CSCI222

Assignment 1

Report

Overview

This project involves the design and implementation of a file management system, it is made for the purpose of providing practice to student doing CSCI222 at the University of Wollongong, the project itself is broken into design and implementation elements, with students required to provide reports and documentation to demonstrate proper use of a rational unified process (RUP) design as well as provide a demonstration of the working finished product.

The programs 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.

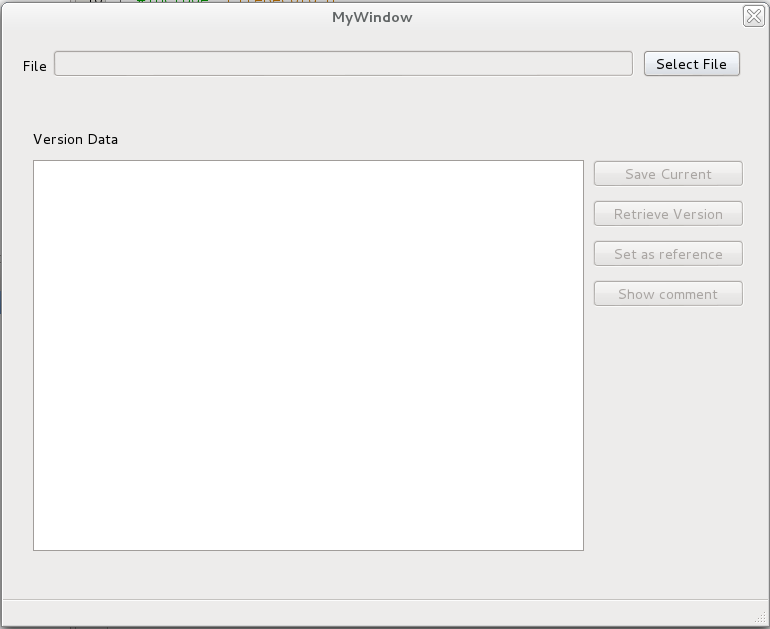
# Program Presentation

List of the features

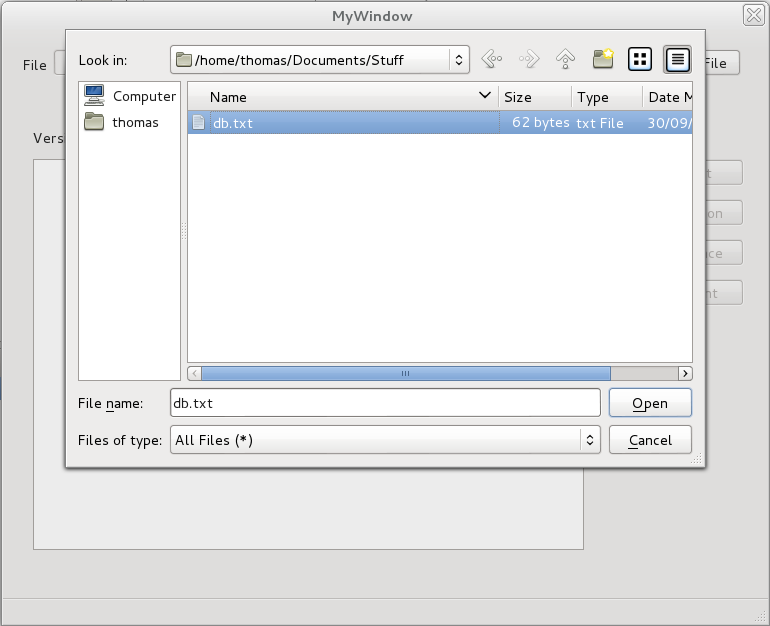
---->list will go here

--->this is how we use features

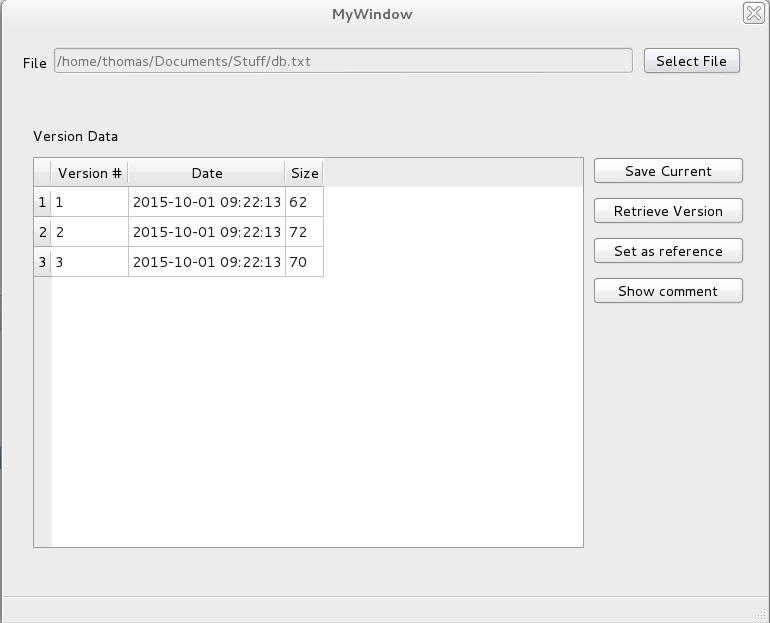
The initial main window gives user an option to select file with “Select File” button.



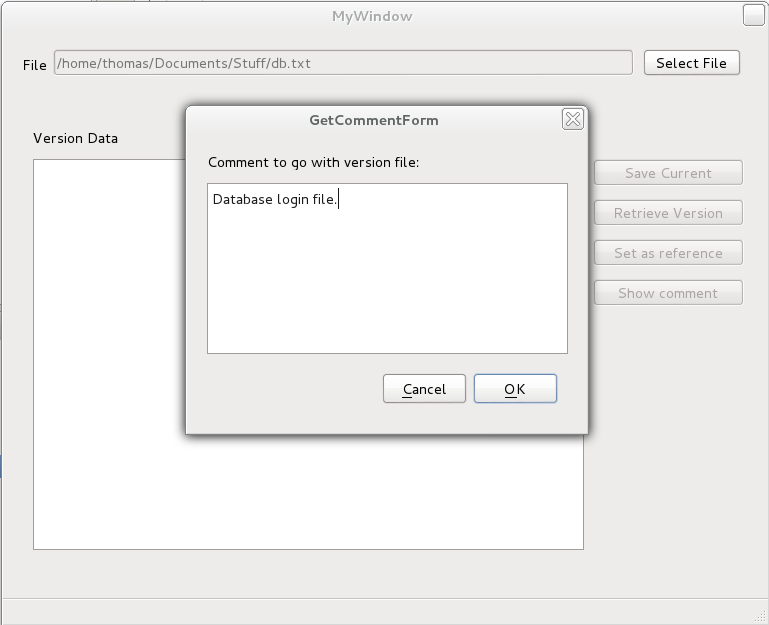
After clicking on “Select File”, the choose file window will appear for user to select file select the file and shows the path chosen.



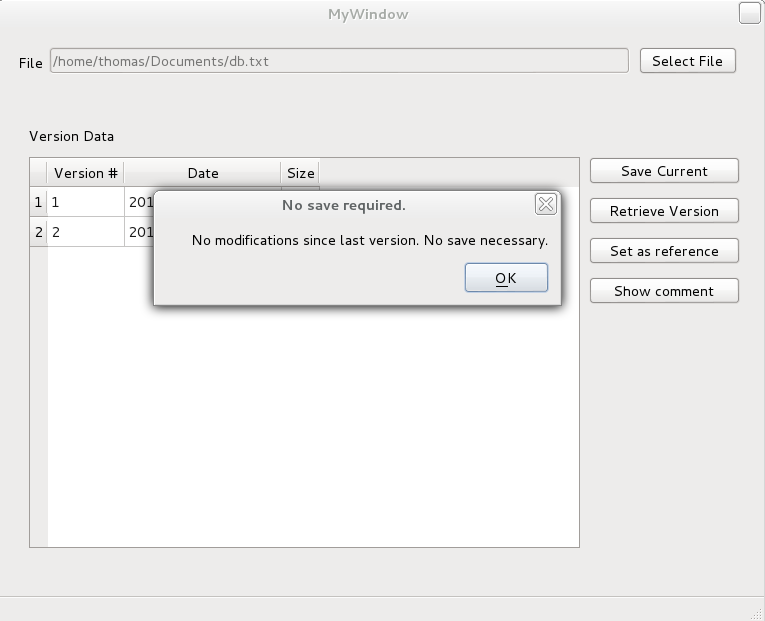
If the file is already saved in the persistent storage, saved versions of this file will be shown in the table.



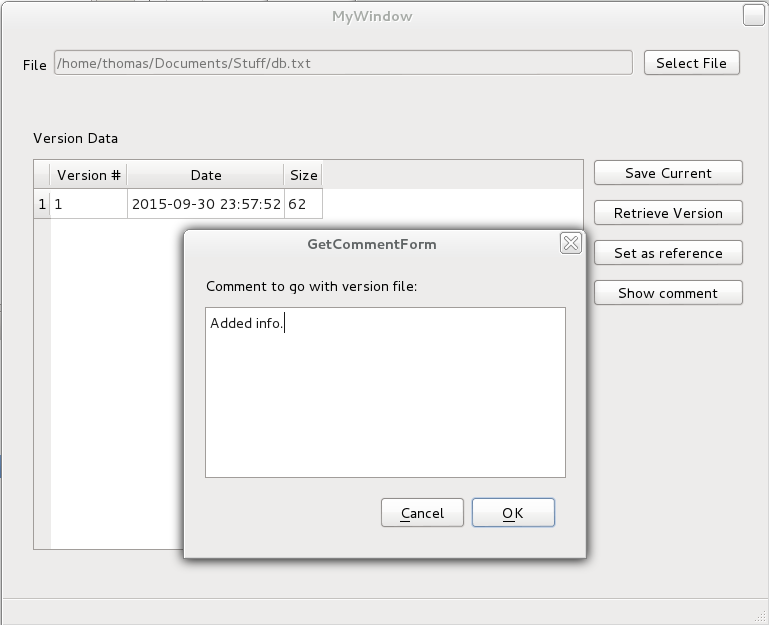
Otherwise, the program will create initial version in the database and ask user for the comment to go with the initial file version.



