A Toolkit For Reporting On Metagenome Assembly Quality

Final Report for CS39440 Major Project

Author: James Edward Euesden (jee22@aber.ac.uk)
Supervisor: Amanda Clare (afc@aber.ac.uk)

4th March 2016
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in
Computer Science (G401)

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 Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.

- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.

- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

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.

Name ....

Date ....

Acknowledgements

I am grateful to...

I'd like to thank...

Abstract

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

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

1.1   Introduction

The project aim was to create an application that could report on the quality of a metagenome assemblies provided by the user, presenting them with feedback about the contiguous reads contained in their data. The requirements for the project topic were very open, as there are multiple techniques for attempting to report the quality of a single species genome assembly, and while some can be used for metagenome assemblies, it was believed that no one tool covered this area yet with numerous techniques. Likewise, the way in which the results could be presented to the user was not established and open to my own interpretation.

1.2   Background

Before the project began, my knowledge of metagenomics was very limited, close to none. However, I was liked the project title and description and thought it would be an interesting and challenging topic, to learn new domain knowledge, use different technologies and attempt to implement an application where I had to learn from the ground up. On top of this, I find DNA to be an intriguing topic even with my limited knowledge, I was curious to learn more as I worked on this project.

My first step was trying to understand what exactly it was I was expected to produce at the end of the project. As the requirements of the resulting application were so open, it was up to me to research what metagenomics is, what is meant by 'quality' within the subject, how this quality might be found and reported on, what technologies would be appropriate and what quality techniques could be used.

1.2.1   Metagenomics

Metagenomics is the study of environmental samples of genomic data where the contents of the data are potentially unknown and unclear. It can be described as 'Open-ended sequencing of nucleic acids recovered directly from samples without culture or target-specific enrichment or amplification; usually applies to the study of microbial communities.' [7] It can be used in the

findings of what an animal gut may contain, what viruses are within a sample when looking into outbreaks and finding what microbial communities exist in a sample area.

Metagenomics is a hot and interesting topic in the bioinformatics field, and its uses grow as more is learned, but there is the issue of quality, and how a metagenomic sample should be processed. I read articles that attempted to provided ways of analyzing metagenomic data to get the best quality results at the end. [19]

1.2.2   Understanding Quality

Considering the nature of metagenomics and the unknowns, it becomes clear quite quickly that when a sample you may have is run through an assembler in an attempt to create a genome for sequencing, without the proper tools to assess your data you cannot be sure if what you are creating is an actual thing that could exist in nature. The processes of taking a sample through to sequencing with metagenomic data can be very error prone, leading to misassemblies with duplicate or short reads, or combining reads together to make chimeric contiguous reads (contig). [6]

A chimieric contig is an instance where an assembler has put together reads from a sample that it believed were part of the same whole, and yet were in fact of different species/sub-species, and so creates a contig that does not actually exist in nature. It can be understood then that if a user were to sequence this, unless that is the result they wanted, it won't be of any use to them, but without the proper tool how would they know that their assembly data contains chimeric contigs and are not just wasting their time and money?

We can visualize this through the diagram below, where we take a number of reads, put them through an assembler and the output is a number of contigs that the assembler believes it has created correctly, but may in fact be chimeric.

Figure 1.1: A very small example of a number of reads being taken into an assembler and the output of a contig where there may be a chimera.

If we imagine each of the colours representing an individual species (though in reality, there would be no colours, only the characters of data, and potentially hundred to thousands of different species and sub species in a sample) we could, by eye, see how we could extract some interesting genome data, for example taking all of the green in the order they are presented and finding the 'green' genome. As naive as that is, it serves then to demonstrate that if we look at the output of the one contig shown in the figure, we can see how the colours are aligned in a way that they are mixed.

It is possible that the resulting contig is a valid genome that could be found in nature, where the particular sequences in the different coloured sections are shared between each colour 'species' and so this contig could work. However, it is also likely that the different coloured sections are widly different than the natural connecting colours were this contig made up of a single colour (in this figure, all yellow, all blue, all red). This would then be a chimera. This demonstrates that without some quality assessment tools to attempt to report on where these misaligns have occurred, a user may never know if what they have is good quality data or not.

1.2.3   Existing Software

There are a number of tools I discovered in my background research that attempt to aid in the quality assessment of metagenomic data, in particular MEGAN, a 'next-generation metagenomic data, called MEtaGenomeANalyzer', which attempts to do a taxonomical analysis comparison to known reference data [4] and PRINSEQ, a tool which provides 'summary statistics of FASTA (and QUAL) or FASTQ files in tabular and graphical form' [13]. When considering what my own application should do, I looked at the techniques used for PRINSEQ most, as these seemed to match up with what I thought would be useful to a user for my own application, and from discussing the topic with my supervisor I found techniques such as the GC Content distribution could be a good place to start.

It was not just tools for metagenomic quality assessment I looked into. I also found the NCBI database and their BLAST tool [3], and kept in mind these may be useful as I progressed through my applications development, and through reading an article that discussed the advantages of k-mer frequency analysis for quality assessment, I looked into the Jellyfish [8] and BFCounter [9] tools for just this role where I could potentially consume the output of their processing, although they took a step back in my mind while I considered what it was I actually wanted my application to be and worked out the requirements.

1.3   Analysis

After understanding the project topic and problem a little better, I decided upon a number of re-quirements of the application to begin with. Some of these were definite goals, some stretch goals and some future development tasks if I were to finish all else or were to continue the application after the project deadline. I broke the problem down into its two core components steps, the analysis and the report. I felt that the resulting list that can be seen below was enough to work with based on the knowledge I had gained from my background reading and what I thought would be appropriate for the time allotted for the project.

1.3.1   Quality assessment

The quality assessment had to use a number of methods suitable enough to produce some data or statistics that could indicate to the user a measure of quality of their assembly data. For this end, I decided upon a number of objectives:

1.3.1.1   Contig length

'Give the user control over the minimum length a contig should be to be considered.'

This measure would be good in displaying where an assembler could not find any reads to match with the current read and so didn't do anything more with it than output it. However, this would most likely indicate that it is of no use to a user, as a contig the size of a read length is unlikely to contain any useful genome data. Allowing a user to set the minimum length threshold lets them set their own size, be it the known read length size of their data, or a size they think would be appropriate to start seeing some usable data from contigs with length over a particular amount/number of read lengths.

1.3.1.2   Number of unknown characters

'The application should count the number of unknown N characters within a contig.'

When an assembler cannot understand what to do with a character, or a sequence of characters, it may insert an 'N' character. Indicating to a user how many of these exist in a contig, and what percentage of the whole they make up is a helpful indication of whether their data is of good quality or not, with the less or no unknowns the better the quality.

1.3.1.3   GC Content

'Conduct a GC Content percentage analysis in sliding windows of a size set by the user.'

The GC content is the percentage of 'G' (guanine) and 'C' (cytosine) characters within a sequence [15]. When the application takes a contig, it should break the contig up into windows of a size set by the user. If a contig was of length 30,000, they might break it into 100 windows of 300, for example. These window sizes should then have their GC content percentage calculated, and the mean of the entire contigs GC content worked out. Using these values it then becomes possible to detect potential anomalies in the percentages of individual windows that have a percentage value drastically different than the mean.

1.3.1.4   Open Reading Frame Locations

'To confirm if a GC window is outside of the mean by misalign, Open Reading Frame (ORF) Locations should be found where the GC content would naturally be higher.'

