**Milestone 5: Updated Design and Near-Complete Implementation**

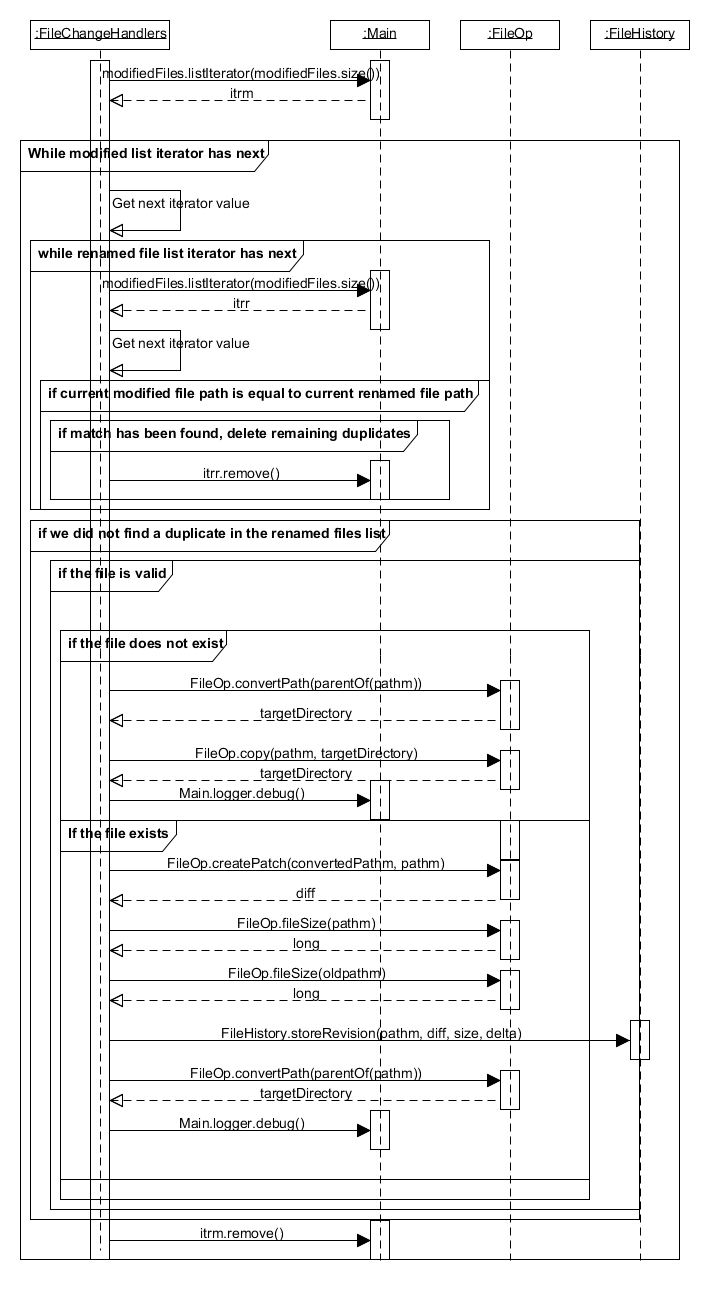
* **Group 06:**
* Mike Hoffert - mlh374
* Syed Ahsan Rizvi - sar457
* Hattan Alsharif - haa775
* Da Tao - dat293
* Michael Butler - mdb815

