Project Plan for CSE2000

K. Kanniainen, P. Ulev, Y. Wang, D. Coban, M. Cristea-Enache

1 May 2020

Contents

Chapter 1

Introduction

1.1 Background

Bacteria, microscopic, single-celled organisms, thrive and appear in many different environments such as in the soil, in the air, and in human bodies. In regard to their DNA, the concept of species is quite vague: some species are made up of nearly identical genomes, while in other cases the whole species only shares 40 % of their total genome content.

This diversity in a genomic content of a species becomes more important in cases such as the E.coli bacteria. Most E.coli do not cause any harm, but some extra genes may transform it into a dangerous pathogen. Thus, having the ability to analyse and identify different strains can be immensely helpful.

Luckily, the bioinformatics field is rapidly developing software to do precisely this. There are tools that compare DNA from a sample to an ever-growing database of known strains with high accuracy and recall.

1.2 Client goal

With this background in mind, the Delft Bioinformatics Lab wants us to develop a more streamlined and intuitive workflow that will enable more biologists to run their DNA analysis with little programming knowledge. The challenge lies in gathering the latest researched algorithms and combining them in an end-to-end, easy to use pipeline. More details about the goals and the list of features will be made clear in the requirements chapter.

1.3 Approach

To ensure a successful end product, our team has started the project with an orientation phase. More precisely, we have spent two weeks in which we have analysed the existing software solutions and interviewed key stakeholders. Furthermore, we have gathered and compared the benefits and drawbacks of the available tools, languages, and architecture choices. We have gauged the risks that could hamper our progress and found some guideline solutions that could minimize them. We have crafted a proof-of-concept solution for demonstration, and last but not least, we have created a preliminary implementation timeline that will help us with assessing our progress during the development phase.

1.4 Structure of this document

As part of our two-week orientation phase, this project plan documents the research and design choices we have made.

Chapter 2 contains a more detailed description of the problem put forward by the client and an analysis of the existing tools and algorithms that could help us build a good end product.

In Chapter 3 we describe the list of features we plan to implement.

Chapter 4 is about the initial choices we have made, both in regard to technical tools and design principles, how we have made those decisions, and the risks we acknowledge.

Finally, Chapter 5 contains a preliminary implementation plan, including a week by week guideline schedule of the features.

Chapter 2

Problem analysis

In this chapter we discuss the problem posed to us, considering some of the existing approaches and exploring why there is still work to be done.

2.1 Background

Often the significant genetic variation even within a single species can have a big impact over the properties of microbes. Different strains, when colonising a hospitalised patient for instance, should be treated in different ways and pose different levels of danger to begin with. This is why it is useful to be able to identify, as accurately and as quickly as possible, different strains of bacteria present in a sample [1].

Modern technologies make it possible to easily extract DNA information from samples, but the limitation is that these methods cut the original DNA molecules into small fragments [2].

2.2 Problem statement and key goals

Our challenge revolves around a computationally efficient tool to identify different strains of bacteria present in a sample of small DNA fragments. We have been asked to develop an end-to-end pipeline which can, given such a sample, estimate the relative abundances of bacterial strains present in the data. Our product should be easy to use for a scientist with limited computer science experience. Preferably, it should run on a cloud platform.

Input data may be given in the form of an identifier or a file path. Our product should download the data if it is not available in local storage already.

The pipeline should then proceed to estimate the strain abundances using initially an existing analysis tool and later on possibly our own implementation. The design should be modular so that the exact algorithms are interchangeable.

In the end, the results should be presented in a way that allows the users to easily draw conclusions from it.

2.3 Existing end-to-end products

The following section will compare some existing tools for metagenomic analysis in terms of their output formats, speed and space efficiency, easiness of use and platforms they run on. We find Sigma [3], Bib [1] and StrainEst [4] most useful for our purpose. Note that the comparison significantly relies on the respective articles about these tools.

Output: it is the result of the calculations made by the algorithms in some kind of output files. StrainEst generates four text files and one pdf visualization file. They represent relative abundance and prediction information. Sigma generates one text and one html file. Both of them

contain information about the estimation of the relative abundances. BIB generates only one file abundance.m_alphas, which contains estimation of the strains and their abundances.