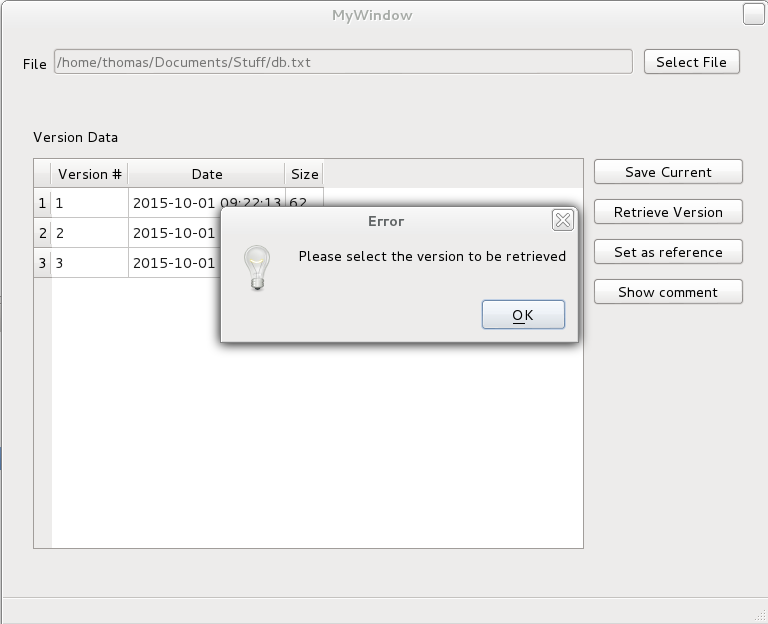
If the user attempts to save the current version of the file that has not been changed since the most recently updated version, the message box will show up to inform the user that there is no need for saving the current version.



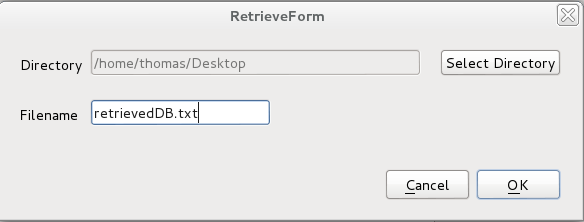
In case the change in the current version was detected, user will be asked to supply the comment for the new version and the new version of the file will be displayed in the table.



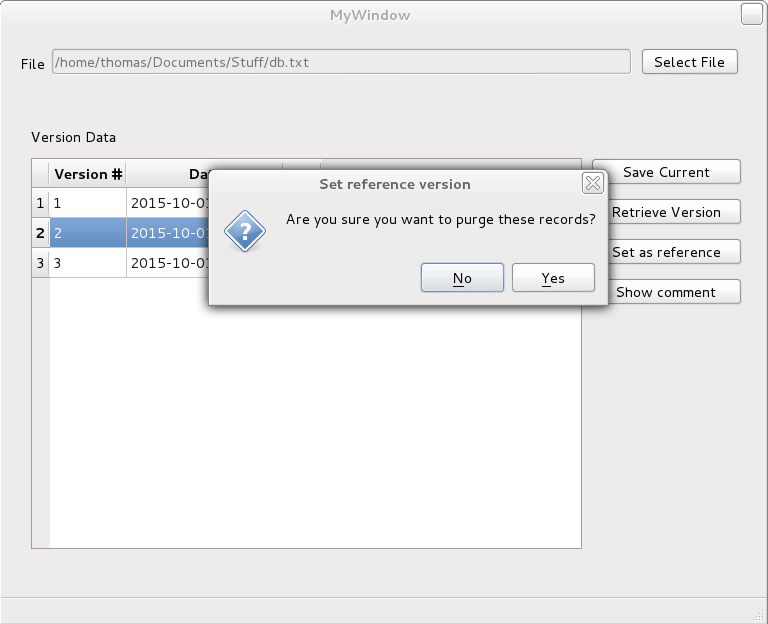
Retrieve version button provides user the option to retrieve the version of the file that is selected in the table view. If the file version to be retrieved was not selected in the table, user will be informed.

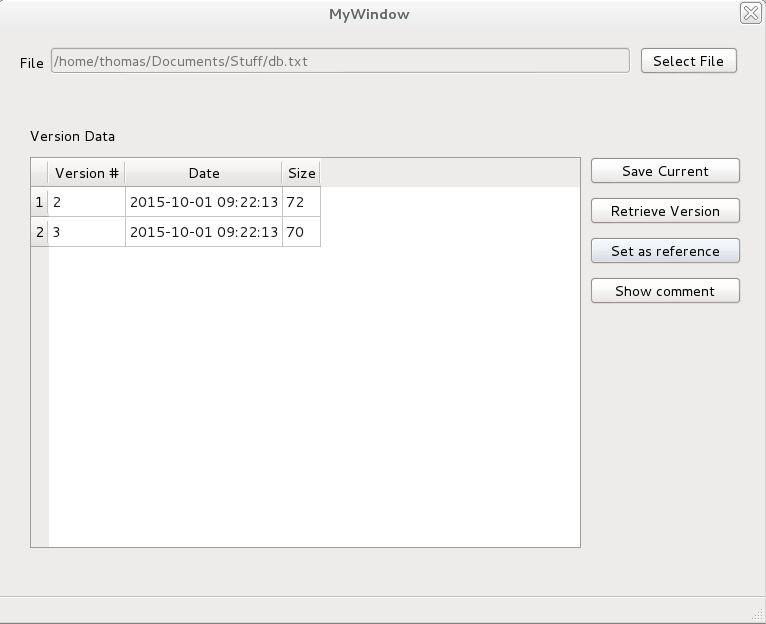


Otherwise, the retrieve form dialog will show up asking user to specify the directory where the retrieved version will be saved, along with the filename.

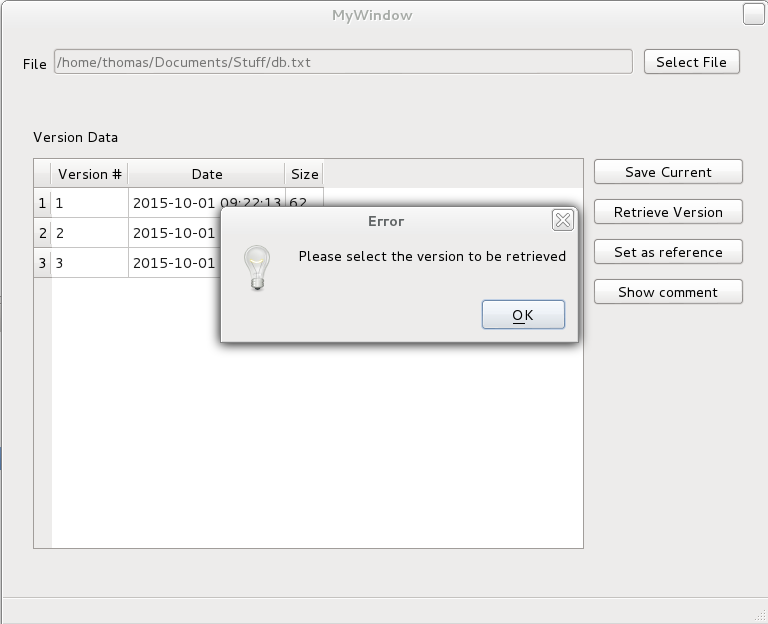


Set as reference option allows user to select version that will be used as a reference to delete all previous versions of the file that are not needed anymore. The program will ask for confirmation to proceed with deletion.

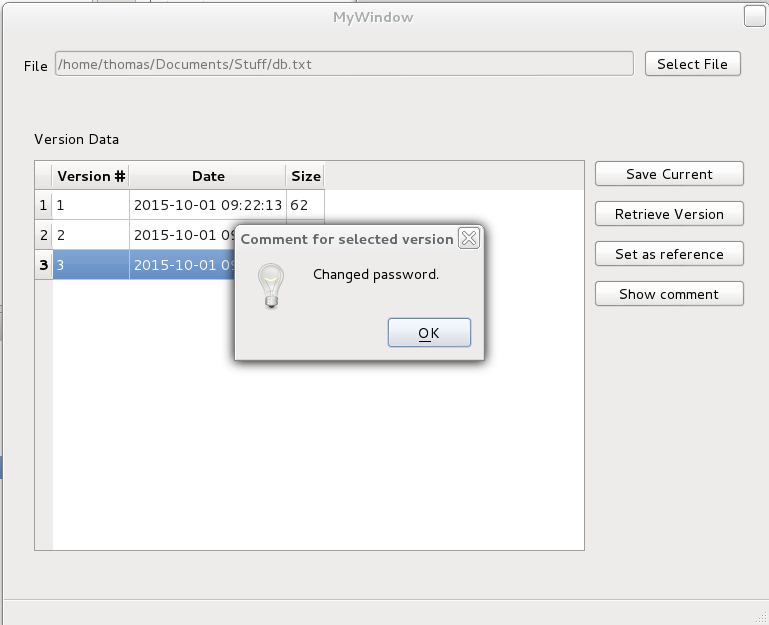




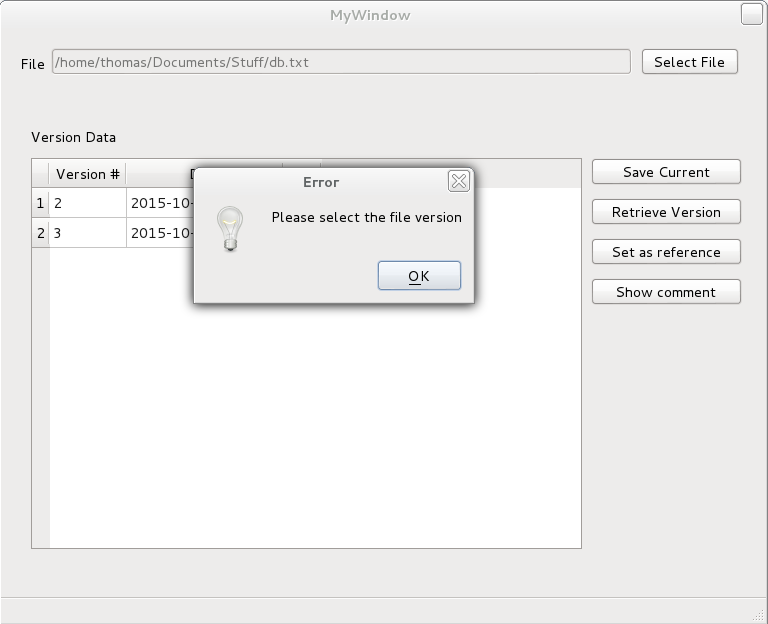
In case the reference version is not selected in the table, the program will notify the user.



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



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



## Functional Requirements

**TODO:** Probably add screenshot of final working instance of these. The text can probably be written beforehand.

Create Initial Archive (completed)

Detect Changes (completed)

Save Modified Version (completed)

Display a summary of versions in storage (completed)

Retrieve chosen version (completed)

Show comment associated with version (completed)

Use compression (completed)

Incremental changes (completed)

Discard old (completed)

# Group

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

## Deliverables of Members

### Thomas Nixon

* 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

### Phil Edwards

* 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

### Ivana Ozakovic

* 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.

