commit any new files or changes made to old ones.

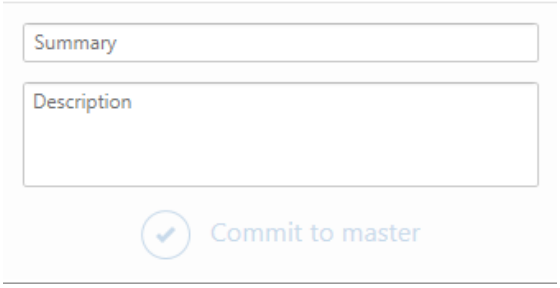
For Windows and Mac users there is a downloadable desktop application (Displayed above) for GitHub which makes this version management much easier.

DETAILS/HOW WE USED IT

This black bar is the top level of the Git repository. It shows the entire timeline of commit history to the repository. Each dot represents a commit and clicking on each takes you back to the previous versions of the repository. This is great for data security to ensure nothing is lost in an accidental commit of wrong work.

The image below shows any uncommitted changes a user has made to a current document or file. To commit it to the repository they must include a summary of what they are adding/changing and a description so that other users know what changes have been made. We used this information to go over each other’s work and make any recommendations to fix any mistakes the user who committed may have made and suggestions on how to fix them, working together to help each other.

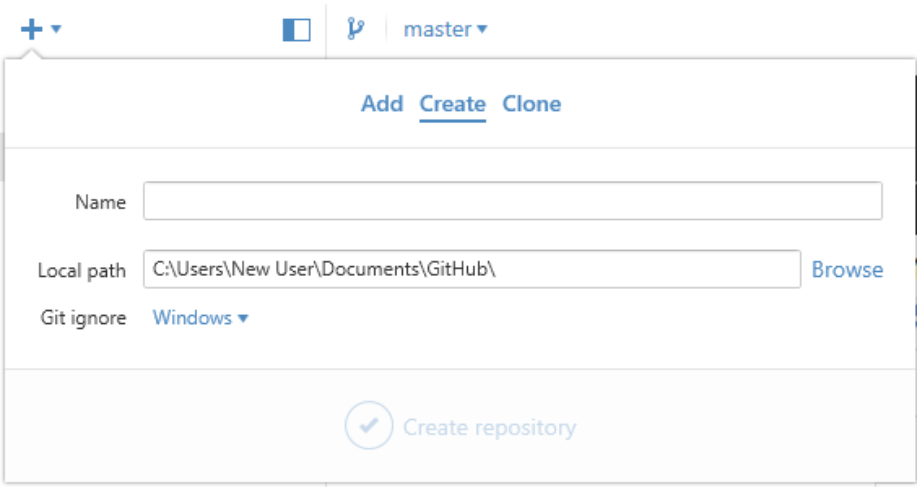




ACCESS AND USAGE INFORMATION

GitHub makes accessing the repository easy whether it's through command line or through the desktop application. The website supplies a clone/checkout URL for HTTPS, SSH and Subversion which makes it easy for all different users of the repository.

The desktop application is a small 100mb download and makes the version management much easier. All the user has to do is click the plus '+' symbol on the top left of the GUI (Pictured below) and that will drop down a box whether the user wants to Add, Create or Clone a repository, then it's as simple as copying the clone URL into the program and then you're ready to go with all files from the repository.



For command line users there are various git commands that do everything needed, from pulling the files down as well as pushing them back up and committing changes. This Git Primer file was created by Phil E to help the other group members who are new to Git. It includes step by step instructions on what exactly we needed to do if we were on Linux or without the desktop application.

Git Primer

If you want to use the command line the majority of commands you'll need are listed here.

Clone the repository

```
git clone https://github.com/boristsr/redsquare.git
```

Update your local copy with changes others have submitted

```
git pull
```

To see locally changed, added or deleted files

```
git status
```

Committing changes

When you want to commit changes you need to stage files for committing, run git status to check what files are changed. Then to add the files you want type `git add <files>` Wild characters work here `git add *.cpp *.h`