Speed and space efficiency: BIB typically runs for around 10 minutes on 1M reads on a single-CPU PC. We tested StrainEst and it ran for around 30 minutes in a Docker container analysing a 28MB fastq file. Sigma is written in C++ and has a specific focus on speed, parallelization and memory optimization. It uses significantly less memory and has lower runtime than other tools (Pathoscope, MEGAN), according to their test described in their article [3]. It took Sigma 10 minutes and 10000 cores to complete the alignment step in their algorithm, but on a huge dataset requiring 20 million simulated reads.

Environments: on high-level, all three of these are just some folder with script files. They can be put in an isolated environment, such as a Docker container. StrainEst already has a dedicated Docker container for it [5], with all the required tools installed in it. Similar Docker containers can be made for Sigma and BIB, and it is also possible to make one container containing all three of the tools and run them as needed. Terra supports Docker containers and automatically delegates all data and computation jobs to the integrated cloud service.

Usability: some basic command-line competence is required to be able to work with these programs. A programmer or a computational biologist should be able to make use of any of them, possibly requiring some time spent researching the APIs. As mentioned, StrainEst contains a set up environment with all the configuration done and additional tools downloaded. It is the most intuitive and easy to get started with, which in many cases is a vital step in choosing which tool to use for any kind of analysis.

2.4 Pipeline

A pipeline in a Unix shell context consists of several commands linked to each other with a pipe ('|') character. This syntax connects the previous command or program's standard out to the next one's standard in to form a "pipe" of data flow. In a bioinformatics context, it means a similar system of analysis and data wrangling tools and commands, but is typically crafted with a dedicated description language that is more human-readable than simple bash and, aside the basic linear chaining, allows for various other "plumbing" settings, that is, data flow configurations such as branches and merges. These enable concurrent execution of suitable tools [6].

These core pipelines can be extended from both ends. In the front-end it is possible to abstract away code and ask the user for data sources and selected options with a graphical user interface. In the other end of the system, the results of the process can also be displayed visually without requiring any further interaction from the user.

A good pipeline should be modular. This means that each part in the pipeline should be replaceable with any piece of software that implements the same interface. In our project, the parts most likely to be replaced frequently are the exact analysis tools and the (cloud) platforms they should run on.

2.4.1 Key features

1. User starts with a simple graphical user interface (GUI) to generate a configuration file for the pipeline. The user can upload this file to a cloud service and proceed from there.

2. User logs into a cloud computing service and clicks to start a workflow that will take the data identified in the configuration file and process it according to the user's preferred settings. The data and algorithms used in the tasks can either be located on the same cloud platform or somewhere else.

3. The final command of the workflow offers to download a report file or possibly automatically opens it in a browser tab. It should provide a visually clear representation of the output data and options to download it.

Of these three, step 2 should have the highest priority and step 1 the lowest, as a small configuration file should be relatively easy to edit even without a GUI.

2.4.2 Related tools

This section presents some useful software tools that can be of use for our project.

WDL (Workflow Description Language) is a language made to describe data processing workflows in a human-readable way. The syntax is user-friendly and structured and is appropriate for non-programmers as well. It can chain numerous tasks together and parallelise their execution. It is appropriate for genome analysis pipelines and developed by Broad Institute and has various tutorials online which further incited us to use it as our workflow language. WDL is not executable by itself, instead it needs to be run on an execution engine such as Cromwell [6].

Snakemake is one of the most famous workflow management system tools out there. Workflows are described with a human readable language based on Python and can be run on clouds, servers, and other environments. Its documentation is more extensive than WDL's and it is highly popular with its 591 citations, as of April 2020 [7]. Snakemake is widely used in bioinformatics and "interoperates ... with any web service with well-defined input and output (file) formats [8]." A more extensive comparison between Snakemake and WDL can be found in chapter 4.

2.5 Algorithm

The computational problem regarding strain identification is as follows: the input data consists of short DNA fragments extracted from the sample, while we also have a reference database containing information about known strains and their DNA profiles.