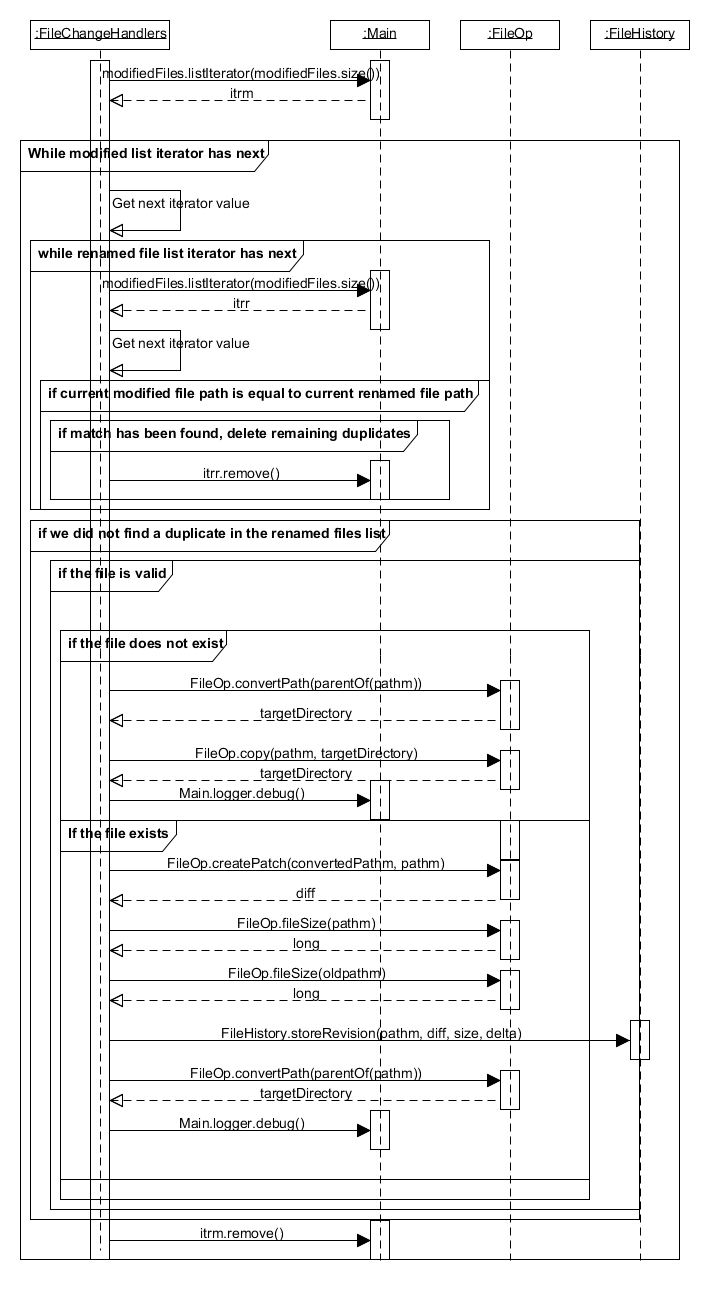
**1 - System operations**

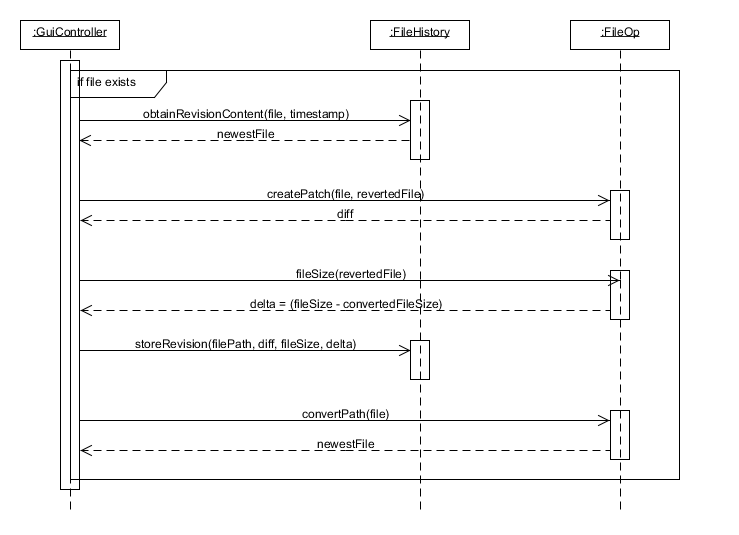
* Startup.resolveBackupDirectory()
* DbManager.setConfig()
* FileOp.createPatch()
* FileOp.applyPatch()
* GuiController.revertRevision()
* GuiController.viewRevision()
* GuiController.changeLiveDirectory()
* GuiController.changeBackupDirectory()
* FileChangeHandlers.handleRenamedFiles()
* FileChangeHandlers.handleCreatedFiles()
* FileChangeHandlers.handleModifiedFiles()
* FileChangeHandlers.handleDeletedFiles()