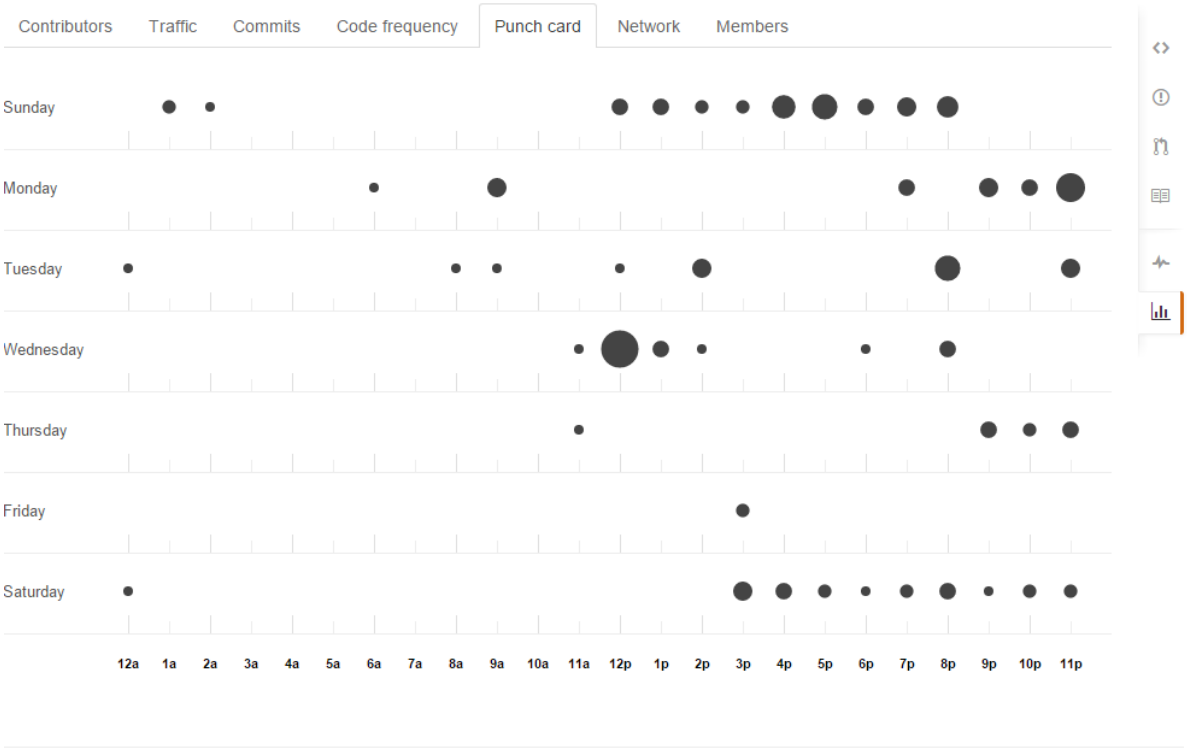
Committing

Since git is a distributed VCS, you need to commit your changes to your repository, then push it to the main repository.

