# 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:

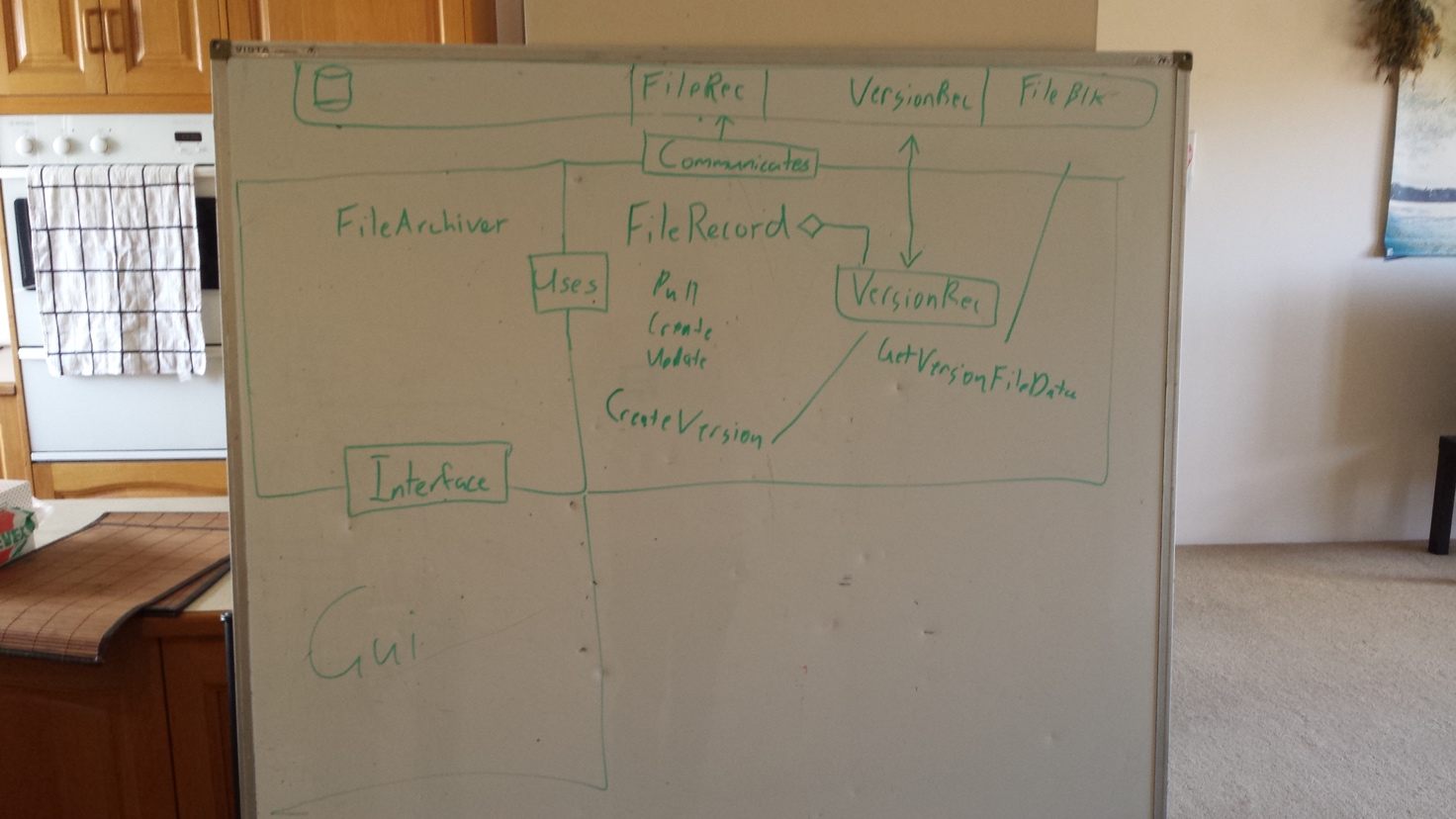
10am-10pm

Discussions:

