Versioning Scientific Research Data

University of St Andrews

CS5099 Dissertation in Computer Science

MSc Advanced Computer Science

Joshua Donohue-Channon

130007105

Word count: 8000

Abstract

In many academic fields, academic publications are based on newly collected data. This data often forms the basis of a publication, it is then necessary to manage this data in a manner speeding up the publication process due to its ability to improve data workflow. A smooth, efficient, and concise workflow is crucial for the management, cataloguing, archiving, and publishing of data.

The University of St Andrews has developed NOMAD, a system to streamline the workflow for NMR based research. Currently in NOMAD, users are unable to edit data because the NOMAD's experiments are immutable. Users can annotate and analyse the machine generated experimental data with meta data, for example, grouping the experiments into compounds and projects. It would be useful if users could annotate the experimental data, but doing so in a manner whereby the original data is not lost.

This project will implement a versioned storage-based service for scientific research data as a solution to the problem faced by NOMAD. The aim is to create a file versioning system for non-computer scientists who are unfamiliar with version control.

This project created a strong data model and developed a novel solution to versioning scientific research data. It can also be applied to generic data, giving it a vast array of possible applications.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 8,000 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona de library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Joshua Donohue-Channon

23rd October 2017

Table of Contents

1 Introduction

1.1 Overview of Research Data Workflow

Research data workflow is the evolution through the stages of a research data lifecycle. The lifecycle of data is the chronological steps research data goes through from the beginning phase, for example planning an experiment or discovering resources associated with anticipated area of research, to manipulation and analysis of the generated data, until publication of the data and report. [1]

Workflow can be categorised into two distinctive segments into which the various stages of the lifecycle fall; active and inactive. Active is chronologically the first group of stages. 'Active' refers to stages that require a significant amount of active input from the researcher. These processes are very different compared to inactive stages because they are the creation and manipulation of data. Examples of regular inactive phases includes publishing, archiving, and sharing. These are stages which do not contain any creation or manipulation of data; but rather the movement of it.

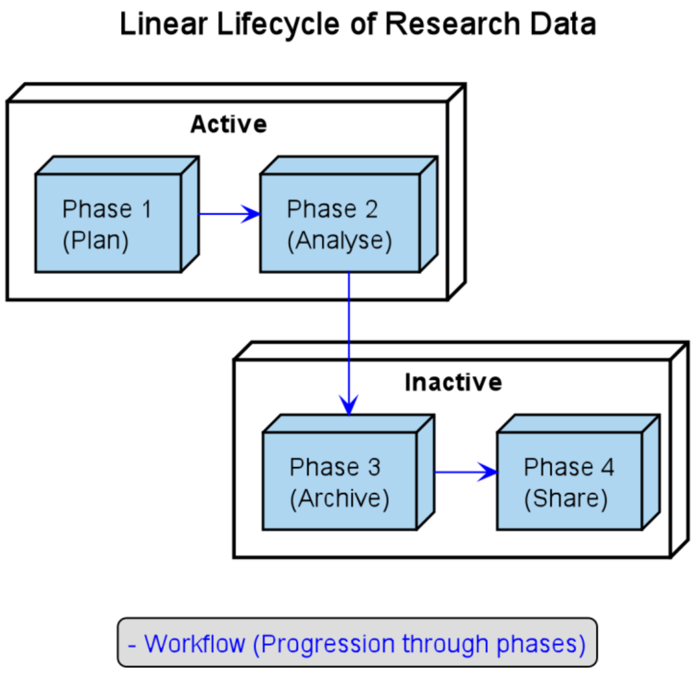


Figure 1: Example of basic linear lifecycle structure.

The workflow and life cycle of data are often considered to be the same thing. There are several categorises of lifecycle or workflow models and these can be differentiated based on either who designed the model, or the use of it. Linear models are the easiest to understand and breakdown, however circular models are likely the most accurate in depicting the reality of research data lifecycles. [1]

In addition, data workflow has essential processes that also occur which do not always fall into a single phase. These are separate processes which occur throughout the cycle in multiple phases. These include the documentation of the workflow process, and the application of metadata, as well as the backup of data produced, in order to prevent the possibility of physical loss or creation of errors. [1]

1.2 Workflow phases

The data produced from the active phases is not normally integrated into data repositories in many cases. Typically, the method for recording this data is the application of physical cataloguing using notebooks, (the online equivalent is also a regular method for recording) or some institutions may have a specialised method/ process introduced for recording data. These are often built with unique needs in mind. As a method of cataloguing the evolution of data, this is an efficient method and if integrated into a data repository correctly they could be an exceptionally effective form of data management. Therefore, the method of recording the product from the active phases affects the transition of date from the active phase onwards, and as such, it has a knock-on effect upon the workflow. Further integration of data from the active phase would result in optimisation with an increased likelihood of being able to avoid compromising the data with avoidable errors. There should be an equal emphasis on both stages of a data lifecycle in terms of being managed correctly. Every phase in a single cycle is critical and should not be overlooked or handled improperly.

It can be concluded that a smooth and concise scientific data workflow is crucial for an institutions ability to catalogue data correctly, and therefore, manage and share their data publicly; as is promoted by the Research Councils UK and the European Commission. [2] The optimum workflow helps researchers to save time, reduce the likelihood of human errors where possible, and further promote data collaboration across an institution or field of research. [1]

1.3 Research Data Management

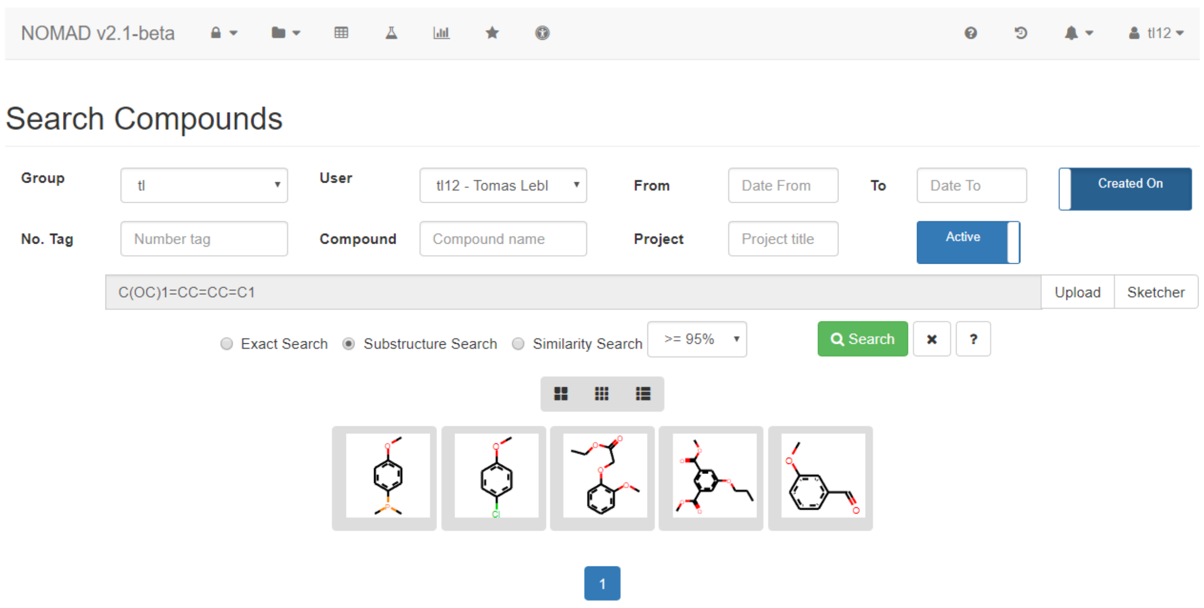
Research data management is an integral part of research data workflow. Classically the general focus of research data management services and systems has been on addressing the final stages of the research data lifecycle; archiving and sharing. [3]

Research data management is the organisation and structuring of a data workflow. Effective management is integral to successful research because it ensures the data that is the basis of the research can be used to validate the research. [4] Successful management of data is measurable by the accessibility and ease of comprehending said data. Therefore, data management can be measured by the ability to obtain information when required and furthermore the data is citable and can be used to verify research. Also, it is so that data with long term value which has been assembled can be retrieved and understood in the future. [5]

1.4 Nuclear Magnetic Resonance

Nuclear Magnetic Resonance (NMR) spectroscopy is a research technique that utilises the intrinsic magnetic properties of atomic nuclei. This type of spectroscopy can be used to extract physical and chemical properties of the atoms and the molecules that contain these atoms. NMR spectroscopy is most commonly used in organic chemistry, but it is applicable to any molecules that contain nuclei with spin.

