**Visualization and Interactive Data Exploration for a Spectral Information System**

Final Report for CS39440 Major Project

*Author*: Andrew Hickman ([anh23@aber.ac.uk](mailto:anh23@aber.ac.uk))

*Supervisor*: Dr. Mark Neil(mjn@aber.ac.uk)

14th April 2014

Version 4.2 (Final)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Science (G400)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

**Declaration of originality**

In signing below, I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the sections on unfair practice in the Students’ Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
* I understand and agree to abide by the University’s regulations governing these issues.

Signature ……………………………………………. (Andrew Hickman)

Date …………………………………………………

**Consent to share this work**

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ……………………………………………. (Andrew Hickman)

Date …………………………………………………

**C:\Users\Andrew\Downloads\anh23-barcode.pngEthics form application number**

Application reference number: 834

**Acknowledgements**

I am grateful to…

Mark Neil for supervising the running of the project throughout and helping with setting up initial contact with the customer

Andres Hueni, the original creator of the SPECCHIO system who’s constant communications have helped to shape the project.

I’d like to thank…

Ben Brooks, Christopher Hardwick, Helen Harman, Sam Nicholls, Victoria Harkness, Alex Stuart, Chris Savill and Rhian Watkins for helping in testing the functionality to help find bugs within the graphical user interface.

I would also like to thank Ian Griffiths for finding a release package of IzPack rc-3 as the rc-4 was a broken installer.

**Abstract**

The updating and upgrading of a spectral information system known as SPECCHIO that was created to manage and update spectral information for use within its relevant fields. The project had a list of functional requirements which were laid out to be worked upon throughout the course of the project with additional functional requirements added throughout. These requirements were as follows:

* Update the graphical user interface
* Add location data exportation of spectra to external mapping software.
* Addition of spectral thumbnails in the form of spectral graphs
* Increased user feedback when creating campaign’s
* Small script to automatically upload spectra to the database
* Upgrade the current systems software deployment software.

The main aim of the project was to increase usability for the end user by giving them additional information, through which adoption of the SPECCHIO system would increase.

This upgrade of functionality would be carried out in two main programming languages Java and Python, the choice of the first was decided upon by the original creator as the application was already developed in java. The second language python was chosen as it is a simple scripting language that was a perfect fit for the functionality that was required when dealing with a MySQL database.

The results of the project fit the functional requirements listed within the project specification and have vastly improved the end user interaction within the graphical user interface, additionally the python script can be adopted by scientists and engineers when collecting spectral information in the future.

In conclusion the application now contains more ways for the end user to approach a spectral information system and has been shaped with guidance from the customer to ensure end user satisfaction.

**Contents**

1. Background, Analysis & Process 8

1.1. Background 8

1.1.1. Background languages 8

1.1.2. Background Systems 8

1.2. Other Software Systems Considered 9

1.2.1. Software Deployment systems 9

1.2.2. Scripting systems 9

1.2.3. Database systems 9

1.3. Background Interaction with Customer 9

1.4. Analysis 10

1.4.1. Spectral Thumbnail Update Overview 10

1.4.2. Exporting Location Data to External Mapping Software Overview 10

1.4.3. Glassfish Analysis and Upgrade Path 11

1.4.4. Additional user feedback on database imports 11

1.4.5. Python Script and why it was required 11

1.4.6. Dealing with an enterprise software system 11

1.4.7. Time management and split of requirements 11

1.4.8. Gantt chart 13

1.5. Process 14

1.5.1. Plan driven approach 14

1.5.2. Diary 14

2. Design 15

2.1. Overall Design Decisions 15

2.2. Design Decisions of Used Programming Languages 16

2.2.1. Java 16

2.2.2. Python 16

2.2.3. XML and MySQL 16

2.3. Updating Existing Documentation for the End User 16

2.4. Spectral Thumbnails Design Decisions 16

2.4.1. Spectral Thumbnail Class Diagram 18

2.5. Location Data Exporting to External Mapping Software Design Decisions 18

2.5.1. Swing Design Decisions 19

2.5.2. Maps Processing Class Diagram 20

2.6. User Feedback When Inserting Spectra Design Decisions 20

2.6.1. Swing decisions 20

2.6.2. Campaign Creator Class Diagram 21

2.7. Upgrade to Glassfish 4 Design Decisions 21

2.8. Python Script Design Decisions 21

2.8.1. Additions to the script 22

2.10. Overall Architecture 23

3. Implementation 24

3.1. Spectral Thumbnail Implementation 24

3.1.1. Implementation Issues 24

3.2. Location Spectral Plotting Implementation 26

3.2.1. Attaining the location 26

3.2.2. Implementing with the SPECCHIO application 26

3.2.3. Multiple Spectra Exporting 26

3.2.4. Thread usage 27

3.2.5. Refactoring code for good design 27

3.2.6. Implementation Issues 27

3.3. Addition of User Feedback into Campaign Creation 29

3.3.1. Implementing Swing 29

3.3.2. Deciding which data to return to the user 29

3.3.3. Implementation issues 29

3.5. Implementing GlassFish 4 into the Current System 30

3.5.1. Implementation issues 30

3.5.2. Upgrading the system 31

3.5.3. Jax RS-2.0 Upgrade 31

3.7. Python Script for Auto Uploading to the Database 32

3.7.1. Code layout 32

3.7.2. Finding correct MySQL plugin 33

3.7.3. Implementation Issues 33

3.9. Review of Implementation 34

4. Testing 35

4.1. Overall Approach to Testing 35

4.2. Test Tables 36

5. Critical Evaluation 41

5.1. Were requirements correctly identified 41

5.2. Were design decisions correct 41

5.4. Were project aims met 42

5.5. Time management 42

5.6. What would I have done differently 42

5.7. Managing a large enterprise project 43

5.8. Testing 43

5.9. Conclusion 43

6. Appendices 44

A. Third-Party Code and Libraries 44

7. Annotated Bibliography 45

# 

The SPECCHIO spectral information system is a java application in which spectral files are read and manipulated through the use of metadata being passed from a MySQL database. These spectral files are then passed to a client application where the data can be further manipulated and spectral plots can be built through MATLAB for detailed analysis of spectral files.

# Background, Analysis & Process

In preparation for the project a range of background research, analysis and detailed design had to be completed in order to set realistic goals and functionality requirements throughout the course of the project.

## Background

### Background languages

Java was a requirement of the system as the currently deployment is almost entirely programmed in java. This required some background reading into swing which is a graphical user interface toolset within the java language and is deployable across multiple operating systems with slightly different look and feels throughout.

For this a range of chapters from Java Swing, Second Edition, O’Reilly Media [1] were used and referenced throughout the code.

Python was chosen for the scripting language used when auto deploying to the database and so many tutorials and webpages were used in order to get an overview of the languages which later was chosen as it could be run individually from the main SPECCHIO application. [2][3][4] Were all used in creating the python script.

### Background Systems

Before the project was set to begin the background requirements that had to be completed were as follows:

* Download, install and familiarise myself with the current implementation of the SPECCHIO application in its current form (V3.1.3)
* Install Python and Pydev into the chosen IDE (eclipse)
* Install Glassfish 3 for use with the SPECCHIO application
* Install MySQL workbench for use with the SPECCHIO application
* Install MySQL plugins for both Python and Java

Initially the current SPECCHIO application was the top priority to get to grips with as this would be the foundation that would be built upon, this required an account to be created with both the existing database and live database.

As python was chosen to be used for the script that would automatically upload spectra data to the database it had to be installed along with any plugins that would be required in order to access the MySQL database.  
  
Glassfish 3 was installed for the initial build of the SPECCHIO system as this is what the original author Andreas Hueni currently have the web application deployed on. This deployment application served as a basis for the update to the most recent version of Glassfish and allowed for configuration data to be compiled that would be passed to the newer version.

MySQL workbench was a necessity for the SPECCHIO database as it allowed data to be viewed directly from the database for consistency checks. It also allowed users to be added to the database initially as there was no default root user to be used for the SPECCHIO application.

Both python and java required external plugins for MySQL which was required in order to deal with any SQL requests that had to be made to the database. These were handled with two different plugins

* The Java plugin used was mysql-connector-java-5.1.34-bin.
* The python plugin used MySQL-python-1.2.4b4.win32-py2.7.

## Other Software Systems Considered

### Software Deployment systems

Google app engine allows for the running of projects throughout the google infrastructure and have the added benefit of not requiring servers as they handle all of the application deployment issues for the user. This was a valid candidate to be used instead of Glassfish for deployment purposes but it was decided that the current implementation of Glassfish would be a safer alternative when it came to upgrading the application deployment as all configurations could be exported out of Glassfish 3 and into Glassfish 4 with little to no issues.

### Scripting systems

Bash could have been used when writing the database auto upload script but this came with its own set of limitations - the most notable was the inability to be used on a Windows machine without the use of Powershell or similar software.

### Database systems

PostgreSQL is another database management system that offers similar functionality to MySQL and would have made the transition from the current SQL database a trivial task. As the database was already created and required no additional functionality that wasn’t already being offered in MySQL it was decided that no changes had to be made with the database.

## Background Interaction with Customer

Before the project even began the interactions with the customer were frequent. Andreas Hueni had a major role to play in how the end product was shaped over the course of the three month period. His interactions were numerous and will be linked throughout the document as his insight was a driving factor behind the overall success of the project.

Andreas Hueni is a research assistant at the University of Zurich and the original author of the SPECCHIO spectral information system. His background is computer science and achieved a BSc in the subject at the University of Applied Science, HTL Brugg-Windisch, Switzerland. He also carries a PGDip in arts in GIS from Massey University, Palmerston North, New Zealand and MPhil (Sc) in Earth Science also from Massey University, Palmserston North, New Zealand. Other notable mentions are his multiple publications and co-publications on scientific papers. He also took part in APEX (Airborne Prism Experiment) PAF, which has been successfully used for operational APEX data processing from RAW to level1 (radiometric, spectral and geometric calibration) since the first test flights of APEX in 2008.

The initial set of functional requirements came from Andreas himself and at the beginning of the project it was pointed out that more functionality although not necessary could be looked at during the project. These additional functional requirements were upgrades to the system as certain pieces of deployment software were now out of date, these upgrades were deemed a top priority for the longevity of the application and were taken onto the project.

## Analysis

SPECCHIO is a spectral database designed to hold reference spectra and spectral campaign data obtained by spectroradiometers. Its front end graphical user interface allows the user to add and maintain spectral campaigns as well as view alter and export any campaign chosen by the user. The current implementation of the graphical user interface is lacking in end user functionality with many overly technical and underwhelming options for the novice user to view campaign data. This is why the project’s main focus is to improve the end user experience to increase the adoption rate of the system.

With the background research complete on the base SPECCHIO software it became obvious that the current implementation required some graphical user interfaces upgrades. Additionally, extensive communication to the end user when information is being inserted into the database and a way for external spectral collection tools to automatically upload data to the SPECCHIO database.

With the background research into the SPECCHIO application and with the functional requirements laid out in the project specification it quickly became apparent where the upgrades within the system needed to take place as the current project was laid out in a very straightforward manner. The query builder in particular required extensions being made as it gave little feedback to the user as to the data that was actually being stored in the database.