This addition to the requirements came later in the projects development as I came to understand GC content and how a window that looks like an area of a misalign may actually not be. We can look at a contiguous read and find protein coding regions through the use of Open Reading Frames [17], where the GC content percentage is often naturally higher than outside of these regions.

By finding the ORF Locations, we can use these in the quality report to match up to windows that may at first seem like anomalies in the contig where it is instead actually a natural occurrence. There is a preexisting tool for this ORF finding functionality by NCBI [16]. While it might not be the case that an ORF Location matches with an out of threshold GC content window, or that

they do match but it is still in fact an anomaly, it was a requirement of the application I wished to include in order to give the user another tool to use for inspecting their data.

1.3.1.5   K-mer frequency analysis

'The application should conduct a k-mer frequency analysis, or use output of k-mer frequency analysis tools.'

Conducting k-mer frequency analysis in window sizes has a similar output as GC content, except we look for the frequency of particular 'k-mer' (where 'k' is a number of how many characters to be considered, e.g. 3mer 'ATG', 4mer 'ATGA'). Through measuring the frequency in windows we could see if there was an even distribution of frequencies across a contig, and if any windows had large changes in the frequencies that could indicate a potentially bad quality contig.

This was set as a stretch requirement, as the process of efficiently conducting k-mer frequency analysis and the research into the possibility of writing my own software to do it or consuming output of another application was expected to take over the time for the project, after my background reading and understanding was completed and the other features previously mentioned implemented.

1.3.1.6   NCBI reference data

'The application should compare the contiguous read data to known reference data in the NCBI database with BLAST.'

Using the known reference data, much like with MEGAN, it would be possible to see if any of the contigs in the user data match to known reference data and so can be considered good quality. This was a stretch requirement though, with the idea more in mind for if the application were to be continued to be developed after the project deadline.

1.3.2   Quality report and data input

1.3.2.1   Read user input in FASTA format

'A user should be able to paste or upload their data in FASTA format.'

The FASTA format is a commonly used format for genomic data. It is 'a text-based format for representing either nucleotide sequences or peptide sequences' [20]. It is formatted in the way that can be seen in figure 1 below. I set myself the task for allowing the upload and paste of data, beginning with pasting the data into a form, as uploading data could add a larger realm of security and data size issues and so I wanted to have the simplest thing possible first.

For the FASTA format my application accepted, I planned for it to begin only taking into account ATGC & N as characters to deal with, where N was 'unknown/unclear' and the other characters were valid. Uppercase and lowercase characters would be preserved in any output display of the application, but capitalization would not taken into account for any of the processes as they did not make a difference.

1.3.2.2 Display a list of the contigs

'A user should be able to see the contiguous reads listed in their FASTA data.'

I decided it would be a good idea to display to the user the list of contigs that their file contained, with small bits of relevant information, such as the size of the contig, the header of it and the previously mentioned 'N' count and percentage. From here a user would be able to select a contig they wish to further inspect for quality issues.

1.3.2.3 User control over GC Content, ORF Location and k-mer frequency analysis parameters

'When conducting a quality inspection, a user should be able to adjust the parameters of the methods based upon their expectations, data input and choice.'

Since the data uploaded by a user would have been assembled from reads of a particular length, the appropriate size of GC content and k-mer frequency windows, minimum size of ORF Location lengths and minim length of contigs should be set by them. It could have been a possibility to attempt to guess what a good size would be for the user (i.e. find the shorted contig in the data and consider that that might be the size of a single read, and suggest length parameters based on that).

1.3.2.4 Visual reports of the quality inspection methods

'A user should be able to see a visual representation of a contigs GC content and any anomalies.'

In order for a user to be shown whether their data is of good quality or not, visualizations would be required that highlight any problem areas, or if areas are not highlighted, at least give them a broad sense of the layout of the results of the quality inspection to allow them to analyze the results from themselves. This should include displays for GC content in windows, ORF Locations broken into individual frames, k-mer frequency analysis in windows and a comparison shown between the GC content and ORF Locations.

I did consider how I could potentially provided a quality confidence factor to a user, or how I might demonstrate to them in a statistical way that their data was of good or bad quality. However, with the amount of methods that could be used, the unclarity of the data itself and the users expectations, I felt that it would be better to instead provide the user with visuals to analyze the results of my application themselves, and I would instead try to highlight problem areas without necessarily saying clearly 'This is/is not a problem area'.
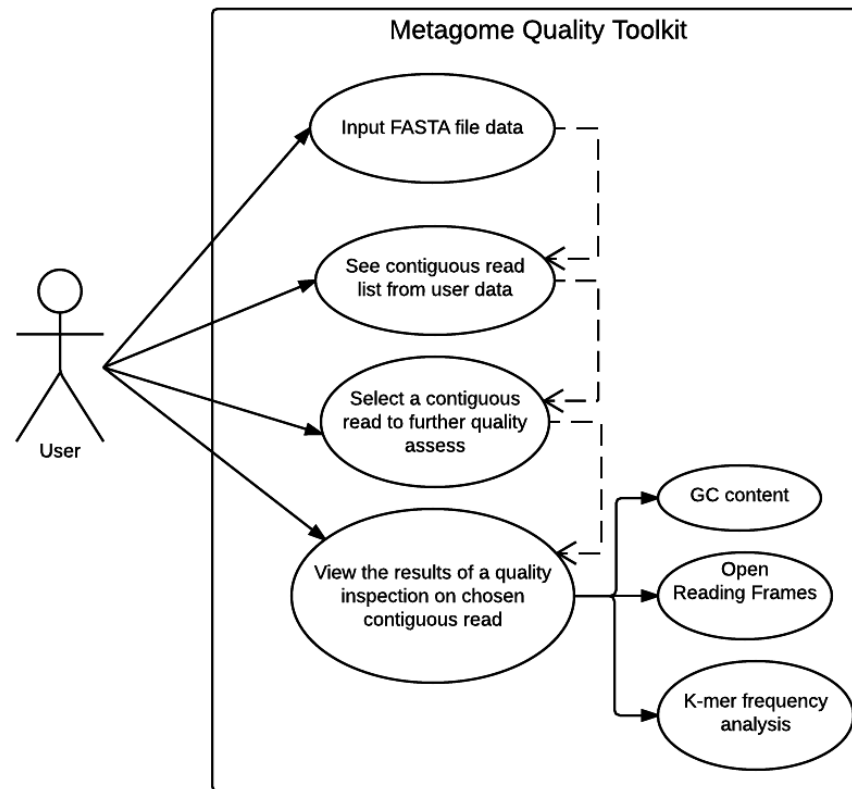
Figure 1.2: A use case diagram demonstrating the expected functionality for a user based on the functional requirements.

1.3.3   Implementation - 3rd Party vs Self-Created

Based on the requirements I had selected, I decided that I wanted to write my own software for each functional component. While solutions for each component exists individually, I felt there would be more benefit to a user if I could produce just my own software to support the functionality over requiring them to user third part software in order to use my application too.

Additionally to this, I wanted the technical challenge of writing my own software, and enabling the application to be maintainable and expanded upon in the future rather than relying on third party support applications in order to process a users input. The process of developing the application from scratch would also give me the opportunity to learn more about the domain and technologies for development than just consuming output of other existing applications, even if achieving the end result would take more time.

While I knew the project would come to a close on development at the end of the semester, I also wanted to envision this as a project that could be taken forward in the future to be further developed and maintained. For this reason it also made sense to develop my own application without consuming output from third party software, excluding the FASTA files from an assembler. If one of these third party applications was no longer available or their output changed, it would render my application unusable or require more future modification to adapt to changes by outside sources.