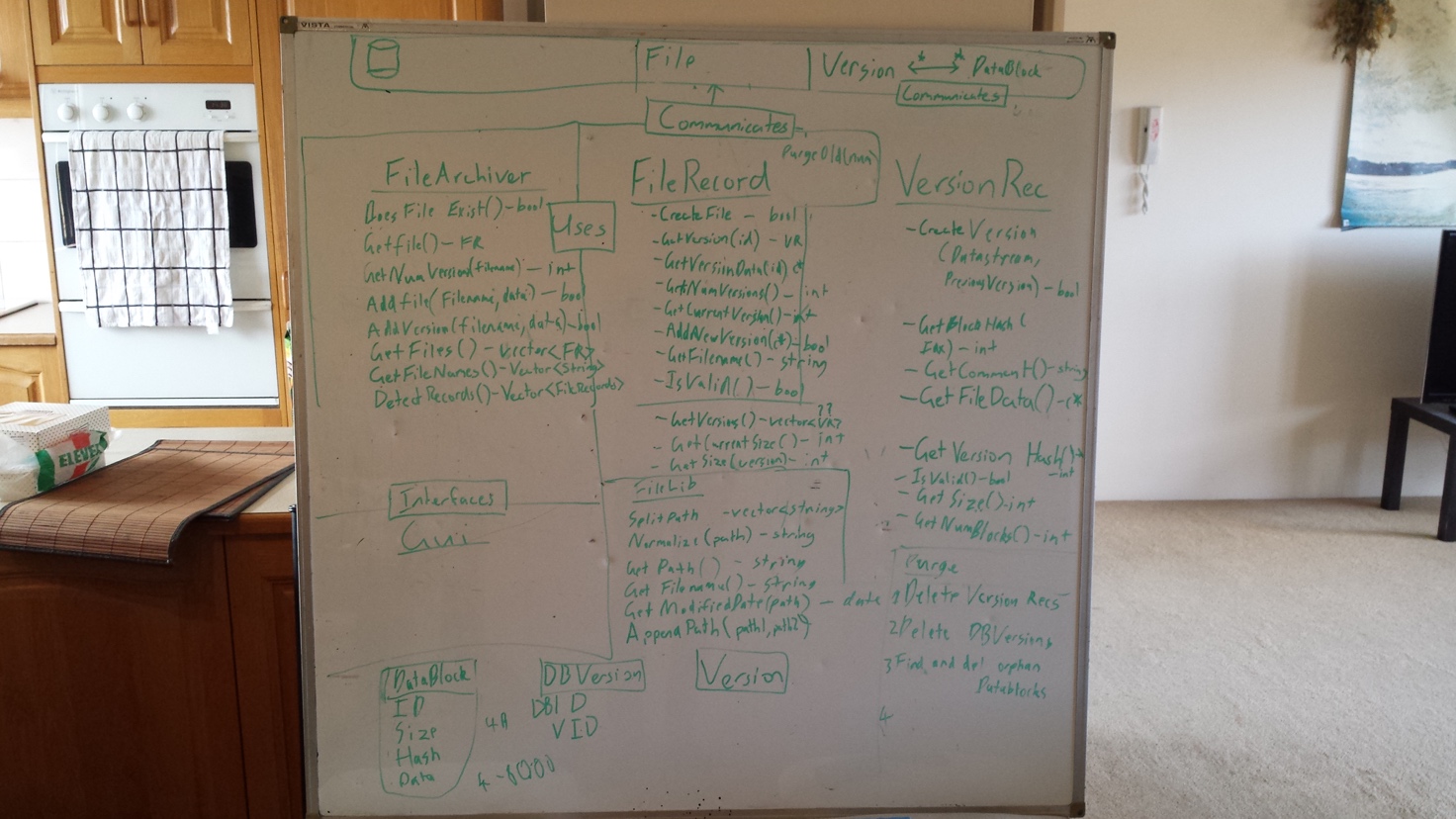
Discussion continued throughout the day, but started with a layout of the planned stages and ultimate goals for the meeting