A statistical model estimating the "mix" of different strains in a sample can be formulated as a multivariate linear regression problem. To formulate this, we first need a measure of similarity. One technique to estimate this revolves around k-mers, which are sequences of length k that are captured from the DNA fragments with a sliding window. The counts of different k-mers can be used as a measure of similarity between genomes and this will serve as the basis for our computation model. Further data analysis techniques may be used to improve accuracy.

The question of computational efficiency comes into play when the size of the sample data and the value of k are considered. The input datasets are typically big, even hundreds of gigabytes, so we cannot rely on fitting everything in the main memory at once; instead, we will want to design an algorithm that processes the data one row at a time. Furthermore, there are 4^k possible k-mers in a DNA sequence, as every character in the k-mer can be any of the four bases A, C, T, and G. As we have to track a count for each of these, we need to pick sufficiently small values for k.

2.5.1 Existing algorithms

This section explains the key steps used in the 3 existing algorithms mentioned in section 2.3. Their way of how to approach and how to solve the problem as well as which tools to use may serve a good reference when we create our own algorithm.

Sigma algorithm [3] for metagenomic biosurveillance uses a stochastic probabilistic model to resemble metagenomic reads sampling from genomes. In order to align metagenomic reads against reference genomes, Sigma adopted a short-read alignment algorithm, Bowtie2 [9], by default. To estimate the probability of sampling a random read from the reference genome g_j, Sigma uses the non-linear programming method implemented in the Ipopt library [10].

StrainEst [4] is a program that uses single-nucleotide variants (SNV) profiles to count and identify coexisting strains and their relative abundances. The pairwise Mash distances [11] are computed between the genomes of the species of user interest and the species representative (SR), in order to filter out unrelated genomes. They use complete linkage hierarchical clustering to select the representative set of genomes, which are aligned onto SR using MUMmer[12] and the SNV profiles are calculated. 10 representative genomes of the species are selected and aligned with

reads using Bowtie2 [9]. Given the SNV profiles, Lasso regression is used to identify the reference strains as well as their relative frequencies which fit best into the observed allelic frequencies.

BIB, Bayesian identification of bacterial strains from sequencing data [1], uses UPGMA method in the MEGA6 [13] to cluster the strains and uses progressiveMauve [14] for multiple sequence alignment, in order to selects reference genomes. Then, BIB uses Bowtie2 [9] to let the reads align against the reference sequences. BitSeq [15] is used to estimate the relative frequency of each strain in the sequenced dataset.

2.5.2 Related tools

This section features some genomic analysis tools that could be used in our algorithm.

GATK is a "genomic analysis toolkit focused on variant discovery". It is widely used software with strong documentation and use cases. Its scope is beyond human genomes, it handles other organisms as well and supports exome analysis. It can be run on a cloud platform or a traditional cluster. GATK supports local clusters such as Docker but can also be run directly. It features ready-made pipelines but also modular tools, including data pre-processing and SNP calls [16], among others [17], [18].

Mash is an alignment-free toolkit to compute the distance between two genome sequences. It extends the MinHash algorithm, which compresses large sequences to sketch representations. Mash then compares the sketches and computes the Jaccard index, the proportion of shared k-mers, acting as a significance value for the match, and the Mash distance, which assesses proportion of the mutation between two sequences [11].

Tensorflow is a machine learning library, mostly used for image processing, reinforcement learning and time-series analysis [19]. It has a powerful and complex set of tools and features with a good API. Its main focus are different types of neural networks and many kinds of real-world problems can be solved with it. Google developers work on it and it currently has state-of-the art novel algorithms in machine learning. It is also well-known for the ability to operate on large scale environments. Tensorflow is available in multiple languages, including Python, Javascript, C++, and Java.

2.6 Cloud platforms

Cloud computing is a large area of research nowadays. It is helpful in numerous fields, including bioinformatics, as the file sizes can be quite big and the algorithms used often require a considerable amount of computational power; cloud environments are built to scale as needed.