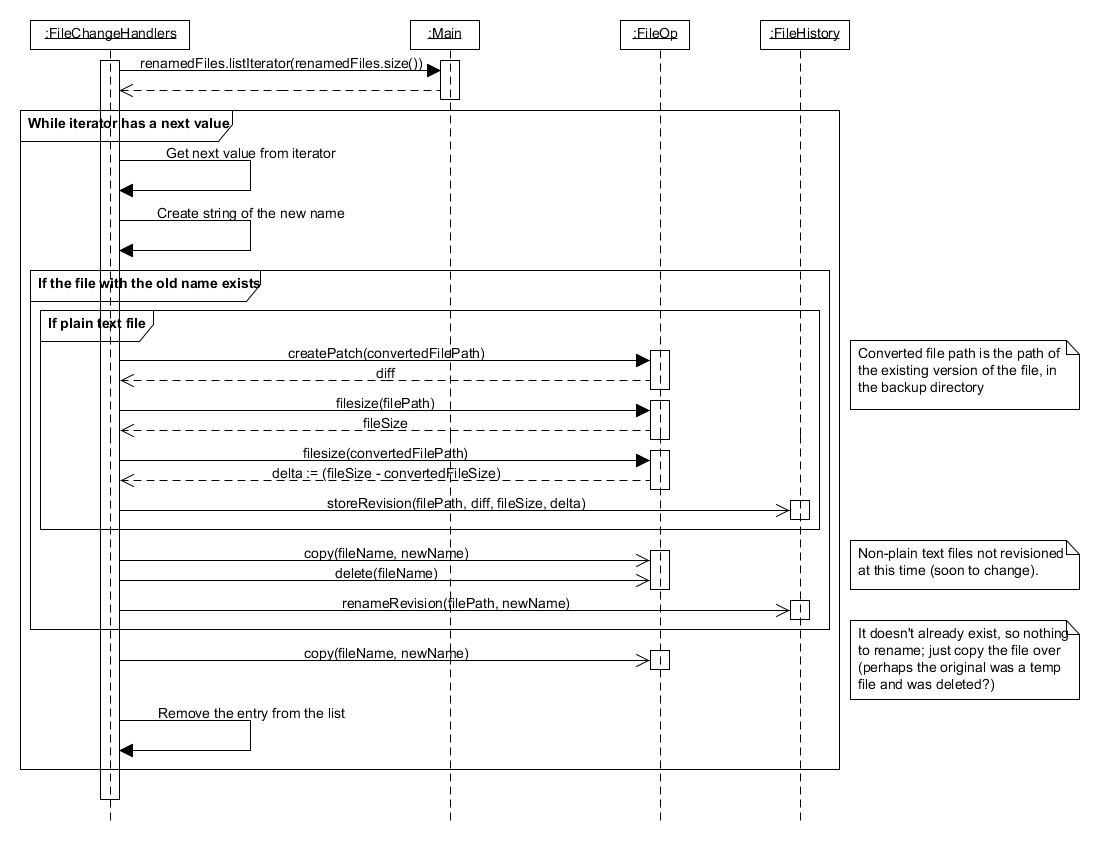
**2 - Sequence diagrams of major system operations**