1.4   Process

While my project could well follow a plan driven approach, such as waterfall or the spiral model, I believed that an agile approach would be suitable for building it. An agile methodology seemed most appropriate due to the unclear requirements (based on my lack of understanding and the open tasks at the start of the project) and would allow me to break down each requirements into stories, and then into tasks and get a thin slice of the application done quickly, and build outwards from there, always ensuring I had some finished product at each functional completion.

I chose to use Scrum for my framework, and used aspect of Extreme Programming for my daily development cycle adapted to a one person project. Considering the nature of my projects requirements, I considered if I should instead use Feature Drive Development, but decided upon Scrum due to my experience with it through my industrial year and in that I could see how I could break the requirements down in to user stories and tasks.

1.4.1   Scrum

Scrum is an agile framework for developing complex products [14]. In brief, it works in iterations (Sprints) where work is broken into user stories and tasks for those stories, there are key members of the team who have designated roles (Engineer, QA, Scrum Master, Product Owner, etc) and where each Sprint has meetings involved in it to help the flow of work: Sprint Planning & Sprint Retrospectives, Sprint Review and Daily Stand-ups. I decided that adopting this framework would help me structure my weekly work and aid me in planning and development for the project.

As a one-person project, a number of the components of Scrum either had to be dropped or adapted, for example, the roles of team members were all adopted by myself rather than having actual team members. I work on a weekly Sprint cycle, where at the beginning of every Tuesday a new Sprint would begin, and I would review the work done and have a Retrospective with myself and plan out the next weeks tasks, often through talking with my supervisor would I understand the Stories a little better to plan them. As for Daily Stand-ups, I was able to carry these out through holding them with my peers, Alex Jollands and Sion Griffiths. Between the three of us we would meet in-person or over Skype on weekday mornings at 10am and discuss our work from the day before, the planned work for the current day and anything we wanted to soundboard off one another.

The Scrum framework had me taking my initial planned out requirements and turn them into Stories with their own tasks. Stories are structured in the way of:

'As a *<type of user>*, I want *<some goal>* so that *<some reason>*.' [11]

Once a user story has been created, I would then consider it and see if I could break it down into a 'narrow slice'; the simplest thinnest possible thing I could do for a feature implementation that would produce results. There is an example of this in Appendix 3 'Examples and Extras' under 'Example User Story Breakdown'. Breaking down the requirements into user stories and then tasks helped me focus on what was important, and gauge what needed to be done and how much effort vs time it would take.

1.4.2   Extreme Programming

For the day to day development outside of the Scrum framework, I decided to adopt a modified Extreme Programming (XP) approach [10]. While I couldn't do all of the XP techniques (pair programming, for example), I strongly took the elements of Test Driven Development (TDD), refactoring, simple design (coding only what needs to be done based on the TDD and refactoring) and continuous integration. I felt that using a combination of Scrum and XP techniques was a suitable process for my application, and would aid me in my design, implementation, testing, meeting the requirements and focusing my motivation.

The motivation for taking the aspects of TDD and Refactoring were that I could build and design my application iteratively, as I understood and refined the requirements and functionality of the application as I went, while still having a usable application from an early stage that could be developed upon in each Sprint.

TDD ensured that the application would be developed in the simplest way possible, and always meet the requirements currently tested for by doing the unit test development up front. Refactoring the work as I went and where needed meant that I would keep my code maintainable while developing in this way. The end result through selecting this methodology was to aim for a robust, maintainable, well-design application with test coverage for all functionality and programmed efficiently through proper time management. Using continuous integration meant that I could be sure that whenever I completed a new bit of code, it would be run against all existing tests to check that nothing had broken and was impacting previously working functionality.

1.4.3   Pomodoro Technique

On the matter of time management during development time, I found it useful to break my work into small cycles using the Pomodoro technique [1] to increase my focus and productivity. The aim of the pomodoro technique is to spend 25 minutes with full focus on the work, and then spend 5 minutes to stop and have a break. They are also useful for keeping track of work done and visually seeing how much time has been invested into the project for the day. To record my pomodoros I used `http://www.tomato.es/`.

Due to the short length of each pomodoro, it was also seen as an effective method for squeezing in work where there was time. By quantifying time into small slots, it made it possible to consider how much time I might have between other events in the day and work out where it could be possible to put in a single pomodoros worth of work. This is instead of perhaps not working because I felt I did not have the time, or working and eating into other daily activities if I didn't block out the time and be aware of when to stop.

1.4.4   Project blog

While developing my application I kept a blog about my project, any milestones I reached or interesting sections. While I did not designate a particular time to blog, I believed it would be a useful tool in reflecting on the work done when writing this report, something for my supervisor to be able to check upon my progress between meetings and give me the time to reflect on the work done and any upcoming tasks to help me mentally process where the project was in its life cycle against the planned objects. The blog may be seen at `http://users.aber.ac.uk/`

`jee22/wordpress/.`

Chapter 2

Design

2.1 Overall Architecture

The design of my application was evolutionary as I was developing it using XP in an agile way. This meant that each step had an initial design and was built upon every time a new feature was added, being refactored along the way. There were a number of prototypes made during the course of the project's development, and this chapter will discuss the design of the end product, and go into detail where the design may have been previously different.

The resulting application evolved into a web service, with a Model View Controller (MVC) framework and resources structured for maintainability. Following XPs guidelines, at each step of the way the applications design was made as such to be the simplest yet most maintainable it could be through refactoring, cutting down on duplicate code, structured logically and choosing smart data structures and Objects to represent aspects of the application and quality results.

2.1.1 Choice of technologies

At the beginning of the application, I felt that the application should be programmed in a language that would be able to be ported across multiple platforms for used by anybody who wished to use it. Additionally to this, I wanted to have ways of representing the components of my application as Objects and would need some way of presenting a UI to the user, even if my initial application would only output command line results, in order to follow the XP value of Simplicity (YAGNI - "You Ain't Gonna Need It", until you do).

For this reason, I selected to develop my application using Java to begin with as it filled the criteria of being Object Oriented and was portable through using the JVM on different operating systems. I resolved that I would select a UI package to present the report of my results once some of the core functionality had been implemented. I considered instead using Ruby or C++, but due to my familiarity with Java I believed it would be a better choice to stick with what I knew. At this point, I did not consider having the application as a web service, and so using Ruby with Rails was not something I had considered.

As the application developed and I started reading in files and outputting results from the GC Content process, I found that I was having a hard time finding a quickly usable GUI I could work with for Java to display the results in a way that I wanted. It was at this point I started considering

a technology change to Ruby on Rails until Sion Griffiths, a peer of mine, suggested I could keep my application in Java and turn it into a web service using Spring Boot [12]. This seemed like the perfect solution to my problem, allowing me to generate my charts and results using HTML5, JavaScript and HTML5 Canvas, along with JavaScript libraries such as Plotly.js.

The conversion to Java Spring Boot was not a painful one, and only involved setting up a new Maven project and declaring it to run as a Spring Boot application in the pom.xml file, then copying over my previous code and structure into the new project, from there I could deploy the application as a web server using Tomcat and was back to my previous position of working out a UI. Thankfully, there is a package called Thymeleaf [18] that works with Spring Boot to allow access to Objects from within the Model of the Java code (placed there by the Controller) using the View dynamically as the View is generated through HTML templates and fragments.

```
model.addAttribute("contiglist", contigList);
model.addAttribute("discardedcontigcount", contigResult.getDiscardedContigCount());
model.addAttribute("contiguousread", new ContiguousRead());
return "list";
```

Figure 2.1: Adding an object to the Model via the Controller to be accessed in the View. The View here is 'list', which is a Thymeleaf template that will dynamically build the page using the data from the Model added here.