### Spectral Thumbnail Update Overview

One of the pieces of functionality that was required by the initial functional requirements was the ability to view thumbnails for any given spectra before the user commits to opening a report for the given spectra. Through the analysis of the system and given the functional requirements it would be sensible to give the user some form of feedback either visually or textually within a panel where spectral information is already being displayed.

### Exporting Location Data to External Mapping Software Overview

One of the main functional requirements of the project was the ability to grab the location data for a single spectra and export the information into external mapping software. Through the analysis of the system it was decided it would be a wise idea to add the add this functionality into the current implementation of the graphical user interface, most likely into the data browser within the program as this is where users can view the contents of the database and have a return of visual and text information. Then within either of these panels if would be useful to give the user the ability to return the location data for given spectra and export them into a web browser using an external mapping website <http://mapcustomizer.com>.

### 

### Glassfish Analysis and Upgrade Path

As an additional piece of functionality, it was posed as a task that the current application deployment server Glassfish 3.1.2.2 be upgraded to its most recent version. Through the analysis of the system it appears that this upgrade should be possible but would require multiple upgrades to plugins that are required by the new version of Glassfish and also upgrades to the servlets that set the deployment path for the SPECCHIO web application.

### Additional user feedback on database imports

As pointed out by the customer and supervisor there is little to no interaction with the user when importing a campaign which makes it very difficult for the user to understand what information exactly is being added to the database.

A simple Swing upgrade would allow for a text box that could give the end user all the information required when data is being read into both the program and database.

### Python Script and why it was required

During the analysis of the system a side project was put forward for the automated addition of spectral information into the SPECCHIO database. This script was required for external spectral collection devices, as it would allow them a way to upload their spectral data directly without having to use the external medium of the SPECCHIO application.

This would be extremely helpful and time saving for the scientists who collect spectral data for the SPECCHIO application and would be an added incentive to upload every piece of data collected instead of just a select few.

Python was chosen for the scripting language as it is ubiquitous enough to be run on any operating system but also lightweight enough in its design to solve complex problems with small scripts through its vast import library.

### Dealing with an enterprise software system

The size of the system being used is substantial with many different systems and libraries being inserted and used throughout. Because of this a section of time had to be dedicated to code analysis.

The current SPECCHIO system uses the following technologies that had to be analysed before coding could begin:

* MATLAB – Used for drawing and plotting of spectral graphs
* GlassFish – Used to deploy the web application
* Java – The basis of the code is written in java and had to be analysed
  + 14 Packages
    - Hundreds of Classes and interfaces
* MySQL database – certain subsections of the database would have to be accessed later in the project when the python script was being created

### Time management and split of requirements

During the analysis of the system, and given the plan driven model of the project a realistic split of functional requirements to time constraints had to be made early on in the project. The chosen method for keeping time management clear was a Gantt chart as it allowed for shifting time frames when budgeting time. This ability to move time constraints came in particularly handy when the additional upgrades the SPECCHIO system were added to the project specification.

The requirements themselves had to be split evenly and be given enough time as to be completed and bug tested so there would be no integration issues when the development code base gets pushed to the live application that is currently circulating. The split of functions were chosen by how useful they would be for the end project as such that if another piece of functionality would not be completed in time that it would give the most value to the end user.

Because of the above choice the list of requirements were split as such:

* **Glassfish 4 upgrade** – This was chosen to be top priority as during the course of development it would allow for any bugs that may arise from upgrading to the latest version to arise throughout normal use of the application.
* **Location data exporting / spectral thumbnails** – As both of these pieces of functionality were swing based they were chosen to be worked concurrently as they would use the same frame to pass information to the end user.
* **Additional feedback on database imports** – Another swing and java logic problem but using a different section of the SPECCHIO application as all information was chosen to be passed into a text area it would take a shorter time to complete than the above requirements.
* **Python script** – As this piece of functionality was in no way related to the SPECCHIO application and would not affect the running of the system this was deemed the last priority in terms of work order.

The beginning of the project was set to start on the 21st of January 2015 but did not commence until the 2nd of February due to no source code being sent [7]. This caused a week delay which pushed back the code analysis.

### Gantt chart

Below is the Gantt chart used throughout the project it was updated a few times during the project as to incorporate the Glassfish upgrade that was added a few weeks into development.

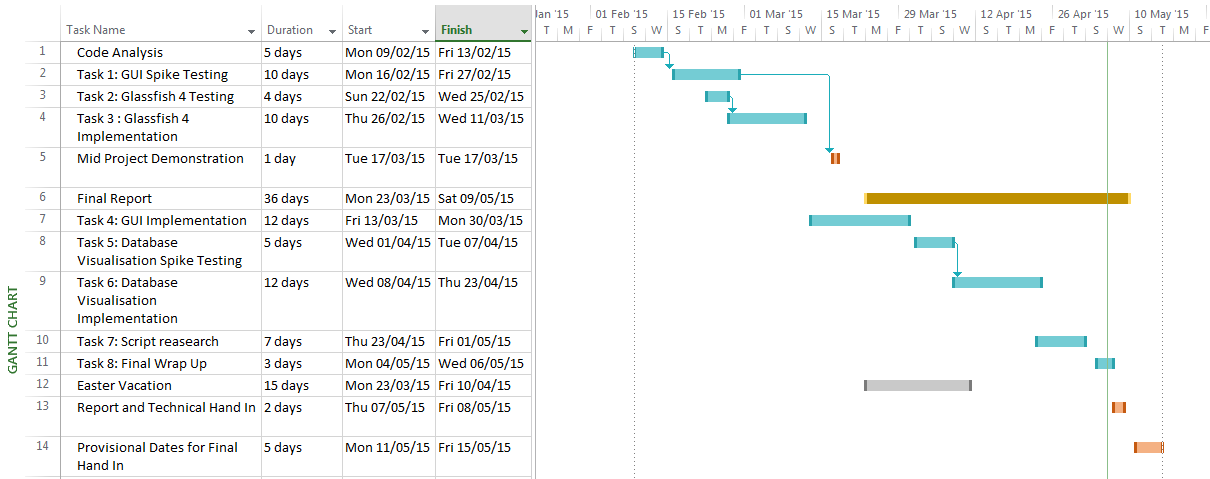


Figure - Final gantt chart for project

## Process

The processes carried out over the course of the project are what defined how the project was run and ultimately directed how the end product was produced. Described below is the software development methodology that was chosen and how using its development strategy shaped the nature of the project.

### Plan driven approach

For this project, a plan driven approach was chosen with new pieces of functionality being produced in an iterative manner to bug test the system with each new iteration.

The reason for this choice was the strict set of functional requirements laid out before the project even began. This approach allowed for concise time management and documentation paths to be laid out prior to any implementation occurring. With the incremental model being used, any initial planning required heavy analysis of the system to determine where new pieces of functionality would be inserted into the existing code base.

### Diary

Throughout the three month project a manual written diary was kept that includes daily updates as to the current state of a given increment or piece of functionality. The option to keep a written diary as opposed to an electronic blog was purely out of personal preference and required thoughts could be written at a moment’s notice without the need for technology.

Standard entries included the day’s work with minutes based on the current bug within the functionality of a system.

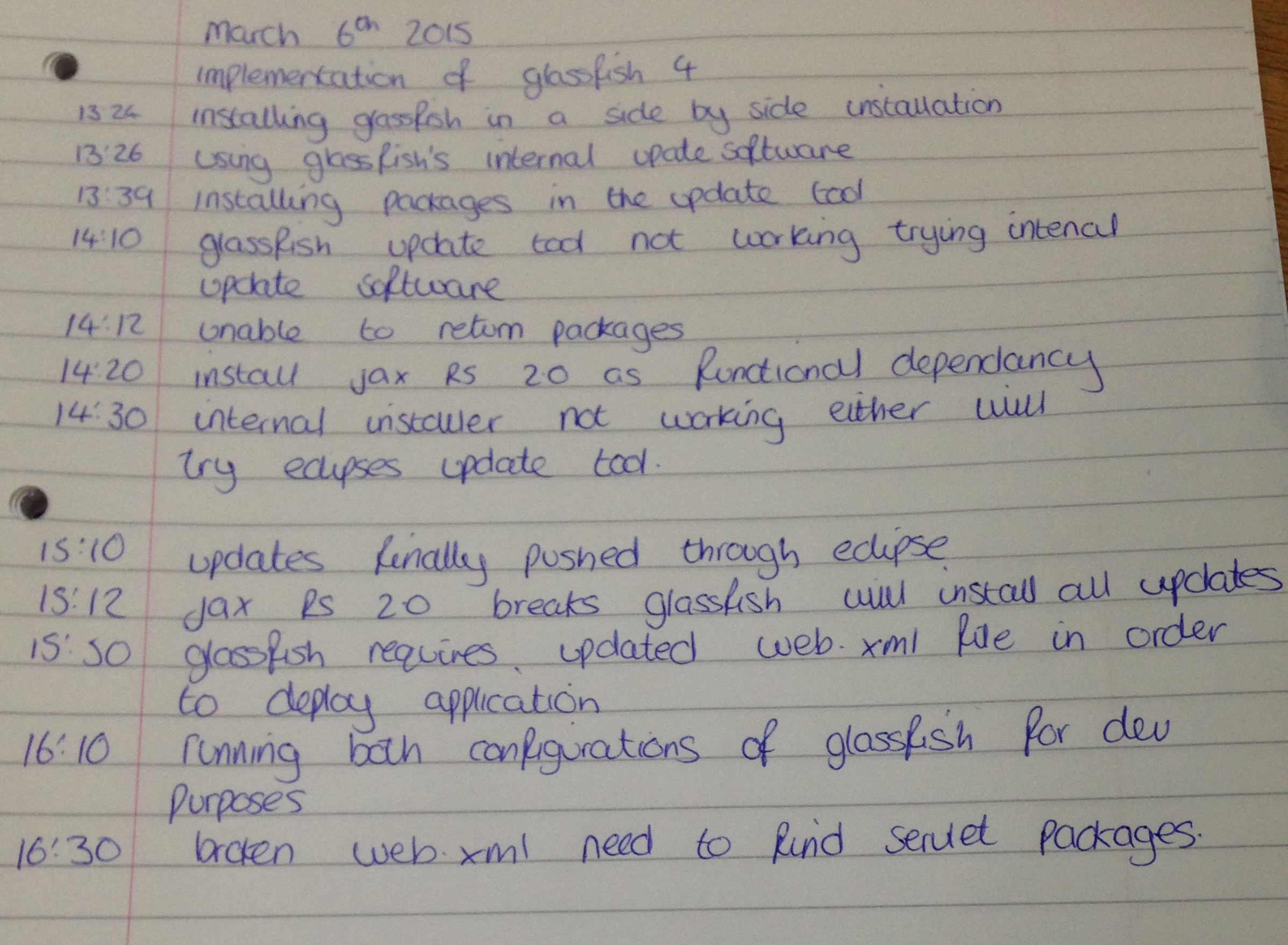


Figure 2- Example of Minutes

# Design

In this chapter a detailed analysis of software systems and design patterns along with individual requirements design choices will be explained and expanded upon for when the base SPECCHIO system incorporates the features listed in the requirements.