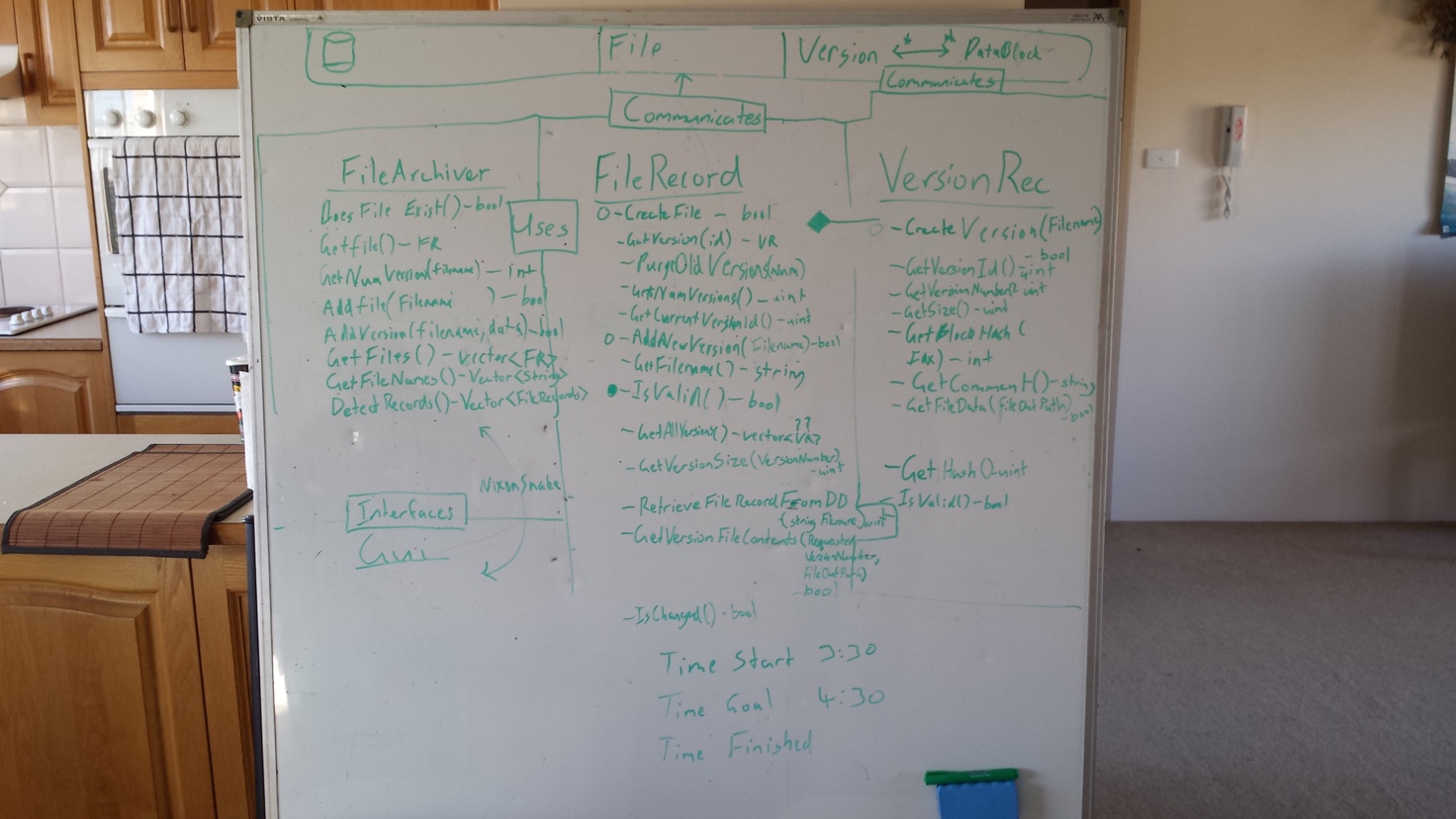
# Program Design

## 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.

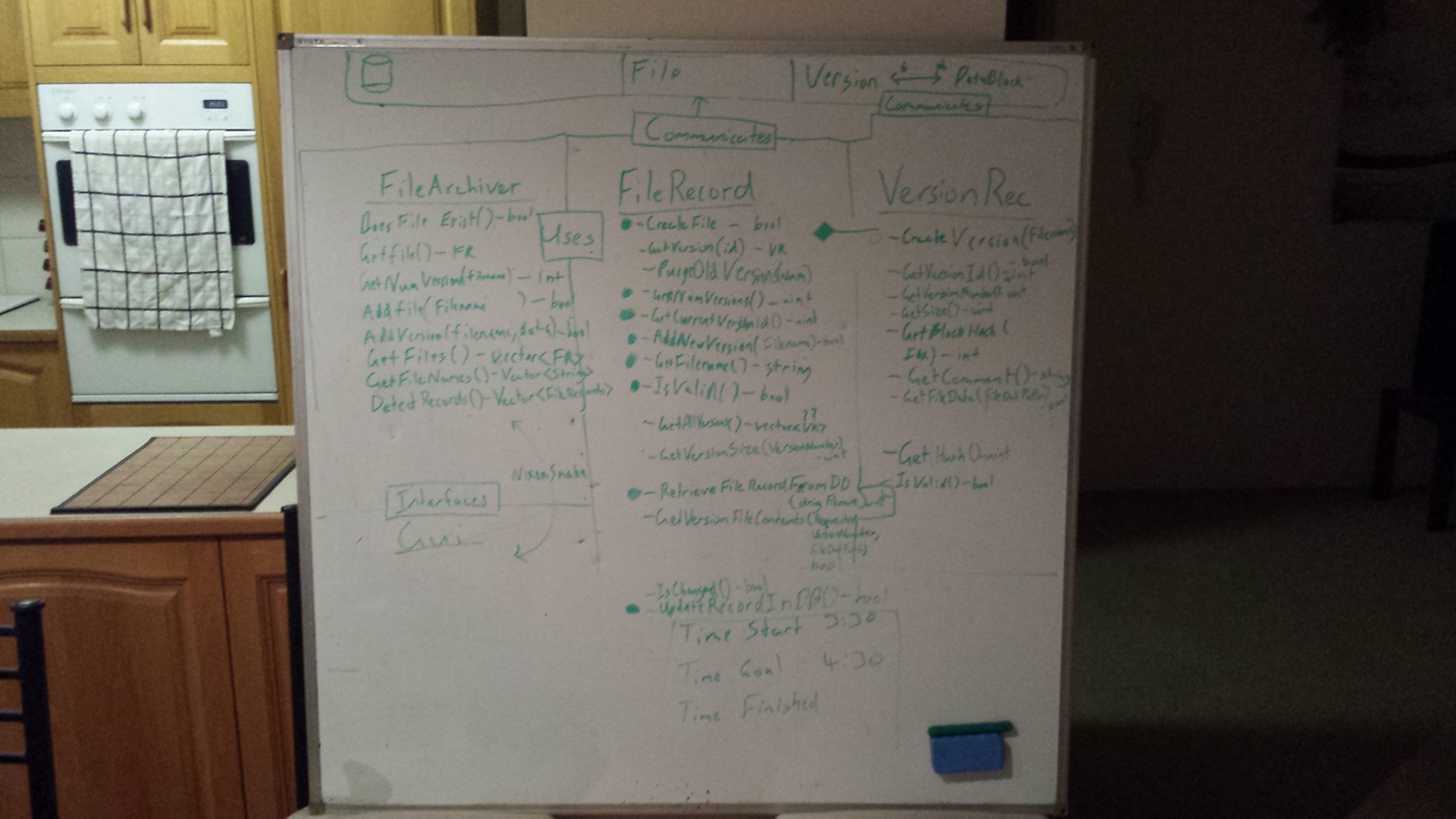
Files are stored in the database when a new VersionRecord is created through FileRecord. The VersionRecord class handles creation of the VersionRecord database entry. It also handles checking for duplicate blocks and creation of VtoB records which associate blocks with a version record.