On top of using Thymeleaf for accessing data put into the Model, it has the additional benefit of using fragments that can be imported into different HTML templates. This design choice made it so I was able to cut down on writing duplicates of code. For example, I wrote the header and footer of the UI design in a Thymeleaf HTML fragment, and then on each page just need to call one line in order to import it into that page, rather than built the entire thing again. An extra benefit to this is that it allowed me to keep the HTML templates clean and easier to maintain by reducing their size and separating out different aspects of the UI. Having the header, footer and a number of forms in Thymeleaf HTML templates meant that these sections could be edited without impacting any page they are imported into.

```
<tr>
    <td>
        <div th:replace="fragments/qualityparameters :: inspectbox"></div>
    </td>
</tr>
```

Figure 2.2: Including a thymeleaf fragment in a page. You can see how the call is made from one 'th:replace' with the name of the html fragment file and then the fragment to be included from that file, in this case 'inspectbox'.
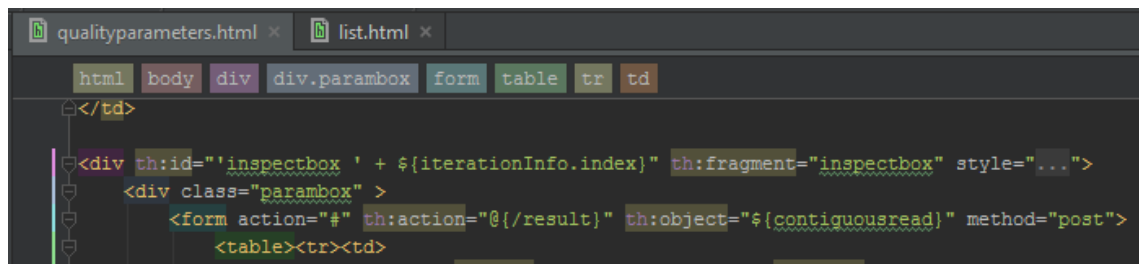
Figure 2.3: Including a thymeleaf fragment in a page. This is the declaration of the fragment being included in the previous figure, declared as such through 'th:fragment'.

For the GUI itself, I elected to use Plotly.js [5] for representing my GC content charts for the level of detail and control it gives a user over inspecting charts, large and small, that worked no matter what GC window size the user selected. For the ORF Locations frames charts I wrote my own code using JavaScript and displayed it with HTML5 Canvas, as it allowed me control over what should be shown depending what a user clicks upon and I could work with the data from the Model in ways I saw fit.

2.1.2 MVC Framework

As I was building an application with a GUI, the application was built using a Model View Controller framework design, separating out the components into their different types. This helped the separation of code and responsibilities. The class diagrams presented in this chapter are based on the final product design and based on a page per page view from the UI and what classes are used on the HTML request of that page.

2.1.2.1 Model

The Model was designed to contain the data and methods for processing a users input. This contains the data structures and objects for handling a users data as they traverse from page to page of the view. I designed the Model in a way that certain objects would be Bean objects (classes with getters and setters for their properties) that could be accessed by the View in the way that Thymeleaf required.

Class Diagram for 'Welcome' Page

Figure 2.4: Class diagram for the 'Welcome Page', and what classes are used upon a Request for the page.
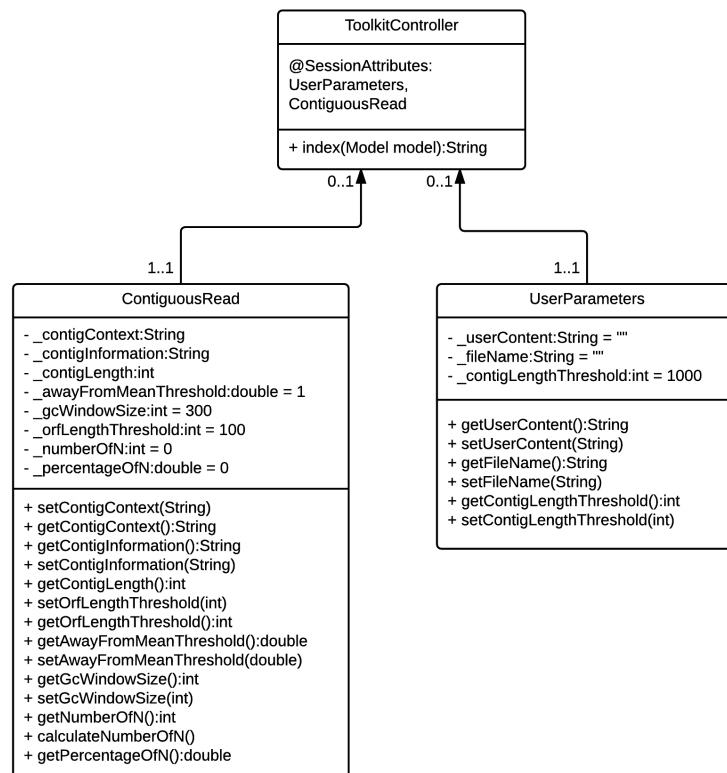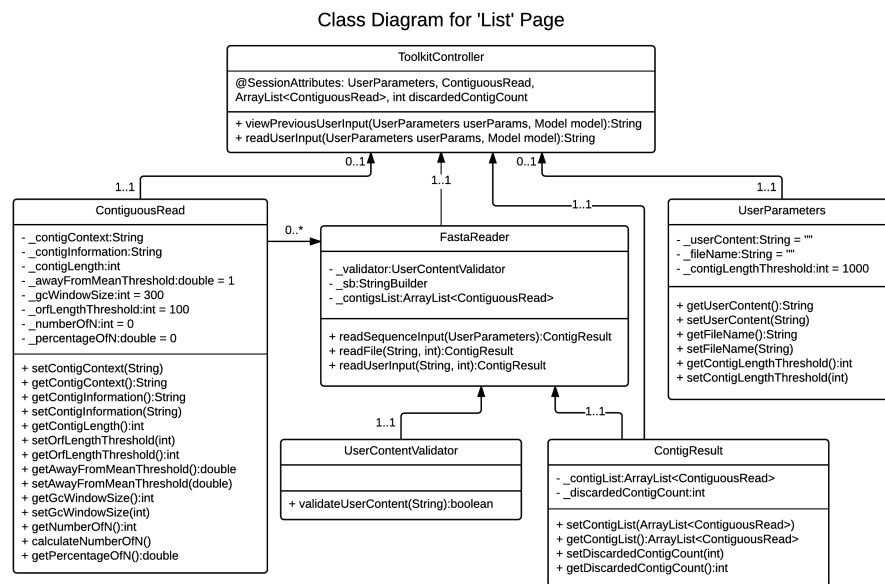
Class Diagram for 'List' Page