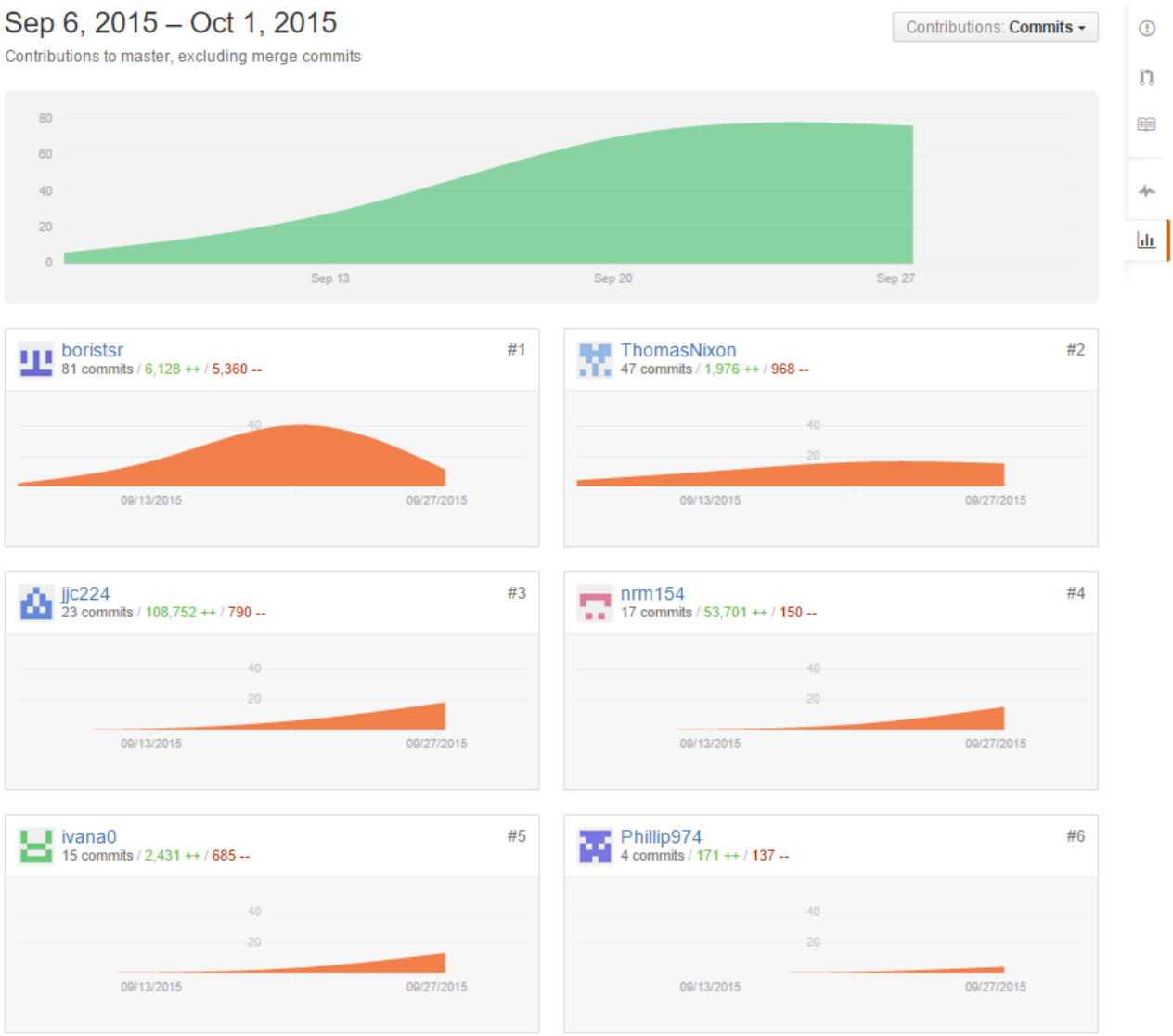
```
git commit -m "Put a comment here about what you've changed"
```

```
git push origin master
```

The image below is from the GitHub website. It shows the virtual punch card of work commitments the group has made. It shows which day and at what time each commit occurred at for our group it shows that most work is done in the afternoons and later parts of the day.



This is the final shot of the contribution statistics for all members, showing all members making successful use of GitHub versioning systems



**NOTE: line count was broken by a mistake in committing a RTF file to the repo, line count for contribution for members nrm154 and jjc224 is inaccurate

GROUP RECORDS

GROUP MEETING SUMMARY

MEETING ONE

Meeting called by: Nicholas
Note taker: Nicholas
Timekeeper: Nicholas
Attendees: All
Report Presenter: Nicholas
Target Meeting Time: 30 minutes

Manager's perception of project state:

Being the first group meeting, my impressions mostly revolved around how I thought we would work as a group, some members had already worked together on projects, so group cohesion was already in a good place, the main adjustments required was introduction of the new people to the group.

Agenda Topic: Role selection

Time Allotted: 10 minutes

Discussion:
We took some time to discuss as a group which kind of role within RUP each member would like to take.

Action Items:

1. Advise of desired role within the group, by next week lab
2. Revise and familiarise yourself with the assignment details

Agenda Review:
Currently no items for review

Agenda Topic:
In preparation for the next week to come, the design of the project had to be outlined and understood by each individual within the group.

Time Allotted: 20 minutes

Discussion:
We took some time here to quickly review the assignment and began to brainstorm some ideas for the design.

Action Items:

1. Discuss and outline initial design thoughts for the project
2. Upload a concise coding style to enhance code consistency

Agenda discussion:

Some concerns were raised about the structure of the current design; a decision was made that we would revise the design in order to streamline it. No date was targeted, merely we acknowledge that we felt there was a need for redesign and agreed to go about it, at some undefined later date.