**FileChangeHandlers.handleModifiedFiles()(Butler)**

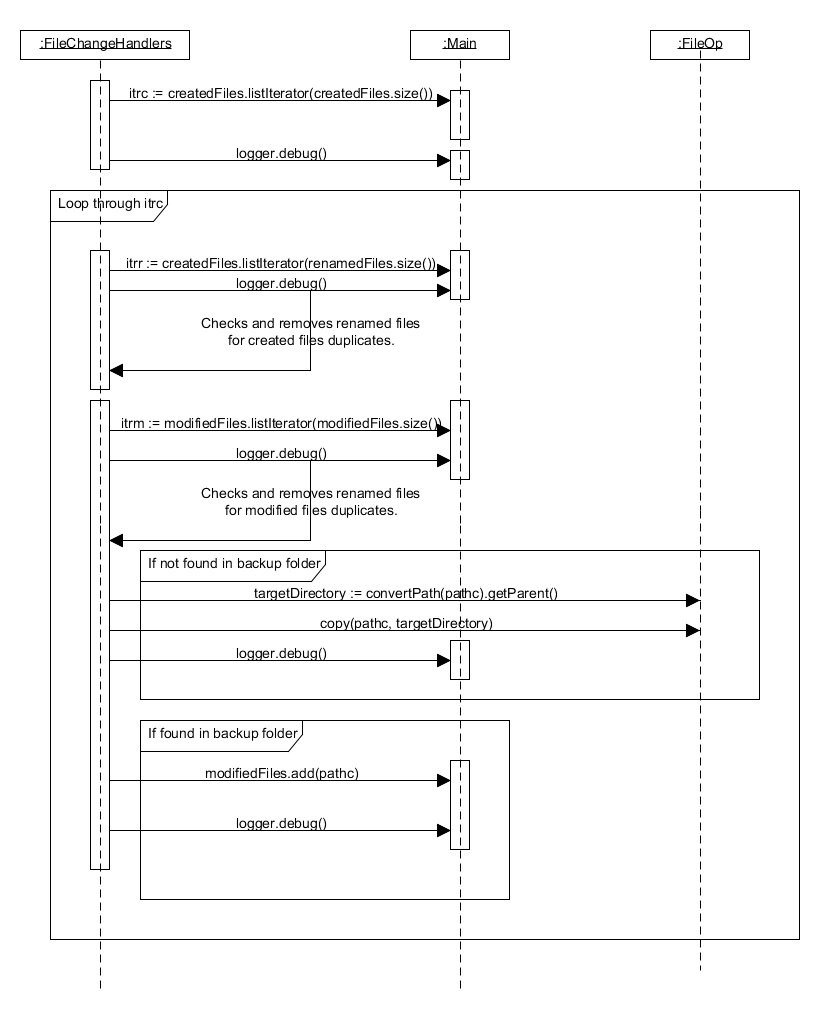
****

** GuiController.revertRevision()(Rizvi)**

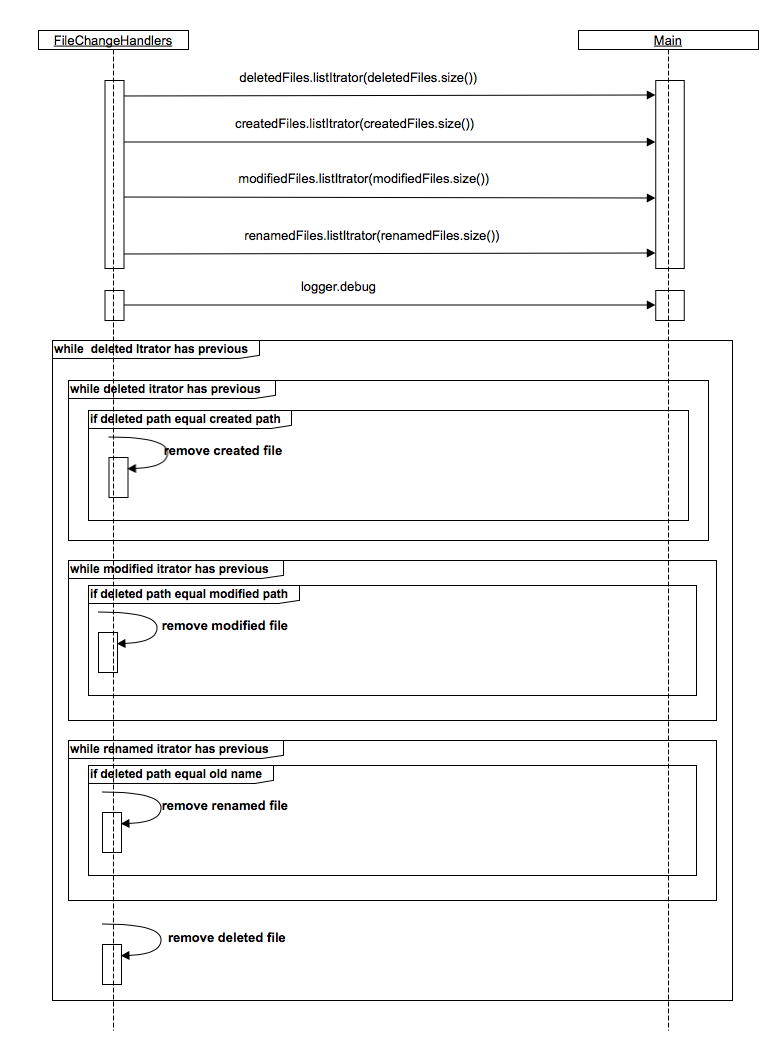
****

**FileChangeHandlers.handleRenamedFiles()(Hoffert)**

**FileChangeHandlers.handleCreatedFiles()(Tao)**

****

**FileChangeHandlers.handleDeletedFiles()(Alsharif)**

****

**3 - Class diagram**

The class diagram is located in the included class\_diagram.png file.

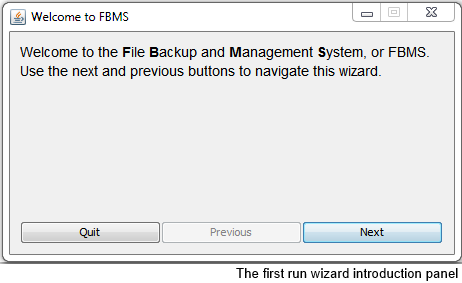
Used GRASP patterns include:

* The use of a GuiController class as a controller to handle system functionality for the front end.
* The FileChangeHandlers class also acts as a controller as it handles System events which are then delegated to other classes.
* The FileHistory class performs gets and sets the file history (its revisions), acting in a high cohesion way: it's focused on dealing with file history and leaves everything else to the other classes (in particular, it uses FileOp to perform operations such as creating the patch files that are the heart of revisions).