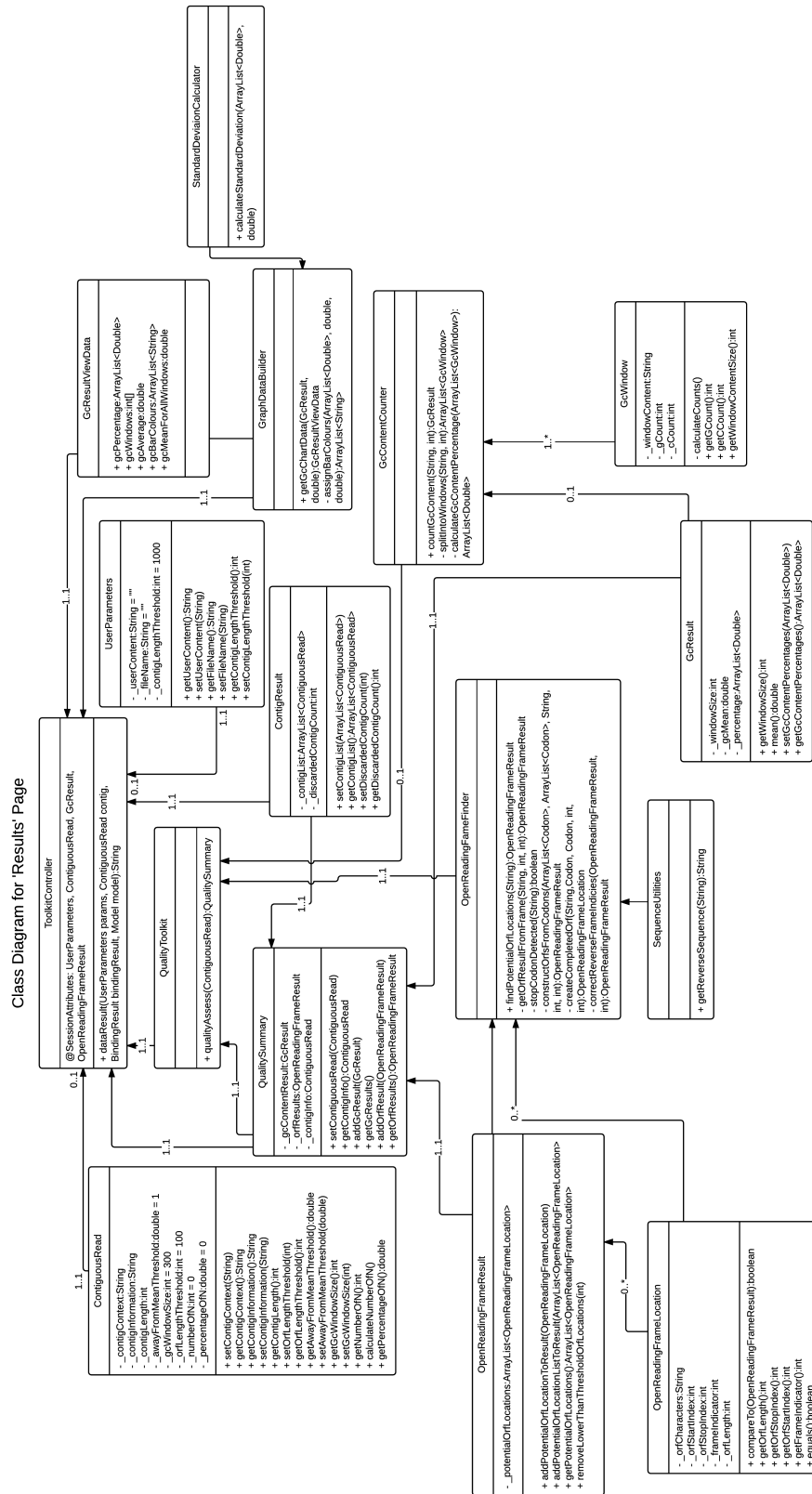Figure 2.5: Class diagram for the 'List Page', and what classes are used upon a Request for the page.

Figure 2.6: Class diagram for the 'Results Page', and what classes are used upon a Request for the page.

The structure evolved over time due to the nature of XP and using an agile methodology. Even so, the result was that there was a logical sense to the structure of classes, methods belong in the class they live in, some utility classes exist for carrying out operations that don't necessarily need to be within a class that needs that utility and this also left the code easier to maintain and separated. By using utility classes and methods, they can be called by other classes in the future, and the class that uses them doesn't need to know about the implementation internals of the methods they use, only the results.

2.1.2.2   View

The View of the application is where the GUI is presented to the user to be seen in their browser. Designing the application to use the browser was chosen to allow anyone to be able to use the application as long as they are using a modern browser. It is using HTML, HTML5 Canvas and JavaScript, dynamically built using Thymeleaf templates. If any of the template does not process or there is an issue with the users data they are presented with an error page and told to return to the menu and try their task again.

The data for the View comes from the Model, passed by the Controller. In line with keeping responsibilities separated, the Model does not care about how the View displays the data, and the View does not care about the content of the Model data it is passed as long as the data types it expects are present in the Model from the Controller. For example, the ORF Location data is sent to the View as an OpenReadingFrameResult Object, that has a list of OpenReadingFrameLocations and other data about the process. It is then the responsibility of the View to use this data in displaying it in HTML5 Canvas with JavaScript. No additional processing or modifications are carried out on the data at this point, the View just picks up the data and places it on the Canvas element where it expects it should go, based on the content of the data, e.g. an OpenReadingFrameLocation with the 'frameIndicator' set as 0 would indicate to the View that it should be placed on Frame 1.

This type of responsibility separation is consistent throughout the application, with one except. There is a GcResultDataView, that is the GcResult data processed into an easy to handle class. This is to make the process for the View far faster than having to get the View to carry out some processing to determine the GcWindows that are out of the mean threshold and additional results to display on the bar chart for the GcResult. While it would be possible to alter the application in order to leave this up to the View, I felt that it would be better to instead serve the View with the window data it needed to put on the chart and the colour codes for each of those window data bars.

2.1.2.3   Controller

The Controller serves as the master for how data flows through the application, making calls to methods from the Model to process the users data, receiving requests from the user via the View and HTML Requests and handling what data is for use in the View. The user may send HTML Requests to the different pages of the application and will receive a response based on the '@RequestMappings' of the Controller.

For example, a GET on the 'list' page will return a page expecting the user to have already submitted some data that has been turned into a list of contigs. A POST on the 'list' page however will be expecting that this is a new set of data and process the data submitted into a list of contigs and put them into the users session attributes.

Through every page, a user carries a Session with them and particular session attributes are expected at certain points in @SessionAttributes. For example, a user must have UserParameters and a ContiguousRead in their session parameters in order to be able to view the List and Results page. If they have not visited the Welcome page and submitted data, however, it is not possible for them to have these attributes, and so they are locked out of those pages. This is good behavior as the user should not be able to try and view results for data they have not submitted.

2.1.3   Naming Conventions

I followed a naming convention of attempting to name classes, methods, fields and variables in such a way that a reader could understand what the function of that particular thing was just from the name alone. Private field names all begin with an underscore, e.g. '_privateField' and everything is written in camel case, starting with a lowercase character, e.g. 'thisWouldBeAMethodName()'.

2.1.4   User Input

Based on the background reading, the decided file format to accept was FASTA format. The design allows for user uploading and user pasting of files in the Model methods. However, in the end I decided to keep the design just to handling pasting of data. This was based on a time constraint and issues with file upload limits, security and the need to test. While the functionality has been left in the code, it has been left as a 'if I were to continue' functionality to be expanded upon if I had more time to work on file uploading as a priority over different techniques for the quality assessment.