Philip Edwards also posted a coding style document for all members to follow; this was to ensure consistency in the code throughout the project.

MEETING TWO

Meeting called by: Nicholas
Note taker: Nicholas
Timekeeper: Nicholas
Attendees: All
Report Presenter: Nicholas
Target Meeting Time: 30 minutes

Manager Perception of project state:

At this stage my perceptions were largely the same as the previous meeting, with no hard work being done, there is no data to base a review on. The cohesion of the group appeared to be strong and members seemed flexible and willing to assist in all parts of the project.

Agenda Topic: Review the requirements of the project and handle role allocation.

Time Allotted: 15 minutes

Discussion:
We took time to outline the requirements of the project in the sense of the overall functionality and the interconnectedness of sub-projects to build the project as a whole.

Action Items:

1. Reviewed assignment specifications
2. Role preferences for the project, and the assigning of them

Agenda Review

1. All members reported familiarising themselves with the assignment specs, questions were raised which lead to the group reading through the assignment spec's to get everyone on same page before beginning planning and implementation

1. Roles
    a. Manager- Nicholas Ross Morgan
    b. Lead Designer – Josh Coleman
    c. Lead Implementer- Phil Edwards
    d. Designers – Assigned to all
    e. Systems Integration – Ivana Ozakovic, Phil Edwards

*Roles were expanded- check detailed report for final roles

Agenda Topic: Identify appropriate consultation times and collaboration methods

Time Allotted: 15 minutes

Discussion:
The group discussed ways to communicate, as well as methods to get in contact with one another if any of the services relied on fail.

Action Items:

1. Assign regular meeting times
2. Establish secondary communication methods
3. Investigate Technologies for use(MySQL/Mongo, GiTHub/provided repo)

Agenda discussion

1. Regular meeting times were scheduled for 6 PM every Sunday, with regular informal check-ins during the week
2. Group chose Skype as a means for communication outside of physical meetings
3. Phone numbers were exchanged as a backup and for on-the-go contact
4. A brief discussion was had about the familiarities of each member with the systems, members reported being more familiar with MySQL and GitHub came out as the preferred version management

MEETING 3

Meeting called by: Thomas Nixon
Note taker: Phil
Timekeeper: Thomas
Attendees: Phillip Edwards, Thomas Nixon, Joshua Coleman, and Ivana Ozakovic
Report Presenter: Nicholas Morgan
Target Meeting Time: 1 Hour

Manager Perception of project state:

I was not present for this meeting, but after receiving the notes and seeing the outcome I was happy with the direction the project was taking, a complete redesign on the structure had been completed.

Agenda Topic:
Redesign of the overall system.

Time Allotted: 30 Minutes

Discussion:
Integrity insurance of systems.

Action Items:

1. Supporting system selection
2. Ensure all members have installed Git to manage/share work on the project

Agenda Review

1. Decisions were made based on the preferences of the group at the previous meeting to go ahead with usage of MySQL and GitHub

2. Integrity of the server/database was confirmed accessible by all members individually, then reported the success to one another as a group.
3. A Git primer was added to the project's GitHub to assist members in their usage

Agenda Topic: Overall project structure.

Time Allotted: 20 minutes.

Action Items:

1. Project structure redesign
2. Database setup

Agenda discussion

1. A lengthy discussion was had about the redesign with a complete redesign of the structure complete by the end of the meeting *refer to detailed report section
2. A database was setup to allow testing of interactions with code and a live database

MEETING 4

Meeting called by: Nicholas
Note taker: Nicholas
Timekeeper: Nicholas
Attendees: All
Report Presenter: Nicholas
Target Meeting Time: 1 Hour

Manager Perception of project state:

At this stage prototyping for functions within multiple sections of the program had been completed, the design phase was completed and members had a clear view of the new direction of the assignment, The project as a whole seemed to be on track to be completed. Group cohesion was slightly lower, with the redesign happening in the previous meeting, however by the end of the week's meeting, people were back on the same page.

Agenda Topic: A continuation of the design to ensure integrity of interconnected components.

Time Allotted: 35 minutes

Discussion:
More thorough outline of the overall design.

Action Items:

1. Discussion on the changes made to design
2. GitHub member familiarization

Agenda Review

1. Phil and Thomas ran the members not present at the previous meeting through the specifics of the redesign

2. Time was spent showing members the proper use of Git to avoid issues and insure all could successfully commit their work without issue

Agenda Topic: Project code and the initial assignment of it.