* The FileOp class is very low coupling. With the exception of getting the live and backup directory from Main (which most of the classes do), FileOp has no dependencies and works on its own as a low end collection of utility methods
* Most of the classes like FileChangeHandlers, GuiController, and DataRetriever make use of the principle of polymorphism as they operate on interfaces, encapsulating the internal implementation of the objects.
* DbManager is a class that provides high cohesion for database access while having low coupling to the rest of our program. By consolidating those functions interacting with the database into one class we avoid unnecessary coupling to the database itself while providing robust SQL interaction.

**4 - Implementation**

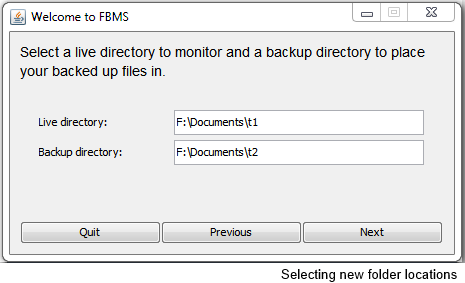
**Instruction manual**

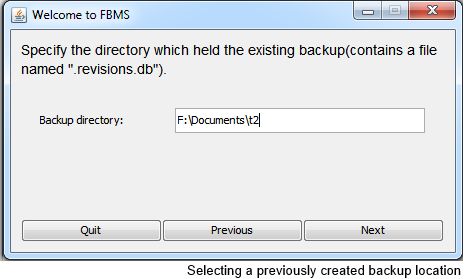
FBMS, or the File Backup and Management System, works akin to an offline Dropbox or Google Drive. When the program is started for the first time, a wizard will guide you through selecting your live and backup directories.



* **Live directory**: The folder that you want to keep backed up and revisioned.
* **Backup directory**: The location to store the backup. Don't use an existing folder, or you risk losing your files.

