Schedule

**Weeks 1-3:**

* liaise with other students and identify potential group members.

**Week 4:**

* Finalize group
* Informal group meeting
* Discuss personal strengths and weaknesses relating to project
* Identify appropriate RUP roles
* Agree on meeting times and places for real group work
* Communications links – email, phone, group communication.

**Week 5:**

First formal group meeting

* Assign roles
* Review project specification
* Assign tasks relating to evaluation of technology etc. Your group should assign to individual members the responsibilities for exploring technologies (including supplementary technologies such as use of subversion).

Second formal group meeting:

* organise work allocations for core algorithms, data persistence, networking, and user interface.

**Week 6:**

* Aim to have at least a crudely implemented version of the core algorithms section

identified for iteration 1.

**Week 7/8:**

* Iteration 2 completed.

**Week 8/9:**

* Iteration 3 completed; Report writing.

**Members**

Thomas Nixon

Philip Edwards

Josh Coleman

Ivana Ozakovic

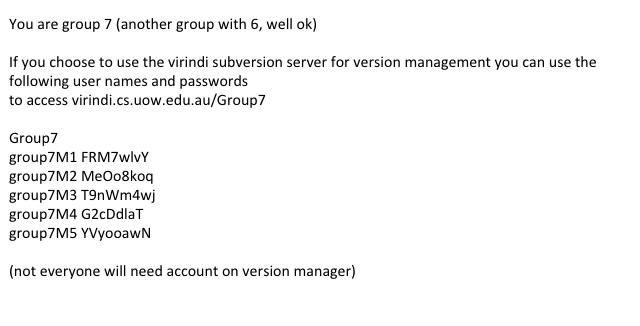
The Other Phil

Nicholas Morgan

**Roles**

|  |  |
| --- | --- |
| **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, The Other Phil |
| Documentation (\*) | many |
| Implementer (\*) | many |
| Document Backup Maintainer (1) | The Other Phil |

Email from the lecturer:



# Mark breakdown

**Presentation of report - 4 marks**

This is a fairly large fraction of the overall mark. Much of “Software Engineering” is really just a matter of communication with your stakeholders. Your markers are stakeholders.

*They want a report that is complete, clear, and concise.*

Factors contributing to this portion of the mark will include: the clarity of the “retrospective project overview”, the overall structure of the report document, appropriate use of embedded images (modeling diagrams, screen shots, subversion reports etc), layout of sections (particularly code layout), indexing of the report, clear sectioning, English grammar and spelling, and general appearance.

**Design and planning - 3 marks**

These marks are for your group’s work on completion of the design supplied, particularly your work on the data persistence and user-interface aspects.

**Lab demo - 2 marks**

Convince a tutor that it really does work and work well with a good demonstration.

**Project management - 5 marks**

These marks are based on evidence for the use of an effective software development process.

Factors contributing to this part of the assessment will include: evidence for an effective group structure and adoption of roles; a well defined iterative construction phase; effective group collaboration as evidenced by meaningful group meetings; disciplined development processes of

individuals.

**Code - 6 marks**

* Based on the quality of the C++ implementation code
* Evidence for testing such as the unit test code and reports.
* Use properly organized header and implementation files for classes
* Good C++ coding style is required
* Elements written by different implementers should be consistent in style (Coding Standard).
* The code should not contain errors.
* Control flow should be well structured.
* Remove shit (All those trace statements *cout << “Start function insertnew” << endl;* they may have been essential during development, but remove them from the final version!)

# Implementation Checklist

[**https://docs.google.com/spreadsheets/d/1cm7PirAqpf5pUjfOZSe6NzmB37ZZvlRI7MVDniZDPc0/edit?usp=sharing**](https://docs.google.com/spreadsheets/d/1cm7PirAqpf5pUjfOZSe6NzmB37ZZvlRI7MVDniZDPc0/edit?usp=sharing)

Coding Convention

<https://docs.google.com/document/d/1JoXoEunYJn6wyRo0nborxpSpfM4oq9f0-SfS7k6ewro/edit?usp=sharing>

# Decision List

MySql

# Git

You need to create an account

<https://github.com/boristsr/redsquare>

# Iterations

**First iteration**

There is an opportunity for work in parallel by those working on the GUI interface and those working on the core components.

The MyWindow GUI can be prototyped. Actually, it can be pretty much completed. It can use a dummy version of the “FileArchiver” that will allow an “initial record” to be created, and subsequent versions to be added, and which will provide made up placeholder data for version numbers, modification times, and lengths so that the version display can be tested.

Most aspects of FileRec and VersionRec can be implemented and tested essentially independently of the rest of the project (working with dummy data inserted using MySQL Workbench or the 'mongo' client application). There is some opportunity for using the cppUnit unit test framework to test the implementation of these classes.

The work on the FileArchiver class could start with an exploration of how to transfer a complete file to persistent storage, and how to build and store a single FileRec with data for an actual file.

At a minimum, completion of the first iteration should see:

* A GUI interface that “works” (just showing dummy data)
* A FileRec class that can read a file and create correct data for length, modification time, collection of block hashes etc.
* A FileArchiver class that can create a “blob” record for a complete file and retrieve that record back to some other local file, and which can transfer a FileRec to/from the persistent store.

Implementation

* Define classes - FileRec, Version Rec, “FileArchive”
* Design GUI
* Implement base functionality for classes

Design

* Use Cases
* Sequence Diagrams
* Activity Diagrams

**fileRec/verRec - phil**

**GUI - Nic**

**FileArc- Nixon**

**CPPUnit testing - ivana, josh, other phil**

**Use cases - ivana, josh, other phil**

**Second iteration**

The second iteration will see the integration of the GUI component with the increasingly realistic implementation of the FileArchiver and related classes. This integration step should be quite simple to achieve.

The main work in this iteration will be sorting out details of how to deal with multiple versions and how to store data correctly in persistent storage and later retrieve chosen versions.

At a minimum, completion of the second iteration should see:

* A GUI interface that really works, operating with an instance of the actual FileArchiver class
* Completed FileRec and VersionRec classes
* A FileArchiver class that can insert new
* Update (possibly still using the initial version of the file as the reference for the update process rather than the version most recently saved)
* Get version info
* Retrieve version

**Third iteration**

If not done earlier, the “compression” aspects should be implemented. The file should be gzipped before transfer to persistent storage (and gunzipped as part of the retrieval process). Individual blocks associated with versions should be compressed and decompressed using the functions from the Qt library.

**Google Drive project folder link**

<https://drive.google.com/folderview?id=0B2hbPevvzoSYfmducV9pSVVFM0o1ZnAzYVVCbVpacGIyT3oxOHROWDFCNGlhRFd1ZHRLamc&usp=sharing>

Ivana- version record.h

Phizo- filerecord.h

basegui - Nic

Everyone - read over coding standard + provide feedback

# Design