The expected user group of our program is biologists. We expect them to have either none or a very basic understanding of programming, so a cloud-native solution with an intuitive workflow that does not require the end user to install software themselves could be really useful and we have not encountered such a tool in our research this far. Another upside is that cloud-native workflows are very easy to share to the larger scientific community. This significantly improves the impact potential of our work.

For these reasons as well as being one of the goals put forward by our client, we have researched one of the more popular cloud-native workflows used by the bioinformatics community.

2.6.1 Terra

The cloud platform Terra allows users to upload their data to the cloud and lets them use each other's published workflows combined with applications like Jupyter Notebook within a simple interface. It has built-in support for Cromwell and various genomic analysis tools. Cloud computing is also supported in the form of Google Cloud integration and the use of containers [20], [21].

Terra offers native support for running scripts written in WDL [20]. WDL in turn "... makes it straightforward to define complex analysis tasks, chain them together in workflows, and parallelize their execution" [6].

Chapter 3

Requirements

This chapter is dedicated to the features our team, in close collaboration with the client, has decided to implement. We briefly discuss the process of feature negotiation, followed by a list of functional and non-functional requirements. In the feasibility study section we motivate some prioritisations we have made.

3.1 Methodology

Using the MoSCoW prioritisation method [22], we were able to reach an agreement with key stakeholders on the importance of each feature we might implement. The most important features are labelled into the **Must Have** category; the ones that are important but not as critical are grouped into the **Should Have** category. The requirements that are desirable but are not necessary for the program are grouped into the **Could Have** category, while the features in the **Won't Have** category will not be planned for during this timeframe.

We have also distinctly marked (with asterisks) some of the requirements in the following section: those requirements have an adjusted priority, which is a result of our feasibility study. We will go in more depth about how we made these choices, in agreement with our client, in section 3.4.

3.2 Functional requirements

These are requirements that concern the features of the software and can be represented as use cases.

1. The program **must** be able to use sample data residing on the same platform as the program.
2. The program **must** accept a configuration file in a machine readable format containing preset settings as well as a file path to the data sample.
3. The program **must** be able to use at least one existing strain estimation tool to compute a prediction of the relative abundances of the strains present in the sample. *****
4. After the strains have been estimated the program **must** offer a report with the results in a standardised machine- and human-readable format.
5. The program **should** offer a graphical user interface for creating the configuration file.
6. The program **should** be able to use sample data from a public database, given an identifier.
7. The program **should** offer the user a choice of multiple existing strain estimation tools. *****
8. The program **should** accept a reference genome database provided by the user. *****

9. The program **should** offer a visualisation of the results.
10. The program **could** offer the user the possibility of running multiple available strain estimation tools in parallel and displaying the results in a combined view. *****
11. The program **could** perform preprocessing on the input data, such as trimming the fragments, in order to increase accuracy. *****
12. The program **could** feature our own estimation process alongside the existing ones. *****
13. The program **won't** offer the user the possibility of starting the workflow with downloading and building a relevant reference genome database by picking from a list of available species.

3.3 Non-functional requirements

These requirements concern the properties and performance of the software that are not specific to any single use case.

1. The software **must** include a workflow description written in a language that is commonly used in the field of bioinformatics.
2. The written code **must** be readable and commented according to industry standard code formatting guidelines.
3. The code **must** be tested where applicable, with branch coverage of at least 45 %.
4. The files **should** include documentation sufficient for a biologist to run the program.
5. The program **should** be available on a widely-used cloud platform, accessible to researchers in the field.
6. The code **should** have branch coverage of at least 75 %.
7. The code **should** have meaningful documentation for each class and method.
8. The code **should** make use of modular design so that it is possible to add new algorithms later on.
9. The program **could** make use of available genome analysis tools and other libraries such as GATK, Jellyfish, Mash, SciKit, and Pandas, in order to craft our own implementation of the strain identification process.
10. The model fitting process **could** include cross-validation and other machine learning tools in order to avoid overfitting.
11. The program **won't** offer significantly better accuracy than the existing state-of-the-art tools. *****
12. The program **won't** offer the user an option of using a cloud platform of their choice.