## Overall Design Decisions

The overall design of the SPECCHIO system and any additions made throughout the project were made to fit the software architectural pattern of the model view controller. This pattern was already implemented when the project began in the original implementation and was continued throughout development. The processes and interactions of the user are split into a variety of classes of methods as to keep the model, view and controller’s separate.

New features were implemented using an iterative design pattern. This pattern was used as the requirements were definitively laid out and could be well documented and planned out prior to any implementation taking place.

Shown in [Figure 3], the iterative design method began with initial planning of the requirement. An analysis of the code base and how to properly implement a code insertion was made. The implementation was then carried out from the analysis and implemented code would follow the model view controller. The process would then continue the flow and the requirement would be tested and evaluated. The requirement would then be analysed again to ensure all functionality was inserted if not the loop would continue until deployment was successful.

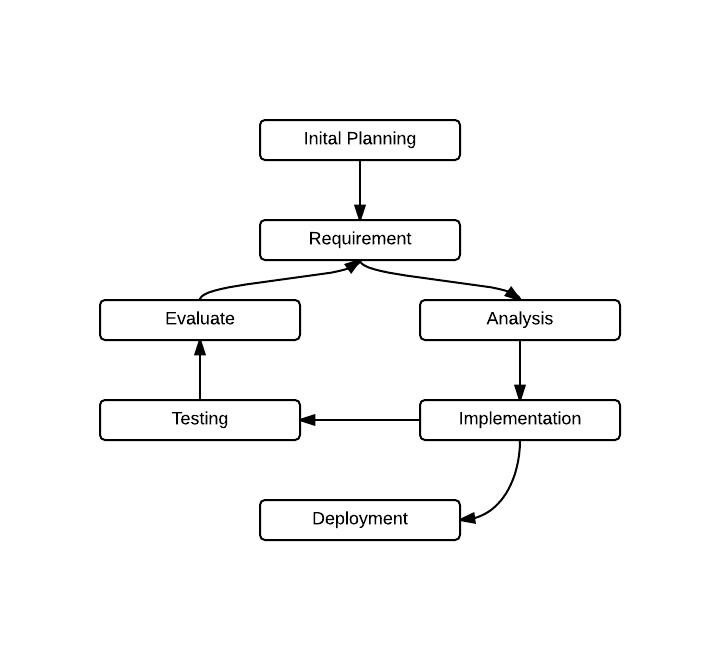


Figure - Iterative Design

Schneiderman’s eight golden rules influenced heavily on the overall design of any new interface properties that were introduced as a bi product of required functionality, three of the golden rules were used heavily as the interface design was only being enhanced and not newly created. [9]

* 1 – Strive for consistency
* 3 – Offer informative feedback
* 5 – Offer simple error handling

These three fit perfectly with major functionality, consistency was key as the goal was for end users to intuitively get to grips with the Swing additions. Informative feedback was important for the database dialog - a user should be well informed on background processes that directly affect the spectra that they are inserting. Simple error handling would be provided through the use of dialog boxes to condense error messages.

## Design Decisions of Used Programming Languages

### Java

Java 1.7 JDK was used for development of the main body of code, this programming language choice was to continue the object oriented design of the original product. The implemented code also allowed for the ‘compile once run anywhere’ philosophy of Java, which is one of the main attractions of the SPECCHIO system as both the developers and end users range across multiple operating systems.

### Python

Python 2.7 was chosen over the newer 3.4.3 edition as the MySQL plugin used within the Python development was incompatible with the latest edition. Python script is easily deployable on multiple operating systems provided python is installed. This scripting language is easily maintainable and as the basis of the code will be passed to the original developers after the completion of the project, it will serve as a good basis for future revisions of the auto uploader script.

### XML and MySQL

For the required functionality of importing data to the database, there was a dependency between XML and MySQL to implement parsing and passing to the database through insert statements.

The export of spectral data to XML allowed for simple search trees to be completed to parse any given file for specific fields and tables. Information could then be inserted into the database and committed through the python script.

## Updating Existing Documentation for the End User

With all the functionality added to the final system the documentation that is handed to developers was required to be updated. These documents list the upgrades made to the SPECCHIO system and how to implement any code that has been added into either existing or new classes.

The Glassfish 4 upgrade required a substantial amount of detailed steps in order for the live version of SPECCHIO to be upgraded. Documentation already existed on how to install a fresh install of Glassfish to the end user and so this piece of documentation was updated to add the upgrade paths and code required.

## Spectral Thumbnails Design Decisions

In the current implementation there was no way to look at a spectra in any form other than its identification number. A choice had to be made on which piece of information would be shown, there were a range of possibilities that were all text based that take from the SPECCHIO clients’ implementation of the getMetaParameters method. This method allows the code to retrieve any of the information about a given spectra. Ultimately this was deemed to be only useful if the correct piece of information was picked for the user. The chosen method of interpreting information was to return a spectral graph. This would give the user the ability to have a graphical representation of individual spectra that gives a perfect overview.

The design choice was to have a separate class to handle given identification numbers in order to plot the spectral graph and return a JPanel that would display the given plot. The information would be gathered from the file browser on the left hand side of the current implementations pane and would be implemented in a thread in order to allow long processes to be undertaken without causing any hangs in the application.

A major concern throughout the project was the ability to push these changes to a live implementation of the SPECCHIO application. This in turn had to shape the design of how the spectral thumbnails would be implemented. In order to reduce processing time when taking multiple reads from the database it was deemed necessary to only plot a single spectra at a time. On development machines where the database is stored locally for testing purposes reading multiple spectra was almost instant as no external connections had to be established, but once these changes hit live every spectra would have to be plotted from an individual read from the database and having multiple spectra being plotted at once theoretically would cause huge wait periods for the user.

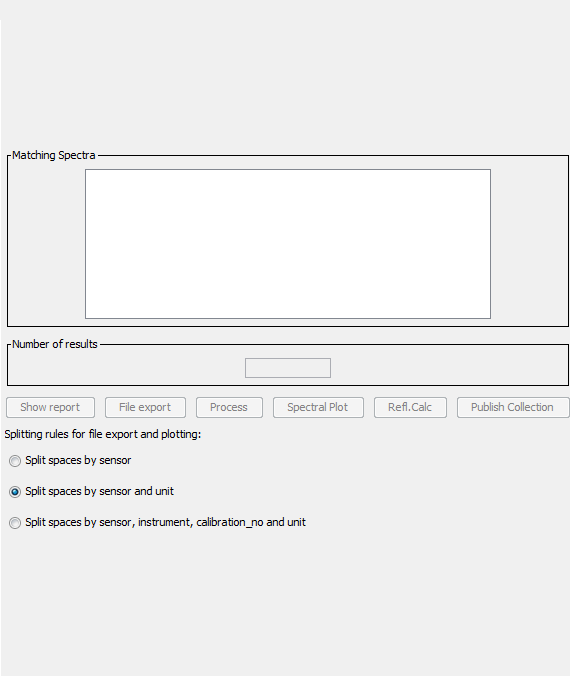


Figure 4 – old specchio query browser Implementation

The graph had to be plotted somewhere within the current GUI and within the initial implementation of the query browser there was a substantial amount of space that could be used if the elements were pushed down [Figure 4], this gave the ability to add a small spectral plot to the query browser.

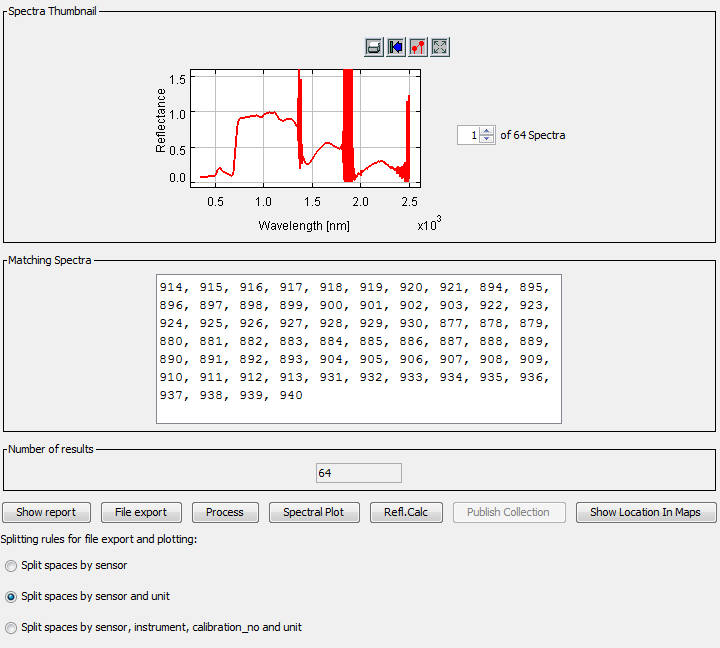


Figure 5 - new implementation of spectral thumbnail

The new implementation used the available free space in order to create a new spectral graph which would be updated as the user clicks on the spinner. This gives the user a quick overview of the spectra within a given set of ranges and allows them to make a more informed decision on what data they wish to take from the database. [Figure 5]  
  
The live application should be free of bugs and this influenced the design decision to have a separate class for handling the data input from the query builder. This way any bugs within the code were quickly fleshed out and error caught and handled.

### Spectral Thumbnail Class Diagram

Shown below is the class diagram for the query builder to add spectral plot interactions. Listed are all the variables and method calls required for each.

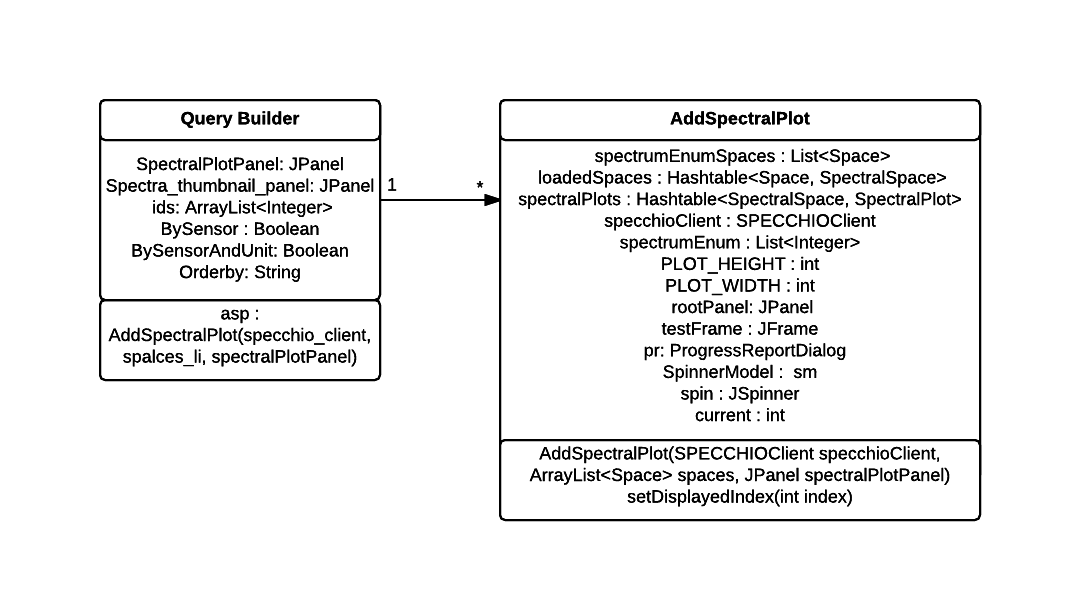


Figure 6 – Query Builder Class Diagram

## Location Data Exporting to External Mapping Software Design Decisions

The design choice for location exporting was to further the idea of giving informative reads of spectral data to the user, the main concern when dealing with the location of spectral data was the ability to have multi point processing as spectroradiometers can be mounted onto the back of a moving vehicle of any variety and plots can span a fair distance. This allows scientists to see small changes in data read outs as the location updated. The ability to read out single spectra to mapping software was just important as location data can give extensive insight into given graphs and Meta data.