### Retrieving a file from the database.

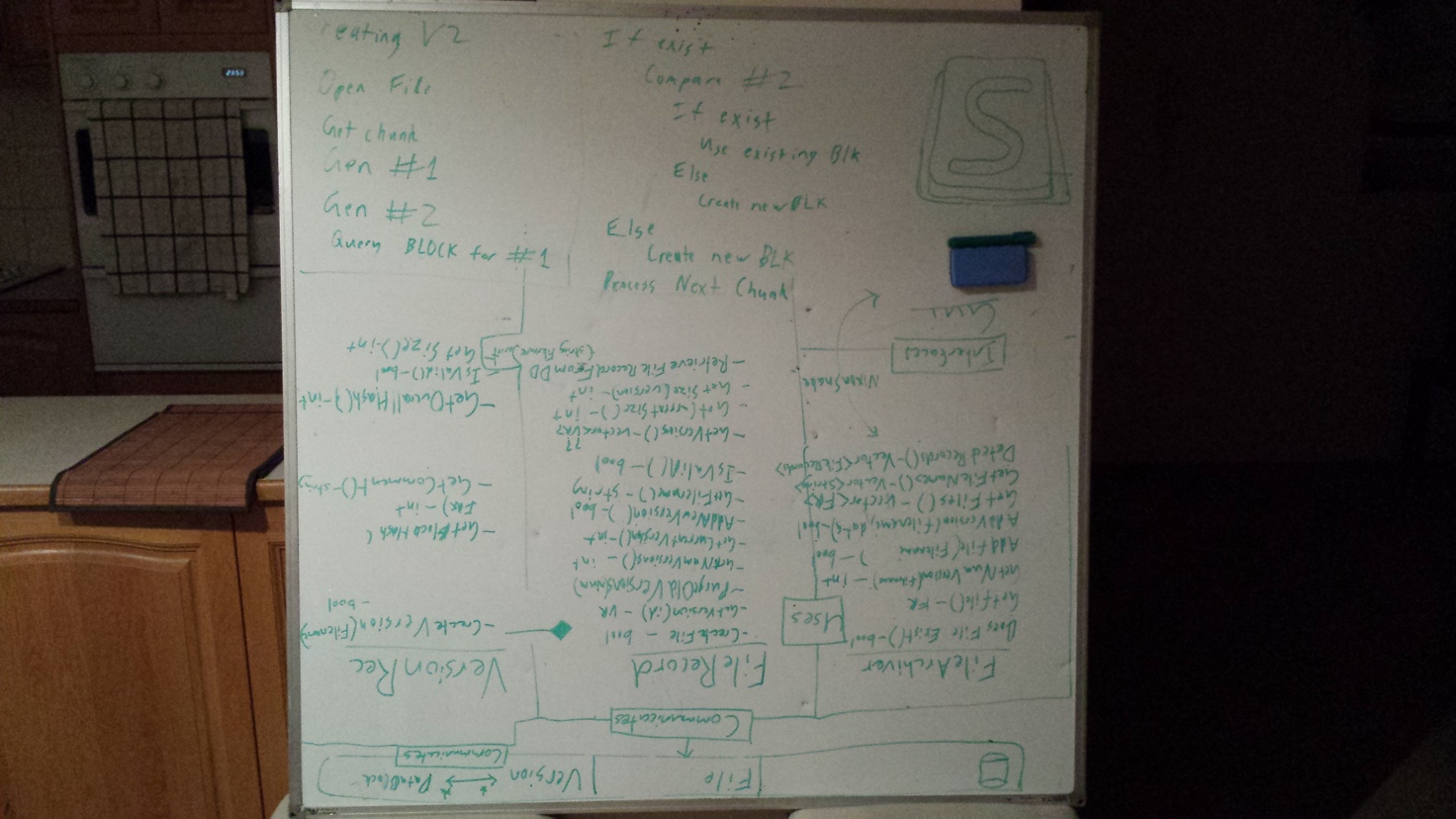
Retrieving a file from the database is accomplished through a valid VersionRecord instance. It will retrieve a list of VtoBs and then gather all required blocks and store them in a file on the database, before finally decompressing the file to the final destination.

