Biocomputing II Project Reflective Essay

Michael Ellis

1 Approach to the project

1.1 Interaction with the team

The first element of the project we decided upon was how our team would communicate throughout. I proposed the team collaboration tool "Slack", of which I had zero experience but had heard good things. This allowed us to share code snippets with each other and to split our conversations into different channels depending on which tier of the project we wanted to discuss. We were generally very satisfied with this decision which was complemented with biweekly face-to-face meeting in the early stages.

The second—equally vital—element settled in our first meeting was which tier we would each work on. When considering how to divide the project: Elizabeth had found the python elements of the course particularly challenging relative to the web page programming; I had escpecially enjoyed python and wanted to gain more experience with it; and Aine was happy to work on any aspect of the project. This left us with the following split:

- Aine:
  - Data parsing
  - Database population
- Michael:
  - Data access
  - Business logic
- Elizabeth:
  - Front end

1.2 Overall project requirements

As a group we very much used the specifications in the project brief as the minimum requirements of our work and then left it up to individuals to decide how far they wanted to take their contribution.

1.3 Requirements for my contribution

In determining the requirements for my contribution I used MoSCoW analysis, placing each possible requirement into one of four categories: what the project *Must have*; what it *Should have*; what it *Could have*; and what it *Won't have*. For my main contribution—the data access and business logic—I decided on the following:

Must have

- SQL queries to access database to retrieve DNA sequence and coding locations for a particular gene
- Python wrappers for SQL
- Python functions to:
  - Output the coding region based on location of the coding region and whole sequence
  - Count the frequencies of codons
  - Identify enzyme cutting locations
  - Label an enzyme good or bad based on its cutting locations relative to coding region
  - Output all the relevent information in table form for use in the web page

Should have

- Function to compare the codon frequency in the gene to the codon frequency in the whole chromosome and produce a p-value and an adjusted p-value with a Bonferroni correction

Could have

- Function to parse chromosome location to facilitate searching by location - though this could be done by the person responsible for the front end

Won't have

- Functions to create rankings of genes by the rate of occurence of selected codons

My contribution was intended to be limited to data access and business logic and perhaps writing some functions to be used in the cgi-script. Unfortunately, however, halfway through the project, Elizabeth had to drop out of the masters course leaving us with a large section of the project to build from scratch. We decided it made the most sense for me to create the web page and cgi script since I was already working in python and I was closer to the front end in the development process. For this I aimed to do the bare minimum to create a functional webpage that met the requirements in the brief.

2 Performance of the development cycle

Within the group, I would say the development cycle generally didn't work that well. It was extremely difficult to deal with losing a team member halfway through the development and it has prevented us producing a website up to the standard that we felt we were capable of. However, having said that, the development cycle worked well between the middle layer and database layer. We were able to test the layers together relatively early on which allowed us to discover oddities of the data that had to be incorporated into the design of the middle layer. These included the existence of 'n's within DNA sequences as well as partial DNA sequences with a length that was not a multiple of 3 and thus could not be represented by an amino acid sequence.

One aspect that had a reasonably significant impact on the development cycle was assigning the data access tier to the middle layer. This allowed for me to have a very clear idea of the API between the data access tier and the business logic tier but it caused problems when the final database design and format of the data that it contained did not match what the data access tier expected to be accessing. This led to the redesign of both the database and the functions written to parse the data recieved.

However, I feel certain that a lot of the problems we might have encountered were mitigated by using Github, this allowed for us to keep track of each others development process much more easily.

3 The development process

The business logic tier had an incredibly smooth development process. It was very clear what information it would receive and what it needed to output. The functions that were written for it within the first week of development went mostly unchanged. I started first with the most important functions (e.g. extracting the coding sequence from a sequence of DNA, having been given the coding locations) and finished with the more complex and less vital functions such as that which assigned a p-value to the relative frequencies of codons. I spent very little time in the design process since it seemed intuitively clear how the parts would fit together. Breaking this section up into lots of small functions definitely helped keep things clear in the process.

The data access tier had a slightly rockier development process due to changing database design and format of the data. Although for the varying design this simply involved changing slightly the SQL queries, while the unexpected format requires some extra lines of code to parse it. These extra lines of code were later removed when the format of the data was changed to be more suitable for a database.

For the development of the front end, the process was a lot less smooth. Because I had decided to produce a web page to the bare minimum of the specification, I did not place much emphasis on planning, I simply wanted to get things working. This led to a design that grew out of the already existing middle tier, with some elements of the front end feeling like they're held together by duct tape. A feature that epitomises this is the method of generation of the summary table for the home page—it is generated by a function *summary_html_table()* that has to be run manually, the html code generated is then manually copied

and pasted into the *index.html* file. This is by no means a disaster, but, with a longer planning phase, the structure of the scripts for the web page would probably be better.