### Josh Coleman

* 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).

### Nicholas Morgan

* Meeting Reports
  + Timekeeping and note taking
* Documentation
  + Elements and unit testing procedure
  + Supporting code samples and listings
  + Version Management – initial
  + Detailed meeting report – revision
* Project collation

### Phil M

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

# Design & Implementation

*A summary of your group’s work on design and the implementation plan. This should cover: any reworking of the proposed implementation classes and give details of decisions relating to data persistence and user interface issues. UML modeling diagrams should be used to illustrate*

*design decisions. If your group decides on a different implementation plan, with different iterations, you should give details and justification.*

* *Details of the construction phase. This part of the report should clarify the work done in each iteration.*
* *Summarize data from defect and integration reports created by the systems integrator.*

## Design Choices

### Database

Diagrams go here

Explanation of why we changed it

### FileArchiver

Diagrams go here

# Diagrams

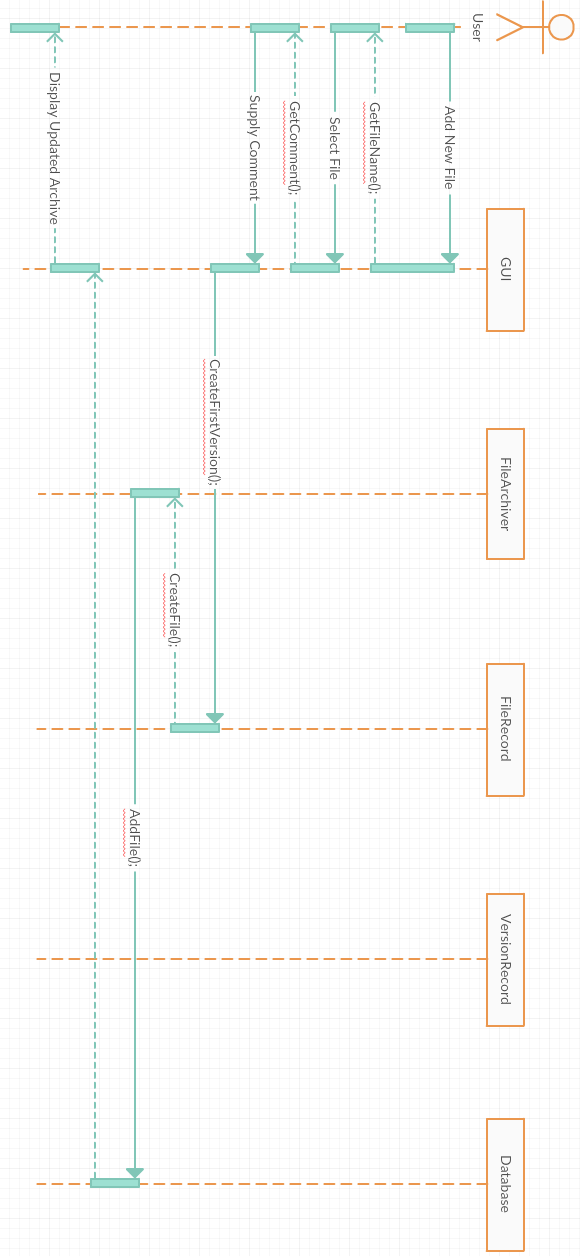
## Use Case

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

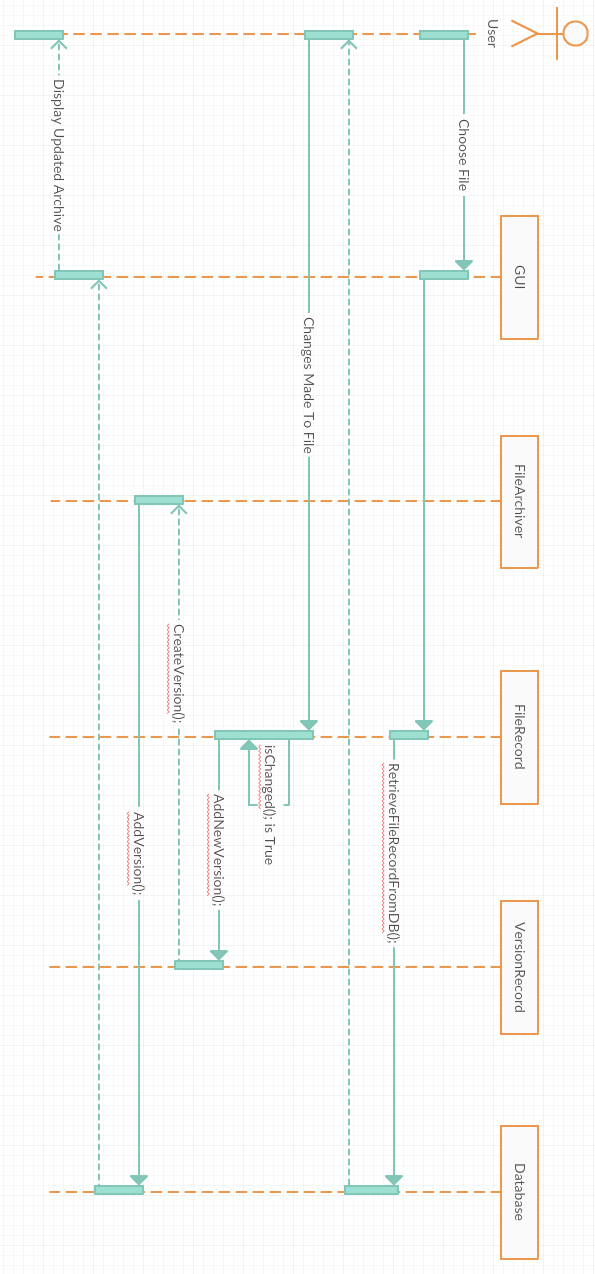
## Sequence Diagrams

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

### Adding the initial file to the FileArchiver System.

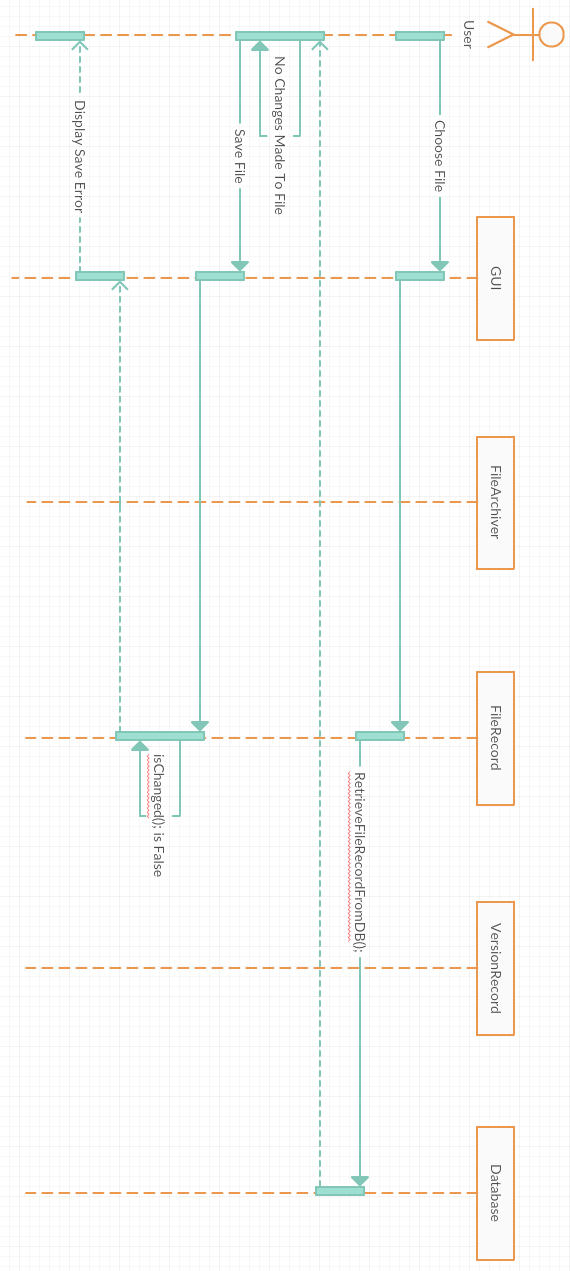


### Updating a current file in the system.

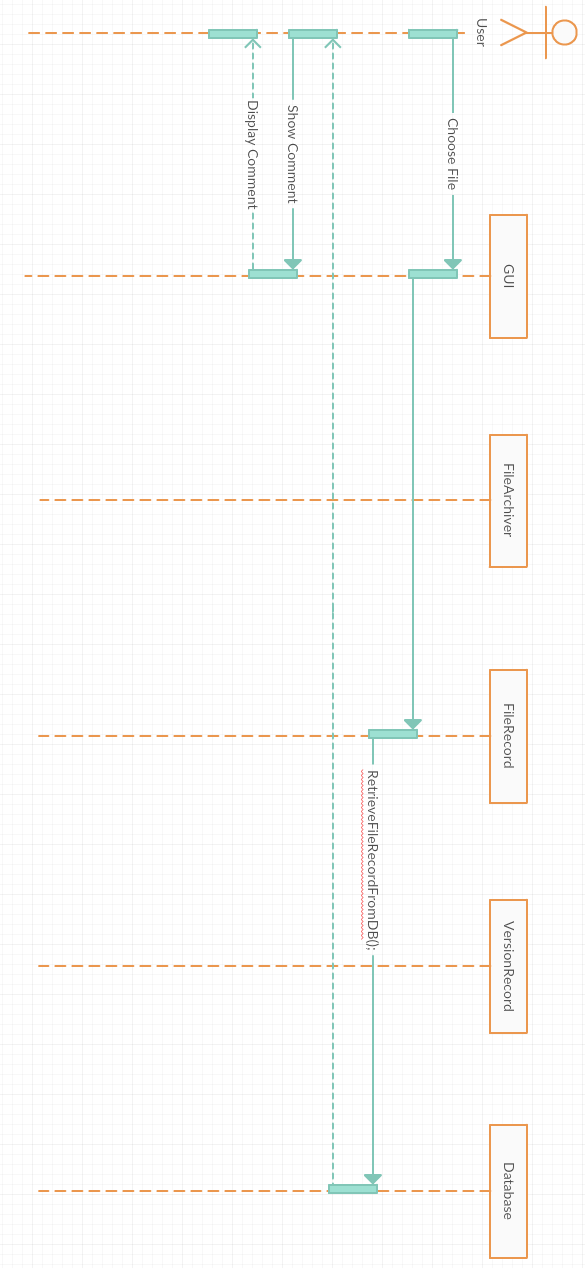


### RetrieveAndSetVersion2Retrieving A Current Version and Setting as Reference

### Attempting To Save Without Having Made Changes



### Viewing The Comment Of A File



# A description...Database structure

Our final database structure and specifics varied from the provided specs within the project documentation