User data is also not kept by the application, it is stored in a user session that expires once they leave the page. This is handled by Java Spring, and set as one of the @SessionAttributes. If the user does not have their session variables, they are presented with the Error page, so they cannot try and access areas of the application/web service where they currently do not have access or the data to do so. This means that the data they have is their own, and not retained by the application or possible for other users to access.

2.1.5   Directory Structure

The directory of the applications structure was designed such that it would reflect the MVC aspect of the application, and make finding resources and particular Classes easier for a reader. The design makes the application easier to maintain in the future and is another way of helping to enforce separation of responsibilities between classes and resource types. In the figure below you can see the way that the directory structure is designed to back this up.
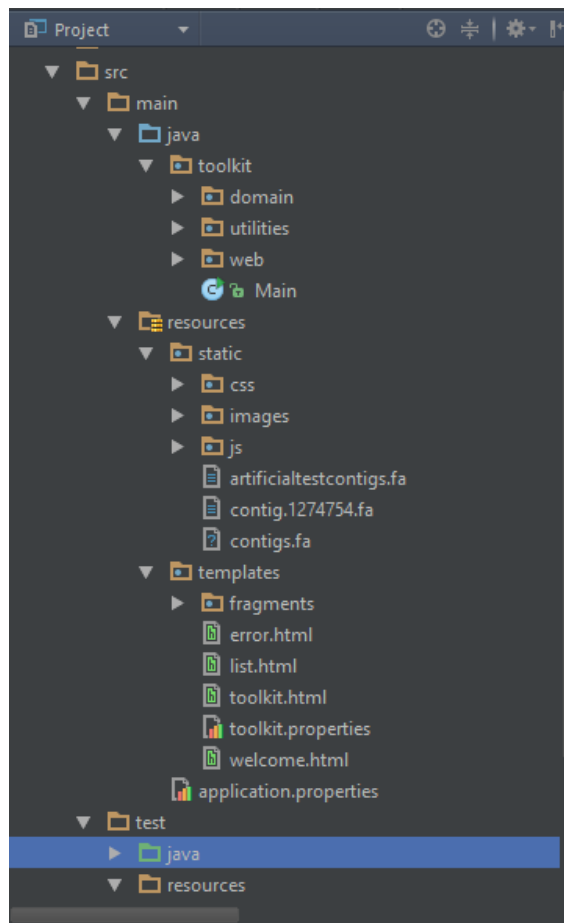
Figure 2.7: The layout of the directory structure for the application.

Within the 'src/main/java' folder, there are the sub folders 'domain', 'utilities' and 'web'. Separating the classes out this way helps with the previously mentioned separation of responsibilities. Domain contains the classes for the domain objects themselves and the data and processing (the Model, so GcResult, OpenReadingFrameLocation, etc), Utilities contains utility classes for use by any of the other classes and to allow their functionality to be independent of any domain class and Web contains the Controller classes (ToolkitController, ErrorController).

2.1.6 QualitySummary and Results

In the application, the quality assessment results are returned in a QualitySummary object. This QualitySummary contains references to GcResult and OpenReadingFrameResult. The is the option to have these two implement an interface that might be called QualityResults, and have just an array of QualityResults. This would allow us to add any type of result without needing to know what is in there. However, I chose not to do this as considering the way in which we serve data to the View it seemed okay to be able to have the View called results from the model directly, and through XP developing this any further would go against YAGNI.

I am aware that in the future if we allowed a user to select what type of processing they wish to use and as more techniques for quality assement are added, this technique would be very useful

to implement. It would be possible to do this and in the View have it check for the existence of the expected results, or check if they were empty/null, and then only dynamically include fragments of results that are present.

2.2 User Interface

Framework mock ups of the user interface. Logo for kicks.

- Input form, why is it this way? - List - Displaying each contig, why, framework, drop downs - Paramters - Why did I pick these Paramters, what use are these for the user? Why GC content windows and what size is reasonable? Can demonstrate domain knowledge about the choice of parameters. Up to the user, allowing them to put in stupid numbers, but their results are their own. - GC Chart - Why does it look like this - Choice of standard deviation and mean, why? What use does this actually have in showing the user error areas? What about just visually seeing the data? Show the screenshots of areas out of GC content threshold, show the areas where obvious chimera (50/50 example) - ORF Location - Why did I develop it to look like this? Representing each of the 6 frames, demonstrate the reverse frames are displayed in the correct direction. How did I achieve this and why display it this way and the inner detail? Other approaches? What about the NCBIs version? Ability to click. Also the list? Why have both? They're both useful to have, as a user may want to look at the shortest ORF Location but not be sure where it is even when they read what frame it is. Likewise they may want to see where an ORF Location lies in the list organized into longest to shortest. Why organized into length. Useful to the user to get an idea to further inspect. Why 6 different canvases for the frames? Easier to modify and display on the page. - Superframe - What is it and why did I consider it. Showing overlaps and where the Red areas may be areas of errors and what to look for. The explanation of it, why explain it this way?

2.3 Support Tools

For writing the code, I used JetBrains IntelliJ Community Edition IDE. This included writing all the code be it Java, HTML, CSS, Javascript, ThymeLeaf and any additional properties for setting up Spring Boot. This was very useful for text highlighting, debugging the code, and tied into my version control, using Git and hosted on GitHub. I also used `http://www.tomato.es/` for counting my pomodoros.

2.3.1 Version Control and Continuous Integration

I used version control to keep a repository of my code in case of losing data in event of hardware crash or software corruption. For this I selected to use GitHub as I already had a repository on there and Git is supported in the IntelliJ IDE I was using. This can be found at `https://github.com/coderghast`.
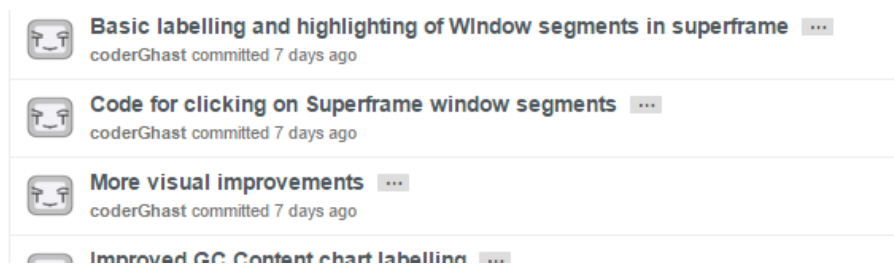
Figure 2.8: A number of the commits made to my major project repository.

In addition, every time I checked in, I used continuous integration with CodeShip [2], allowing me to receive e-mails any time I made a commit to my repository that didn not pass the tests written. This was very helpful in discovering issues with failing builds when checking in a change and forgetting to update tests or breaking previously written code as the design changed.

Figure 2.9: Examples of the continuous integration on CodeShip running my tests every time I committed to my repository.