Time Allotted: 20 minutes

Discussion:
Discussion of the actual code and who should be assigned the relative sub-tasks.

Action Items:

1. Initial Code Assignment and discussion

Agenda discussion:

1. Code Assignments
   - **a.** VersionRecord – Thomas
   - **b.** GUI/Reporting – Nicholas
   - **c.** FileRecord – Phillip E.
   - **d.** CPP Unit testing– Ivana/Josh
   - **e.** Gui - Phillip Mihajlovski.

MEETING 5

Meeting called by: Nicholas
Note taker: Nicholas
Timekeeper: Nicholas
Attendees: All
Report Presenter: Nicholas
Target Meeting Time: 1 Hour

Manager Perception of project state:

Project is currently in crunch mode, some deliverables of members hadn't been committed on time, other group members had been working hard and a large portion of coding was completed, at the beginning of this meeting CPP Unit testing, GUI and some touch ups on the VersionRecord and FileRecord functionality needed to be completed, reports also needed to be completed and the design document fleshed out.

Agenda Topic: Identification of possible issues with the functionality of the project thus far.

Time Allotted: 20 minutes

Discussion:
Identify issues in the design/functionality and handle as necessary, as well as identify what else follows.

Action Items:

1. Update internal TODO list
2. Discuss issues in functionality

Agenda Review:

1. As a group we sat down and updated the TODO list, this updated all members perspective of the current state and gave us a clear idea of what was remaining to finish
2. We took some time as a group to discuss any functionality issues we had in an effort to see if any group member could provide insight or a new look on the problem

Agenda Topic: An update of what components need to still be completed.

Time Allotted: 25 Minutes

Discussion:
An overall update of the project: what components have been completed.

Action Items:

1. GUI design and implementation finalized
2. CPPUnit testing code finalization
3. Makefile fixes
4. FileRec and VersionRecord implementation finalized
5. Reports
6. Presentation

Agenda discussion:

1. GUI design was passed onto Ivana and Josh
2. CPP Unit testing was assigned to all members
3. Makefile fixes for Unit testing
4. Report writing assigned to Nicholas
5. FileRecord and VersionRecord finalized assigned to Thomas and Philip Edwards
6. Presentation for inclusion in report assign to Ivana
7. Final meeting

Meeting called by: Nicholas

Note taker: Nicholas

Timekeeper: Nicholas

Attendees: All

Report Presenter: Nicholas

Manager Perception of project state:

Project is currently in crunch mode, some deliverables of members hadn't been committed on time, other group members had been working hard and a large portion of coding was completed, at the beginning of this meeting CPP Unit testing, GUI and some touch ups on the versionrec and filerec functionality needed to be completed, reports also needed to be completed and the design document fleshed out.

Agenda Topic: Final check to ensure all tasks have been completed.

Time Allotted: 20 minutes

Discussion:
Verify with one another what has been completed and check that no incomplete tasks remain.

Action Items:

1. Final check of TODO list
2. Report update with finalized info

Agenda Review:

1. As a group we went through the TODO, ensure all items checked off were completed and assigned final tasks

2. Tasks
    a. GUI – Joshua and Ivana
    b. Report – Phil Mihajlovski and Nicholas
    c. VersionRecord – Philip Edwards
    d. FileRecord – Thomas
    e. CPP Unit testing – Phil Edwards, Ivana Ozakovic, and Joshua Coleman

GROUP MEMBER WORK JOURNAL SAMPLES

Group members maintained a work journal to track work times, the primary goal of these journals was to identify and shortcoming pertaining to time management and to ensure an organized approach to the design and implementation of the project.