4 Code testing

There were two main approaches I used in code testing. The first and most important was the use of dummy data. I filled a database with dummy data that matched what I expected to recieve from the database in order to test the data access tier. I also used this approach in the testing of the business logic tier, feeding it the type of data it expected to receive from the data access tier. I also combined these two tiers, testing both together on the dummy database. This approach was very useful in developing the bulk of my functions.

The second was to add some small doctests to my functions. However, rather than use test-driven development, I added these relatively late in the development process and while comforting to know my function still worked as intended, they did not contribute much to the quality of code or the speed of development.

5 Known issues

There was one large issue—the website didn't work (when I originally wrote this essay). There was a problem importing the modules into the cgi script. This included the *Pandas* module and the *Pillow* module. I haven't been able to fully test the website because of this. I spoke to Dave Houldershaw and he properly installed the modules on Hope. However, while I haven't been able to test them properly, I feel confident that the business logic tier and data access tiers function as expected.

All of the genes included in the database have coding regions. However, for many of them, they do not have any listed in the database despite them being included in the Genbank file. Because no coding region means no codons, a division by zero error is produced when calculation for the percentage of codon use is attempted.

6 What worked and what didn't - problems and solutions

What didn't work well

Within the group, generally when things didn't go well it was because of a lack of sufficient communication. I think we were too hesitant to get involved in each other's work at the start of the project for fear of stepping on each others toes. I think this manifested itself most clearly between the database and data access tier - rather than state clearly something along the lines of: "I think there should be an entry in a table in the database for each coding region with a start position column and an end position column. This would help a lot with simplifying the data access tier." I didn't make it clear what my needs were. This problem could also perhaps been avoided by assigning the data access tier

to the person handling the database, as would, perhaps, be natural. This would have produced a situation where the interactions between individuals in the group matched with the APIs between layers.

What worked well

Within the group, what worked well was our use of collaboration tools such as github and slack. Without these it would have been much more difficult to maintain the level of communication necessary to produce a functioning website. While we definitely could have made better use of these tools, they were, nonetheless, invaluable.

Within the middle layer, I don't think there was a solution that I was particularly proud of. It seemed to be that the problems here lent themselves to straightforward obvious solutions. Thinking in more general terms, the approach of breaking the layer up into many small functions was extremely effective and made for easy debugging and clear development.

Within the front end, there were a couple of solutions that I was vey happy with. The first was the generation of the exon diagrams. I didn't want to use anything other than html and python, in order to produce a functional website in the shortest time without having to complicate things with new languages. My approach was to generate a .png diagram with a python module called *Pillow* using rectangles and ellipses, and to display this image file in the webpage. It was not a particularly elaborate approach, and I'm sure there are better, more conventional, ways of handling this but I was satisfied with this solution using the tools immediately available to me.

The second—more general—solution that I similarly satisfied with was the use of the module *pandas* and its data structure, the *DataFrame*. It allowed for easy handling of tables in the cgi script and straight forward production of html tables. Rather than needing to write a function to produce html tables of varying length based on python lists, the DataFrame object has a method *to_html* which renders the DataFrame as an html table.

7 Alternative strategies

Generally, I was happy with the project design and implementation. However, there were a few areas where I would consider using alternative strategies if I had to tackle this project again.

The first is in the storing of chromosome-wide information for the codon tables. The approach I used was to store them as lists in the *config.py* file. A neater approach might have been to store the information in a csv file. This would have probably been more important if there was more information attached to each codon.

The second was in the generation of the summary table for the home page. To generate this, I pulled all the data from the database into a DataFrame, including the DNA sequence for each gene, and then manipulated this DataFrame. For a larger dataset this might have been unfeasable, and dealing with it row-by-row would be more appropriate. As it stands, the amount of data is easy manageable, and a summary DataFrame is probably the simplest approach.

The last area in which I might consider an alternative strategy is in the cgi script. In my approach, the summary page for a gene is only accessible from its accession number, even in the cases where it has a unique name, location or protein product. When a search is carried out by gene name, a list of genes with the name is produced even if it only has one item in it. I felt this was the simplest approach due to handle the non-uniqueness of these three features for many of the entries in the database though it may have been more desirable for the summary page for a gene to appear when these searched for features are unique.

8 Personal insights

This project has made me a much more competent python programmer and given me a much better overview of how the different pieces of a website fit together. While I knew the importance of good communication and planning in an abstract sense, this experience has given me a much greater appreciation of what can go wrong when those elements of the development process aren't emphasised.

Furthermore, I have more of an understanding of how projects should be structured, how to collaborate with others on software development, and the value of test-driven development.