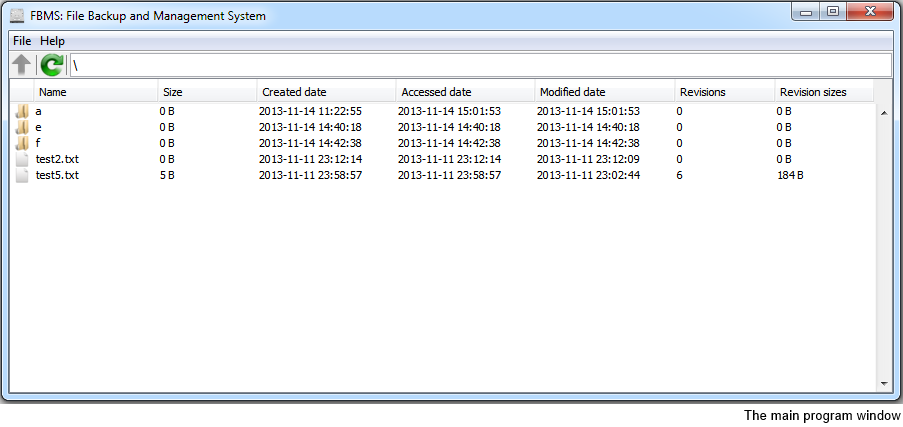
You can opt to start from scratch (specifying a backup and live directory), or import an existing backup directory (which will determine the live directory from previously saved settings).



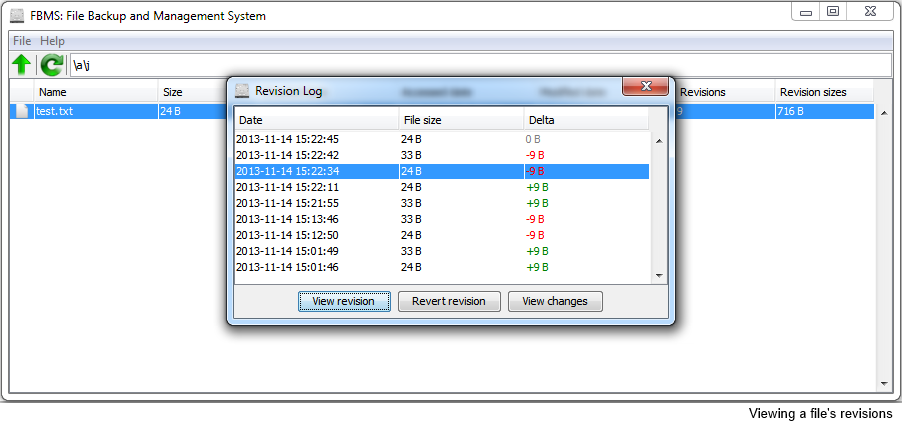


Once folders are chosen and the wizard is completed, the program will run quietly in the background. An icon will be created on the system tray. Double clicking this icon will open the program's interface.





The interface shows a special file browser, which displays the contents of the backup directory along with information such as the number of revisions stored. This file browser can be navigated with the keyboard or a mouse, and has functionality similar to other file browsers, such as Windows Explorer or Dolphin. It is, however, a specialized browser intended only to be able to restore files and view revisions.



Double clicking a file will open the revisions dialog, displaying a list of past revisions (if there are any). These revisions can be viewed (which opens the revision in your default program) or reverted (which sets that file in your live directory to the specified revision). You can also view the changes introduced by a revision, which will open in your default web browser.

The file browser is also able to:

* Copy the selected file to a chosen folder
* Restore all files
* Change the live or backup directory

FBMS is also able to detect when files have changed since the last time you ran the program. When starting the program up, FBMS will perform a one time scan for file changes. This scan is slow, and can be disabled in the settings if FBMS is always running.

The settings dialog is accessed under the File menu of the GUI. It provides a few choices such as:

* Whether or not to display non-fatal error messages (which don't stop the program from running, but notify the user of issues).
* Whether or not to run the previously mentioned first run scan
* Whether or not to enable the trim feature, which removes revisions older than a specified date.