Initially the plan was to export multispectral plotting into Google Maps but as the Google Maps API has a limit to the amount of geocoding that can be passed to their servers at one time.

* 2,500 per 24 hours [8]
* 5 Requests per second [8]

This limitation when being rolled into live deployment of the system could have cause significant issues for the end user as the application increases in popularity or if for example hundreds of spectra are passed to be processed at once. Due to this, an alternative method for exporting multiple spectra had to be constructed for use within the final live system. The barrier to adopting the Google Maps API was due to a paid license which could cost upwards of £10,000 for using such a small subsection of the API. In future if the use of multi spectral plotting was to grow, this paid option would be a viable alternative to the implemented system.

### Swing Design Decisions

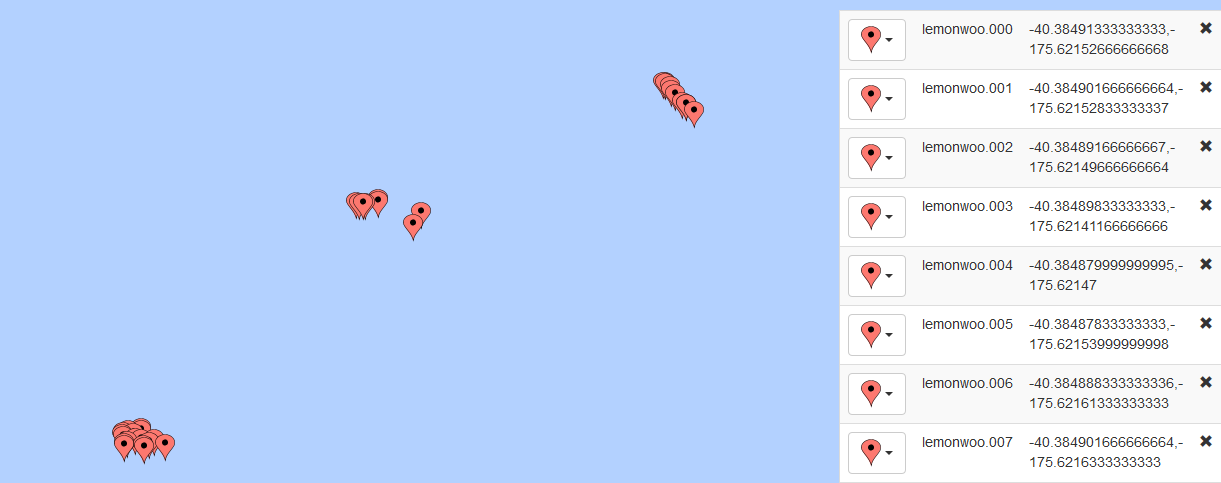
Within the SPECCHIO system, as with the spectral thumbnail panel the Query Builder was a clear choice for the addition of swing components to deal with the location data exporting.

A new button had to be added to deal with any of the selected spectra from the tree browser. This extended out the pane but with the new query browser frame’s full screen state this ended up not being an issue.

Within an early implementation of the multi spectral plotting function a new JFrame was opened which contained a lone JTextArea with all the location data selected passed into a string. The idea was for the user to copy and paste this text and then close the frame. This was quickly changed when automatic copying to clipboard was discovered as it removed the need for a new frame to be displayed to the user. The removal of this Frame allowed for consistency in the initial design and allowed for less user interaction to reach the same desired goal.

The text copied into the clipboard is pre-formatted to include the location data and file names for spectra, this as show in [Figure 7] allowed for each specific point to be named and the user can track the movement between each read easily.

**Figure 7 – Multi Plot Exporting**



### Maps Processing Class Diagram

Below the interactions, variables and method calls are shown between the query builder and maps processing class:

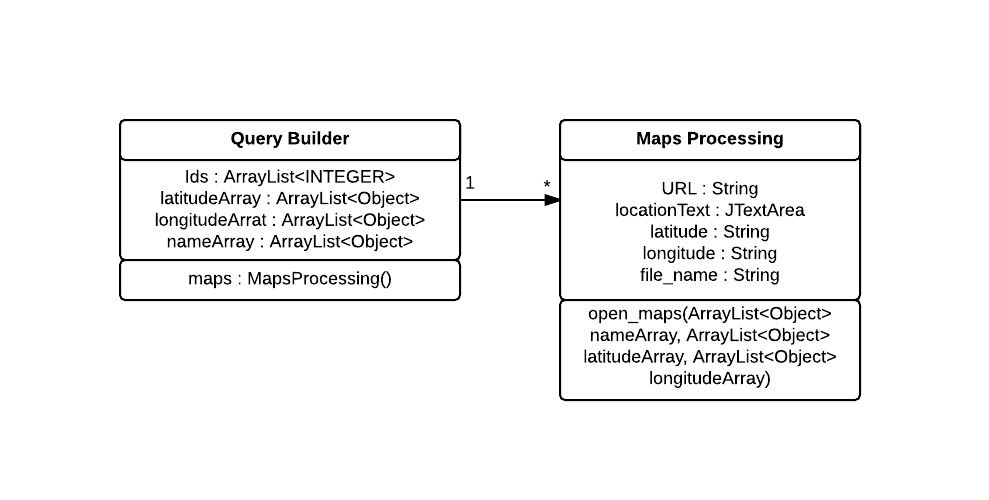


Figure 8 –Maps Processing Class Diagram

## User Feedback When Inserting Spectra Design Decisions

The design decision was clear with this particular topic. When using the system for the first time during background research it was immediately apparent that there was a lack of information being displayed to the user on background processes taking place. The main question then to be asked was where is the most convenient and consistent place to add this feedback?

The SPECCHIO system has five different menu items on the insertion and removal of spectral campaigns and two different methods for creating campaigns, they are also spread across two different menu bar tabs which confuses matters further for the user.

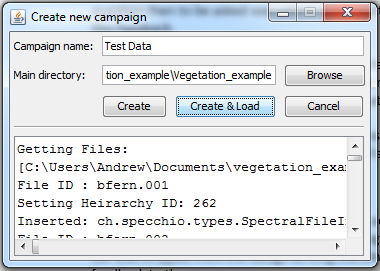


Figure 9 –Campaign Creator Diagram

Eventually the campaign creator frame was chosen to contain the new swing upgrades as it’s where the first interaction with the insertion of spectral data takes place and thus would yield the most feedback to the user. The end result is shown in [Figure 9].

### Swing decisions

The original design for the Swing layout was to add both a text box and a progress bar to the user interface to display progress as it was being made, quite quickly though the progress bar was stripped from the design as long standing processes would not yield any informative feedback to the user.

A JTextArea separated by a JSeperator were added and were set to shown when the user choses to load a campaign. The JTextArea would display information on the file names being imported, the file directory being imported, the hierarchy being set and finally the insertion into the database. This information was gathered from the SpecchioCampaignDataLoader as the text are was passed in and all appends done to the text were done within this class. This gives the user all the information required on the data flow. The JTextArea was inserted into a JScrollpane to give a consistent window size with the viewpoint being set to the bottom of the text area which allowed the user to scroll through and see any progress made.

### Campaign Creator Class Diagram

Shown below are the variables and method calls used in implemented the new functionality into the NewCampaignDialog and SpecchioCampaignDataLoader classes.

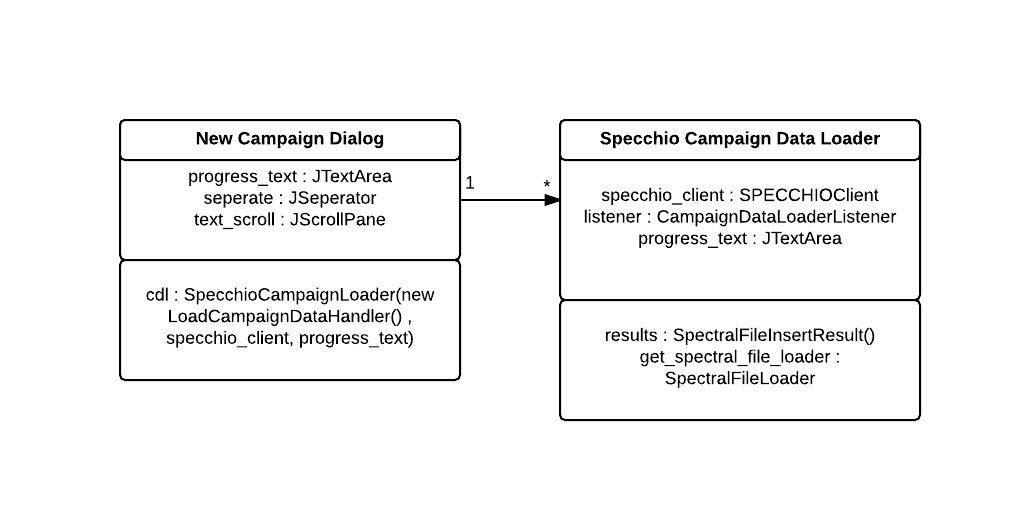


Figure 10 –Campaign Creator Class Diagram

## Upgrade to Glassfish 4 Design Decisions

Unlike other parts of functionality, the upgrade to Glassfish 4 was an addition to the original functional requirements, but was seen as a big upgrade for the longevity of the system for future iterations. Due to the nature of the upgrade this was actually completed prior to any graphical upgrades to the system as the software deployment system could be tested through regular use when developing the application.

This did come with its own set of challenges though as servlet declarations had to be changed and additional fields had to be added to the web pool within Glassfishes configuration.

## Python Script Design Decisions

The python script implementation was chosen to support the use of scientists in the field or with a large collection of spectral data. The main driving force behind the addition of this script was the ability to bypass the need for a third party (SPECCHIO system) and upload information directly to the spectral database. The only dependencies would be an account with the database which users uploading data would have already.

This lightweight script requires little input from the end user in the form of choosing the file location for the spectra. From there the program will either auto exit due to incorrect credentials or run through the given file searching for spectra of the XML file type.

The main aim was to reduce the amount of time it takes for a user to upload a substantial amount of spectra to the database. Due to the nature of the script the user interaction could be reduced to nothing if file locations are already known.

### Additions to the script

The script in its current form will only parse XML documents as this was the test data used to insert correctly. The foundations of the script have all of the required insert statements for the SPECCHIO database and thus extending of functionality for additional file types would be a trivial task. The other file types that could be read in are as follows (taken from the SPECCHIO application)

* HDR, SLB,SLI
* TRM
* XLS
* SED
* SPT
* SPU
* OUT
* H5
* DAT
* TXT
* CSV
* ASD

These file types can currently be parsed through the SPECCHIO application from spectroradiometers and would only require a transition from the Java language to Python.