Explanation of why we changed it

# Element List

1. File record functionality – (Full functionality Completed during iteration 2)
   1. Create file records – (Completed iteration 1)
   2. Update file records – (Completed iteration 2)
   3. Show version of files – (Completed iteration 2)
   4. Remove versions - (Completed iteration 2)
   5. Add new versions of files – (Completed iteration 2)
   6. Show size information – (Completed iteration 1)
   7. Check if a version has changed –(Completed iteration 1)
2. Version record functionality
   1. Retrieve version record from database (Completed iteration 1)
   2. Update Version information from a database (Completed iteration 2)
   3. Create new version records (Completed iteration 1)
   4. Show file size information (Completed iteration 1)
   5. Remove a version from the database (Completed iteration 2)
3. Gui Functionality (full functionality with data integration completed iteration 3)
   1. Provide graphical interface for workings of the program (Gui base completed at iteration 1)
4. File Archiver (Full functionality achieved during iteration 3)
   1. Check for already existing files (Completed during iteration 2)
   2. Get version info (completed at iteration 2)
   3. Insert a new file into database (Completed at iteration 2)
   4. Add versions to files (completed at iteration 2)
   5. Retrieve files and information from database (completed iteration 2 – at iteration 1-local file storage)
   6. Compressing and uncompressing files from the database (Completed at iteration 2)

# Unit Testing Procedures

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

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

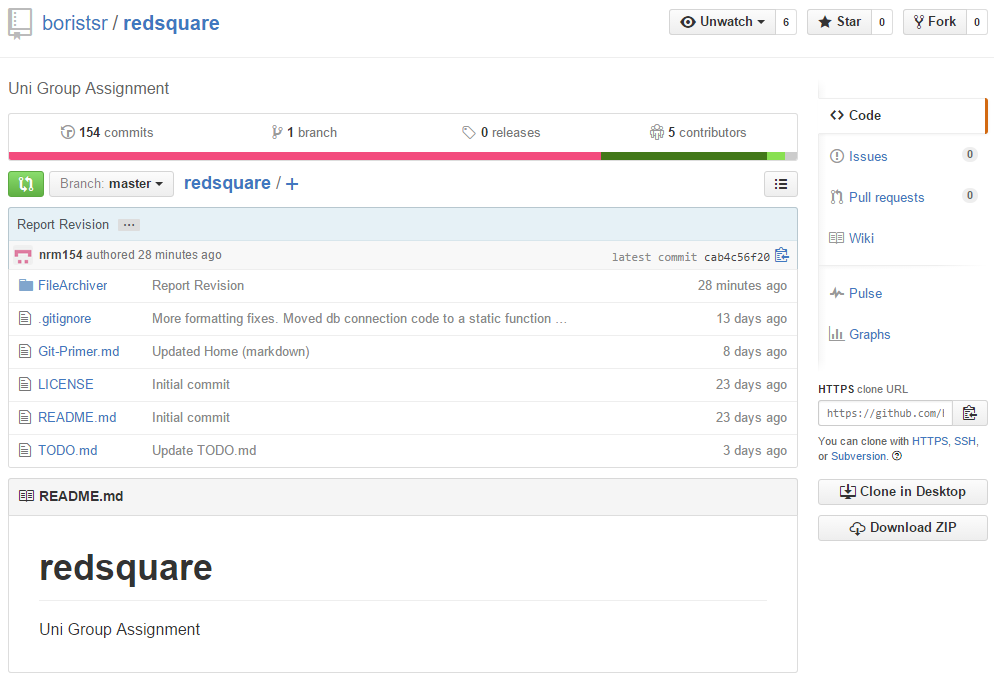
The additional testing functionality created by members was:

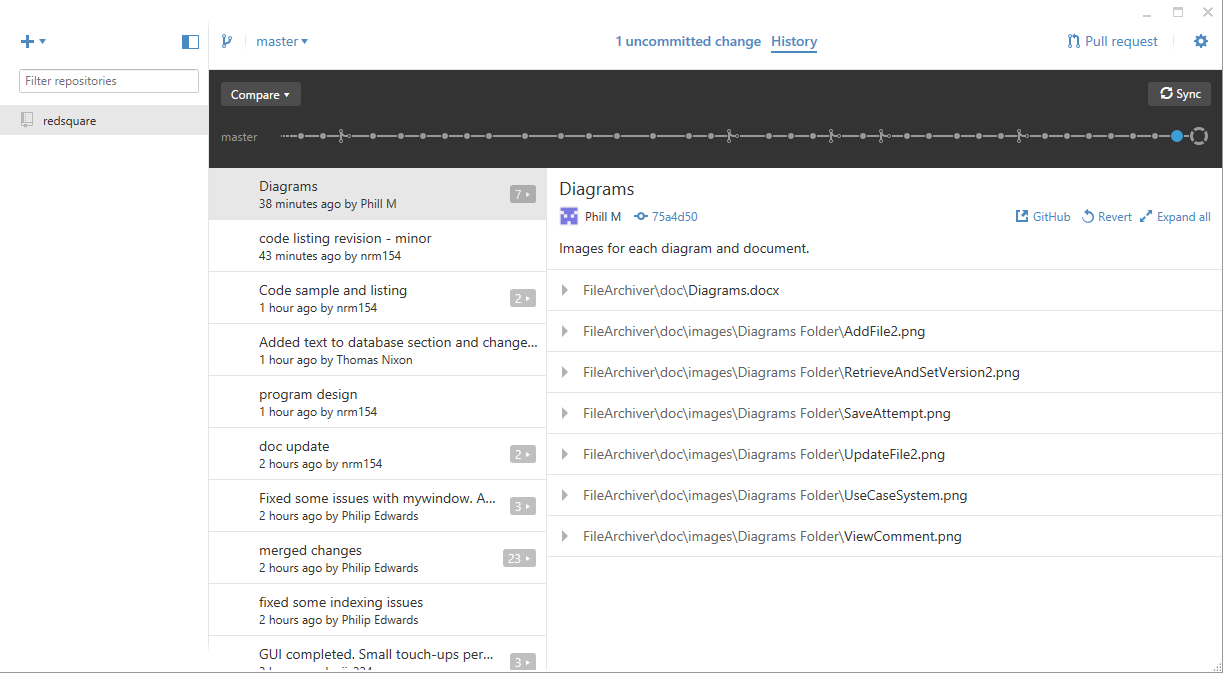
1. Testing of functionality when appending a path
2. Testing of functionality when querying a file name
3. Testing for proper hashing of files
4. Testing for accuracy of files modify dates
5. Testing for accuracy when querying the path of a file
6. Testing of successful file compression/decompression

# CSCI222 – Group x – 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, and the second picture shows the desktop application screen.





## 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.

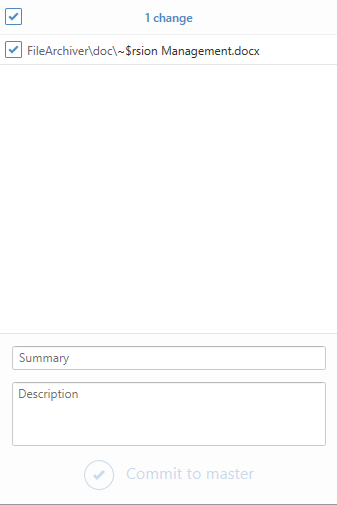
For Windows and Mac users there is a downloadable desktop application (Displayed above) for GitHub which makes this version management much easier.

## 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.



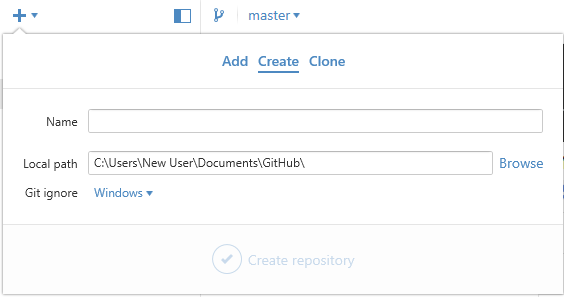
The image below shows any uncommitted changes a user has made to a current document or file. To commit it to the repository they must include a summary of what they are adding/changing and a description so that other users know what changes have been made. We used this information to go over each other’s 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.