3.4 Feasibility study

In this section we briefly discuss the process of requirement negotiation. Based on the key goals put forward by the client, we came up with an initial list of requirements, most of which can still be found in the previous section. However, after another interview with the client, the key stakeholder, as well as an internal feasibility discussion, some requirements have gained or lost priority. Furthermore, additional feature suggestions from our client have been added to the list.

The main adjustment our team had to make after reviewing and negotiating the requirements has been a shift of priority between the pipeline and the algorithm (the respective concepts we have analysed in depth in Chapter 2). The opinion of our client has been that developing a better pipeline would be more preferable than a fast, memory efficient, algorithm. For this reason, the functional requirements concerning the pipeline have gained priority (Functional requirements 3 and 7), while the process of creating our own estimation process has lost priority (Functional requirement 12 and non-functional requirement 11)

Another adjustment we have made after the interview with our client has been adding some of his suggestions as features our end product will contain. We have carefully deliberated the expected amount of resources those new features will require, after which we placed them in the categories of non-essential but important or desirable features (Functional requirements 8, 10, and 11)

The changes we have made in our requirements have been on the cautious side. We consider it a better idea to mark more requirements as important but non-essential in order to not run out of time and resources for this project. With these requirements we are comfortable of finishing all **Must Haves**, most **Should Haves** and some **Could Haves** as the MoSCoW method imposes.

Chapter 4

Project approach

This chapter is dedicated to exploring several possible tools that we will use during our development phase. After finishing the list of requirements, our team has considered multiple choices in order to get the desired end product. Thus, each subsection contains a brief discussion about the tool we have chosen and the advantages over other possibilities.

4.1 Development process framework

One of the first choices we had to make was which development process framework to use. In broad terms, there are 2 main ways in which we can guide the process: a waterfall approach or an agile one.

The waterfall model [23] is based on minimizing risks through a systematic and thorough linear approach. More specifically, the waterfall model offers a structured approach, with clearly defined phases (requirements, design, implementation) while also emphasizing the importance of documentation. Its main drawbacks are the lack of working prototypes until late in the process as well as being unable to change the requirements during the implementation phase.

An agile approach [24] supports an incremental, iterative process. It offers a working product throughout the whole project, slowly adding more features upon it. It promotes early and often collaboration between the client and the developers in order to build the best possible product.

For these reasons, our team chose an agile approach, more specifically the Scrum process [25]. In order to adapt the scrum approach to our needs as best as possible we will work with one week long sprints; the planning for each sprint will happen on Monday morning and the retrospective on Friday afternoon.

4.1.1 Main architecture

The existing tools to analyse metagenomic data-sets are quite hard to use and there is a need for a unified solution. For our program we decided to create an end-to-end solution to improve the user experience. We will have a simple GUI that generates a configuration file so that the user can tweak the workflow parameters without having to know how the workflow is generated or used. The configuration file will specify at least one strain estimation algorithm to use and the parameters needed for the algorithm, including the input data location.

We will use a cloud based platform for scalability and easiness of use, as this is one of the key goals the client has put forward. More about our choice for the platform in section 4.1.3.

Our goal is to have the user upload the generated configuration file to the platform, where it will set up the strain estimation process as needed. We will strive to write modular, interchangeable code. This will help us achieve some of the could have requirements, namely being able to freely choose a strain estimation tool and run multiple ones in parallel. After the results are computed, the program will let the user view and download the results in a variety of formats.

4.1.2 GUI frameworks

We will use HTML5 as the general GUI framework for all graphical user interfaces we make in this project. It is highly customisable, can be implemented in standalone `.html` files, and aside a reasonably new web browser does not require any additional software to use.

The data visualisation view that shows the results of the process will make use of the Chart.js framework, which makes it easy to display all kinds of graphs from a dataset.

4.1.3 Cloud platform

We consider Terra an appropriate platform for our project for several reasons. It has strong support for multiple workflows (pipelines), it has a friendly GUI and easy-to-use options. Terra works with Google Cloud, so all the data and computations will be uploaded and run there. We do not have a specific preference for a cloud as long as it offers a reliable and secure service that can meet the user's requirements.