NMR Online Management and Datastore (NOMAD) is a cloud service principally designed to automate and streamline the workflow for NMR spectroscopy. NOMAD handles lab management, data acquisition, and data access. The system is currently deployed to the School of Chemistry at the University of St Andrews, which has 6 NMR spectrometers and over 600 active users.



Group: tl
User: tl12 - Tomas Lebl
From: Date From
To: Date To
No. Tag: Number tag
Compound: Compound name
Project: Project title
C(OC)1=CC=CC=C1
Exact Search [✓ Substructure Search] Similarity Search >= 95%

Figure 2: Nomad compound search. [http://nomad.wp.st-andrews.ac.uk]

The NMR facility in St-Andrews University must manage large quantities of data and NOMAD, an online cloud based research data management system, has been implemented to ensure a suitably fast and reliable workflow. Before NOMAD was introduced the system was much more haphazard. Experiments were booked on physical paper rather than digitally, and the resulting data was spread across several storage units. This setup made it more likely for errors to be made, as well as more difficult to find and collect data, reducing the efficiency and reliability of the workflow.

NOMAD's function is to improve the workflow of data produced from the NMR analysis. It falls into several stages within this specific workflow. This data workflow is mainly handled by the Pure Data Repository. NOMAD's first role is for lab management, this is in the form of it controlling the booking of experiments. Its second function is to collect the raw data from the Pure data repository. Its final function is to allow interaction between the user and the research data through an online portal. [2]

1.5 The Problem

Currently in NOMAD, users are unable to edit data because the NOMAD's experiments are immutable. Users can annotate and analyse the machine generated experimental data with meta data, for example, grouping the experiments into compounds and projects. It would be useful if users could annotate the experimental data, but doing so in a manner whereby the original data is not lost. Implementing versioning is a possible solution.

1.6 The Proposed Solution

This project will implement a versioned storage-based service for scientific research data. The aim is to create a file versioning system for non-computer scientists who are unfamiliar with version control. The task will involve making the user interface simple and intuitive, this is to try to prevent the users having to learn new technologies or version control. The end goal of the project is to build data versioning capabilities into the NOMAD system.

2 Objectives

2.1 Primary

2.1.1 Versioning

V1 Design and implement an interface for a version control system (VCS) that can be integrated with NOMAD, and which will allow for versioning at the compound-level tracking changes in experimental data.

2.1.2 Client-side application

A1 Design and implement a desktop application that will provide a mechanism for the user to stage the annotated data for uploading and versioning. The application will run on the client's computer and will be a stand-alone application that can be installed on the client's computer without the need for any additional third-party software.

A2 Extend the application's capabilities to manage the uploads in a manner that avoids race conditions between multiple users uploading data simultaneously.

A3 Collect qualitative feedback from selected expert users of NOMAD.

2.1.3 Web interface

W1 Design and implement an interface to visualise and manage the versions of the compounds and projects from within the NOMAD web interface.

2.2 Secondary

2.2.1 Versioning

V2 Extend the versioning capabilities of the VCS to the project-level tracking changes in Compounds.

V3 Extend the versioning to include tracking changes in meta-data.

2.2.2 Client-side application

A4 Conduct usability testing about how users interact with the system, collecting quantitative data and more qualitative data.

2.2.3 Web interface

W2 Implement the desktop application's functionality in the NOMAD web interface, to remove the need for the desktop application.

2.3 Tertiary

2.3.1 Versioning software

S1 Design and implement a system that will allow for the VCS to be applied generically to any form of scientific data or project. The system will also contain a RESTful API that will be used to fetch changes from the user, either through a web interface or a desktop application. The system will be self-contained and can be deployed with minimal configuration.

3 Context survey

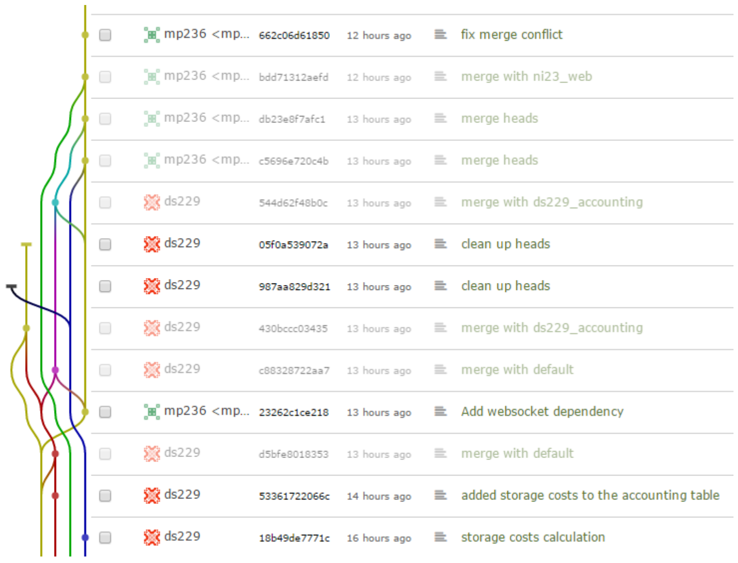
3.1.1 Existing solutions

Versioning of scientific data is often primitive, often only consisting of a basic file naming system [6]. Standards for archiving data and maintaining records differ significantly between different science disciplines, fields and even research groups. There are solutions for versioning databases [7] but none for versioning smaller datasets or individual experiments. Datahub [8] is a collaboration tool for scientific datasets, that offers tools to process, view and share data. The versioning capacities and revision management of Datahub are developmental and do not provide a comprehensive solution for scientific data versioning [9]. The closest solutions are pure version control systems, but version control has a significant learning curve for those outside the realm of computer science. Hence, a more bespoke solution is required.

3.1.2 Version control

Version control systems (VCS) track changes to documents and allow users to save versions of files, often termed as revisions. VCS provides the ability to restore old revisions of files. This allows users to track other users' changes, correct errors, and improve collaboration. Some of the more popular VCS are Git [10], Mercurial [11] and Apache Subversion (SVN) [12].

Version control systems work by finding changed files in the project directory, often using a hashing function [13]. A hashing function will generate a universally unique identifier (UUID) [14]. Each file save in the repository will have a UUID. When a file is changed the generated UUID from that file will be different and so it is easily identifiable as changed. The changed file can be uploaded to the repository. More advanced VCS will track the specific changes within the individual files and not just the changed files. This can be done using a diff utility [15]. Manging of revisions is often done using source trees. Source tress are a way of visualising the version control. An example of a source tree can be seen below, in Figure 3.



```
mp236 <mp...  662c06d61850  12 hours ago  fix merge conflict
mp236 <mp...  bdd71312aefd  12 hours ago  merge with ni23_web
mp236 <mp...  db23e8f7afc1  13 hours ago  merge heads
mp236 <mp...  c5696e720c4b  13 hours ago  merge heads
ds229         544d62f48b0c  13 hours ago  merge with ds229_accounting
ds229         05f0a539072a  13 hours ago  clean up heads
ds229         987aa829d321  13 hours ago  clean up heads
ds229         430bccc03435  13 hours ago  merge with ds229_accounting
ds229         c88328722aa7  13 hours ago  merge with default
mp236 <mp...  23262c1ce218  13 hours ago  Add websocket dependency
ds229         d5bfe8018353  13 hours ago  merge with default
ds229         53361722066c  14 hours ago  added storage costs to the accounting table
ds229         18b49de7771c  16 hours ago  storage costs calculation
```

Figure 3: Example of a source tree, taken from the NOMAD repository. Courtesy of Shyam Reyal.

3.1.3 Storage based services

Storage of research data is just as important as versioning. Working in a collaborative and transparent environment is becoming crucial in the modern research community. Possible solutions to this are simple file versioning or data attached storage. A more advanced version is a cloud based storage system such as Dropbox [16], Google Drive [17], or OneDrive [18]. These services all have limited version control capabilities, but are more for the use of data recovery. There are many online data platforms that provide versioning capabilities such as GitHub [19] and Bitbucket [20].

4 System design

There is currently no solution on the market that will solve the problem that NOMAD is facing, nor is there any solution that abstracts the versioning functionality of current version control system technology. Therefore, it would be useful to create a generic versioning solution for research data that can be applied universally.