**Meetings**

|  |  |  |
| --- | --- | --- |
| Date | Attendance | Discussion |
| 29 October | Mike, Da | Discussed progress of project and others' work; Handling of binary files |
| 2 November | Mike, Ahsen | Optional meeting for implementation; GUI design; GUI file browser table; Taskbar icons |
| 4 November | Mike, Ahsen, Hattan | Personal meeting for development of GUI and file diff system |
| 5 November | All | Minor implementation; testing; GUI revision table; Obtaining specific revisions; Handling file changes; Settings dialog |
| 9 November | Mike, Da, Ahsen | Optional meeting for implementation; GUI wrap-up; bug handling; testing; Milestone 5 plans; Milestone 4 review; Non-backwards compatible changes; Logging |
| 12 November | Mike, Da, Ahsen | Optional meeting for milestone 5; testing results |

As with before, scheduling meetings is very difficult due to the very full and very different schedules of the group members. We had optional integration meetups on Saturdays in Spinks, which were mostly a way to get everyone coding with easy access to their group members.

The majority of communication stems from email contact. We've exchanged well over 100 emails since.

**SVN log**

The SVN log is located in the included file, svn\_log.txt.

**5 - Project plan listing**

**Spent time**

Mike:

- Planning and documentation: 5 hours

- Milestones: 15 hours

- Utility demo: 8 hours

- Skeleton implemention and

PoD classes: 2 hours

- Watcher class: 1 hour

- DbManager.init(): 3 hours

- FirstRunWizard class: 6 hours

- Data.getFolderContents(): 2 hours

- Various Control code related

to startup: 5 hours

- JUnit testing code: 4 hours

- Errors class: 3 hours

- MainFrame table: 5 hours

- MainMenu functionality: 2 hours

- MainMenu functionality: 2 hours

- MainToolBar functionality: 1 hour

- RevisionDialog table: 2 hours

- SettingsDialog: 3 hours

- Redid FileOp.createPatch and

FileOp.applyPatch: 2 hours

- Data.getTableData and

Data.getRevisionData: 3 hours

- Data number formatting: 1 hour

- Trim database feature: 3 hours

- FileOp utility functions: 2 hours

- Miscellaneous bug fixes: 4 hours

Total: 84 hours

Da:

- Rough design of modules: 1 hour

- Rough design of functions: 1 hour

- Work on previous milestones: 4 hours

- Work on FileOp: 7 hours

- Milestone 4(text): 5 hours

- Milestone 4(graph): 2 hours

- Test case: 2 hours

- Work on FileHistory: 6 hours

- Miscellaneous bug fixes: 2 hours

- Redo operation contracts: 1 hour

- Build.xml: 4 hours

- Milestone 5(graph): 5 hours

Total: 39 hours

Ahsen:

-Milestone 4 (documentation): 5 hours

-Frontend implementation: 4 hours

-GUI Brain storm(in total): 7 hours

-GUI implementation: 10 hours

-Testing: 3 hours

-Milestones: 9 hours

Total: 38 hours

Hattan:

- Milestone 4: 9 hours

- Work on Presentation slides: 3 hours

- FileOp.creatDiff: 4 hours

- FileOp.fileToList: 4 hours

- Test FileOp.creatDiff: 2 hours

- Test FileOp.fileToList: 3 hours

- FileOp.applayDiff: 6 hours

- Milestone 5: 4 hours

- Instruction manual pics: 2 hours

Total: 37 hours

Michael:

- Milestones: 8 hours

- getConfig(): 3 hours

- setConfig(): 3 hours

- handleCreatedFiles(): 6 hours

- handleDeletedFiles(): 2 hours

- handleModifiedFiles(): 4 hours

- handleRenamedFiles(): 3 hours

- testing(general): 4 hours

- debugging/optimization(): 4 hours

Total: 37 hours

**Future assignments**

At this point of time, the program is largely completed in terms of functionality. There is much testing, quality assurance, and bug fixing to do, however.

We've tagged a stable version as version 1.0. We're currently working on improving MIME type detection, revisioning of binary files, and making the system more object orientated.