SAMPLE OF WORK DIARY FROM THOMAS NIXON

```
Work Diary


Author: Thomas Nixon


6/09/2015
Read documentation
Created Google doc to start putting outlines of what needs to be done in.
Researched MySQL and its use with c++
Time: 3:00pm-7:00pm

Created FileArchive.h & FileArchive.cpp.
Defined class members and modified names.
Insured code was correct.
Time: 7:00pm-7:30pm

Created a rough version of the database MySQL code.
Currently untested and needs to be checked.
Time: 7:30pm-8:00pm


14/09/2015

Put SQL in MySQL server.
Fixed a few small errors in the code.
Tested operation.
Database ready to be used.
Time: 9:00pm-10:00pm


15/09/2015

Wrote the code to connect to MySQL server.
Tested operation over internet and returned data.
Updated some of the class in the FileArchiver with the code.
The functions are not complete but have the frame work to used.
Time: 9:00pm-11:00pm
```

Work Diary

Author: Ivana Ozakovic

Work#:1

Short description: Declare functions and data members for VersionRecord.h and set up function stubs in VersionRecord.cpp

Planned work schedule

Date: 12/09/15, Saturday

Time: Anytime

Actual work time

Date: 12/09/15

Time: 12:30pm - 1:30pm

Summary of work completed

Part of the Project: Implementation

Files modified: VersionRecord.cpp (created), VersionRecord.h

Functions modified: /

Description: Declared data members and functions in the VersionRecord.h.

Created VersionRecord.cpp file and set up stubs for declared functions.

Defects/problems: Functions have to be defined.

SUPPORTING CODE SAMPLES

SUPPORT CODE FOR CONNECTING TO THE DATABASE

```
// Connect to database
try
{
        driver = get_driver_instance();
        dbcon = driver->connect(host, user, pw);
        dbcon->setSchema(dbname);
        bInitialised = true;
}
catch (sql::SQLException&       e)
{
        log("ERROR: ");
        log(e.what());
        log(boost::lexical_cast<string>(e.getErrorCode()));
        log(e.getSQLState());
}

//dbcon->setTransactionIsolation(sql::TRANSACTION_READ_UNCOMMITTED);
return dbcon;
```

SUPPORT FOR UTILITIES FOR LOGGING DEBUG INFO

```
#include "ProjectConstants.h"

void logToFile(std::string message)
{
#ifdef DEBUG_LOG_TO_FILE
        std::ofstream outFile("debuglog.log", std::ios::app);
        if(outFile.is_open())
        {
                outFile << message << std::endl;
                outFile.close();
        }
#endif // DEBUG_LOG_TO_FILE
}

void log(std::string message)
{
#ifdef DEBUG_LOGGING
        std::cout << message << std::endl;
        logToFile(message);
#endif //DEBUG_LOGGING
}
```

Support code sample - hasing - Murmurhash

```
void MurmurHash3_x86_32 ( const void * key, int len,
                                                    uint32_t seed, void * out )
{
        const uint8_t * data = (const uint8_t*)key;
        const int nblocks = len / 4;

        uint32_t h1 = seed;

        const uint32_t c1 = 0xcc9e2d51;
        const uint32_t c2 = 0x1b873593;

        //----------
        // body

        const uint32_t * blocks = (const uint32_t *)(data + nblocks*4);

        for(int i = -nblocks; i; i++)
        {
                uint32_t k1 = getblock32(blocks,i);

                k1 *= c1;
                k1 = ROTL32(k1,15);
                k1 *= c2;

                h1 ^= k1;
                h1 = ROTL32(h1,13);
                h1 = h1*5+0xe6546b64;
        }

        //----------
        // tail

        const uint8_t * tail = (const uint8_t*)(data + nblocks*4);

        uint32_t k1 = 0;

        switch(len & 3)
        {
        case 3: k1 ^= tail[2] << 16;
        case 2: k1 ^= tail[1] << 8;
        case 1: k1 ^= tail[0];
                                        k1 *= c1; k1 = ROTL32(k1,15); k1 *= c2; h1 ^= k1;
        };

        //----------
        // finalization

        h1 ^= len;

        h1 = fmix32(h1);

        *(uint32_t*)out = h1;
}
```