### 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.

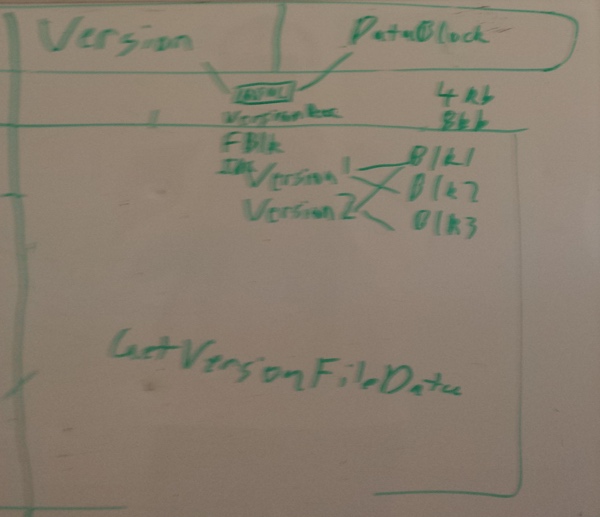


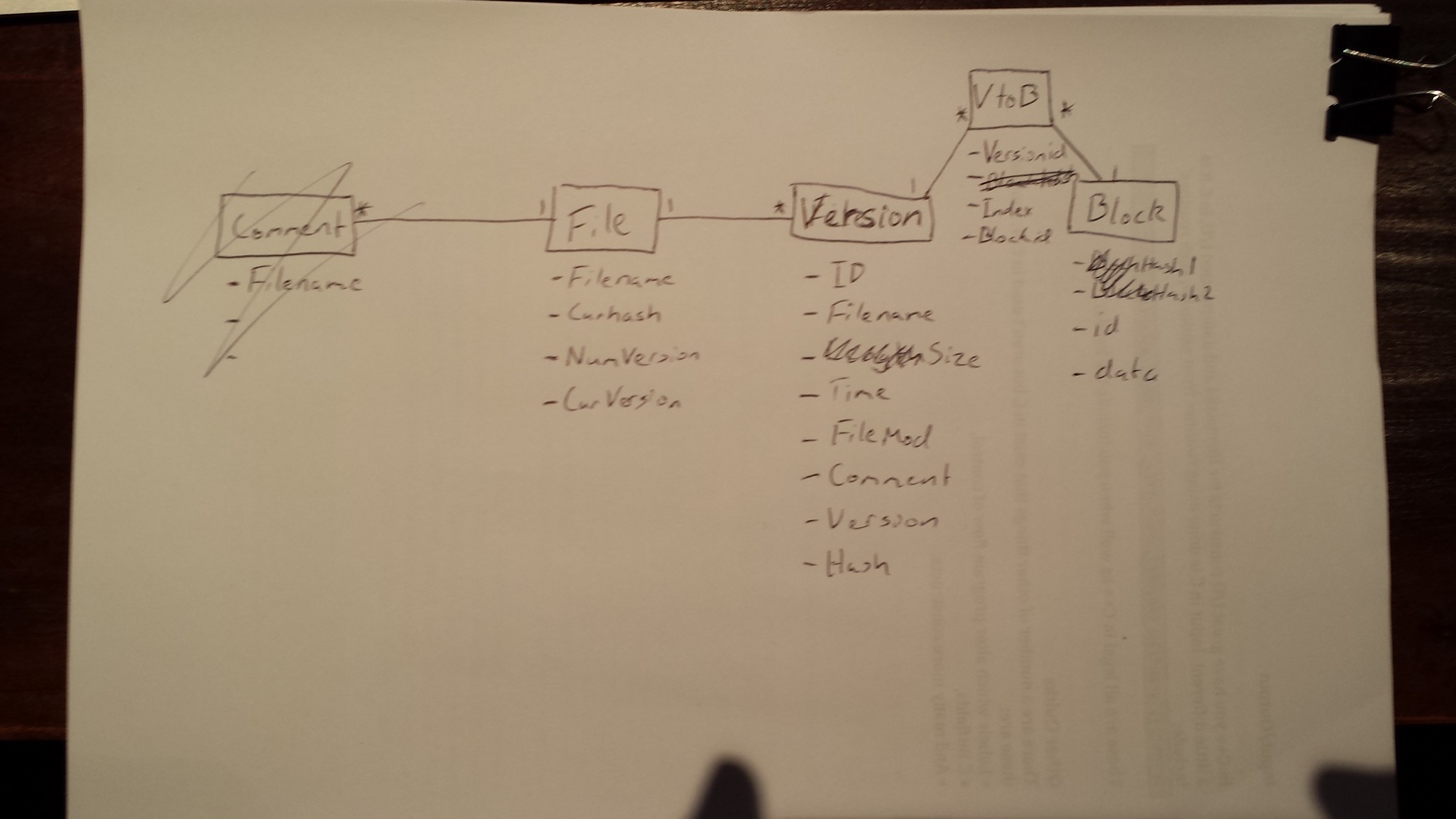
The whiteboard was also used to help plan out the logic in some of the functions. This enabled many people to see the design as it evolved, have input and ensure that it was clear before implementing.

