**Visualisation 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 1.0 (Draft)

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 ……………………………………………. (Your Name)

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 ……………………………………………. (Your Name)

Date …………………………………………………

C:\Users\Andrew\Downloads\anh23-barcode.png

**Acknowledgements**

I am grateful to…Mark Neil, Andres Hueni

I’d like to thank… Ben Brooks

**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. These requirements were to update the graphical user interface, add location exportation of spectra to external mapping software.

FINISH LATER SHOULD BE WRITTEN LAST.

Include an abstract for your project. This should be no more than 300 words.

**Contents**

1. Background, Analysis & Process 8

1.1. Background 8

1.1.1. Background languages 8

1.1.2. Background Systems 8

1.2. Unused systems 9

1.2.1. Software Deployment systems 9

1.2.2. Scripting systems 9

1.2.3. Database systems 9

1.3. Analysis 9

1.3.1. Spectral Thumbnail Update Overview 9

1.3.2. Exporting Location Data to External Mapping Software Overview 10

1.4. Process 10

2. Design 11

2.1. Location Data Exporting to External Mapping Software design decisions 11

2.2. Spectra thumbnails design decisions 11

2.2.1. Spectral Thumbnail Class Diagram 12

2.3. Python Script Design Decisions 13

2.4. User Feedback When Inserting Spectra Design Decisions 13

2.5. Upgrade to Glassfish 4 Design Decisions 13

2.6. Overall Architecture 13

2.7. Detailed Design 13

2.7.1. Even More Detail 13

2.8. User Interface Design 14

2.9. Other Relevant Sections 14

3. Implementation 15

3.1. Spectral Thumbnail Implementation 15

3.1.1. Implementation Issues 15

3.2. Multi Location Spectral Plotting Implementation 16

3.2.1. Attaining the location 16

3.2.2. Getting the correct string for the location 17

3.2.3. Implementing with the SPECCHIO application 17

3.2.4. Thread usage 17

3.2.5. Refactoring code for good design 17

3.2.6. Implementation Issues 18

4. Testing 19

4.1. Overall Approach to Testing 19

4.2. Automated Testing 19

4.2.1. Unit Tests 19

4.2.2. User Interface Testing 19

4.2.3. Stress Testing 19

4.2.4. Other Types of Testing 19

4.3. Integration Testing 19

4.4. User Testing 19

5. Critical Evaluation 20

6. Appendices 21

A. Third-Party Code and Libraries 21

B. Code Samples 22

7. Annotated Bibliography 23

# 

# Background, Analysis & Process

In preparation for the project a range of background research, analysis and processes 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 creation tool 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 code.

Python was chosen for the script 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 the given scripting language within its own self-contained script that can 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 learn the current implementation of the SPECCHIO application in its current form which is (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 implementation that would be built upon, this required an account to be created with both the test database and the live database that are used.

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 they current 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 the adding of users 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.

## Unused systems

### 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 and the reason it was not chosen was the inability to be used on a windows machine without the use of powershell or a similar software.

### Database systems

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

## Analysis

With the background research complete it became obvious that the current implementation of SPECCHIO required some graphical user interfaces upgrades. Additionally more information 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 SPECCHIO application was the ability to view thumbnails for any given spectra before the user commits to opening a report for the given spectra. This data had to be inserted into an existing part of the graphical user interface within the Query Builder frame with all backend logic being dealt with in a separate class.

### 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 an external mapping software, this came with a set of problems and implementation issues that needed to be worked out. The design choice for this requirement is that the location can be passed through a single class and with the correct method calls open a new browser window that displays location of given spectra.

Taking into account the problem and what you learned from the background work, what was your analysis of the problem? How did your analysis help to decompose the problem into the main tasks that you would undertake? Were there alternative approaches? Why did you choose one approach compared to the alternatives?

*Upgrading to glassfish 4 could have been done in another deployment software but the convenience of using glassfish due to its SQL ties pre implemented.*

*Updating graphical user interface*

*Calling correct spectra*

*Auto running script when plugged into computer.*

*Adding to the graphical user interface with appropriate spectra thumbnails*

There should be a clear statement of the objectives of the work, which you will evaluate at the end of the work.

In most cases, the agreed objectives or requirements will be the result of a compromise between what would ideally have been produced and what was felt to be possible in the time available. A discussion of the process of arriving at the final list is usually appropriate.

## Process

You need to describe briefly the life cycle model or research method that you used. You do not need to write about all of the different process models that you are aware of. Focus on the process model that you have used. It is possible that you needed to adapt an existing process model to suit your project; clearly identify what you used and how you adapted it for your needs.

# Design

//TO DO

You should concentrate on the more important aspects of the design. It is essential that an overview is presented before going into detail. As well as describing the design adopted it must also explain what other designs were considered and why they were rejected.

The design should describe what you expected to do, and might also explain areas that you had to revise after some investigation.