The system will be designed in a modular manner to provide flexibility. Designing the system in this manner makes it easier to integrate the versioning functionality into NOMAD. The project can be tested independently and is not dependant on any given version of NOMAD, allowing it to be developed in its own development cycle. This also improves code maintenance along the line, as problems with versioning can be found as they are isolated to the versioning submodule. Any changes within the submodule or the functionality of the submodule can be tested too, so that changes can be confidently made knowing that there will not be any adverse effects or bugs. This is important as NOMAD is currently deployed and constantly developing with functionality being added regularly.

Finally, developing in this manner will keep the options open for the project to change if needed. For example, it will allow the tertiary objective S1 to be achieved more easily, as the application will not need a restructure nor extensive rewriting to fulfil the objective.

NOMAD is used for interfacing primarily with NRM machines. The user can automatically schedule their data collection. The raw data from the machines is then uploaded to a centralised data storage. Users are notified when their data has successfully been uploaded, they may then access their data through the NOMAD web portal.

The compounds, once stored, have their meta data stored in a MySQL database. This provides the search functionality for the NOMAD portal. The optional interaction with the meta data database allows for the changes to be searchable; for the data base to allow versioning it will need to be managed in a manner that will allow that. Lambda architecture is a possible solution. [21] This is where all entries to the database are maintained, and deletions or undo actions are stored in the database as a new entry. This helps to protect against loss of data and removes the need to replace or modify database entries. This way the database can store information about the versioning without having to query the versioning information storage.

To keep the project generic and allow it to be integrated easily with other programs, the Strategy design pattern will be used. [22] This way the user can choose how they wish to manage their meta data, by simply implementing their own class. This way they will not need to interact with the other functionality of the project, or change how they currently manage their meta data. This also allows the client to choose their database solution allowing for both SQL or NoSQL databases.

For the versioning functionality of this project to be added to the current workflow of NOMAD users, it will have to be integrated to allow users to upload their processed data. One of the main requirements is to maintain the integrity of the raw data and ensure that there is no risk of data loss of the original data. This proved to be a key design challenge, keeping the raw data separate to the versioning information. This increases the amount of data stored as the raw data will be duplicated in the versioning information. But this is viewed as beneficial, as it is an added redundancy against data loss, especially if the versioning information is stored in a different network location.

There are three main components to the project: versioning server, desktop application, and web interface.

The versioning server is the main component for the project and manages the versioning of the compounds and experiments. This section will utilise a version control system to help manage the versioning without having to implement a new versioning solution.

The desktop application will be used by the user to upload their modified data. The application will allow the user to add the changed data files to the staging area of the application. The application will find the changed files within the data, and let the user determine which changes they wish to save and upload to the server. Once the user is happy, the application will then upload the changes to the server to be versioned. The desktop application will use Model-view-controller (MVC) pattern [23] with JavaFX [24], to allow for the GUI to be changed if the client desires.

The desktop application has been deemed necessary, rather than having a completely web based service. Uploading entire datasets to the server over the internet to find the changed files can be an expensive operation. Comparatively, simply finding the changed files by hashing them is not resource intensive. This method will need to be implemented regardless of where the changes are calculated. By doing this client-side it will reduce the time needed to upload the changes. This improves the user's overall experience.

The final component to the project is the web interface and consists of two parts. The first part, a RESTful API [25] that will expose the versioning server to the web. The second part of the web interface is the hosting of a web page that will allow the user to manage the data.

The full architecture of the system can be seen in Figure 4. The blue components are the components that are within the scope of the project. The components in green are component external to the project and in this case, show how the project will be integrated with NOMAD.

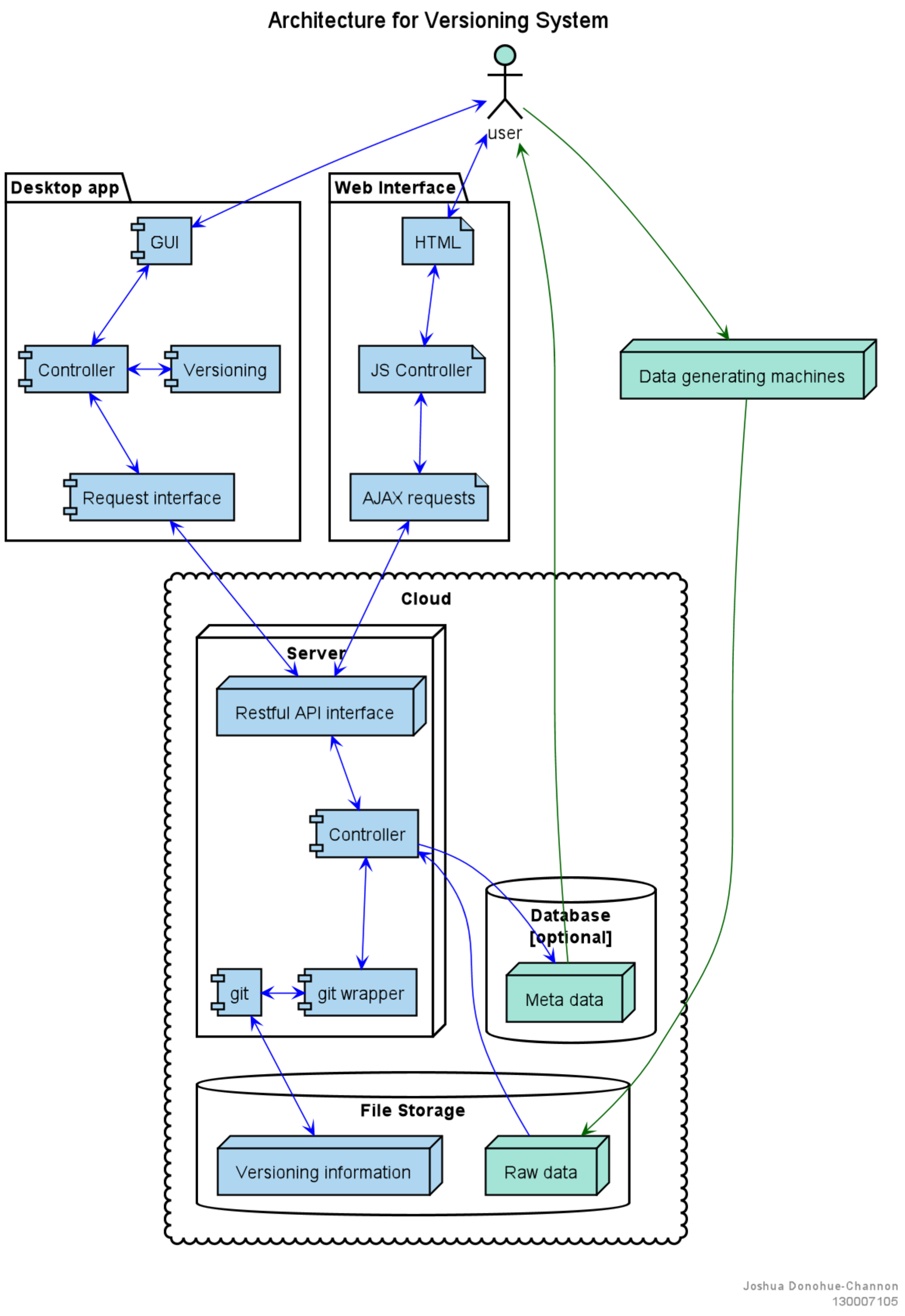


Figure 4: Diagram of the complete system architecture. The system components are in blue and the external components in green.

5 Implementation

The objective of the project, as laid out in section 2, above. But to successfully to achieve them there is a set of fundamental functionality that needs to be fulfilled. These functionality were determined by working with the expert users of NOMAD. For the versioning to be useful for NOMAD it needs to satisfy the following core functionality:

1. Add an experiment to a compound
2. Add a second experiment to a compound
3. Add an experiment to a second compound
4. Modify an experiment
5. Modify an experiment that belongs to two compounds
6. Remove an experiment from compounds

This core functionality is primarily based on the structure of data used by NOMAD. NOMAD is used to help classify compounds for chemistry research. To do so conclusively it is often necessary to conduct multiple experiments, each producing a new data set. The data sets generated by the experiments are called experiments. When these experiments are grouped they are described as a compound, which can also include associated meta.

The data this is project can manage is not simply limited to experiments and compounds. It can manage any form of data that can be described as a project with sub-projects. For consistency we will use *compound* to describe a collection of data sets, with *experiment* to describe the data sets. Keeping the type of data that this project can handle generic, allows the project to be versatile and increases the number of possible uses.