## Overall Architecture

The overall architecture of the SPECCHIO system through my inputs had to remain consistent with the original authors design and follow a strict model view controller architectural pattern. Due to this many classes that required background logic to be completed were split off and called through the base graphical user interface class.

Shown in [Figure 11] are any additional classes and their interaction with the base SPECCHIO program along with how logic was split into additional classes that split from the user interface.

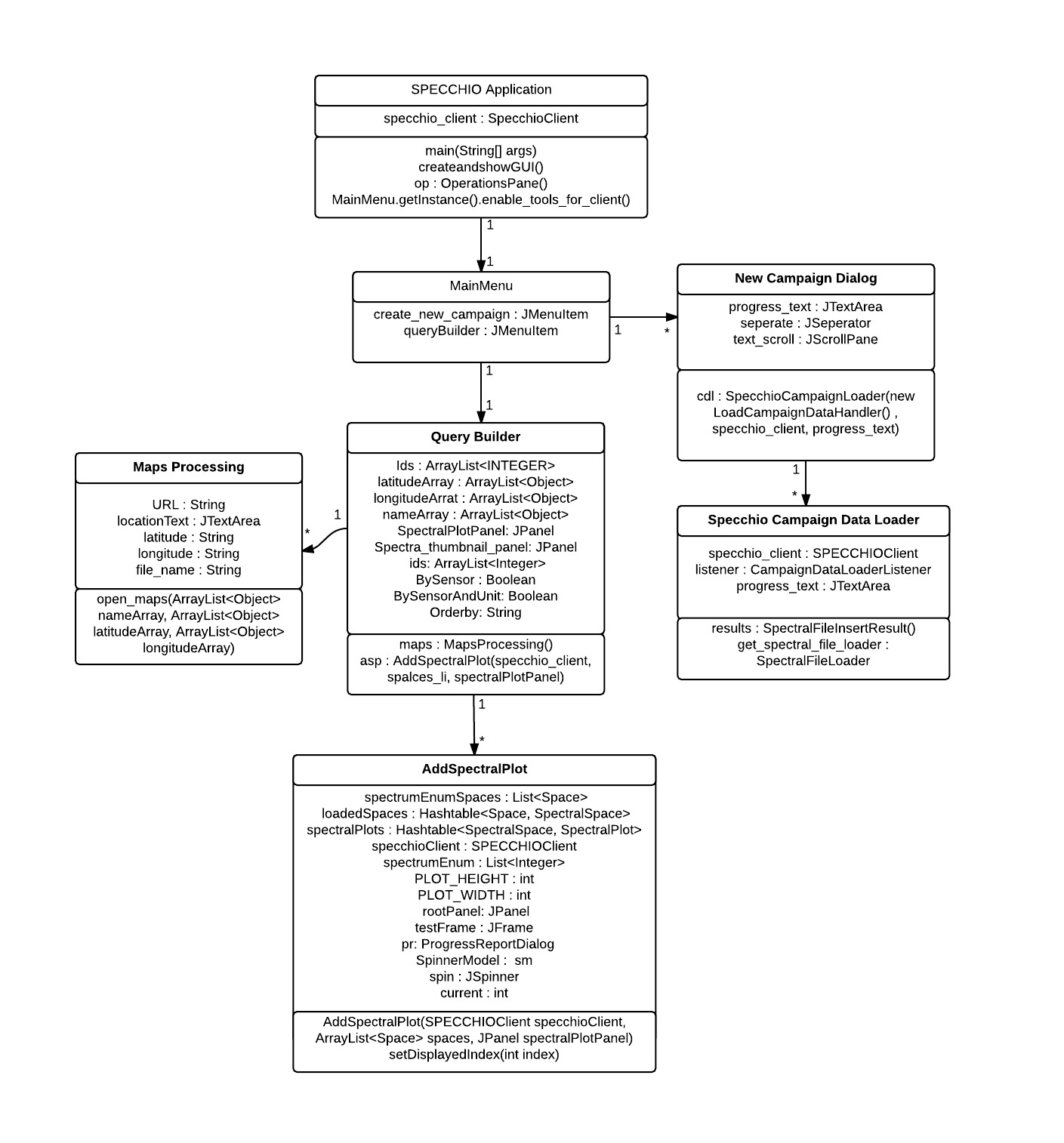


Figure 11 –Class Diagram for system interaction

# Implementation

Throughout the course of the project, a list of functional requirements were given. This section details the reasoning and methods used to produce the different types of functionality giving sufficient justification for the choices made in relation to the design decisions.

## Spectral Thumbnail Implementation

The implementation began by passing in the selected spectra’s from the query builder frame into a thread that is called by a tree listener within the pre-existing file browser. It was tasked that this tree listener be updated to pass the unsorted selected spectra ids into a thread, this thread was created to handle four pass variables.

1. ArrayList of spectra identification numbers
2. Boolean of ‘by sensor’
3. Boolean of ‘by sensor and unit’
4. String of current order

The variables are then passed into a class whose sole purpose is to create a spectral plot for given ids. Within this class a method was created that checks the id’s against the database’s listings for the given id. It would then pass the information to a series of ArrayLists and Hashtables that were created by the applications creator Andreas Hueni which sort the ids and handle the Meta data for each associated id.

A separate function set displayed index is called and checks against a JSpinner with a change listener for the currently selected id and creates a new spectral line plot from MATLAB’s library with a defined width and height set for correct viewing within the GUI.

The selected spectral plot is then drawn to a new JPanel and redrawn and revalidated to ensure correct display within the GUI. This JPanel is then passed back out to the Query Builder and updated and shown.

### Implementation Issues

When creating the thumbnails, a number of issues when dealing with the collection and representation of correct spectral id’s and plots had to be dealt with, along with this Java swing also had limitations with handling the panel redrawing and revalidation when adding the spectral plots as well as the spinner having a correct change listener implemented.

A list of known implementation issues are as follows.

1. Swing and layered panels
2. Swing override of panel sizes
3. Single drawing of plots as to reduce program load
4. Implementing a spinner number model to stop index out of bounds exception

#### Swing and layered panels

When creating new instances of spectral plots they had to be cast to a panel which was being passed from the query builder. These plots were then being drawn to a panel called spectra\_thumbnail\_panel but after the first had been drawn an issue was found that any subsequent panel would be drawn behind the original panel thus causing unusable plots when multiple spectra had been selected.

A new check had to be implemented when the tree’s action listener was called which would remove any current instance of the spectra\_thumbnail\_panel and redraw and revalidate a new panel when the spinner had been given a new value either typed or clicked on. This change also relied heavily on the design decision to only draw one panel and one spectra at a time in order to reduce load on the system if a huge array of spectra were selected.

#### Swing override of panel sizes

Within the swing framework any child panel will inherit the sizes of the parent even when specifically setting a size for the child panel. This meant that when the panel had no current listing for a spectra thumbnail the panel would be a single pixel wide on initial creation and would not update. It was later found that setting a preferred size for a child node would automatically update the parent node to allow for the child’s preferred size.

#### Single drawing of plots to reduce program load

The choice to only draw a single spectra at a time was decided upon after a 200 spectra test showed that rendering every drawing at once would cause small hangs in the application even with the database being run on the local development machine.

Due to this issue, the spectra spinner was altered to pass a single value to the spectral plotting method. This substantially reduced the load and reduced the hang in the system to a fraction of a second. With the later addition of a thread to handle solely the spectral thumbnail plots the system has no visible slowdown.

#### Implementing a spinner number model to stop index out of bounds exceptions

The JSpinner created had the ability to have any number set by the user and due to the change of the spinner directly influencing the drawing of a spectral plot this would create Index Out of Bounds exceptions. A spinner number model was implemented that had an upper limit of the maximum amount of spectra this is determined by checking the ids ArrayList that is passed into the AddSpectralPlot class. The size of this ArrayList was then checked for its size and then added to the spinner number model. This spinner number model also doesn’t allow for input outside of the models value range which would have also caused an Index Out of Bounds exception.

## Location Spectral Plotting Implementation

### Attaining the location

In order to get the location of the individual spectra, the correct spectra had to first be attained through the graphical user interface of the SPECCHIO system. Within the current implementation of the spectra data browser there is the ability to check the state of the files that are currently highlighted by the user through a tree selection listener, this would in turn give a local array list of current ID’s of spectra that are selected by the user.   
The implemented code takes the current id selected by the user and runs the SPECCHIO client’s method to return both the latitude and then longitude for the spectra with that given id (through two different function calls as the latitude and longitude are separate fields). This SPECCHIO client method returns an AdaptedArrayList which contains the latitude or longitude for any spectra selected with the given ids. The returned array list contains the latitude and longitude for any of the selected spectra and skips any spectra in which there is no location data.

### **Implementing with the SPECCHIO application**

A new class had to be created that was named MapsProcessing, this class exists to deal with the backend logic issues that are required for dealing with the processing of the latitude and longitude as well as opening the new browser window. The other reason for having this new class is to strive for consistent design layout throughout the program as the QueryBuilder class is strictly front end graphical user interface along with the threads used to control any logic from the backend. So the MapsProcessing class was created and placed in the proc\_modules package of the SPECCHIO application.

To ensure consistent design decisions that kept the vision of original authors of the SPECCHIO application it was decided upon that the implementation of the location would be called through the QueryBuilder and threads would be used to ensure the program can continue if any long standing processes are found.

### Multiple Spectra Exporting

Added later within the project the ability to export multiple spectra’s location data was added. This required a full overhaul of the existing method of attaining location data from a given spectra and removed all instance of the toString() method as it was deemed unable to handle multiple sets of location data and was also a bad coding practice when handling multiple sets of data from an array as it would require appending a string and then decomposing a string for each individual spectra.

The data was now being passed into the MapsProcessing class in two array lists which consisted of both the latitude and longitude arrays.

#### Single Spectra

If the passed array lists consisted of only a single entry it was deemed that using Google Maps would be sufficient and more native to the user’s previous experience with mapping software. A simple check is carried out that both of the passed in array lists have only one entry, and if this is deemed true, a URL is generated with the latitude and longitude. These values are inserted into the end user’s default web browser.

#### Multiple Spectra

When multiple spectra are passed into through the QueryBuilder class a check is done to ensure that both the array lists are the same size as to prove that consistent location data was passed and that there is more than one entry in each array list. A simple for loop is then used to iterate through the latitude and longitude array lists, as well as third array list which contains the name of the given spectra.

The name, latitude and longitude are then converted to strings with the ToString() function and appended to a JTextArea with a line break after each read in spectra. This text area is drawn but not shown to the user and the data is inserted into it. When there is only a single spectra left within an array list the contents of the text area are then copied to the users clipboard and a dialog box is displayed to the user informing them that the data should be pasted in the bulk creation tab of the multi plot point mapping website. <http://mapcustomizer.com/> [6]  
A new instance of the users web browser will then open once the dialog has been clicked to receive the users acknowledgement.