4.1.4 Pipeline phases

The workflow script(s) will be written in WDL as a result of our platform choice and will contain the following steps (some are optional and implementation may vary):

1. read the metagenomic sample data specified as a file path in the configuration file;
2. build the reference genome database as specified in the configuration file;
3. do preprocessing (e.g. quality control) of the provided metagenomic sample;
4. run the strain estimation tool the user indicated in the configuration file;
5. offer a report of the results from the strain estimation tool and a visualisation of these results.

In short, our pipeline links different steps needed for conducting strain estimation using different available tools. It should be easy to start even for users who barely have programming experience.

In addition, the available tools facilitating the constructing of the pipeline are listed below. For step 1, NCBI Reference Sequence Database (RefSeq) [26] offers high-quality complete genomes for a range of species, which can be used as reference genomes in the strain identification process. There is a useful tool called ncbi-genome-download [27] that can let users to choose the species of their interest to download from RefSeq. For step 2, in terms of a quality check of the genomic dataset to have strain-level identification, FastQC can offer a report showing the quality of the raw reads [28]. We would adapt the report results shown in FastQC to meet the research purpose of our users. According to the adapted report results, then apply Trimmomatic, "a flexible read trimming tool for Illumina NGS data" [29], on the metagenomic dataset for quality trimming and adapter clipping. In the end, apply FastQC on the preprocessed data and see how much the data quality has been improved.

4.1.5 Algorithm language

We are planning to write our own strain estimation algorithm as a "could have" feature, and we chose Python for this as the main programming language. Python is widely used for data analysis [30] and machine learning problems. Its readable syntax and low complexity allow programmers to focus more on logic and less on implementation details. We consider this as a key factor. In addition, it has numerous libraries for really efficient mathematical operations, such as NumPy, which is particularly good at matrix manipulations, Pandas for dataframes, and matplotlib for visualisations. It also has good package manager (pip), which we can use to deal with external libraries. Python is therefore an appropriate choice for metagenomic analysis tasks.

4.2 Quality assurance

In our GitLab instance, the master branch is protected so that team members cannot push to it directly. Rather, we will update the code in our master branch with the Merge Request functionality by cross-reviewing each other's code. This will result in better code quality in our master branch. Pushes to all branches will trigger a continuous integration pipeline which executes tests and static analysis tools and notifies users on failure. A branch failing the pipeline cannot be merged.

4.2.1 Software testing

We will test the Python scripts we create for the project with the `py.test` module, which features a nice and simplistic syntax. Test coverage will be measured by the `pytest-cov` tool. We do not plan on automating tests for the WDL script or the HTML5 user interfaces.

4.2.2 Static analysis tools

The code readability and quality in each file will be moderated by the appropriate tools for each file type; `pylint` for Python files and `jshint` for any HTML files containing embedded JavaScript.

4.2.3 Version control

We are required to use the TU Delft's GitLab server for version control. It offers us an issue board, which we will use as a virtual Scrum board, while also supporting the basic git features. For writing the associated documentation, we use LaTeX on the collaborating platform Overleaf.

4.2.4 License

The client required us to use Apache v2 as a license for this project [31]. Users of our software are able to use, modify or distribute source code on the condition that all original files carry a note stating the authorship as well as all changes made by the user in that particular file. However, the future users are able to select a different license for their changes and overall project.

4.3 Risk analysis

Two of the main risk factors during the process of implementing the software is communication within the group and communication with the client. To ensure we do not have problems with group level communication we have daily scrum meetings as well as a retrospective meetings at the end of each week. Additionally, we have weekly meetings with the teaching assistant to help us with any problems we might encounter. We also have weekly meetings with the client to ensure we are on track with what the client wants.