The way the data is structured and handled is the most important aspect of implementing the project. The time spent upfront determining how to manage the data model has improved the development process. It allowed for solutions to the more complex actions that might be completed by the user to be found early on. This has helped to avoid reworking large sections of the programs as there was no change in the data structure. Diagrams that show the interactions between experiments and compounds throughout a series of user actions have been a very effective tool for visualising the abstract data structure. These are included as Figure 5 through Figure 10, below.

The diagrams show the compounds in blue, denoted as C_nV_m, where n is the compound index and m is the compound version number. The experiments are shown in green, denoted as $E_kV_{Cn,l}$, where k is the experiment index, n is the index of the compound that the experiment is associated with, and l is the version of the experiment belonging to the compound. The legend shows the sequence actions that occur when the user completes a given action. The solid arrows show a change in version, whereas the dashed arrows show a reference.

The compounds and the experiments will be versioned separately, and a reference to the experiment is stored with the compound. This reduces the duplication of data as a copy of the raw data only needs to be copied once to the versioning server, then the experiment is branched and the branch refence is stored with the compound.

This method means that the client's data untouched, this is this beneficial as it will make clients more comfortable adopting the system if they know that their data will be safe. Additionally, it removes liability if the client loses their data as the versioning system does not have access to the data and thus, making it impossible for it to be the cause of data loss.

5.1 Add an experiment to a compound

The most basic action that the user will need to complete, is to add an experiment to a compound. The user will start by initialising a new compound. Then the user will then add an experiment to the compound. This will create a copy of the experiment from the read-only raw data on the versioning server. Then, will create a branch of the experiment associated with the compound. Then a new version of the compound will be created as the reference to the experiment is added. This can be seen in Figure 5.

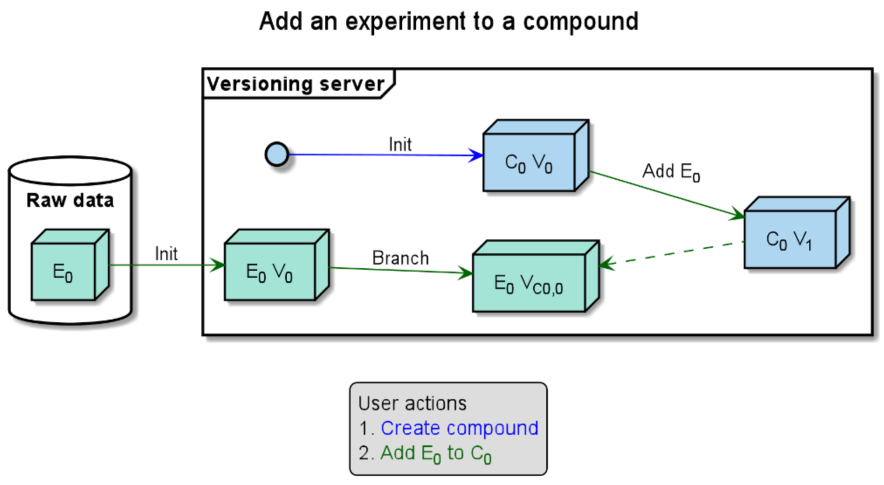


Figure 5: Sequence diagram for adding an experiment to a compound. The solid arrows show the actions that take place, colour coded to the user action. The dashed arrows illustrate a reference to an experiment being stored by a compound.

5.2 Add a second experiment to a compound

The next functionality is the adding of a second experiment to compound. The first two stages are the same as only adding a single experiment. The next stage is a repeat of the second stage. Note that now the compound holds two references one to each of the experiments. This can be seen in Figure 6.

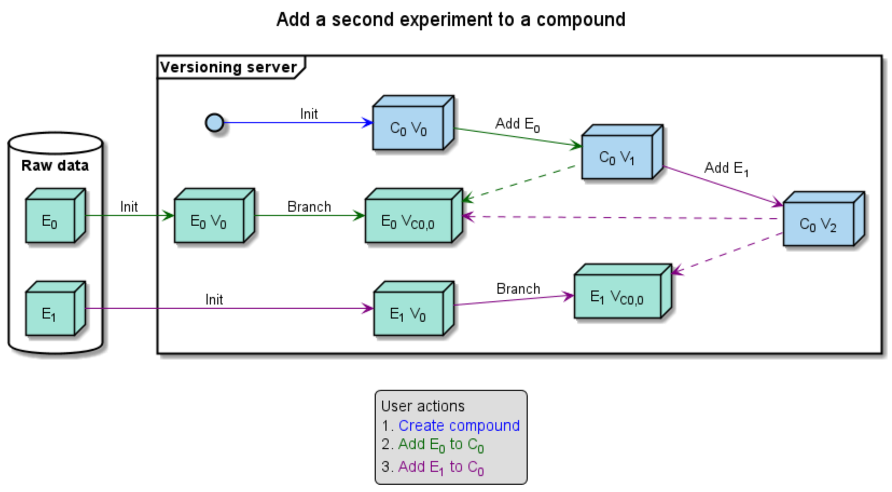


Figure 6: Sequence diagram for adding a second experiment to a compound.

5.3 Add an experiment to two compounds

Demonstrates how the data structure handles an experiment being added to two compounds. The compounds are created as usual and the experiment is added to the first compound. Now, adding the experiment to the second compound will simply branch the experiment again from E_0V_0. This saves duplication of data. This can be seen in Figure 7.

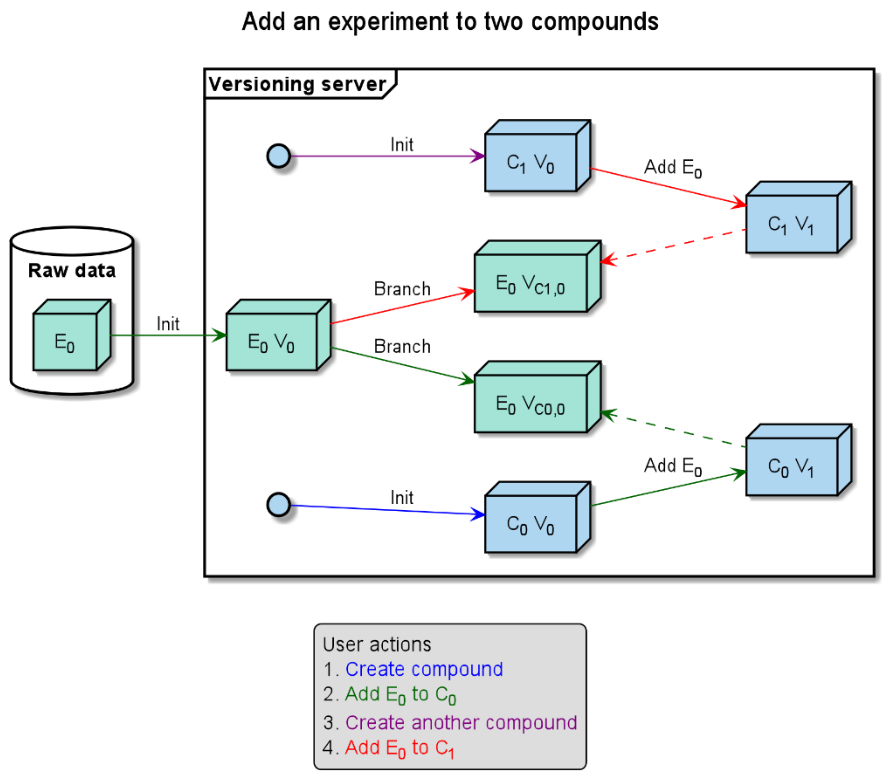


Figure 7: Sequence diagram for adding an experiment to two separate compounds.

5.4 Modify an experiment

This functionality, is the basis of the project allowing changes to the data to be tracked. Once the experiment is modified this then creates a new version of the experiment and updates the reference in the compound also creating a new version of the compound. This can be seen in Figure 8.