UNIT TESTING SAMPLES

TESTING FOR CORRECTLY APPENDED PATH TO FILE

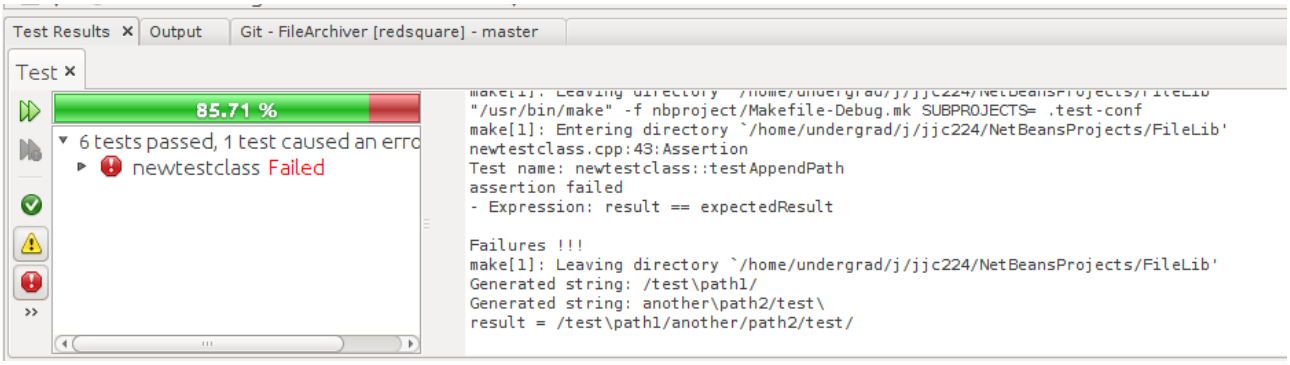
```
void newtestclass::testAppendPath()
{
    const std::size_t NUM_PATH_ELEMENTS_1 = 2;
    const std::size_t NUM_PATH_ELEMENTS_2 = 3;

    std::string pathElements1[NUM_PATH_ELEMENTS_1] = {"/test", "path1"};
    std::string pathElements2[NUM_PATH_ELEMENTS_2] = {"another", "path2", "test"};

    std::string path1          = generateUnormalizedPath(pathElements1, NUM_PATH_ELEMENTS_1);
    std::string path2          = generateUnormalizedPath(pathElements2, NUM_PATH_ELEMENTS_2);
    std::string expectedResult = "/test/path1/another/path2/test/";
    FileLib fileLib;
    std::string result = fileLib.AppendPath(path1, path2);

    std::cerr << "result = " << result << std::endl;
    CPPUNIT_ASSERT(result == expectedResult);
}
```

TESTING FOR RETRIEVING FILENAME FUNCTIONALITY

```
void newtestclass::testGetFilename()
{
    const std::size_t NUM_PATH_ELEMENTS = 2;
    std::string pathElements[NUM_PATH_ELEMENTS] = {"/some", "path"};

    std::string file = "file.ext";
    std::string path = generateUnormalizedPath(pathElements, NUM_PATH_ELEMENTS) + file;
    FileLib fileLib;
    std::string result = fileLib.GetFilename(path);
    std::string expectedResult = file;

    CPPUNIT_ASSERT(result == expectedResult);
}
```

BUG/TESTING LOG SAMPLES

Sample from a run of some implemented unit testing code



We took an iterative approach to our design, starting from basics and improving upon them, during our design we encountered bugs, the two diagrams below show the occurrence of a bug (improper extensions for a boost function) the first resulted in some unexpected behaviour, with the second image showing the solution after the bug fix

```
// Replaces one or more backslashes and two or more forward slashes with a single forward slash.
string FileLib::Normalize(string path)
{
        boost::regex re("\\\\+|//+");
        path = boost::regex_replace(path, re, "/");

        return path;
}
```

```
// Replaces one or more backslashes and two or more forward slashes with a single forward slash.
string FileLib::Normalize(string path)
{
        boost::regex re("\\\\+|//+");
        path = boost::regex_replace(path, re, "/", boost::match_default | boost::format_all);

        return path;
}
```

APPENDICES

CODE ELEMENTS

FileArchiver.cpp / FileArchiver.h

FileLib.cpp / FileLib.h

FileRecord.cpp / FileRecord.h

CompressUtils.cpp / CompressUtils.h

DBConfigurationFileUtility.cpp

DBConnector.cpp / DBConnector.h

GetCommentForm.cpp / GetCommentForm.h

MurmurHash.cpp / MurmurHash.h

MyWindows.cpp / MyWindow.h / MyWindow.ui

ProjectConstants.h

RetrieveForm.cpp / RetrieveForm.h

TestUtilities.cpp / TestUtilities.h

Utilities.cpp

VersionRecord.cpp / VersionRecord.h

FileLibTester.cpp / FileLibTester.h

BackendTests.cpp / BackendTests.h

CODE LISTING