One risk factor in the planning stage of the project is the appropriateness of the requirements. Each group member conducted research after the client stated their goals for the end result of this project. Since none of the members of the group are bioinformatics experts, the end goal and how to achieve it were unclear and ambiguous at the start. Due to our limited understanding of bioinformatics and genomics, our requirements could prove to be unsatisfactory. This would result in the client not getting the program they asked for. On the opposite end of the spectrum, again due to our limited knowledge on the biology side of the project, if the requirements are too ambitious and not feasible the end product will be absent of the features the client wanted.

One such feature is the algorithm. The algorithm could prove to be too difficult and time consuming to implement. Therefore, after consulting the client, we decided to first implement the core pipeline software and make it usable with existing algorithms. This way we limit the risk factor such that the client will get the pipeline they want immediately, and afterwards we will be developing the algorithm for the remainder of the project.

Chapter 5

General planning for the remainder of the project

After analysing the task that was given to us in Chapter 2, by clearly specifying the features we are going to implement and the tools we are going to use in Chapters 3 and 4, we are ready to start the development phase.

By using the insight we have gathered and shared in this project plan, we feel ready to create the schedule we are going to follow for the remaining weeks. We are going to work in weekly sprints, having regular meetings to track our progress; not only within the team through the weekly retrospective but also with the client and our software project coach.

Taking inspiration from a project plan example the course coordinators have provided us [32], we have created a week by week plan, attached below, highlighting when we will work on different parts of the product as well as various deadlines and course-related assignments. Green blocks represent the time our team will use to work on the product, red blocks represent deadlines our project has to follow and the blue blocks serve as reminders to start working on certain course assignments.

Week #	Week 4.1	Week 4.2	Week 4.3	Week 4.4	Week 4.5	Week 4.6	Week 4.7	Week 4.8	Week 4.9	Week 4.10	Week 4.11
Monday	20-04-2020	27-04-2020	04-05-2020	11-05-2020	18-05-2020	25-05-2020	01-06-2020	08-06-2020	15-06-2020	22-06-2020	29-06-2020
Friday	24-04-2020	01-05-2020	08-05-2020	15-05-2020	22-05-2020	29-05-2020	05-06-2020	12-06-2020	19-06-2020	26-06-2020	03-07-2020
Orientation Phase											
Project plan		03-05-2020									
Interview stakeholders											
Problem analysis											
Requirements engineering											
Choose tools & languages											
Implementation Phase									21-06-2020		
Implement must have requirements					19-05-2020						
Implement should have requirements											
Create own algorithm											
Implement pipeline could have requirements											
Documentation											
Technical writing assignments		29-04-2020	08-05-2020	14-05-2020		25-05-2020		10-06-2020	21-06-2020		
Ethics					24-05-2020				21-06-2020		
Information Literacy				11-05-2020							
Project skills				available					21-06-2020		
Responsible computer science		available			first draft	available			21-06-2020		

Figure 5.1: General Planning

Bibliography

[1] A. Sankar, B. Malone, S. C. Bayliss, B. Pascoe, G. Méric, M. D. Hitchings, S. K. Sheppard, E. J. Feil, J. Corander, and A. Honkela, "Bayesian identification of bacterial strains from sequencing data," *Microbial Genomics*, vol. 2, no. 8, e000075, 2016. DOI: https://doi.org/10.1099/mgen.0.000075.

[2] G. van der Auwera. (2017). Mpg primer: High throughput sequencing and variant calling pipelines (2017), Youtube, [Online]. Available: https://www.youtube.com/watch?v=2bAsFyJ4BbI.

[3] T.-H. Ahn, J. Chai, and C. Pan, "Sigma: Strain-level inference of genomes from metagenomic analysis for biosurveillance," *Bioinformatics*, vol. 31, no. 2, pp. 170–177, 2015.

[4] D. Albanese and C. Donati, "Strain profiling and epidemiology of bacterial species from metagenomic sequencing," *Nature Communications*, vol. 8, no. 1, p. 2260, Dec. 2017, ISSN: 2041-1723. DOI: 10.1038/s41467-017-02209-5.

[5] [Online]. Available: https://hub.docker.com/r/compmetagen/strainest/.

[6] *Openwdl*. [Online]. Available: https://openwdl.org/.