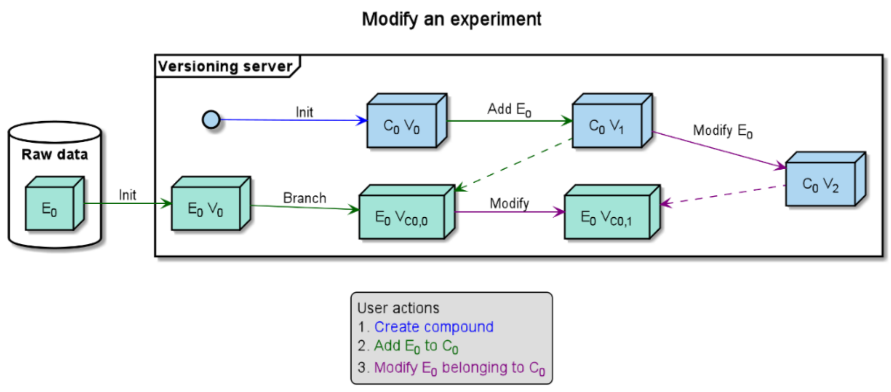


Figure 8: Sequence diagram that demonstrates the changes that occur when an experiment is modified.

5.5 Modifying an experiment that belongs to two compounds

Modifying an experiment that belongs to two compounds. Figure 9, shows how this works. The modified experiment belonging to a given compound only affects the compound to which it is associated.

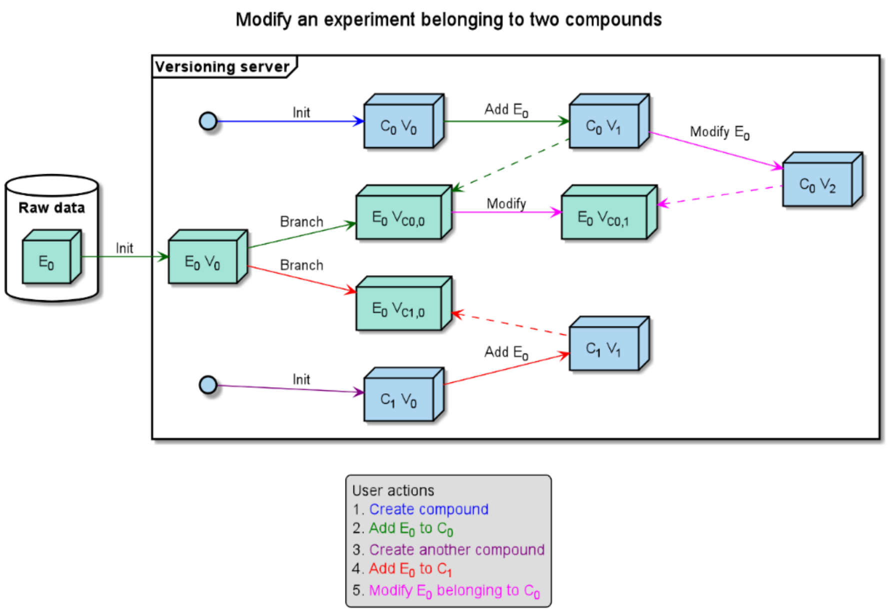


Figure 9: Sequence diagram that shows the effect of modifying an experiment that belongs to two compounds.

5.6 Remove experiment from compound

The final functionality is to be able to remove an experiment from the compound. If the user wishes to remove the compound it simply removes the reference to the experiment from the compound, this also creates a new version of the compound. This can be seen in Figure 10.

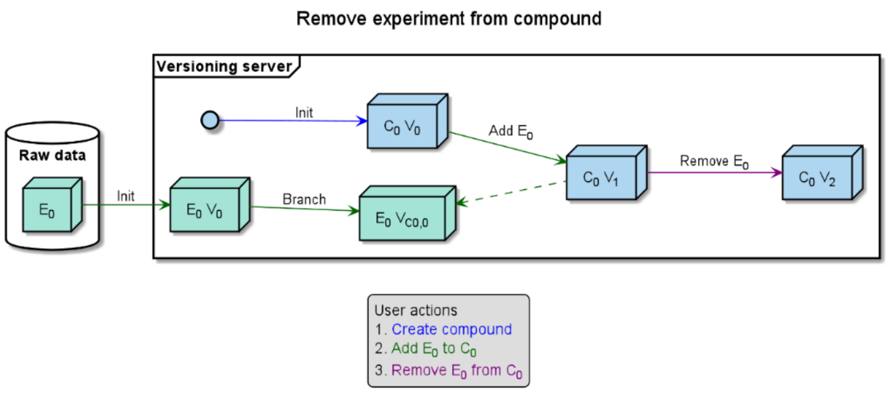


Figure 10: Sequence diagram for removing an experiment from a compound.

Combinations of the above actions will work together and form the basis for the use of the program. The implementation of this uses *git* to manage the versioning of the experiments and the compounds. To utilise the benefits of git, the JGit [26] library which is a pure java implementation of git will be used. Git uses hashes to create a unique id for each commit. The compound will store its experiments and the relevant version hash in a custom file. Git submodules were explored but were determined not to be an ideal solution as it would copy the whole of the experiment into the compound. Using the method of storing the reference only requires a list of experiments and version hashes to be stored.

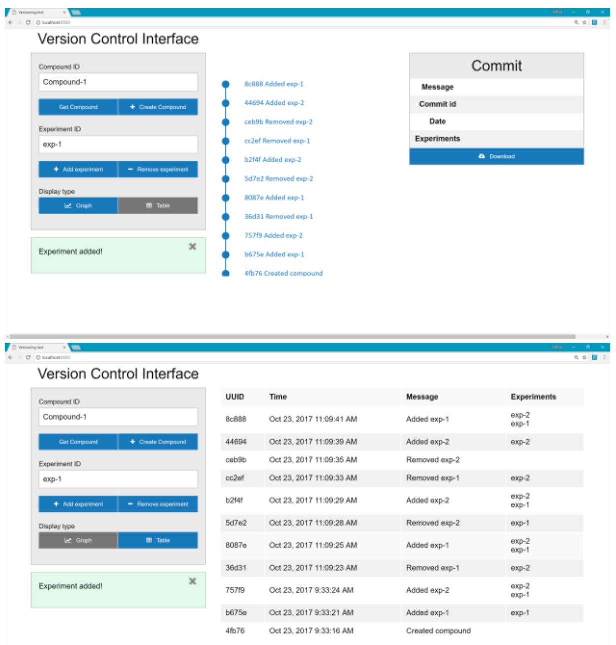
Git allows the git directory and the working directory to be different, this is a very useful feature and is the primary reason for using git. The git directory contains the versioning information and the working directory is where the files are managed for versioning, for example, files changed here can be saved as a new version. By being able to specify the git directory separately to the working directory, there are two main benefits. Firstly, it allows read-only data files to be added. Secondly, the working directory can be placed in a temporary location, reducing file duplication.

5.7 Versioning server

The versioning server handles access to the versioning capability. The versioning server provides access to a web portal to manage the compounds using an Apache Tomcat server. [27] A RESTful API is also provided using Jersey. [28] This is to respond to the web portal queries and to also handle uploads from the desktop application. Tomcat and Jersey were chosen because NOMAD already uses them. Once versioning is integrated with NOMAD, there will be less of a learning curve if the development team wishes to make changes.

5.8 Web interface

The web interface is used to manage the compounds. The interface will allow users to add and remove experiments from compounds. It uses the web framework, Foundation, for rapid prototyping. The web interface allows users to view the version information of the compound via a source tree. A JavaScript library called *gitgraph* [29] is used to display the source trees. The web interface can be seen in Figure 11.



Compound ID: Compound-1
Experiment ID: exp-1
Experiment added!

8c888 Added exp-1
44694 Added exp-2
ceb9b Removed exp-2
cc2ef Removed exp-1
b2f4f Added exp-2
5d7e2 Removed exp-2
8087e Added exp-1
36d31 Removed exp-1
757f9 Added exp-2
b675e Added exp-1
4fb76 Created compound

Message
Commit id
Date
Experiments

Compound ID: Compound-1
Experiment ID: exp-1
Experiment added!

UUID	Time	Message	Experiments
8c888	Oct 23, 2017 11:09:41 AM	Added exp-1	exp-2 exp-1
44694	Oct 23, 2017 11:09:39 AM	Added exp-2	exp-2
ceb9b	Oct 23, 2017 11:09:35 AM	Removed exp-2	
cc2ef	Oct 23, 2017 11:09:33 AM	Removed exp-1	exp-2
b2f4f	Oct 23, 2017 11:09:29 AM	Added exp-2	exp-2 exp-1
5d7e2	Oct 23, 2017 11:09:28 AM	Removed exp-2	exp-1
8087e	Oct 23, 2017 11:09:25 AM	Added exp-1	exp-2 exp-1
36d31	Oct 23, 2017 11:09:23 AM	Removed exp-1	exp-2
757f9	Oct 23, 2017 9:33:24 AM	Added exp-2	exp-2 exp-1
b675e	Oct 23, 2017 9:33:21 AM	Added exp-1	exp-1
4fb76	Oct 23, 2017 9:33:16 AM	Created compound	

Figure 11: Screenshot of web interface. Shows two ways of displaying the compound versions: source tree (top) and table (bottom).