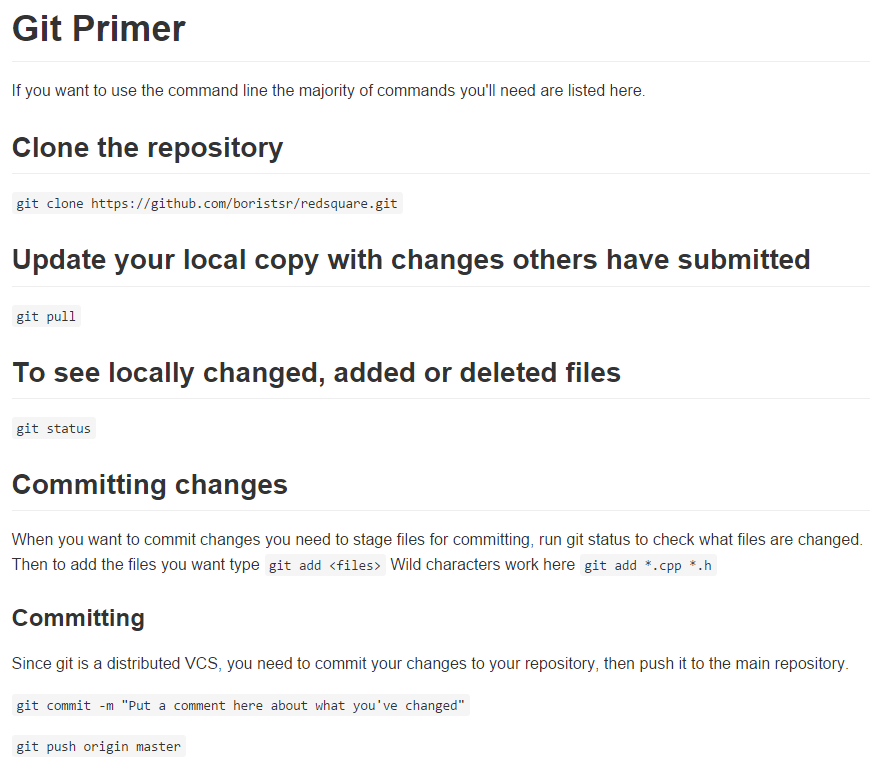
## Access and usage information

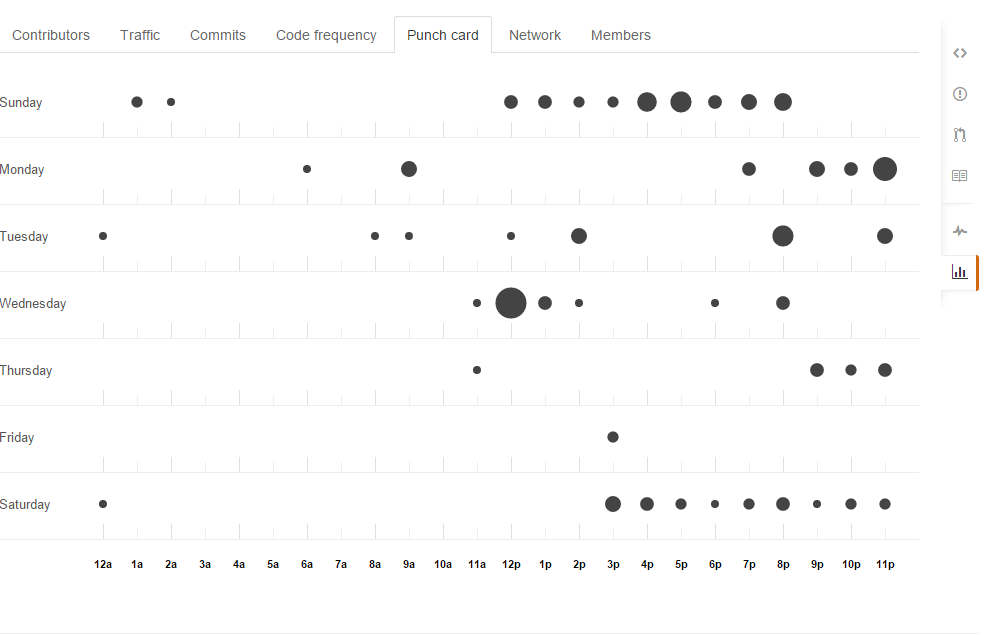
GitHub makes accessing the repository easy whether it’s through command line or through the desktop application. The website supplies a clone/checkout URL for HTTPS, SSH and Subversion which makes it easy for all different users of the repository.

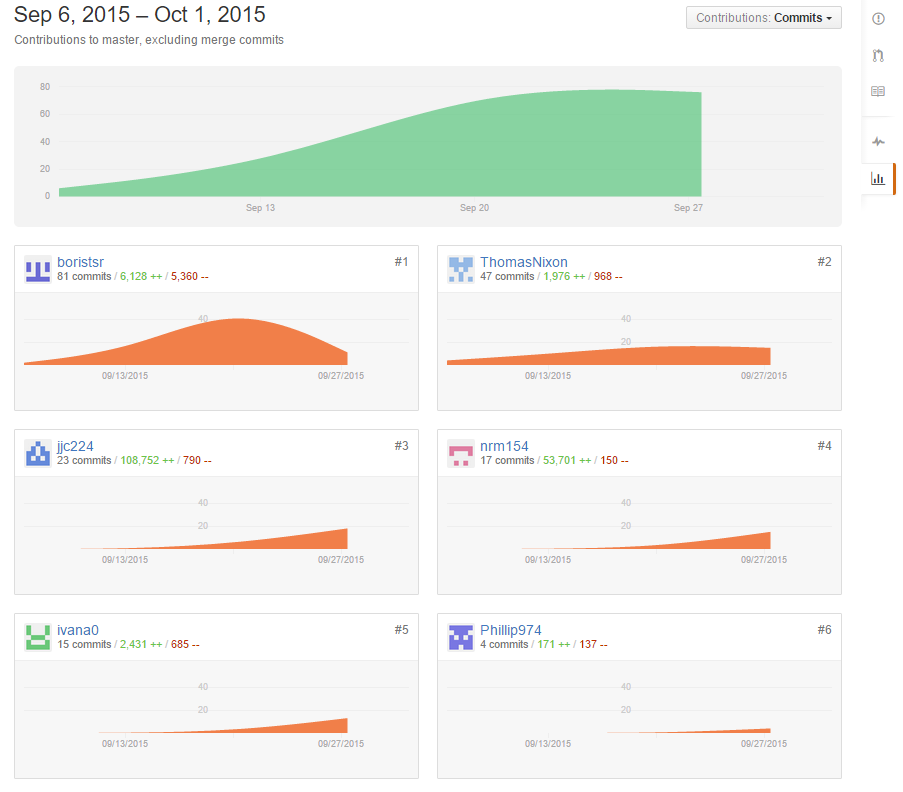
The desktop application is a small 100mb download and makes the version management much easier. All the user has to do is click the plus ‘+’ symbol on the top left of the GUI (Pictured below) and that will drop down a box whether the user wants to Add, Create or Clone a repository, then it’s as simple 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 a 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 GitHub. It includes step by step what exactly we needed to do if we were on Linux or without the desktop application.



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 member nrm154 is inaccurate

# Group Records

*Group meeting records and individual diaries:*

* *There should be samples from bug logs and testing logs*

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 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 needed was introduction of the new people to the group.

**Agenda Topic:** Role Selection  
**Time Allotted:** 10  
**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**

1. Currently no items for review

**Agenda Topic:** Week in plan- design discussions   
**Time Allotted:** 20

**Discussion:**  
We took some time here to quickly review the assignment and began to brainstorm so idea’s for the design.

**Action Items:**

1. Discuss initial design thoughts for the project

2. Create a sort of coding style guide – Phillip Edwards

**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 E. 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 of all parts of the project.

**Agenda Topic:** Week in review

**Time Allotted:** 15 minutes

**Discussion:**

**Action Items:**

Reviewed assignment specifications

Preferences for Role within the team

**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
   1. Manager- Nicholas Ross Morgan
   2. Lead Designer – Josh Coleman
   3. Lead Implementer- Phil Edwards
   4. Designers – Assigned to all
   5. Systems Integration – Ivana Ozakovic, Phil Edwards

\*Roles were expanded- check detailed report for final roles

**Agenda Topic:** Week in plan

**Time Allotted:** 15 minutes

**Discussion:**

**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 6pm every Sunday, with regular informal check in’s during the week
2. Group chose skype as external communication platform
3. 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: <fill this in > Phil and Thomas

Timekeeper: <fill this in > Thomas

Attendees: Phillip Edwards, Thomas Nixon, <anyone else?>

Report Presenter: Nicholas Morgan

Target Meeting Time: 5 Hours <edit this>

**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:** Week in review

**Time Allotted:** 10 Minutes

**Discussion:**

**Action Items:**

1. Supporting system selection
2. <any other items you think should be added here>

**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. <discussion for other items if any>

**Agenda Topic:** Week in plan

**Time Allotted:** 4 hours 30 minutes

**Discussion:**

**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:** Week in review

**Time Allotted:** 35 minutes

**Discussion:**

**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:** Week in plan

**Time Allotted:** 20 minutes

**Discussion:**

**Action Items:**

1. Initial Code Assignment and discussion

**Agenda discussion:**

1. Code Assignments
   1. VersionRec – Thomas
   2. Gui/Reporting – Nicholas
   3. FileRec – Phillip E.
   4. CPPUnit – Ivana/Josh
   5. Gui - Phillip M.

# 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 CPPUnit 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:** Week in review

**Time Allotted:** 20 minutes

**Discussion:**

**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:** Week in plan

**Time Allotted:** 25 Minutes

**Discussion:**

**Action Items:**

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

**Agenda discussion:**

1. Gui design was passed onto Ivana and Josh
2. CPPUnit testing was assigned to all members
3. Makefile fixes for Unit testing
4. Report writing assigned to Nicholas
5. FileRec and VerRec finalized assigned to Thomas and Phil. E.
6. Presentation for inclusion in report assign to IvanaFinal 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 CPPUnit 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:** Week in review

**Time Allotted:** 20 minutes

**Discussion:**

**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
   * + - 1. Gui – Jason and Ivana
         2. Report – Phil M. and Nicholas
         3. Version Rec – Phil E.
         4. File Rec – Thomas
         5. CppUnit testing – Phil E. and Thomas

## Detailed Meeting agenda and Report

# Detailed Meeting Report

[Date] – Date where this happened

Agenda Items:

1. Visualize the current design and structure of the program
2. Redesign area’s we feel can be improve.
3. Design and setup the database for use with testing throughout the remainder of the project

Time allocation:

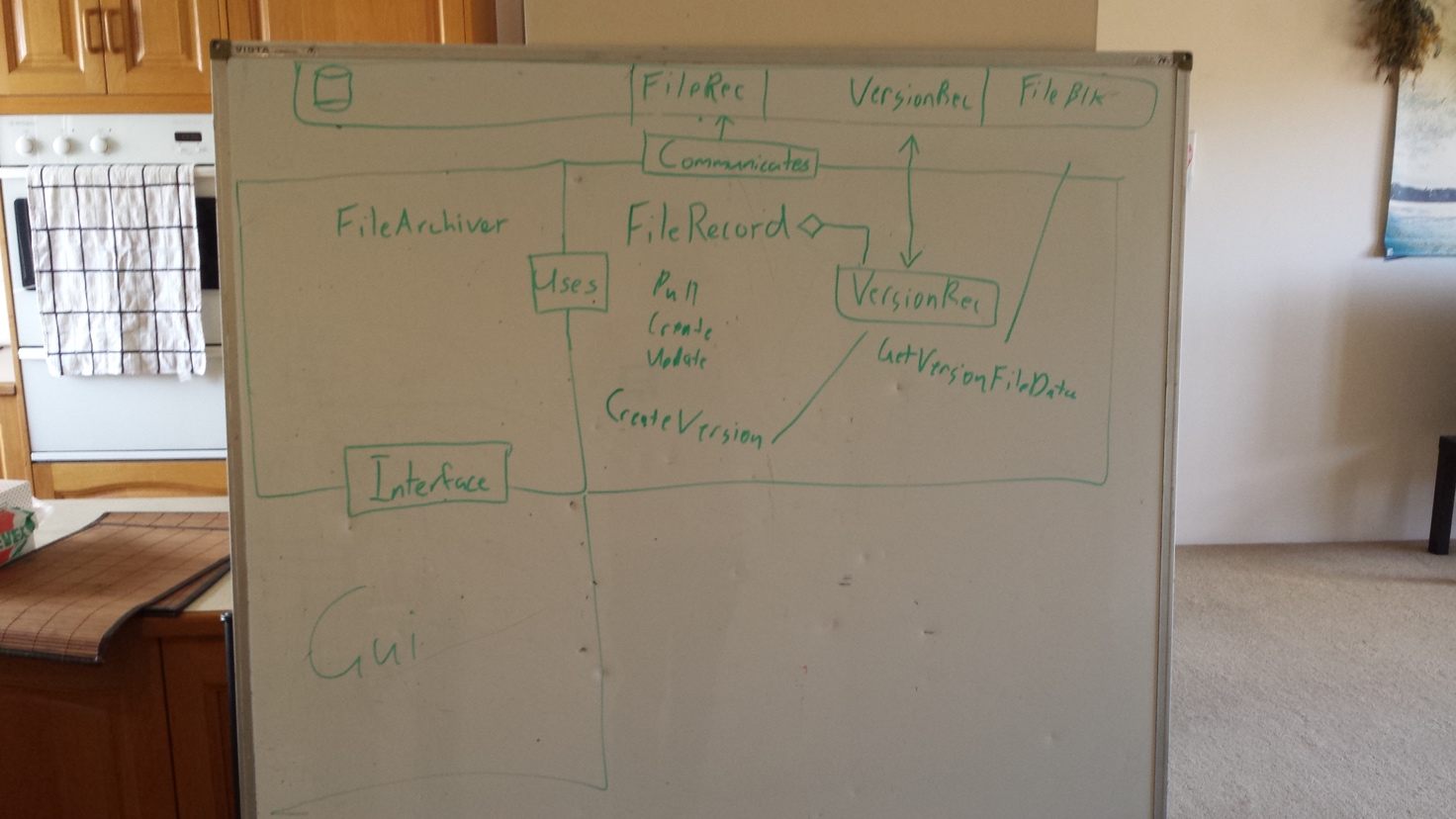
10am-10pm

Discussions:

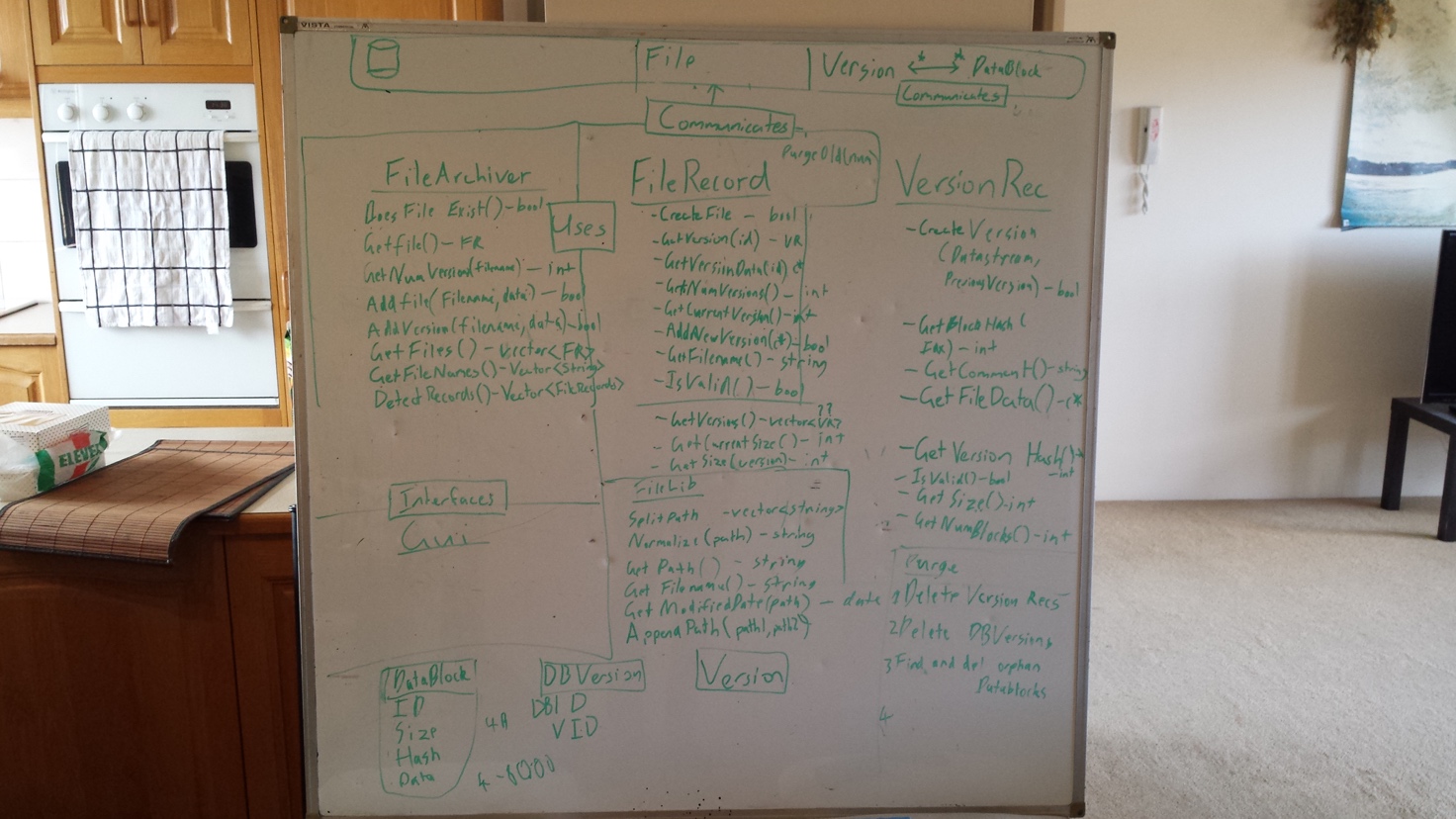
Discussion continued throughout the day, but started with a layout of the planned stages and ultimate goals for the meeting