Typically, for an object-oriented design, the discussion will focus on the choice of objects and classes and the allocation of methods to classes. The use made of reusable components should be described and their source referenced. Particularly important decisions concerning data structures usually affect the architecture of a system and so should be described here.

*Internal pane*

*Mapping Processor*

*Thumbnail processor*

*Database verification*

How much material you include on detailed design and implementation will depend very much on the nature of the project. It should not be padded out. Think about the significant aspects of your system. For example, describe the design of the user interface if it is a critical aspect of your system, or provide detail about methods and data structures that are not trivial. Do not spend time on long lists of trivial items and repetitive descriptions. If in doubt about what is appropriate, speak to your supervisor.

*Updated documentation of pre existing software*

You should also identify any support tools that you used. You should discuss your choice of implementation tools - programming language, compilers, database management system, program development environment, etc.

*Java*

*Xml*

*SQL*

*Python*

Some example sub-sections may be as follows, but the specific sections are for you to define.

## Location Data Exporting to External Mapping Software design decisions

//TO DO

## Spectra 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 MATLAB’s implementation of the getMetaParameters. This method had the possibility to pull back 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 that was decided upon 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 of the given spectra.

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 SPECCCHIO 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.

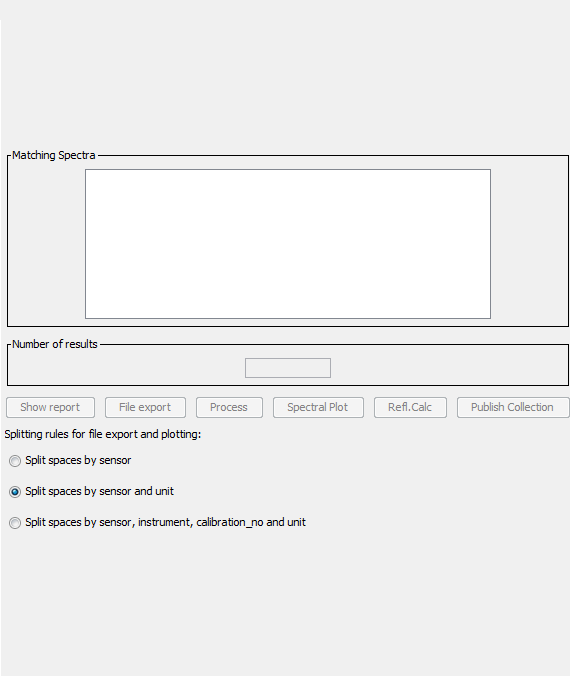


Figure 1 – old specchio query broswer mplementation

The graph had to be plotted somewhere within the current application 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 1], this gave the ability to add a small spectral plot to the query browser.

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 2]  
  