5.9 Desktop application

The desktop application is where the user is able to upload their changed data files to the server. It contains a staging area and shows which files are been modified, created, or deleted. JavaFX is used for the GUI, and can be seen in Figure 12.

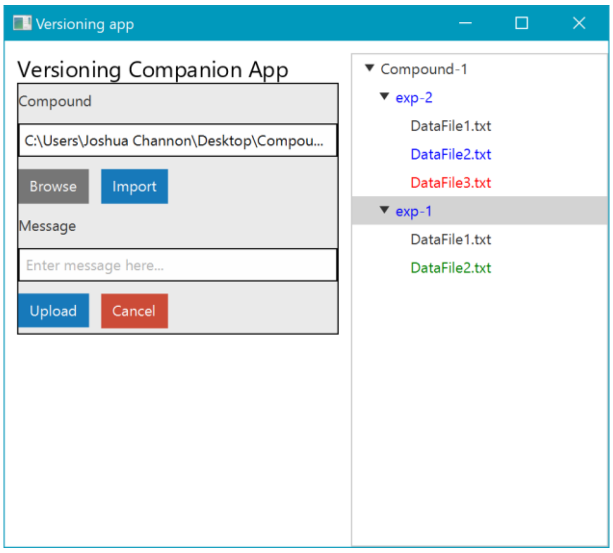


Figure 12: Screen shot of desktop application. The staging area shows modified files in blue, new files in green and deleted files in red.

6 Evaluation

6.1 Expert Users

A demo of the project was conducted for Simone Conte an expert NOMAD user and CTO of NOMAD. As Simone is a mentor for the dissertation the ethics artefact covers the interview, see appendix.

Qualitative feedback was collected as Simone interacted with the demo. Simone's feedback on the web interface:

"The is no drop down for the compounds".

This is a valid point, there is no way to search the disk location for compounds and experiments. Implementing a sample metadata strategy with a database would help to solve this problem and help to make the project closer to a marketable solution.

"It is not clear which one is the most recent [version], you could mark one of the dots a with different colour or make arrows".

This statement was about the source tree. The source tree created with gitgraph is symmetric when displayed vertically. The idea of arrow would be useful and help the user visualise the changes. Gitgraph is on open-source project on github so it would be interesting to investigate this further to see what the options are to show the order of the heads, and if there is not a suitable solution is it worth the time to contribute to the project.

"Missing the ability to set the current head, at the moment the head is always the most recent one, but you don't have the ability so say that I want my head to be this one"

Final thing of note from the interview, is a comment about setting the head of a compound. Currently, compounds are not given the option to be branched. This was due to a concern in the initial stages of the project that having multiple copies of the same compound might be too complicated for users who have had no experience with VCS. Now that the interface is at a function demo stage, it might be worth revisiting this.

A common thread throughout the interview was there was the issues with the user interface. The interfaces were not always intuitive, and there was no supporting documentation or help to guide the user. This needs to be address before providing the project to the client. Further information needs to be collection about how users interact with the versioning program.

6.2 Objectives

The project success can objectively be determined by comparing it with the objectives set out for the project at its outset. The primary objectives outline the fundamentals of the project and completion of the primary objectives does not signify a successful project when considered alone.

6.2.1 Primary

ID	Objective	Status
Versioning		
V1	Design and implement an interface for a version control system (VCS) that can be integrated with NOMAD, and which will allow for versioning at the compound-level tracking changes in experimental data.	Completed
Client-side application		

A1	Design and implement a desktop application that will provide a mechanism for the user to stage the annotated data for uploading and versioning. The application will run on the client's computer and will be a stand-alone application that can be installed on the client's computer without the need for any additional third-party software.	Completed
A2	Extend the application's capabilities to manage the uploads in a manner that avoids race conditions between multiple users uploading data simultaneously.	Ongoing
A3	Collect qualitative feedback from selected expert users of NOMAD.	Completed
Web interface		
W1	Design and implement an interface to visualise and manage the versions of the compounds and projects from within the NOMAD web interface.	Completed

Table 1: Status of primary objectives.

All the primary objectives are completed. The three categories of primary objectives: Versioning, Client-side application and Web interface reflect the three parts of the system as seen above in the system design. These objectives encompass the minimal functionality required of the system.

Versioning has been successfully implemented using JGit developed by Eclipse. This allows it to be used to track changes to compounds. The versioning occurs at compound level and tracks the changes in the raw data associated with any given compound.

This sub-section of the project has been written in a modular fashion, allowing it to be run as a stand-alone application but also allows it to be easily added to any project. Research data is written directly to NOMAD storage by machines, as such the data is read-only by the users. For this project to integrate with NOMAD it needs to work with read-only raw data. This requirement has been met as it stores the versioning information in a different location to the raw data. This way, the original raw data can be versioned while keeping it read-only. This fully satisfies the requirement V1.

The next sub-section of the project, the desktop application, handles the uploading of the user's modified files to the server. The primary objectives for the client-side application are aimed at producing a working desktop application. This is achieved when considering each of the three objectives: A1, A2, and A3.

Starting with A1, this objective is completely fulfilled as the desk-top application portion is coded in Java. This allows the application to run on a Java Virtual Machine (JVM) on the client's computer. The benefit of running the application in this manner allows the application to be run on a computer regardless of the operating system. The JGit library is also used in this portion of the project, the benefit of this is that the changes to the compound can be determined without having to install additional version control software like Git. Though the client still needs Java this is deemed a reasonable request due to its popularity and it is likely that prospective users already have it installed on their machines.

Additionally, the application provides a staging area allowing the user to view the local changes and to select which changes they wish to upload to the server.

Basic qualitative data was collected from a NOMAD user, this can be seen in the section 6.1, above. Though only one user was asked for feedback. The interview was a source of rich qualitative data. Asking more expert users could be useful in flagging up more problems early on.

The web-interface is the final part of the project and ties the other two parts together. Objective W1 dictates the main functionality of the web interface. The web interface uses *gitgraph,* a JavaScript library that allows source trees to be displayed on html pages. The web interface displays the changes to the compounds and allows management of the experiments within the compounds. The interface allows various versions of the compounds to be downloaded.

The project has successfully satisfied all the primary requirements. The competition of these objectives marks the completing of a basic working prototype.

6.2.2 Secondary

ID	Objective	Status
Versioning		
V2	Extend the versioning capabilities of the VCS to the project-level tracking changes in Compounds.	Completed
V3	Extend the versioning to include tracking changes in meta-data.	Ongoing
Client-side application		
A4	Conduct usability testing about how users interact with the system, collecting quantitative data and more qualitative data.	Pending
Web interface		
W2	Implement the desktop application's functionality in the NOMAD web interface, to remove the need for the desktop application.	Pending

Table 2: Status of secondary objectives.

The secondary objectives are an extension of the primary objectives, primarily an extension of the versioning portion of the program. The way that the project developed meant that the versioning at the compound level was naturally the first step. The capability to version the meta data is in place, but is not currently displayed to the user.

6.2.3 Tertiary

ID	Objective	Status
Versioning software		
S1	Design and implement a system that will allow for the VCS to be applied generically to any form of scientific data or project. The system will also contain a RESTful API that will be used to fetch changes from the user, either through a web interface or a desktop application. The system will be self-contained and can be deployed with minimal configuration.	Ongoing

Table 3: Status of tertiary objectives.

There is only one tertiary objective and it is to keep the project generic, so it can be easily deployed to an array of systems. A focus of development was to keep the project modular and this has helped to make progress with this objective. The project needs to be more self-reliant, for example, the option to have its own user and permission system would be a bit step towards achieving this objective. Additionally, making the web interface more modular, for example, making widgets that can be easily added to web pages would also be helpful.