Chapter 3

Implementation

3.0.2 Development Environment

What did I develop it in? My laptop, IntelliJ, Windows 10, PC specs

3.0.3 Features

3.0.3.1 Reading User Input

Implementation What did I do to implement this? Did I do one thing and find it worked for everything? Did I change my implementation. Think about changing from reading in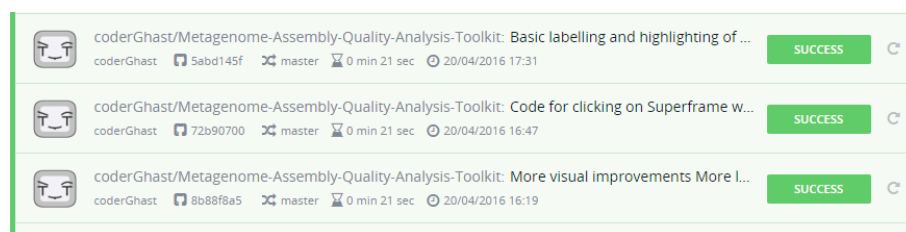 a file to reading user input. Started with a couple of parameters, to big list of parameters, to the UserParameters class (could mention this in Design section too). **Issues** What issues did I encounter, if any? The file names, file sizes being large, formatting of the file and escape characters in user input, symbols in the file? ATGC and default anything else to N. Originally during the reading of the input, would then do the quality assess per contig here, instead moved it to pass back a list of contigs and quality asses on request by controller. Better way of data flow and less class responsibility

3.0.3.2 Counting GC Content & Percentage

3.0.3.3 Displaying GC Content percentage

Prototyped with Plotly. Issues getting data to actually display as I was unfamiliar with plotly and thymeleaf getting data from Objects. Getting the Mean line to show, and setting the colours from within the application - Is this bad? Should probably be done in the View and the model just outputs the data. Decided instead to calculate this in the model itself and output the colours directly. Can still be consumed by parts of the view, and later if a change happens the data is still output

3.0.3.4 Finding Open Reading Frames

Understanding what they were, how they are built. How do I find them in strings? Issues - Finding the longest, how to know when to stop. Originally had the wrong idea and believed all Stop

Codons would be included up until the next Start Codon. Developed against this until feedback from Amanda made me realize the problem was far simpler. Lost developer time, but functionality to fix didn't take too much time. How do we now find ORFs, and store their data? What does it look like, zipping Start and Stop Codons together and getting substrings from within the full contig to create the ORF Location Object.

3.0.3.5   Displaying ORF Locations

Implementation through each ORF Location knowing where it starts and ends, the characters between in a list. Each has a frame indicator, allows them to be kept in a single array. Javascript goes through the list and uses HTML5 canvas to draw them out in 6 different canvases. Working on finding user clicks within the canvas objects, getting data from Thymeleaf to display it from the Objects. Formatting the display and using colour to highlight the start and stop codons. Issues included finding the clicks, highlighting both the menu and the item on canvas. Took some time to get the formatting in the table correct

3.0.3.6   Superframe Comparisson

Making the view of superframe to help the user compare the ORF and GC content. Able to be built upon with k-mer frequency analysis and other techniques as added. The overall 'compare all' chart. Borrowed code from the ORF canvases. Breaking things up into windows from the GC content.

Not really any issues, just trying to find the best way to display it to the user and convey any areas where things don't line up properly. Would like to work on this more and develop a system to detect where the issues really are. Need more techniques.

3.0.3.7   User Interface Organisation

User input section originally having all parameters was messy, only dealing with one contig, then multiple contigs, having parameters appear underneath any contig you wish to inspect, having all charts on the toolbox page displayed in one long list looked ugly, implemented tabs from (credit to the guy here).

3.0.4   Implementation Review

Found that GC Content percentage and ORF Location finding took a lot longer than expected, especially considering that I implemented an additional part of ORF Finding that wasn't needed, and got refactored 3 times until I reached that point. Delayed me to the point I felt I wouldn't have time to implement k-mer frequency analysis. User interface design and programming also took a lot longer. Wanted to present the information in a clean and useful way, without cluttering the page. Getting used to Thymeleaf and how it took the data from the Objects took a while to get used to. Felt that my implementation process was acceptable but issues slowed me down significantly. Everything else I felt I would implement got implemented, but the Superframe needs additional consideration. A lot of the quality assessment is still on the user to look at the produced reports and

their content to consider whether they themselves feel their assembly file is good or not through factors highlighted by my application.

Chapter 4

Testing

4.1   Overall Approach to Testing

Using TDD and refactoring, only did as much development as was required to make tests pass based upon my requriements for each bit of functionality. Also developed test files, both completely artifical to see expected output results, and using real assembly data that I then combined and modified together to result in artifical chimeras to see if I could detect them by looking at the output of my charts.

4.2   Automated Testing

4.2.1   Unit Tests

Used a number of unit tests, writing tests before writing any functional code. Did not write tests for setters and getters, but wrote tests for expected functionality. Made test cases for things such as the ORF Finding (show screenshot of test and tests passing).

4.2.2   Integration Testing

Testing of reading in files. Just asserting the data is as expected, and what happens with blank files, or erronous data files. Does the system handle it gracefully? (Show examples of those test files and what happens with bad files)

4.2.3   User Interface Testing

Tested to see that it worked using Chrome's Developer Tools utility, testing in Chrome, Firefox and Microsoft Edge. (screenshots of how each page looks for each, including results and charts) Testing via a test table (Provided test table) - Tested files with known content and expected results to see response. Tested by viewing what I expected to happen, backing this up with the test table e.g. Click to view information, expected graph results based on artifical test files

4.2.4 Stress Test

Set goal of dealing with large files. Tested with large files to see if the system could process them on my laptop. Could be expanded to deal with much larger files on a system that could handle larger data files with more memory. For my laptop though, dealt with the files fine, even if some took some time to process due to my laptops processing speed. Pasting large amounts of data in a single contig file - Result and time to process file and then inspection time Pasting large amounts of data spread across multiple contigs - Result and time to process file and then inspection time

Chapter 5

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

Appendix A

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 clients 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 [**?**]. The library is released using the Apache License [**?**]. This library was used without modification.

Appendix B

Ethics Submission

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.

Appendix C

Examples and Extras

3.1   Example User Story Breakdown

NOTE: This break down is from my blog entry at `http://users.aber.ac.uk/jee22/wordpress/?p=147`.

```
As a researcher
I want to get a report on the quality of my metagenome
So that I know whether it is of good or bad quality
```

Okay, super high level. This can be broken down into:

```
As a researcher
I want to get a report on the GC content of my metagenome
So that I can see where there might be inconsistencies
```

So, that could be explained better (i.e. what are inconsistencies? Areas where there might be a split/chimera, or just gene encoding regions and completely natural).

```
As a researcher
I want descriptions of the GC content of my metagenome
So that I can pinpoint areas of interest
```

Perhaps a better way of making a story for GC content in this instance. What about the report?

```
As a researcher
I want a textual and graphical description of my metagenome quality
So that I can see and understand where there might be quality issues
```

Again, quite high level, but not too bad. This could be broken down further.

```
As a researcher
I want a graph plotted to show me the GC content in my metagenome
So I can visualise the distribution of GC content to better understand my me
```

From some of these, further tasks can be broken down, so, lets take one and do that with the last story I defined. I suppose, before we can do that though, since we dont have an application developed, we might need some initial setup stories.

```
As a researcher
I want an application to read in my metagenome assembly
So I can see it outside of the FASTA file
```

Maybe thats pushing it a little. Theres not really much to be gained from this in business value, but, as far as development goes it can give us some nice little tasks:

```
Read in FASTA file
Output display of metagenome visually for researcher to understand
```

Thats just two simple tasks. Read in a file type, and with the contents, display it. It might not be much, but its a start where we can say to a hypothetical researcher Okay, weve taken your file, and we can show you that your metagenome looks like this. Theres no processing done to it, but you can see how with this visualisation, theres the room for labelling and noting the interesting points later. What do you think?