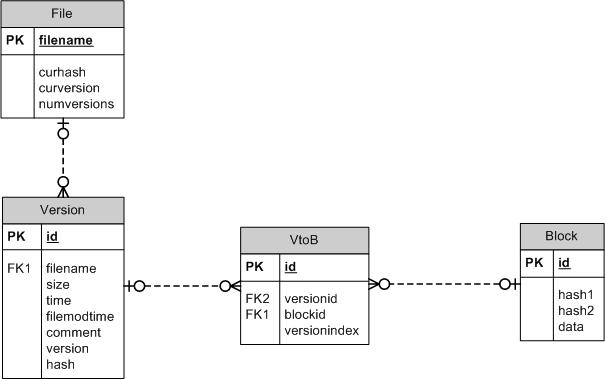


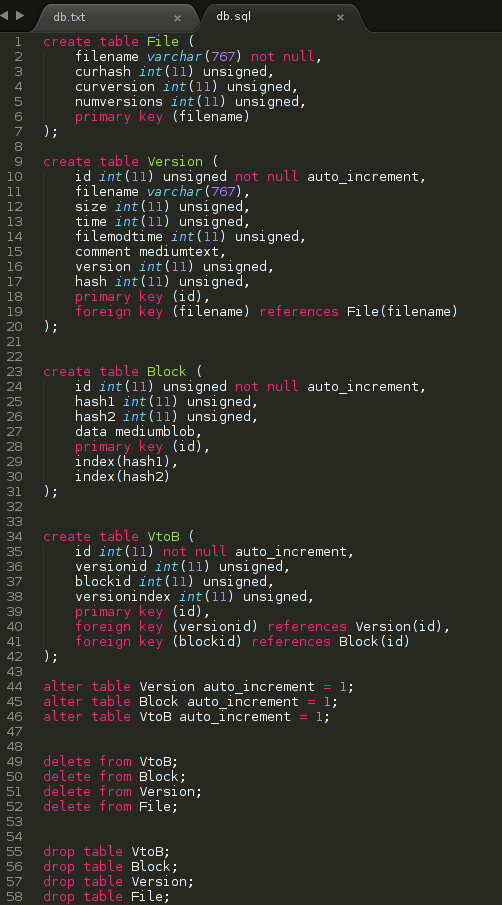
## Database

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.

Agenda Items for next meeting

1. X
2. Y
3. Z