6.3 Core versioning functionality

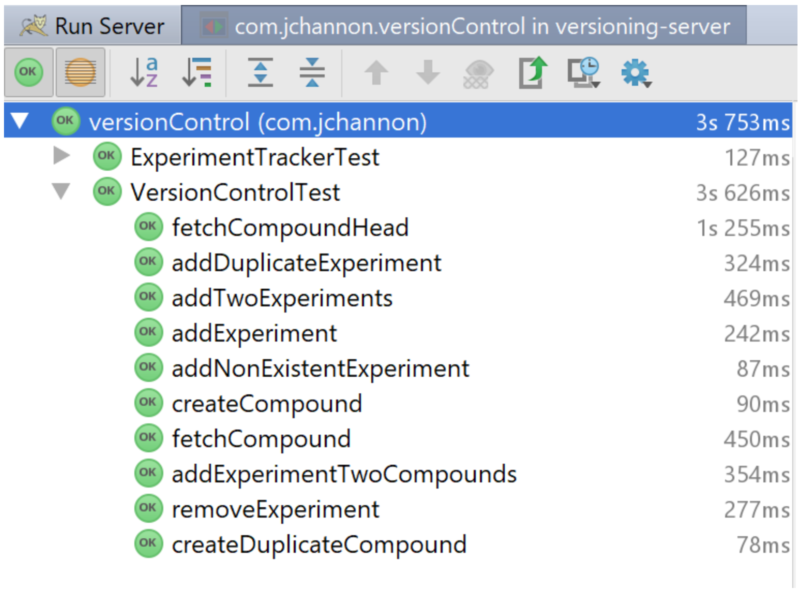
Functionality	Status
1. Add an experiment to a compound	Completed
2. Add a second experiment to a compound	Completed
3. Add an experiment to a second compound	Completed
4. Modify an experiment	Completed
5. Modify an experiment that belongs to two compounds	Completed
6. Remove an experiment from compound	Completed

Table 4: Status of functionality requirements.

All the core versioning functionality has successfully been implemented. This proves that a comprehensive versioning platform has been developed. This provides the backbone for the project and ensures that versioning of compounds and experiments are conducted properly.

6.4 Code Quality

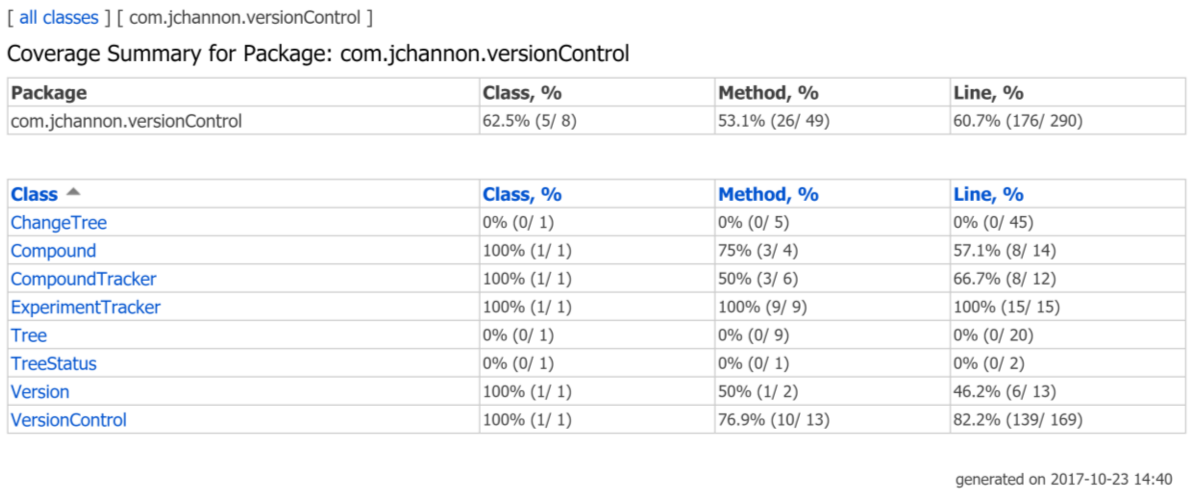
Code quality is an important facet of this project because the motivation of the project is to eventually integrate it into NOMAD. Since NOMAD is an active system, that is constantly in development, it is important that project is thoroughly tested. Figure 13 shows Junit tests used in the project.



```
versionControl (com.jchannon)         3s 753ms
  ExperimentTrackerTest                 127ms
  VersionControlTest                  3s 626ms
    fetchCompoundHead                 1s 255ms
    addDuplicateExperiment              324ms
    addTwoExperiments                   469ms
    addExperiment                       242ms
    addNonExistentExperiment             87ms
    createCompound                       90ms
    fetchCompound                       450ms
    addExperimentTwoCompounds           354ms
    removeExperiment                    277ms
    createDuplicateCompound              78ms
```

Figure 13: Passing JUnit tests for versionControl Class.

Having tests is not the only way to ensure code quality. Code coverage helps to ensure that all lines of code and paths though a program are tested, this helps to avoid writing bad tests.



[all classes] [com.jchannon.versionControl]

Coverage Summary for Package: com.jchannon.versionControl

Package	Class, %	Method, %	Line, %
com.jchannon.versionControl	62.5% (5/ 8)	53.1% (26/ 49)	60.7% (176/ 290)

Class ▲	Class, %	Method, %	Line, %
ChangeTree	0% (0/ 1)	0% (0/ 5)	0% (0/ 45)
Compound	100% (1/ 1)	75% (3/ 4)	57.1% (8/ 14)
CompoundTracker	100% (1/ 1)	50% (3/ 6)	66.7% (8/ 12)
ExperimentTracker	100% (1/ 1)	100% (9/ 9)	100% (15/ 15)
Tree	0% (0/ 1)	0% (0/ 9)	0% (0/ 20)
TreeStatus	0% (0/ 1)	0% (0/ 1)	0% (0/ 2)
Version	100% (1/ 1)	50% (1/ 2)	46.2% (6/ 13)
VersionControl	100% (1/ 1)	76.9% (10/ 13)	82.2% (139/ 169)

generated on 2017-10-23 14:40

Figure 14: IntelliJ generated code coverage report.

Just over half of the classes, method, and lines are tested. The project mostly acts without bugs, but before the project is integrated into other systems more tests should be written.

6.5 Summary

The underlying versioning has been satisfactorily completed, as the primary objective V1 and the secondary objective V2 have been completed. The completion of the core versioning functionality shows that the project has a strong underlying versioning capability, even though the project does not have the features to provide a comprehensive versioning service.

Though the project is light on functionality, the combination of a strong data model and a good underlying versioning capability enables the scalability of the project. Additional functionality will provide significant improvement.

7 Future work

7.1 Advanced functionality

To improve this project as a solution to versioning research data, more advanced functionality needs to be added. The main weakness in the project is the lack of user functionality. Currently the system will only work on a secure network and allows all users to access all data. This is not ideal, as research data is extremely valuable and needs to be protected. A system of users with a permissions system allowing users to access only data they are authorised to view would be a very useful feature. This functionality would need to be implemented in such a manner to encourage collaboration while still protecting the integrity of the data.

Maintaining more complex metadata would also be useful. Allowing users to annotate files, and comment on their contents and reason of a given version would be very useful for users who return to data after a while the initial data collection process. Useful for allowing users to get more value out of currently held data, especially when producing new data is an expensive enterprise. Tagging versions would also be useful, for instance, labelling a version of a compound that has led to an academic publication.

The web interface is useful but is not easy to integrate with other systems. If the web interface was modular and a set of widgets was created, clients could simply plug in readymade components into their current system. This allows them to integrate versioning into their current data managements systems in a user-friendly manner, with minimal investment in upfront development.

The objective W2 would also be a big improvement to the user experience. Creating a web application that could perform the actions of the desktop application in the browser, would make it easier for users to use. This will also make the application more versatile and allow it to be used easily over multiple computers and even provide the option for use on mobile. This will increase the types of data that the project can be used with.

7.2 Integration into NOMAD

The initial motivation for the project was the need for a way to upload modified data to NOMAD while maintaining the original data, thus there is already a demand for this project. Integrating this project into NOMAD would have the most immediate impact and improve the service that NOMAD provides to its over 900 active users. If integrated into NOMAD, it will continue to have impact as NOMAD is actively maintained and is looking to be deployed to other institutions.

7.3 Software as a Service

Implementation of an online platform that would provide online storage for data that also had a comprehensive versioning system, would also be a possibility for future work. The market for online storage already has many established companies. The additional versioning capability would make it stand out in the already congested market as it provides a unique service for research data management. This has the most potential to affect a large client base if executed and marketed correctly.