### **Thread usage**

The current implementation of the SPECCHIO application for the graphical user interface extends the subclass of Thread. This allows the program to continue working should a thread not work. Checks will be in place if a thread does not complete but this allows the program to not hang entirely. The design decision here is clear in that the system has to deal with a lot of graphical renderings and long running SQL processes.

The location code - although not as CPU intensive - can cause hangs as it has an SQL check against the identification codes that the spectra data uses within the database. The choice was made to continue using threads to implement the location checks.

The thread is simple and calls for the thread to begin when a button is clicked within the GUI. This thread then gets the location data based on the current ids and passes them to a new instance of the maps processing class so the data can be manipulated and a new browser can be opened.

### **Refactoring code for good design**

Upon initial creation, the logic based problems for the code were inserted into the thread. Although this did initially serve as a good basis to ensure that the location data was being collected correctly, implementation could not continue this way as the application has a clear design for both the front and back end. All logic would be passed through to the Maps processing class and would in turn be called by the thread.

### Implementation Issues

When implementing the location based spectra collection, there were a series of challenges that had to be overcome to ensure the correct data would later be used.

The issues were as follows:

* Null spectra locations from SPECCHIO clients getMetaparameterValues method
* Compatibility issues with different operating systems

#### Null spectra locations

Within the given spectra the location data may not be present but this does not return a null when the ArrayList method is called on the getMetaparametesValues, this problem has been dealt with through a simple if array list = 0 check and then a JDialog box is shown to the user informing them that no location data exists for the given spectra/ set of spectra.

#### Compatibility issues with different operating systems

The ability to open a new web browser has issues within different operating systems but within Java there is a Desktop class that allows for checking of applications that run natively within the current setup. This allows for the users default browser to be opened with the URL that is defined with the location data. [5]

## Addition of User Feedback into Campaign Creation

The additional user feedback implementation relied on the ability to pass significant information to the user through Swing components. Because of this the SpecchioCampaignDataLoader class extended its functionality to include updating of strings for the operation that is currently taking place.

### Implementing Swing

The Swing components that were added were as follows:

* JTextArea – progress\_text
* JScrollPane – text\_scroll
* JSeperator – separate

These components were all that were required to give the end user a detailed feedback area. Implementing these components required the already implemented GridbagLayout to be updated with simple variable updates to extend the state of the frame.

These components however would not be required to be viewed at all times by the user and they only became relevant as a campaign is being created. In order to show them correctly they had to be instantiated at the time of creation of the frame. This then allowed them to viewed when the create campaign or create and load campaign button was pressed.

### Deciding which data to return to the user

Within the SpecchioCampaignDataLoader class, many pieces of information could have been returned to the user ranging from file extension names to successful file counters. Distilling the information down to the point where it is human-interpretable was a large part of the implementation within this requirement.

The final choice for the information returned was:

* The selection of files being read
* The individual file identification name as it is being read
* The insertion into the database along with its insertion ID
* The hierarchy set by the SPECCHIO application for file placement

### Implementation issues

As with many swing components a variety of issues arose when updating the original implementations frame as the new components were being created in a different way to the previous graphical user interface components. The class itself was extending the JFrame class so as to only set the view for the user with all logic being placed into the SpecchioCampaignDataLoader and the LoadCampaignDataHandler. Through testing of the frame and ensuring that the viewpoint was correctly being set to the JTextArea, one concise view was created which didn’t hinder the original view of the NewCampaignDialog.

The strings returned by the various variables being parsed by the ToString () Function had to have additional prefixes for the user to understand which piece of information was being added along with new line spaces being added after each text entry had been added.

## Implementing GlassFish 4 into the Current System

The implementation and upgrade to GlassFish 4 was the first piece of functionality that was added to the system as the requirement was added shortly after the code analysis was completed. An entirely new version of GlassFish install required different facets within the SPECCHIO Web Application to be updated as shown in [Figure 12]

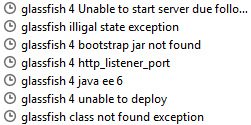


**Figure 12 – GlassFish Facets**

Also importantly the declarations for the Jersey client were required to be updated as the servlet containers definitions were changed during the update and thus the application would fail to deploy with exit code 1. This required an updated web.xml files to be created with update servlet declarations.

### Implementation issues

As shown in [Figure 13] the search history that was required to upgrade the system shows how much of an arduous process upgrading was.  
  
The main implementation issues were deployment and publishing issues as well as incompatibility errors with the java version or servlet decelerations.



**Figure 13- GlassFish Search History**

Within the JDBC Connection Pool, speechio\_web\_pool the username and password fields had to be added manually when upgrading the system with the default root password and username used within the MySQL database.

### Upgrading the system

The new GlassFish installation was carried out from oracles website and not through the eclipse package manager as some dependencies are not passed to eclipse or at least are rolled out at a later date to oracles. Once the user has installed the new version of eclipse and followed the updates path laid out in the upgrade documentation the new version should fit seamlessly into the SPECCHIO application.

### Jax RS-2.0 Upgrade

The Jax RS-2.0 upgrade requires honourable mention as it is one of the lesser known upgrade requirements when updating glassfish versions as the eclipse downloader of GlassFish does not include this upgrade. This required a considerable amount of research to find the error in question as GlassFish in no way points this out to the user as an upgrade requirement.

The following passage is taken from the updated documentation that was created to go alongside the upgrade process as this particular step at the time of writing could only be carried out in this specific order.

“Glassfish 4 with the current implementation of SPECCHIO will not work due to its incompatibility with JAX-RS 2.0. Following the steps below the user will be able to upgrade the standard version of glassfish install with the newest version of JAX and glassfish (along with any minor updates that may come in from the time of writing)

1. Right click and run your new glassfish installation.
2. Head to the glassfish tab and click glassfish update centre.
3. This will then prompt you to install the update centre software through the command line of the eclipse launcher. (This process will take a long time to complete)
4. Right click the glassfish 4 implementation under the server tab within eclipse and select the glassfish tab and then select update centre again.
   1. If this does not work head to your installation folder of glassfish and double click on your bin folder, within this folder there will be a batch file called updatetool. Run this file and it will open a command window then type y when instructed to do so.
5. This new window that pops up will have application images in the left hand pane, within this tab click on the available updates tab.
6. Install all updates as this is not done during the download through eclipses marketplace. (Must ensure that the JAX-RS 2.0 update is within this list of updates)”

## Python Script for Auto Uploading to the Database

The python script is a standalone application that has been developed to be run from an external flash drive. The script uses Python 2.7 as the newer version of Python was incompatible with the MySQL plugin that was used when inserting spectra into the database.

Its functionality includes:

* The ability to ask the user for a given file directory and create folders if they do not exist.
* Parse an XML document for the required fields, primary keys and foreign key constraints used with the specchio database.
* Insert data into the specchio database with the correct hierarchy decelerations, primary keys and foreign keys.

The implementation include a vast array of imports which were required to carry out functionality such as:

* Connect to the specchio database
* Exit the script if a connection is not established
* Ask the user for a spectra directory
* Create new folders for spectra to be passed into once they have been read
* Check for files containing a given extension
* Parse an XML document with the use of the ElementTree import
* Execute MySQL statements to insert data from Python variables

All the pieces of functionality are designed and coded for scalability, the function definitions can be changed to search for a specific file type and then run the correct parsing function to attain the correct data for the database.

### Code layout

The program begins by asking the user for the folder in which the spectra are stored for test purposes this was the E: drive where the test XML file was being stored. This uses the tkFileDialog import which is built into Python, the output is then stored to a variable called input\_folder and the string return then needs to replace all of the back slashes with forward slashes within a windows setting.

A try block is then used to connect to the database with the credentials predefined within the script, an exception is caught if the connection failed and if the user runs the Python code from the command line the exact error will be printed and this causes the system to exit thus ending the script prematurely.

A function is then called which parses xml documents called xml\_to\_database\_for\_spectrum. This function holds the majority of the Python code in order to insert the spectra into the database. This function has variables created for the current id, current hierarchy and current eav as they are primary or foreign key constraints that are defined when parsing the document.

For example the spectrum field when read in contains the primary key constraint spectrum\_id, which is defined when being read into the database and is then passed back out into the current spectrum variable to be used in the insertions with spectrum\_x\_eav.

The code is designed to loop through each field within a given table and return the columns and values associate with a given tables these are then inserted into the database through a MySQL insert statement. This is executed 5 times for each of the required tables needed for the database insert and in a specific order to not include any primary or foreign key constraints.

Finally the database connection has to be closed and the read files are passed into a new folder called Target folder which is created to be nested inside the original folder chosen by the user.

### Finding correct MySQL plugin

Initially the Python script was being created in Python 3.4.3 as the XML parser was created first. But the MySQL plugin chosen had a Python 2.7 dependency, this is a minor point but it shows the poor adoption of the newer version of Python as at the time of writing it has already been released for 3 months [10] and no MySQL functionality exists other than an interpreter [11]. This plugin however is the only import that does not come natively with Python 2.7 and is the only constraint other than installing Python when running the script.

### Implementation Issues

The major issue when developing this script was the correct order in which to insert the data into the database, as for example the spectra could be read in without issue as it had no dependencies but when the spectra\_x\_eav table was read it failed due to foreign key constraints. This caused a need to drop the database as certain spectra were individual from the rest of the database.

## Review of Implementation

The implementation of the each of the five major pieces of functionality lived up to the expectations that were set during the analysis phase, from an implementation stand point. As to be detailed during the critical analysis, one initial requirement to upgrade the user interface did not end up being as successful due to static calling of Swing components within the main SPECCHIO application.

The main objective from a personal standpoint when developing and implementing the requirements was to have core functionality with extensive error handling and user feedback as the live version should include these in an enterprise system.

The spectral thumbnails and maps processing were implemented simultaneously as they shared many thread uses, and required the same spectral information to be passed in (the maps slightly less). The fact that the logic is being dealt with in an external package, and classes makes any changes or extensions simple to implement if Andreas or his co-developers decide to expand upon the functionality that have been pointed out in the design and implementations of each.

The upgrade to GlassFish 4 took up a far more substantial amount of time than first allocated. This was primarily due to the error dialog messages being vague at best the majority of the time and required more research into fixing issues than actually implementing the system.

The Python script was actually less difficult than first imagined as the language was new to myself and time allocation was more than was required for the functionality that ended up being produced.

# Testing

## Overall Approach to Testing

Throughout the course of the project testing was carried out in a manual form as most of the test carried out were Java swing related and would require external tools to test the swing. It was deemed more time consuming to test the swing with an external tool than to manually test the swing components.

The other test that were carried out were done throughout the project, the strategy was to insert and compile code and deal with any issues and add them to the test tables as they became an issue.

The tests carried out were more than adequate for the functionality implemented and were dealt with accordingly with the amount of error handling that was implemented throughout the system.

The application was passed to a group of students within the university in which they were tasked with breaking the implemented functionality, these test cases pointed out the flaws in functionality and are pointed out in the (a) sections of the test table.

Unfortunately the system would have been tested on the end user and original creator of the system but due to a version of the SPECCHIO system being pushed in early May the addition of the implemented functionality would not be added till a later version of SPECCHIO. Constant communication with the customer on their updated functionality requirements ensured that when the system finally makes it into a later version that testing that has already been carried out should be sufficient when being pushed to the live application.