This story with others could then help us propose a Sprint Goal, as below:

Sprint Goal: To display a metagenome in an application after reading in a FASTA file, with the look at implementing GC Content counting should time allow.

Annotated Bibliography

[1] F. Cirillo, *The pomodoro technique*. Lulu, 2009, 1445219948.

> The pomodoro technique, used during the development of my application for efficient time management.

[2] Codeship Team, "Continuous Delivery with Codeship: Fast, secure and fully customizable." [Online] Available: https://codeship.com/, 2016, [Accessed on: 27 April 2016].

> CodeShip, the web service I used to help provide me with contiurous integration for my project, connected to my GitHub repository to be run every time I checked in to the repository. This was integral to developing my application as it highlighted any time that I broke my build through forgetting to modify tests when content was updated, or when I broke parts of the application and so old tests would fail.

[3] N. C. B. I. R. Coordinators, "Database resources of the National Center for Biotechnology Information," *Nucleic Acids Research*, vol. 41, no. D1, pp. D8–D20, Jan. 2013.

> The NCBI database and resources was extremely useful to me while developing my application. It gave me access to papers and research articles, data sets for downloading sequences to play with and explore and tools for techniques I wanted to implement in my own application. It's safe to say that without the existence of the NCBI resources, I would have understood a lot less about my project while beginning my development.

[4] D. Huson and S. Mitra, "Introduction to the Analysis of Environmental Sequences: Metagenomics with MEGAN," in *Evolutionary Genomics*, ser. Methods in Molecular Biology, M. Anisimova, Ed. Humana Press, 2012, vol. 856, pp. 415–429.

> A useful chapter in reading about a tool that has its own attempt at analysing metagenomic data. While the tool does not look at the quality, it looks at what the user provides and indicates to them if anything they have is already existing in reference data, and how their data compares to similar sequences already known. This could be considered as a quality measure, as if a sequence can be found to match a reference sequence, we know it is most likely not chimeric.

[5] P. T. Inc. (2015) Collaborative data science. Montral, QC. [Online]. Available: https://plot.ly

[6] V. Kunin, A. Copeland, A. Lapidus, K. Mavromatis, and P. Hugenholtz, "A Bioinformatician's Guide to Metagenomics," *Microbiology and Molecular Biology Reviews*, vol. 72, no. 4, pp. 557–578, Dec. 2008.

This paper was invaluable to me at the beginning of the project, understanding what metagenomics is, and how the quality of an assembly can be full of errors, and why. While much of the paper was not necessarily relevant to my project as it deals with the full process of working with a metagenomic sample, for my background reading it was very useful.

[7] N. J. Loman, C. Constantinidou, M. Christner, H. Rohde, J. Z. M. Chan, J. Quick, J. C. Weir, C. Quince, G. P. Smith, J. R. Betley, M. Aepfelbacher, and M. J. Pallen, "A Culture-Independent Sequence-Based Metagenomics Approach to the Investigation of an Outbreak of Shiga-Toxigenic Escherichia coli O104:H4," *JAMA*, vol. 309, no. 14, pp. 1502+, Apr. 2013.

This paper was useful in understanding what the study of metagenomics can be used for, and so helped me better understand what metagenomics is and how it could be useful to have a tool for quality control.

[8] G. Marçais and C. Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinformatics*, vol. 27, no. 6, pp. 764–770, Mar. 2011.

Jellyfish, a tool for k-mer frequency anaylsis. A tool I looked into briefly at the beginning of my project when considering what techniques to use for my application.

[9] P. Melsted and J. K. Pritchard, "Efficient counting of k-mers in DNA sequences using a bloom filter," *BMC Bioinformatics*, vol. 12, no. 1, pp. 333+, 2011.

BFCounter, for k-mer frequency analysis. A tool I considered when thinking about the requirements and techniques involved in the development of my application.

[10] Mike Cohn, "Extreme Programming: A Gentle Introduction." [Online] Available: http://www.extremeprogramming.org/, 2016, [Accessed on: 26 April 2016].

The description of Extreme Programming (XP) that helped me understand the XP methodology and adapt it for my own use in this one person project.

[11] ——, "User Stories and User Story Examples by Mike Cohn," [Online] Available: https://www.mountaingoatsoftware.com/agile/user-stories, 2016, [Accessed on: 26 April 2016].

An explanation of how a User Story should look, when utilizing them to plan tasks for use with implementing functions of my application.

[12] Phillip Webb, Dave Syer, Josh Long, Stphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sbastien Deleuze, "spring-projects/spring-boot," [Online] Available: https://github.com/spring-projects/spring-bootl, 2016, [Accessed on: 27 April 2016].

The GitHub repository of Spring Boot and documentation for using Spring Boot. Used by me in order to build my application as a web service with Java using Spring.

[13] R. Schmieder and R. Edwards, "Quality control and preprocessing of metagenomic datasets," *Bioinformatics*, vol. 27, no. 6, pp. btr026–864, Jan. 2011.

A tool that does what my project topic describes, with more understanding than I had at the start of my project. An interesting look at what other software is out there to do what my application also wanted to do and was a baseline for me analysising what sort of techniques I may wish to use to give the user of my own application a report on the quality of their assembly.

[14] K. Schwaber and J. Sutherland, "The Scrum guide," 2001.

The Scrum Guide, useful for knowing how to use the Scrum framework as part of my projects lifecycle model.

[15] J. C. Segen, *The dictionary of modern medicine*.   Parthenon Pub. Group, 1992, 1850703213.

For the reference of what GC Content is. The most referenced link seen else where online (e.g. from Wikipedia) is to a web link that no longer exists. This reference backs up the description of GC Content, and is how I understand what GC Content is outside of Wikipedia's reference to the missing web link.

[16] Tatiana Tatusov and Roman Tatusov, "ORF Finder," [Online] Available: http://www.ncbi. nlm.nih.gov/projects/gorf/, 2016, [Accessed on: 25 April 2016].

A tool from NCBI for finding Open Reading Frames within a pasted sequences. I liked using this to compare my own ORF results against and get an idea of how I might want to display my own results in the quality report too. It was also nice to use for helping me understand what an Open Reading Frame actually is.

[17] The Board of Regents of the University of Wisconsin System, "Translation and Open Reading Frames," [Online] Available: http://bioweb.uwlax.edu/GenWeb/Molecular/Seq_Anal/ Translation/translation.html, 2008, [Accessed on: 25 April 2016].

I started to understand what an Open Reading Frame was from reading this web page. It helped me get a grasp of what ORFs were, how they were constructed and what I should do to write code for my own application in how to find ORF Locations.

[18] The Thymeleaf Team, "thymeleaf/thymeleaf," [Online] Available: https://github.com/ thymeleaf/thymeleaf, 2016, [Accessed on: 27 April 2016].

The GitHub repository of Thymeleaf, for use with Java Spring Boot. Invaluable for me when accessing objects from the Model via the View of my web service.

[19] T. Thomas, J. Gilbert, and F. Meyer, "Metagenomics - a guide from sampling to data analysis." *Microbial informatics and experimentation*, vol. 2, no. 1, pp. 3+, 2012.

A useful article in understanding what metagenomic study was. It was helpful to read as I tried to understand metagenomics at the beginning of the project.

[20] Zhang Lab, "FASTA format," [Online] Available: http://zhanglab.ccmb.med.umich.edu/ FASTA/, 2016, [Accessed on: 25 April 2016].

A description and example of what a FASTA file is and what they do. Gives a nice example of a fake FASTA file and helped me know what to expect with the file format when developing my application.