8 Conclusion

A demand for versioning of scientific research data was identified by users of NOMAD, users wanted to upload processed data to NOMAD without loosing the original raw data. There is no solution on the market that will fit the need presented by NOMAD, nor is there solution that will allow for scientific research data to be versioned. This project has tackled the challenge by producing a solution for generic data.

The project focused on creating a robust data model that can handle all forms of data in a manageable way. Core functionality for versioning compounds and experiments. Once the core functionality was established a basic working solution was created. The project proves the point that versioning of research data can be useful, and the strong data model created has really helped.

The project needs more work before it is a viable large-scale solution. But adding functionality such as user support and permissions, as well as creating widgets for the web interface, will go a long way towards this goal. This project is very versatile and has many exciting application, most of all, an integration with NOMAD.

9 References

[1] T. Wissik and M. Durco, “Research Data Workflows: From Research Data Lifecycle Models to Institutional Solutions,” *Linköping Electronic Conference Proceedings,* no. 123, 2015.

[2] S. I. Conte, F. Fina, M. Psalios, S. Reyal, L. Tomas and A. Clements, “Integration of an active research data system with a data repository to streamline the research data lifecycle: PURE-NOMAD case study,” *IDCC17,* no. Practice Paper.

[3] V. Van den Eynden, C. Louise, M. Woollard, L. Bishop and L. Horton, “Managing and Sharing Data: Best Practice for Researchers,” UK Data Archive, Essex, 2011.

[4] “Managing research data in your institution,” Jisc, 2 October 2017. [Online]. Available: https://www.jisc.ac.uk/guides/research-data-management.

[5] “Research Data Management,” Sudamih Project, Oxford University Computing Services, [Online]. Available: http://sudamih.oucs.ox.ac.uk/docs/Research%20Data%20Management%20Factsheet.pdf.

[6] “Data versioning | Stanford University Libary,” [Online]. Available: https://library.stanford.edu/research/data-management-services/data-best-practices/data-versioning.

[7] A. Seering, P. Cudre-Mauroux, S. Madden and M. Stonebraker, “Databases, Efficient Versioning for Scientific Array,” MIT, 2011.

[8] “Datahub,” [Online]. Available: https://datahub.csail.mit.edu/www/.

[9] A. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden and A. Parameswaran, “DataHub: Collaborative Data Science & Management at Scale,” Cornel University Libary, 2014.

[10] “Git,” [Online]. Available: https://git-scm.com/.

[11] “Mercurial SCM,” [Online]. Available: https://www.mercurial-scm.org/.

[12] “Apache Subversion,” [Online]. Available: https://subversion.apache.org/.

[13] “Cryptographic hash function - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Cryptographic_hash_function.

[14] “Universally unique identifier - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Universally_unique_identifier.

[15] “diff utility - Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Diff_utility.

[16] “Dropbox,” [Online]. Available: https://www.dropbox.com/en_GB/.

[17] “Google Drive,” [Online]. Available: https://www.google.com/drive/.

[18] “Microsoft OneDrive,” [Online]. Available: https://onedrive.live.com/about/en-gb/.

[19] “The world's leading software development platform - GitHub,” [Online]. Available: https://github.com/.

[20] “Bitbucket | The Git solution for professional teams,” [Online]. Available: https://github.com/.

[21] N. Marz and J. Warren, Big Data: Principles and best practices of scalable realtime data systems, New York: Manning Publications Co., 2015.

[22] “Strategy Design Pattern,” Source Making, [Online]. Available: https://sourcemaking.com/design_patterns/strategy.

[23] “Model–view–controller,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[24] “JavaFX: Getting Started with JavaFX,” Oracle, [Online]. Available: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784.

[25] “Representational state transfer,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer.

[26] “JGit,” eclipse, [Online]. Available: https://www.eclipse.org/jgit/.

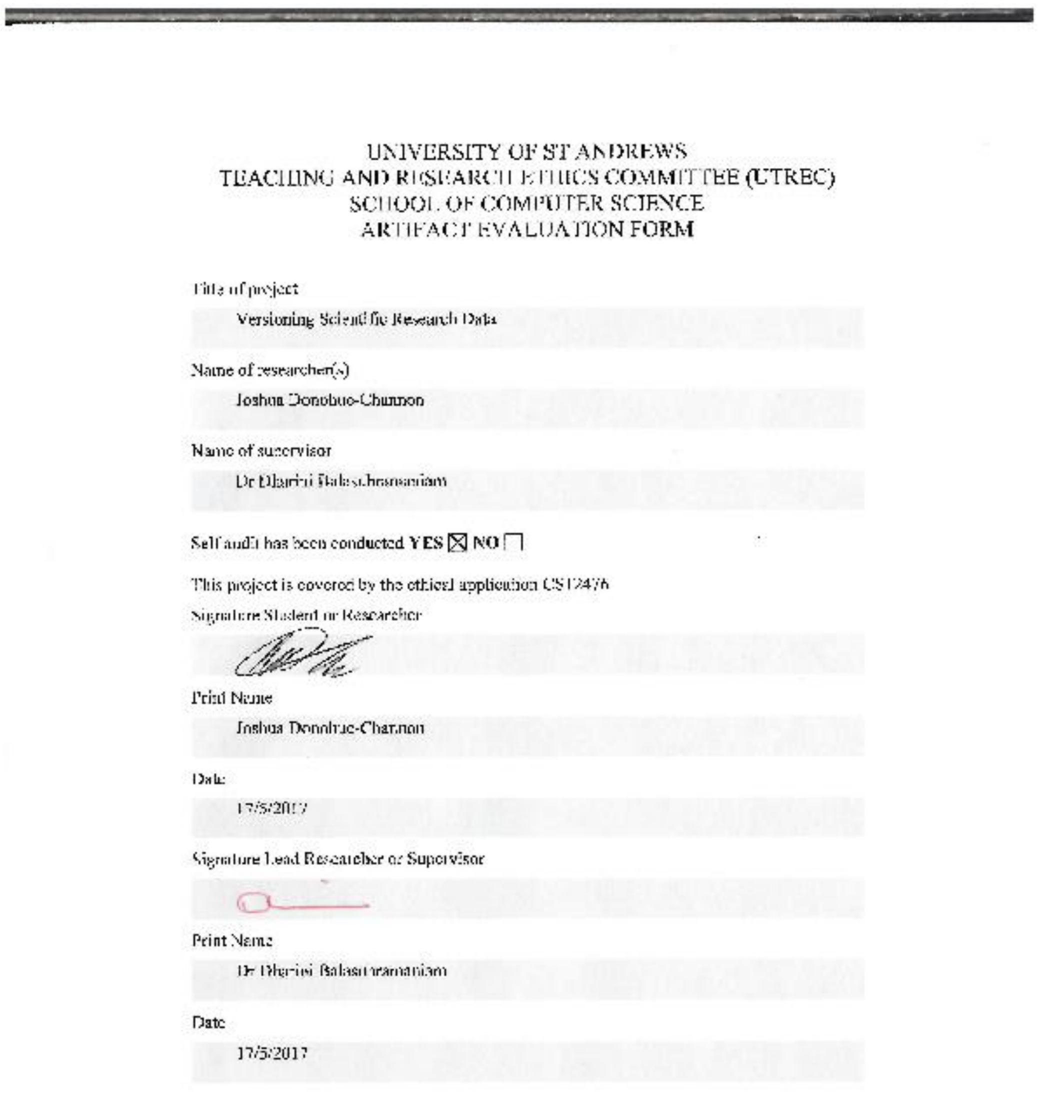
[27] “Apache Tomcat,” Apache, [Online]. Available: http://tomcat.apache.org/.

[28] “Jersey - RESTful Web Services in Java.,” Oracle Corporation, [Online]. Available: https://jersey.github.io/.

[29] “GitGraph.js,” [Online]. Available: http://gitgraphjs.com/.

10 Appendix

10.1 Ethics artefact



UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
ARTIFACT EVALUATION FORM

Title of project

Versioning Scientific Research Data

Name of researcher(s)

Joshua Donohue-Channon

Name of supervisor

Dr Dharini Balasubramaniam

Self audit has been conducted **[✓ YES] NO**

This project is covered by the ethical application CS12476

Signature Student or Researcher

[illegible]

Print Name

Joshua Donohue-Channon

Date

17/5/2017

Signature Lead Researcher or Supervisor

[illegible]

Print Name

Dr Dharini Balasubramaniam

Date

17/5/2017