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

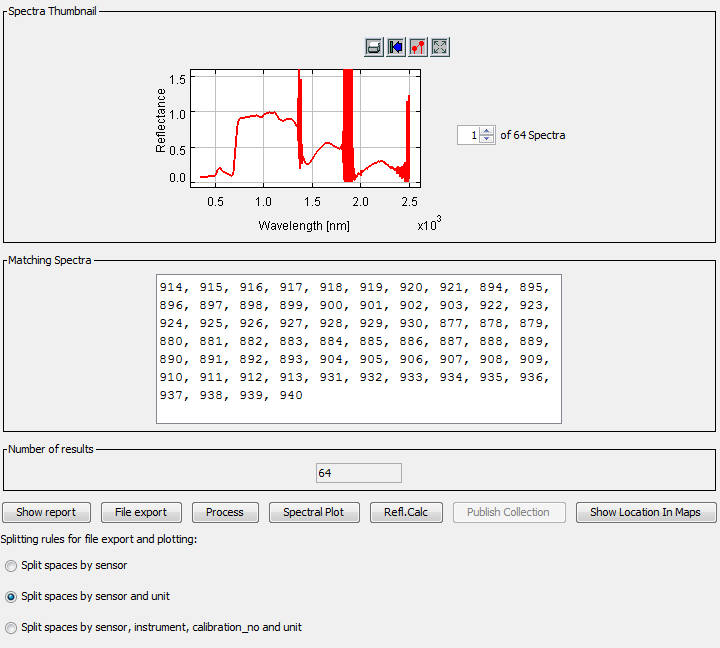


Figure 2 - new implementation of spectral thumbnail

### 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.

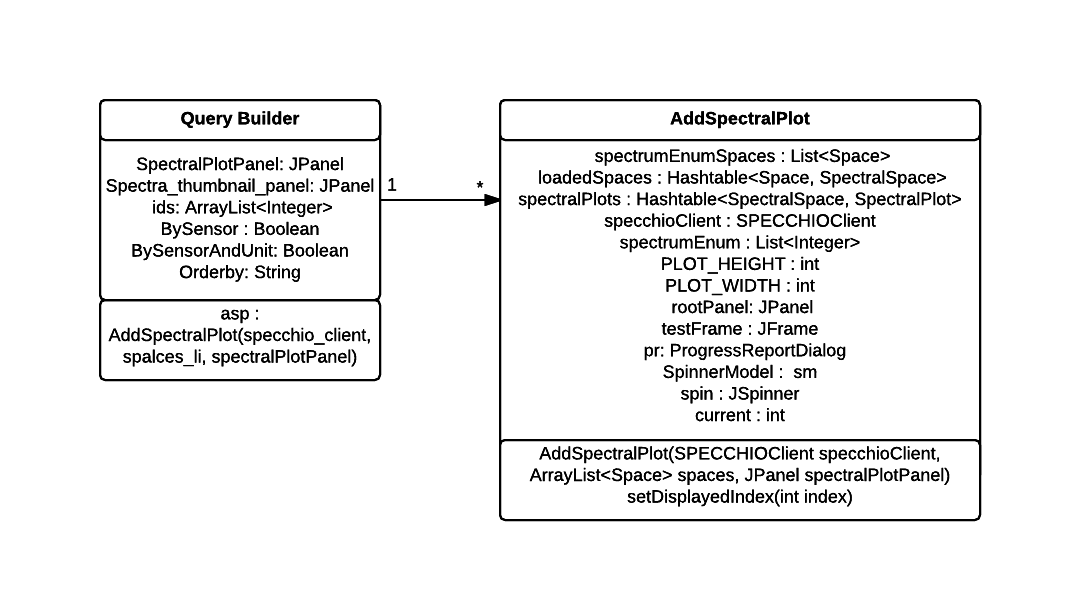


Figure 3 – Query Builder Class Diagram

## Python Script Design Decisions

## User Feedback When Inserting Spectra Design Decisions

## Upgrade to Glassfish 4 Design Decisions

## Overall Architecture

## Detailed Design

### Even More Detail

## User Interface Design

## Other Relevant Sections

# Implementation

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

## 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 a file browser that was already part of the programs core functionality. It was tasked that this tree listener be updated to pass the unsorted selected spectra id’s into a thread, this thread was created to handle four pass variables.

1. ArrayList of spectra id’s
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 that’s sole purpose is to create a spectral plot for given id’s. 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 creators (add reference) 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 showing 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 when a huge array of spectra had been 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 which 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.

## Multi Location Spectral Plotting Implementation

### Attaining the location

In order to get the location of the individual spectra first the correct spectra had to 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, 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 a matlab method to return both the latitude and longitude for the spectra with that given id. This matlab method returns a MatLabAdaptedArrayList which contains the locations for any spectra selected with the given id’s.   
This matlab method would have to be run twice once for the latitude and once for the longitude to get the correct location data.

### Getting the correct string for the location

After attaining the location of the given spectra it was then required that it be converted to a string in order to be inserted into a url that would later be passed onto the web browser.   
The chosen way of parsing the correct String was to run javas toString method on the MatLabAdaptedArrayList. This would have to be run twice for both the latitude and longitude.   
This would return the results below.

The output contained the useful information required about the given spectra but the string would have to be altered to remove the brackets that contained the information.  
In order to complete this the java method of substring was called to remove both the first and last parts to each string to get the desired results. To ensure that a latitude and longitude of any length would always be consistent with MatLabs output the string was checked for length before removing the end of the string.   
These strings are then saved in their local variable equivalents of their desired output and are then passed through to a new class called maps processing.

### **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 maps processing class was created and placed in the proc\_modules package of the SPECCHIO application.

To keep follow the design decisions laid out by the 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.

### **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, obviously 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 so 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 to be dealt with and would in turn be called by the thread.

### Implementation Issues

When implementing the location based specta collection there were a series of challenges that had to be overcome to achieve the correct data that would later be used, bulleted below are a series of issues that had to be dealt with.

* Null spectra locations from Matlabs getMetaparameterValues method
* Multiple spectra per toString
* 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 toString method is called on the getMetaparametesValues, this problem has been dealt with through a simple method that checks the string for “[]” which is what would be returned if the location data is null.

#### Multiple spectra per toString

The implementation that already exists within the SPECCHIO application has already checked for current id’s that are selected by the graphical user interface. It was decided that instead of implementing addition checks on identity that a button would be created that’s state would only be set to enabled if only one spectra was selected.

#### 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 application 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.

*Glasshfish updating issues*

*Graphical user interface static implementation issues*

*Returning correct spectra issues*

*Overly complicated threading issues*

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

# Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on ’real users’? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

## Overall Approach to Testing

## Automated Testing

### Unit Tests

### User Interface Testing

### Stress Testing

### Other Types of Testing

## Integration Testing

## User Testing

# Critical Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

* Were the requirements correctly identified?
* Were the design decisions correct?
* Could a more suitable set of tools have been chosen?
* How well did the software meet the needs of those who were expecting to use it?
* How well were any other project aims achieved?
* If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

# Appendices

* 1. Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [5]. The library is released using the Apache License [6]. This library was used without modification.

* 1. Code Samples

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

# 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.