The iterative nature of the project allowed for testing to be completed after each revision to the code, and then an evaluation to be carried out as to set the state of the code to “done done” otherwise it would complete another analysis and implementation cycle to improve the final product.

## Test Tables

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Description of Test | Expected output | Observed output | Comments |
| 1 | GlassFish 4 deploys web application | Web application is deployed onto glassfish and program runs | Error thrown  com.sun.jersey.spi.container.servlet.ServletContainer | The application doesn’t appear to deploy on Glassfish the servlet container is not being picked up by Glassfish. |
| 1(a) | Glassfish 4 deploys web application | Web application is deployed onto glassfish and program runs | Application deploys and the server shows that the application is published and ready to be used | It appears that Glassfish 4 requires an updated web.xml file in order to deploy an application.  The declaration of the servlet was required to be updated to org.glassfish.jersey.servlet.servletContainer |
| 2 | Database connection can be established with new deployment software | Attributes from the database are returned to the user and passed into the data browser | Authentication failure 401: unauthorised | It appears then when upgrading to GlassFish 4 the root user account has to be added in manually |
| 2(a) | Database connection can be established with the new deployment software | Attributed from the database are returned to the user and passed into the data browser | Authentication successful, progress bar appeared as expected and data was passed from the database to the data browser | Within Glassfish 4’s web configuration menu the specchio\_web\_pool required manual addition of the username and password for the root user being accessed by the database |
|  |  |  |  |  |
| 3 | Spectral thumbnail appears in panel | When the tree browser is updated with a user’s click the GUI updates with the correct spectral thumbnail | The thumbnail is returned correctly and is drawn and repainted to the screen | The implementation of the passing of the spectral thumbnail panel was implemented correctly through the use of passing a panel between classes and updates correctly |
| Test Number | Description of Test | Expected Output | Observed Output | Comments |
| 4 | Spectral thumbnail updates with the spinner | When the spinner is updated with more than one spectra the panel redraws correctly | The panel does not redraw correctly and in turn does now show a new spectra to the user | Need to figure out why the panel is not being updated to the user the input data is correct from the array list and spectra id but the frame is not updating so it must be a swing error |
| 4(a) | Spectral thumbnail updates with the spinner | When the spinner is updated with more than one spectra the panel redraws correctly | The panel now redraws correctly after the initial panel is removed as they were being stacked on top of each other | There was a swing issue where a new panel was being passed to the query builder but was not being updated removing and redrawing solved this error |
| 5 | Spectral thumbnail spinner goes under or over the allowed limit | The spinner should not return any information when it goes above or below its limits | The spinner throws an index out of bounds exception as it is passed the array list entries and it causes them to go out of bounds | Need to limit the length of the spectral spinner as to stop an index out of bounds exception will probably need a spinner number model |
| 5(a) | Spectral thumbnail spinner goes under or over the allowed limit | The spinner should not return any information when it goes above or below its limits | The spinner is now limited to how many spectra are currently highlighted by the user and thus does not throw an index out of bounds exception | After adding in a spinner number model the spinner cannot go out of bounds as the model is locked to the amount of entries within the array list |
| 6 | Spinner is removed when only one spectra chosen | The spinner will be removed from the panel and only one spectral plot will be shown | The spinner was removed and the spectral graph is now centred | A quick check had to be carried out for when there is only one spectra inside the array list and it was just a case of removing the spinner from the swing component |
|  |  |  |  |  |
| 7 | Show in maps button is not enabled when no spectra selected | The button for showing and outputting location data into mapping software is greyed out when no spectra are selected | The button is not enabled when there are no spectra selected | This was quite a simple problem to solve as the initial implementation of SPECCHIO already had this functionality set with their buttons it was just a case of adding my own to the list |
| Test Number | Description of Test | Expected Output | Observed Output | Comments |
| 8 | Show maps button shows error when no location data exists | A dialog is shown when there is no location data for a given spectra | The Dialog opens warning the user that there is no location data available for the given spectra | Doing an array list check for an empty array list for the location data and then running it through an if statement that pops up a JDialog box |
| 9 | Show maps button leads to google maps if only one spectra selected | The output when clicking on the show in maps button with only one spectra is opening a new web browser for the user importing the spectra’s location data into google maps | The spectra opens within google maps but the output string has unknown brackets within the string | It appears that the array list is returned as follow : [-140.00009] The braces need to be removed from the string when being input into a URL |
| 9(a) | Show maps button leads to google maps if only one spectra selected | The output is now imported into google maps with the correct url | The output is now correct for the given single spectra | Running a substring method on the given toString() returned method removes the front and back of the string. |
| 10 | Multiple selected spectra get passed to clipboard and then exported to mapping website | When the show maps button is clicked with multiple spectra selected the location data will be exported to the users clipboard and be ready to be pasted into the mapping website | As expected all of the spectra with location data have been passed to the clipboard along with the correct formatting and names and in turn can then be shown as a multi plot map | Using a JTextArea made the appending to the strings required to make this copy to the clipboard possible, multiple spectra can be exported and shown in maps. This is useful to the user when the spectrograph device is mounted to a moving vehicle. |
|  |  |  |  |  |
| 11 | Additional swing components added to the campaign creator | A new separator added to the campaign creator and new JTextArea when a new campaign is created | The new swing components are not being drawn correctly to the campaign creator | The new campaign creator frame does not update with the new swing component |
| Test Number | Description of Test | Expected Output | Observed Output | Comments |
| 11(a) | Additional swing components added to the campaign creator | A new separator added to the campaign creator and new JTextArea when a new campaign is created | The new components are now added to the campaign creator | The new campaign creator frame needed to be repacked when the new components were added as they were being added but the frame was not being extended |
| 12 | Addition of data being inserted into the JtextArea | As the data is being read into the database the file names are added to the campaign creator text box | The data is being added to the campaign creator | The file names are now added to the campaign creator text box and this gives a user more of an idea of what data is being inserted into the database |
| 13 | The hierarchy for the given spectra is added and printed to the JTextArea | As the spectral data is being inserted into the database it is given a hierarchy for use within the data browser this information should be relayed to the end user | The spectra is having the hierarchy added to it and the output into the JTextArea is being displayed to the user | As data is being inserted into the database it requires a hierarchy number as to be placed into the correct sub folder for each given spectra this is not relayed to the user currently and is added to the JTextArea for the users information |
| 14 | The JTextBox scrolls to the bottom as data is being relayed to the user | As information is being added to the JTextArea the user should see the bottom entry to see the current progress | The viewpoint of the JTextArea is set to the bottom as new information is added this can be scrolled through in order for the user to see all processes that have been carried out | This addition was created in order for the user to get a second by second update on how information is being added to both the application and the database |
| 15 | The file name for the currently being read in piece of information is displayed within the JTextArea | As the files are being read into the SPECCHIO application the file name is updated and shown to the user so they can see what file is currently being read | The file name is added to the top of the currently being read piece of information this allows the user to see what is happening to each file as it is being read in | The output previous to this addition did not include the file names and thus was almost useless information to the user as it was just printing an id for the database read |
| Test Number | Description of Test | Expected Output | Observed Output | Comments |
| 16 | Python script connect to the database under the given credentials or an error is thrown if the details are not correct | When the script is run the database will be connected to or the an error will be shown to show the credentials are not correct | As expected nothing happens if the database connection is correct or an error is thrown if the credentials are not correct | The reason why nothing is displayed to the user if the connection is established is that there is going to be no console available to the user when this script is run. And the import sys closes any further operation |
| 17 | Files that have been read are automatically moved to a new folder within the initial folder chosen by the user | Tk file dialogue asks the user for the folder with the given spectra once they have been read the files automatically get placed into a new folder | The file browser works correctly but because a folder does not exist with the given name it does not automatically create a new one a thus places the files into a binary file which cannot be accessed | I will need to add in an automatic folder creation in order to deal with the moving of files |
| 17(a) | Files are now read into a new folder | The script will now create a new folder if there is not a folder of the name Target Folder within the given folder for spectra | Now when the user selects a folder where the spectral files are stored as they are read and moved the script will now create a new folder of the name Target Folder | This was implemented in order to stop repeat reads of the same file |
| 18 | All files selected are of the given file format | When the user selects a folder all XML files should be read and processed by the script | As expected the script runs through a for loop reading in each of the xml files and processing them accordingly using glob | Glob was imported within python as it automatically returns any files with the given input : input\_folder+\\*.xml |
| 19 | Files are read and inserted into the database | As files are processed by the xml reader they will be inserted into the database | The files are read and inserted but any additions are not committed | It appears that the inputs into the database are not being committed. |
| 19(a) | Files read into the database are now committed | The cursor that was being created for the read to the database required a commit function to be called | Now the database is being updated with the newly inserted spectra with the foreign key constraints | The database just required a commit function to be called in order for any changes to be processed |

# Critical Evaluation

## Were requirements correctly identified

For the most part the requirements were laid out very well, but one in particular could have been expanded upon more. This was the return of information to the end user on database inserts. Although I am thoroughly content with the way the implementation for this particular requirement there was already a basic implementation showing the user what is being inserted I just chose to develop upon this in a more formal manor that would be seen in other projects.

The addition of the GlassFish upgrade came as a surprise but as it was quite early on within the development life cycle [12] it ended up not only, not being an issue, but a substantial upgrade for the SPECCHIO system and will serve as a fantastic basis for future upgrades.

Apart from the two written, the requirements were correctly listed and I received a good deal of guidance from both my supervisor Mark and customer Andreas on where to set the scope for the basis of my analysis.

Although the initial project specification was daunting, as I have no previous experience with spectroradiometers or spectral data, it quickly shaped itself around my core java programming skills and allowed me some exciting functionality implementation to help new users such as myself to understand and manipulate spectral data.

## Were design decisions correct

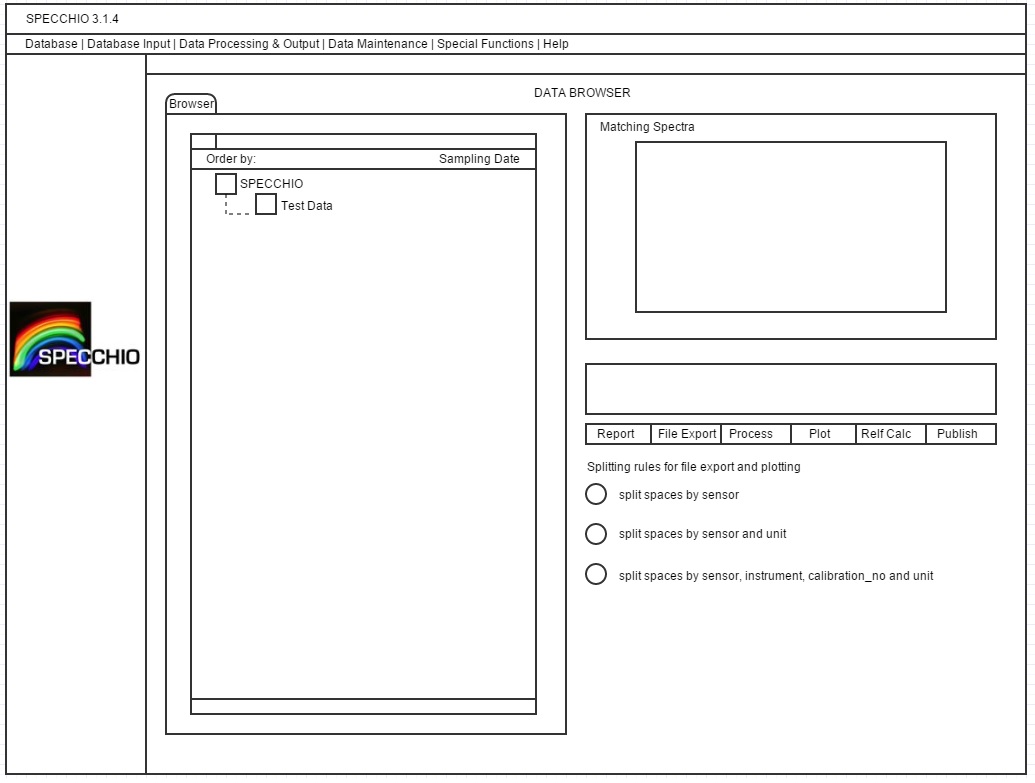
The design decisions chosen, were done so based on the initial project specification and through the correspondence of Andreas. But given an ideal situation for the project it would have been a fantastic opportunity to try an agile methodology, as with a project such as this where functionality is essentially the majority of the project held together with a GUI.