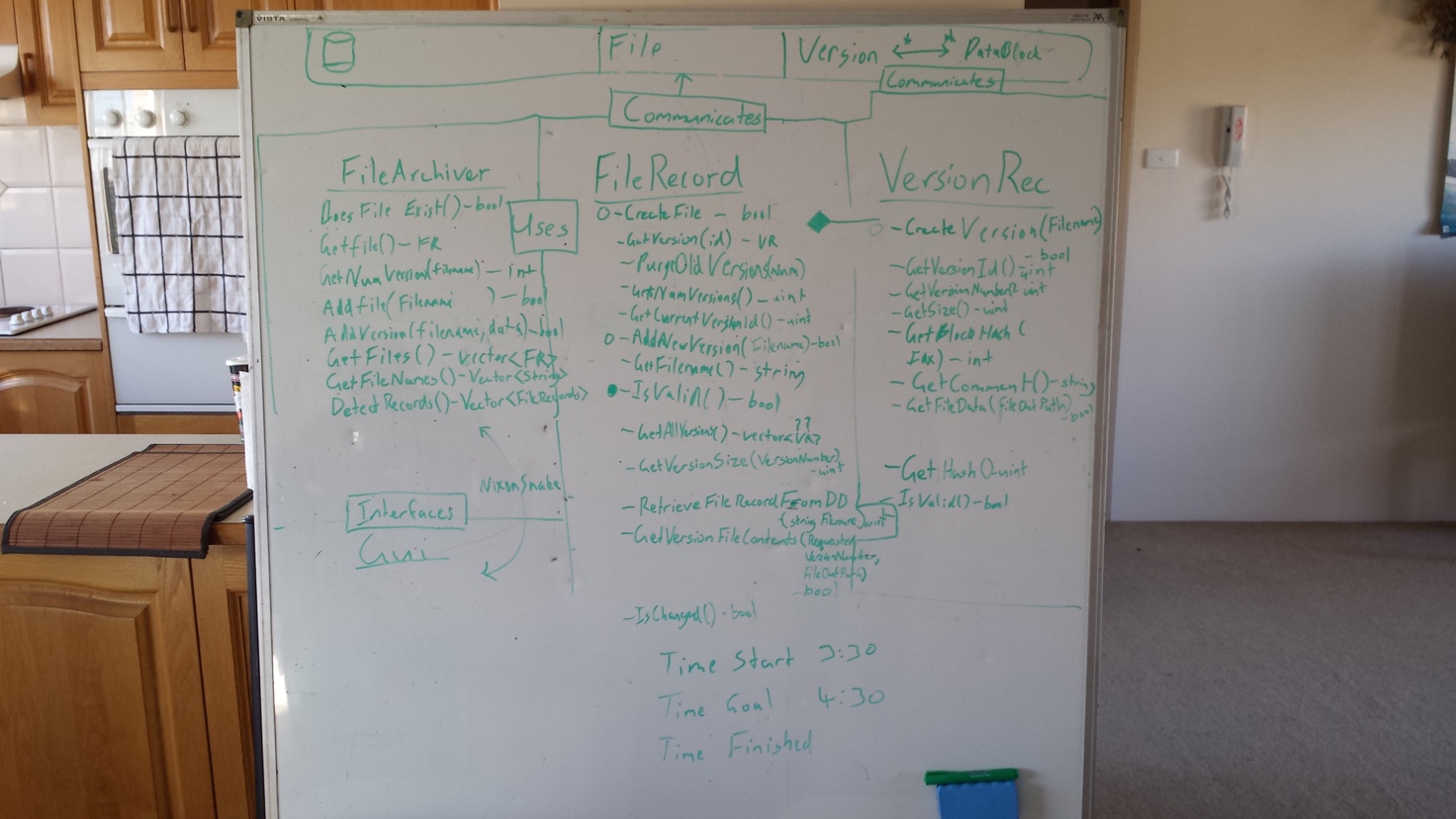
# Program Design

## 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.

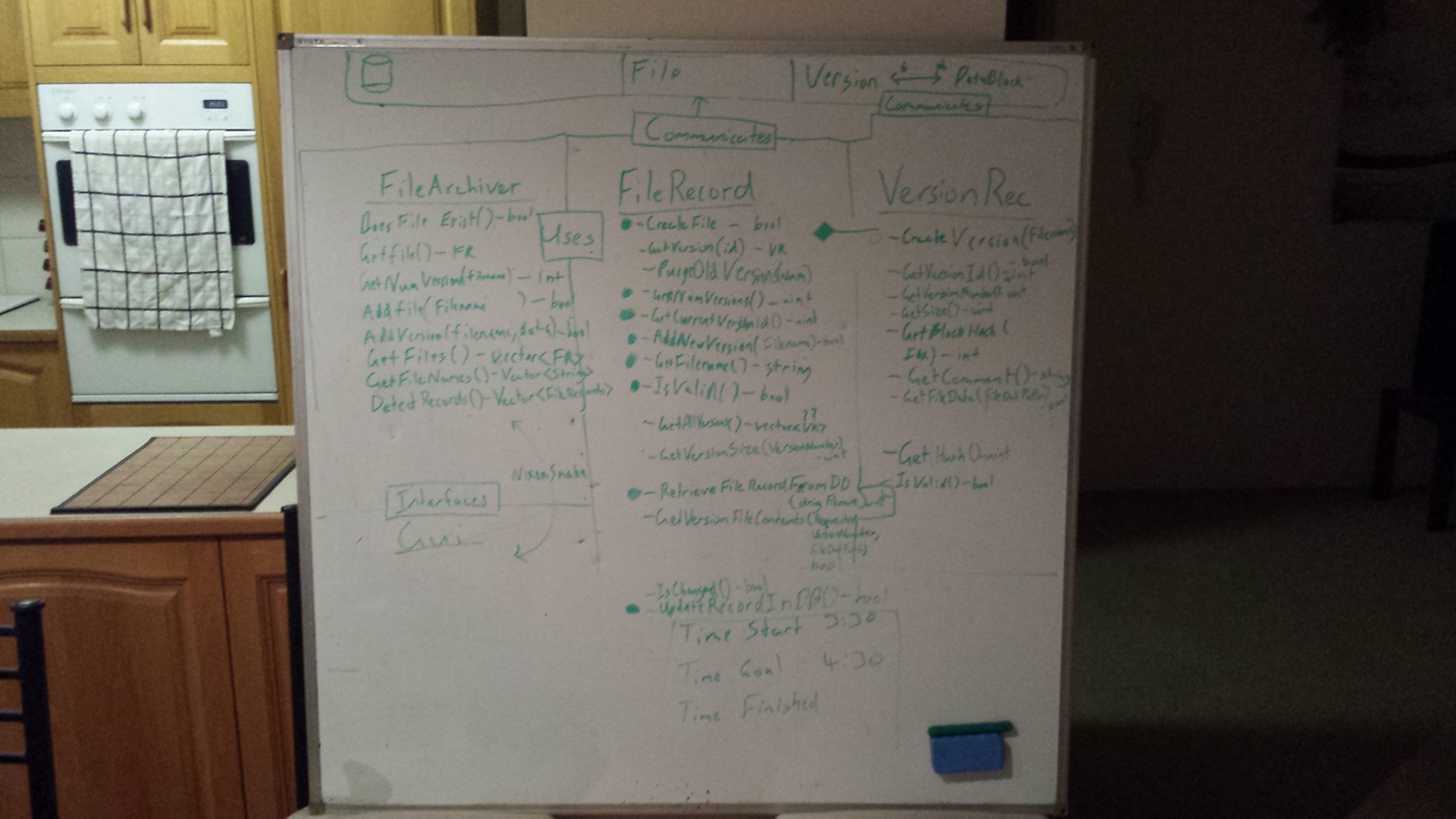
Files are stored in the database when a new VersionRecord is created through FileRecord. The VersionRecord class handles creation of the VersionRecord database entry. It also handles checking for duplicate blocks and creation of VtoB records which associate blocks with a version record.

### Retrieving a file from the database.

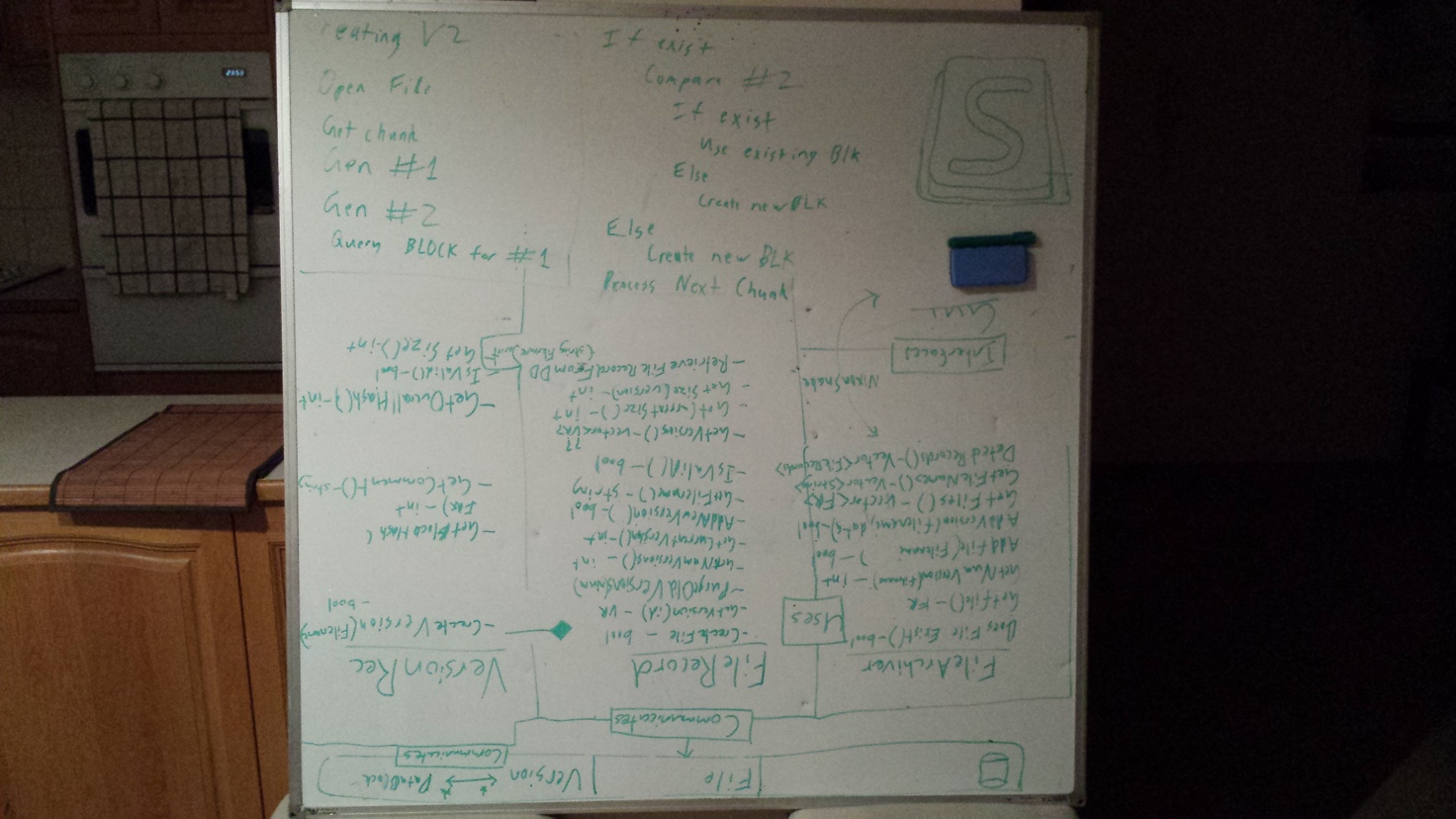
Retrieving a file from the database is accomplished through a valid VersionRecord instance. It will retrieve a list of VtoBs and then gather all required blocks and store them in a file on the database, before finally decompressing the file to the final destination.

### 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.

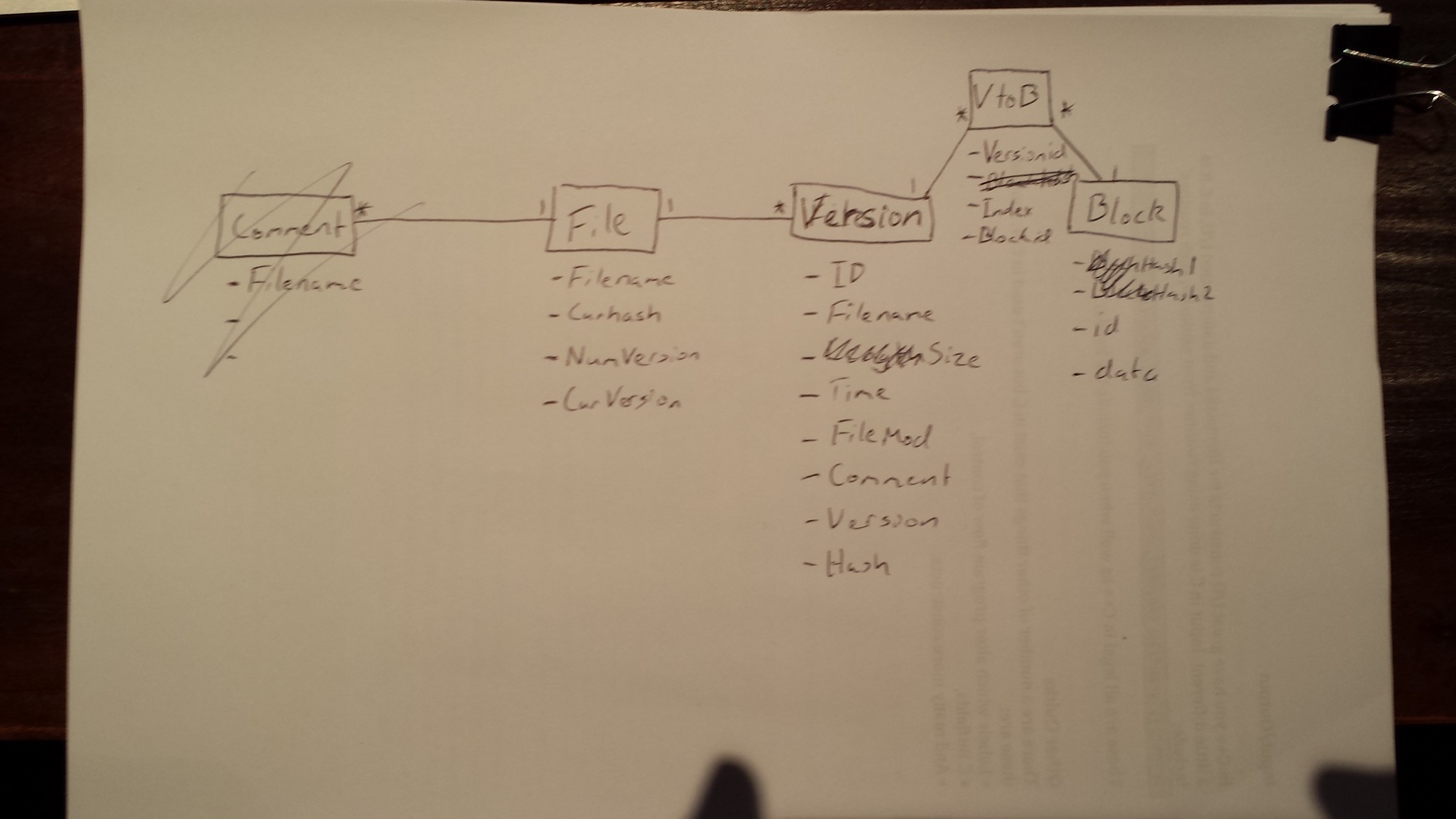
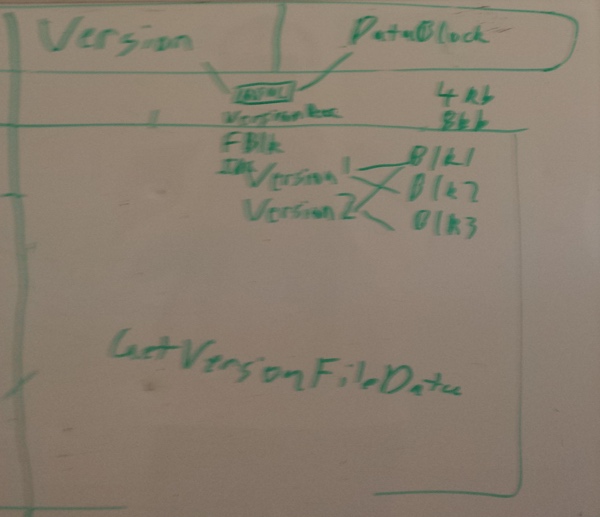


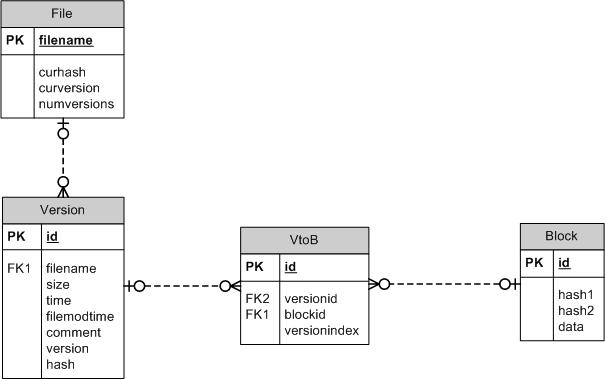
The whiteboard was also used to help us come up with logic in some of the functions. That  
way two people could look at it and make sure it made sense before implementing.