[7] *Dimensions*. [Online]. Available: https://badge.dimensions.ai/details/id/pub.1018944052.

[8] J. Köster and S. Rahmann, "Snakemake—a scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, Aug. 2012, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts480. eprint: https://academic.oup.com/bioinformatics/article-pdf/28/19/2520/819790/bts480.pdf.

[9] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, Mar. 2012, ISSN: 2041-1723. DOI: 10.1038/nmeth.1923.

[10] A. Wächter and L. T. Biegler, "Strain profiling and epidemiology of bacterial species from metagenomic sequencing," *Nature Communications*, vol. 8, no. 1, p. 2260, Dec. 2017, ISSN: 2041-1723. DOI: 10.1038/s41467-017-02209-5.

[11] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy, "Mash: Fast genome and metagenome distance estimation using minhash," *Genome Biology*, vol. 17, no. 132, Jun. 2016. DOI: https://doi.org/10.1186/s13059-016-0997-x.

[12] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg, "Versatile and open software for comparing large genomes," *Genome Biology*, vol. 5, Jan. 2004. DOI: https://doi.org/10.1186/gb-2004-5-2-r12.

[13] T. K, S. G, P. D, F. A, and K. S., "Mega6: Molecular evolutionary genetics analysis version 6.0.," *Molecular Biology and Evolution*, vol. 30, no. 12, pp. 2725–2729, Oct. 2013. DOI: 10.1093/molbev/mst197.

[14] A. E. Darling, B. Mau, and N. T. Perna, "Progressivemauve: Multiple genome alignment with gene gain, loss and rearrangement," *PLoS ONE*, vol. 5, no. 6, Jun. 2010. DOI: 10.1371/journal.pone.0011147.

[15] J. Hensman, P. Papastamoulis, P. Glaus, A. Honkela, and M. Rattray, "Fast and accurate approximate inference of transcript expression from rna-seq data," *Bioinformatics*, vol. 31, no. 24, Aug. 2015. DOI: https://doi.org/10.1093/bioinformatics/btv483.

[16] B. H. Tang, "Single nucleotide polymorphism (snp) and its preview," *Journal of Cell Signaling*, vol. 01, no. 04, 2016. DOI: 10.4172/2576-1471.1000129.

[17] *Genome analysis toolkit*. [Online]. Available: https://gatk.broadinstitute.org/hc/en-us.

[18] O. U. Sezerman, E. Ulgen, N. Seymen, and I. M. Durasi, "Bioinformatics workflows for genomic variant discovery, interpretation and prioritization," *IntechOpen*, DOI: 10.5772/intechopen.85524.

[19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283, ISBN: 978-1-931971-33-1. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[20] *Terra.bio*. [Online]. Available: https://terra.bio/.

[21] B. Institute, *Home - cromwell*. [Online]. Available: https://cromwell.readthedocs.io/en/stable/.

[22] D. Clegg and R. Barker, *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[23] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," in *Proceedings of the 9th international conference on Software Engineering*, 1987, pp. 328–338.

[24] M. Fowler, J. Highsmith, *et al.*, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[25] K. Schwaber, *Agile project management with Scrum*. Microsoft press, 2004.

[26] *Refseq*, https://www.ncbi.nlm.nih.gov/refseq/.

[27] *Ncbi-genome-download*, https://github.com/kblin/ncbi-genome-download.

[28] *Fastqc*, http://www.bioinformatics.babraham.ac.uk/projects/fastqc/.

[29] B. A. M., L. M., and U. B., "Trimmomatic: A flexible trimmer for illumina sequence data.," *Bioinformatics*, vol. 30, no. 15, pp. 2114–20, Aug. 2014. DOI: 10.1093/bioinformatics/btu170.

[30] W. McKinney, *Python for data analysis*, Feb. 2013.

[31] *Apache license, version 2.0*, https://www.apache.org/licenses/LICENSE-2.0, Jan. 2004.

[32] (2016). Example project plan, TU Delft, [Online]. Available: https://brightspace.tudelft.nl/d2l/le/content/193732/viewContent/1648295/View.