I believe that extreme programming, in particular pair programming would have been, and could still be an exciting method of implementing new functionality as the problems, to myself at least were exciting puzzles. I believe truly great and innovative functionality can come from this program and will monitor the success of the application long after the dissertation is over.

The coding carried out underwent massive revisions as refactoring was the key to my success. I mercilessly refactored any classes that I implemented in order to make them as small as possible, and as readable as possible, as this code was being passed off to another developer / team of developers. My main concern throughout this project was not to pass off bad code in hopes that someone else would fix it, on the contrary I would Andreas to view a class instantly see its worth and implement it into the system.

## Were project aims met

All of the core requirements were met to a degree, there is in particular one piece of functionality that I wanted to implement but did not come to fruition. This was the implementation of additional windows into the operations pane to reduce the ‘clutter’ of the system (the operations pane is the main body of the system). Unfortunately the Swing methods were being called in a static manner and every attempt that I made to implement windows into the operations pane ended with frustration or with broken Swing components that could not be updated in any manner. It truly upset me as I had big plans for how the application would look after this change had been made.



**Figure 14- Initial GUI Concept**

As shown in [Figure 15] the aim was to have one main application window that held all of the major functionality of the system such the data browser. As it stands at the moment every menu item is opened in a new frame and I felt like this could be improved significantly.

I foolishly dedicated more time to this than was really justified for the progress that was being made and as such feel more than 5 working days behind my initial schedule.

## Time management

Apart from the above listed, my time management was actually very well split, and used to achieve everything that I set out to do. When it came to implementing code I set myself less time to implement than would initially be required but would give myself a buffer period to handle anything that may have run over slightly. The reasoning behind this is when I code a piece of functionality, I don’t want this to be split over weeks where the inevitability of forgetting something will happen. This allowed myself to implement requirements quickly and deal with any issues there and then, instead of coming back to clean up afterwards. Also I know I work well under pressure so setting short deadline gave more incentive to create good work.

## What would I have done differently

Given an unlimited amount of time and resources ‘ergo’ the most beneficial circumstances to produce and produce under, I would have chosen a single person to try out some pair programming and this would have been beneficial experience with a lot of companies choosing some form of an agile methodology.

An exciting prospect that could function as a new project or dissertation for another student would be to create a web interface for the system. The pieces are all pretty much there, the database where the information is stored is elsewhere and accessed remotely. The application is already deployed through the use of servlets and MATLAB functions can be deployed to the web. It just requires a web minded individual to make it happen. Also the password encryption method would have be improved slightly if making the application accessible to everyone.

## Managing a large enterprise project

This section requires very special mention as previously to the beginning of this project I had no experience what so ever in managing a project larger than 20 classes within a single package. This required a vast amount of time to be spent in version control systems (GitHub) analysing and poking sections of code to see how they interacted with other systems. It was a daunting prospect at first, but once the general code layout was realised and how functionality was split from the user interface it became not only manageable but actually quite easy to find the next section that had to be worked upon.

## Testing

Testing was not as thorough as I wanted it to be and was defiantly the weak link within the project, it’s not to say that the functionality implemented isn’t everything that I wanted it to be. But I would have liked checks to be in place as I pass off the code to Andreas and his development team.

Swing Testing would have been a fantastic addition as the majority of the updates made influenced the GUI in one way or another.

Unit testing would have also been useful for passed information, I did at one point actually try to implement some unit tests, but due to the nature of some of the data being passed in I couldn’t get conclusive results through the unit tests other than getters and setters which did not make for good unit testing.

## Conclusion

In conclusion I believe that the project was a huge success, the requirement I didn’t fully develop was quickly replaced with an alternative and just as important update so I feel like that time was put to good use.

The code I have created should easily slot in with the current system, at the time of writing the code is not being pushed until after a minor version update slightly after mid-May. I would have liked this to be sooner for the sake of having some end user’s testing the functionality that I created but I understand that releases can only be as frequent as the developers can produce.

I sincerely anticipate the future updates that include my functionality and look forward to seeing how the application develops and expands in future iterations.

# Appendices

* 1. Third-Party Code and Libraries

Although I personally did not add any libraries to the system there were a few that are required for the project to run and they are as follows:

* Appbundler-1.0 – Java application bundler downloadable from java.net
* JCalendar – java date chooser for graphically picking a date, covered by the GNU Public license [21]
* JCommon – Swing upgrades and additions, covered by the GNU Public license [20]
* Jersery-client, Jersey-core – used for web client interactions, covered by the GNU Public license [19]
* Jfreechart – used for create graphs within java, covered by the GNU public licence [18]
* Jgraph – Open source graphing, covered by GNU public licence [17]
* Jhdf - API for HDF viewer, covered by GNU public licence [16]
* Joda-time - Used for date time configuration [15]
* Jxl – Used for modifying excel spreadsheets [22]
* Ptplot – Used for graphing purposes [14]
* Qccchart – Used for quality control charts [13]
* Ujmp – Universal java matrix package [23]

# Annotated Bibliography

[1] “J. Elliott, R. Eckstein, M. Loy, D. Wood, and B. Cole, Java Swing, Second Edition, 2nd ed”. O'Reilly Media, Nov. 2002. [Online].

Available: <http://www.worldcat.org/isbn/0596004087>  
  
This book was used for swing referencing when using new components.

[2] “How do I connect to a MySQL database in python? - stack overflow." [Online]. Available: <http://stackoverflow.com/questions/372885/how-do-i-connect--a-mysql-database-in-python>  
  
This stack overflow question allowed for the basis of the python database connection.

[3] “Find all files in directory with extension .txt with python - stack overflow."

[Online]. Available: http://stackoverflow.com/questions/3964681/find-all-files-in-directory-with-extension-txt-with-python

This stack overflow question helped with finding all of the xml files within a given directory.

[4] “E. Bendersky, Processing XML in python with ElementTree - eli bendersky'swebsite." [Online]. Available:<http://eli.thegreenplace.net/2012/03/15/processing-xml-in-python-with-elementtree>

This webpage was used for understanding the basics of pythons element tree function.

[5] “How to open the default web browser using java - stack overflow." [Online].

Available: <http://stackoverflow.com/questions/5226212/how-to-open-the->default-

webbrowser-using-java

This code snipped was used and copied exactly into my MapsProcessing class as it is described to work in multiple operating systems and as I had no way of testing the result with another operating system I felt it best not to alter something that is proved to work.

[6] “swing - copying text to the clipboard using java - stack overflow." [Online].

Available: http://stackoverflow.com/questions/6710350/copying-text-to-the-

clipboard-using-java

This webpage was used as a code basis for copying the location data automatically to the clipboard from a JTextArea within the MapsProcessing class.

[7] “Hueni, A. (Feb 2015). *Desktop Spectrometry kit*.” [email]

Email from Andreas on link to github where source code was given.

Reads: “Hi Andrew, The current source code is Available onGitHub:<https://github.com/ahueni/SPECCHIO>

Let me know if you got further questions that need answering.

Cheers Andy”

[8] “The google geocoding API - google maps API web services - google developers."[Online].

Available: <https://developers.google.com/maps/documentation/geocoding/>

Used to show the usage limits on the free google maps API

[9] “Shneiderman's eight golden rules of interface design." [Online]. Available:http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html

Schneidermans eight golden rules influenced heavily on the interface design.

[10] “Python release python 3.4.3 | python.org." [Online]. Available:

<https://www.python.org/downloads/release/python-343/>

Shown release date for python to show the adoption rate

[11] “Connecting python 3.4.3 to MySQL - stack overflow." [Online]. Available:

<http://stackoverow.com/questions/28754563/connecting-python-3-4-3-to-mysql>

Python 3.4.3 python interpreter example

[12] “Hueni, A. (Feb 2015). *Project beginning.*” [email]

Email from Andreas in which he asked me to look into the GlassFish 4 implementation. Reads:

Hi Andrew,

Most excellent, I’m happy to hear you are excited about it!

I’m attaching a draft of a paper that outlines the concepts of the latest system; that should give you an overview about the general concepts.

All the source code should be online to setup the system.

The document SPECCHIO\_ServerInstallation.docx contained in the doc directory describes how to install the system.

One thing I’d like to learn is if the system works also on the latest version of Glassfish; we have tested it on version 3.1.2.2, so, if you would do an installation on your development machine using the latest version that’d be great.

Let me know if you need any help, we can talk on Skype as well.

Cheers ,Andy”

[13] “qc chart 3 - google search." [Online]. Available:

https://www.google.co.uk/webhp?sourceid=chrome-

instant&#38;ion=1&#38;espv=2&#38;ie=UTF-8#q=qc+chart+3

Library reference

[14] “Ptolemy II ptplot." [Online]. Available:

<http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII8.1/ptII/ptolemy/plot/doc/index.ht>

Library reference

[15] ”Joda-Time - home." [Online]. Available: <http://www.joda.org/joda-time/>

Library reference

[16] “Debian { details of source package jhdf in squeeze." [Online]. Available:

<https://packages.debian.org/source/squeeze/libs/jhdf>

Library reference

[17] “JGraph - wikipedia, the free encyclopedia." [Online]. Available:

<http://en.wikipedia.org/wiki/JGraph>

Library reference

[18] “JFreeChart." [Online]. Available: <http://www.jfree.org/jfreechart/>

Library reference

[19] “Jersey - legal information." [Online]. Available: <https://jersey.java.net/license.html>

Library reference

[20] “JCommon." [Online]. Available: <http://www.jfree.org/jcommon/>

Library reference

[21] “JCalendar." [Online]. Available: <http://toedter.com/jcalendar/>

Library reference

[22] “JExcelApi." [Online]. Available : <http://jexcelapi.sourceforge.net/>

Library reference

[23] “Universal java matrix package | a java matrix library for linear algebra and high

performance computations." [Online]. Available: <https://ujmp.org/>

Library reference