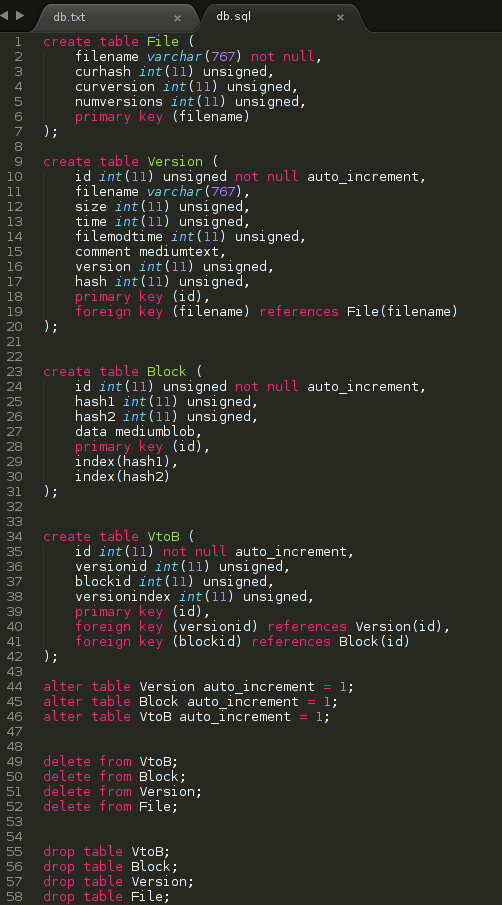


## Database

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.

An ERD was created to show how the database worked. This was useful if anyone needed to refer to the database or see under which table data was stored. 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.

This is the database code at our first implementation. The insertion and retrieval of a File, Version, Block, and VtoB worked perfectly.

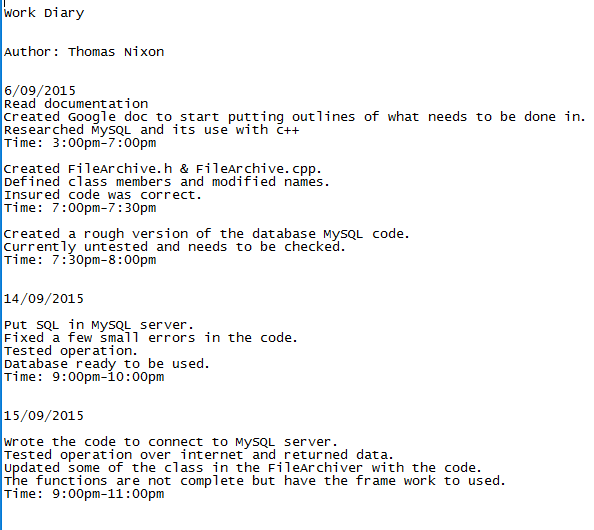
# Discussion – Final

At the end of the meeting, we were happy with the current state of the project, we identified some redundancies in the program and trimmed them, the database was setup and ready to be utilized and the current version of the version record and file record functionalities were implemented.

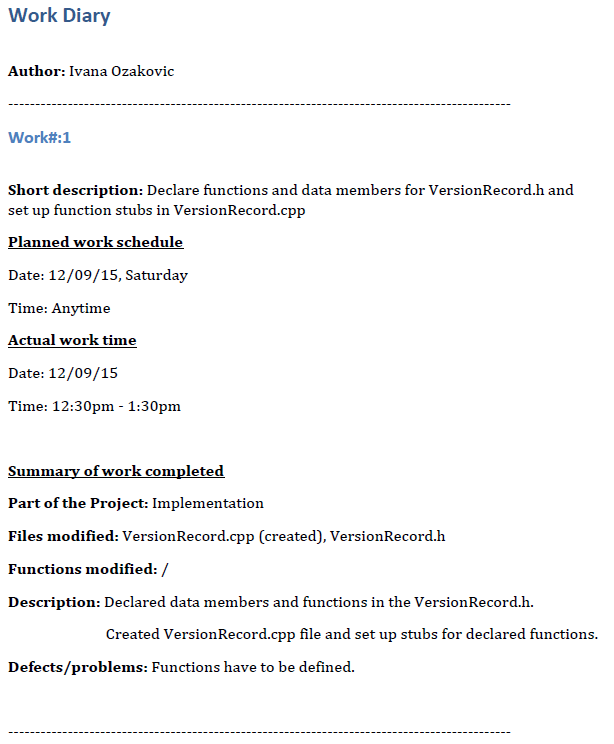
# Group member work journal Samples

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

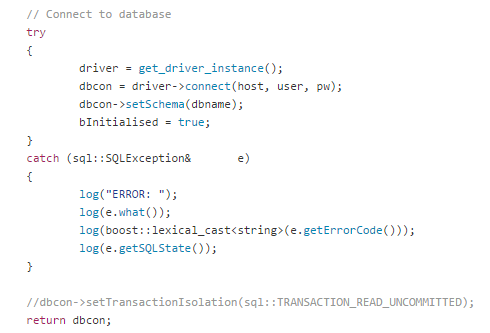


## Sample of work diary from Ivana Ozakovic

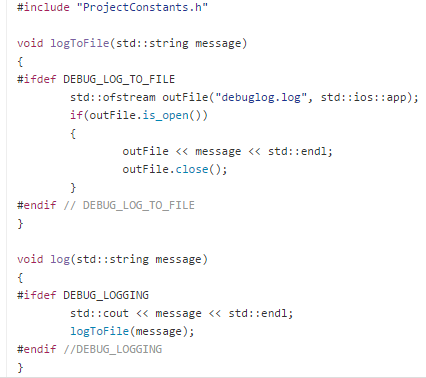


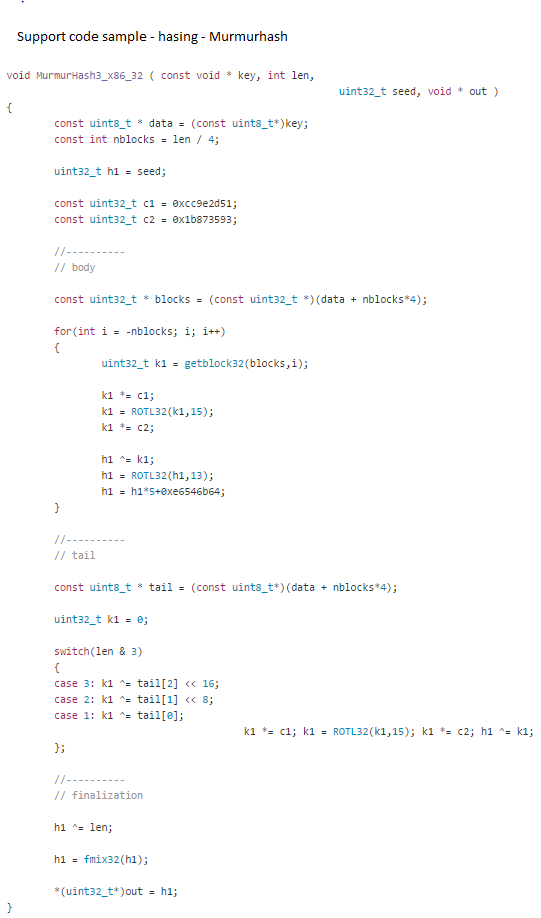
# Supporting Code Samples

## Support code for connecting to the database



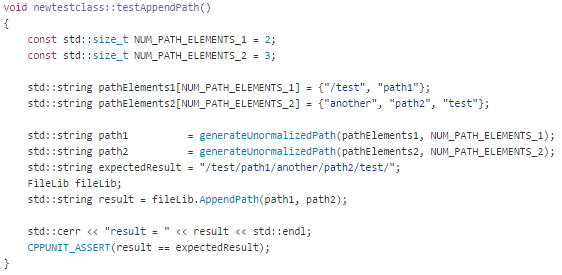
## Support for utilities for logging debug info



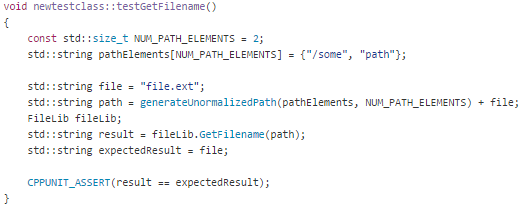


# Unit Testing Samples

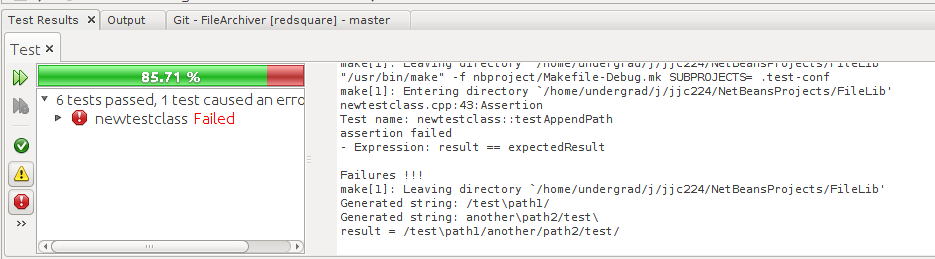
## Testing for correctly appended path to file



## Testing for retrieving filename functionality



## Bug/Testing Log Samples



# 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