PANTZERFAUST
PRODUCT PLANNING

Wouter Smit, 4401409
Justin van der Krieken, 4357116
Casper Athmer, 4329066
Faris Elghlan, 4341538
Cas Bilstra 4381084

29/04/2016

Table of contents

Introduction

Currently, there does not exist a good visualization tool to look at mutations of DNA strings. Our goal is to create such a visualization tool in order to assist the research on the bacterium causing Tuberculosis. A previous attempt has been made last year, but there were some flaws which kept the researchers from using those tools. This work builds on those attempts. This document describes the planning of the project.

High-level product backlog

This backlog provides high-level epics that the product will satisfy.

- The application is able to load sequence graphs and phylogenetic trees
- The application is able to load potentially any data set in the correct format, regardless of the size
- The application shows both the graph and the tree
- The application allows for navigation through the tree and graph
- The application at all times keeps an overview of the current position
- Navigation through the data utilizes the concept of semantic zooming
- The user is able to select subsets of the data, which the application will display
- The application detects mutations between (selections of) the genomes
- The application annotates interesting areas of the sequence graph with information found in metadata or external data
- The application hides useless data from the user to retain overview and sanity

Roadmap

Milestones are set for every week starting from week 2, and a release should be made every week based on the milestone of that week. The project should be finished in week 9, so there are 8 milestones. Every week, we will deliver a sprint plan (backlog) for the coming sprint, a sprint retrospective, and working source code in the form of a release in github.

Milestone 1: Loading 10 genomes
In this week, we should be able to load and show 10 genomes.

Milestone 2: Loading 340 genomes
In this week, we should be able to load and show 340 genomes, yet without semantic zooming though.

Milestone 3: Loading 1000 genomes with a basic interface
Now, we should be able to load 1000 genomes with some sort of semantic zooming. Two graphs for comparison and the phylogenetic tree for navigation should be visible.

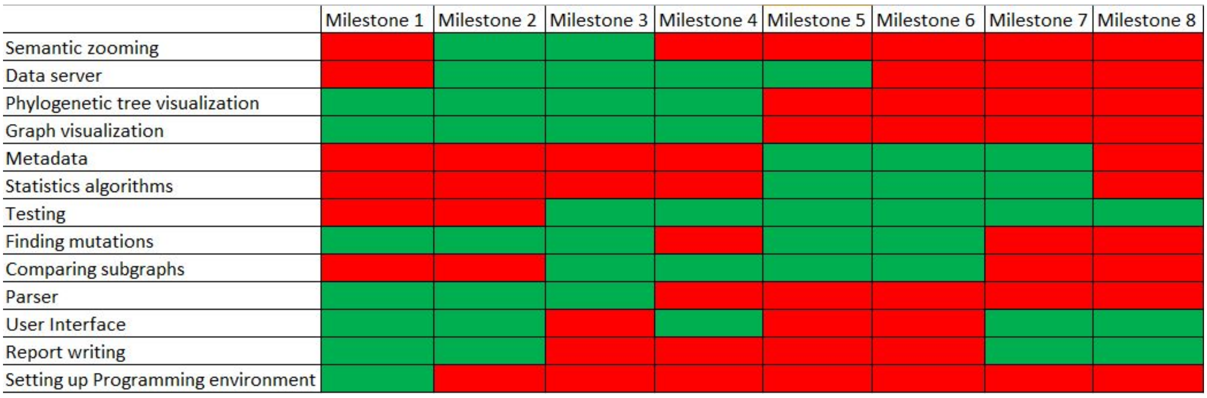
Milestone 4: Loading 3000 genomes
Here, we should be able to load 3000 genomes and have finished the data preprocessing part. Also, the interface should be fine-tuned.

Milestone 5: Loading 6000 genomes, statistics and annotations
We want to be able here to load all genomes and show relevant information about them.

Milestone 6: Advanced interface views
We now want to be able to give more views on the data and add some quality of life to the program (e.g. hotkeys)

Milestone 7: Finalizing interface
The user interface should be finished.

Milestone 8: Finalizing product
The last few small adjustments and a product to deliver which is ‘done’.



	Milestone 1	Milestone 2	Milestone 3	Milestone 4	Milestone 5	Milestone 6	Milestone 7	Milestone 8
Semantic zooming	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟥
Data server	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥
Phylogenetic tree visualization	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥
Graph visualization	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥
Metadata	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥
Statistics algorithms	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥
Testing	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩
Finding mutations	🟩	🟩	🟩	🟥	🟩	🟩	🟥	🟥
Comparing subgraphs	🟥	🟥	🟩	🟩	🟩	🟩	🟥	🟥
Parser	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥
User Interface	🟩	🟩	🟥	🟩	🟥	🟥	🟩	🟩
Report writing	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩
Setting up Programming environment	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥

‘Table 1: roadmap’

Table 1 contains the main subjects on which we will work for every milestone. It can be derived from this table that phylogenetic tree visualization and graph visualization have much to do with each other, as is the case for the ‘Metadata’ and ‘Statistics algorithms’ subjects. In general, there is a big link between milestone 1, 2, 3 and 4 because they are mainly about loading and displaying the data correctly, and milestone 5, 6, 7 and 8 are more about creating a business-grade application that provides useful analytics.

Functional requirements

In this section we will discuss the desired features of the application using the MoSCoW model. This means that we will split all features in four groups:

1. **Must haves:** These are all of the features which must be present in the final version of the application at the very least. It is not acceptable if any of these features is missing.
2. **Should haves:** These are all of the features which are important, but will not make the application unusable if they are missing.
3. **Could haves:** These are all of the features which are nice to have, but not very important if they are missing. These features will only be implemented if we have enough time.
4. **Won't haves:** These are all of the features which are not really important and generally take a lot of time to implement. We expect we will not have time to implement these features.

In the following sections we will globally list the features of these four sections. In the product backlog chapter we will discuss a few of these features in more detail using user stories.

Must haves:

- The application must be able to load files in the GFA format into the application so the data can be analysed.
- The application must be able to load any phylogenetic tree in newick format into the application so the data can be used in combination with a GFA graph.
- When the data is loaded, the application must be able to show both the loaded tree and loaded graph for a good overview of the DNA samples.
- The application must be able to zoom in (when the user scrolls using the mouse wheel) on the graph and reveal a more detailed representation (semantic zooming) to make it possible to keep an overview of big graphs more easily.
- The application must show (meta)data about visible parts of the graph to make it possible to easily determine interesting parts of the graph to zoom in on.
- The application must be able to display the graph of the DNA samples which are selected in the tree, to allow for comprehensive navigation of the graph.
- The application must show which part of the tree is selected while browsing the graph, to make clear what is currently being shown in the graph.
- The application must make use of the phylogenetic tree as a navigation device for the set of loaded DNA samples.
- The application must implement an easy way to navigate through the phylogenetic tree.
- The application must be able to load a set of 10 genomes in less than 1 second.
- The application must be able to preprocess data and afterwards access the preprocessed data to efficiently load user interface elements without any noticeable delays.

- The application must be able to preprocess a graph containing up to 6000 genomes in less that 2 hours.
- The application must be platform independant.

Should haves:

- The application should be able to show relevant information about mutations, so the user won't have to look them up manually.
- The application should be able to bind custom annotations to bubbles in the graph to mark them.
- The application should be able to show useful statistics about the data to indicate if the data is interesting for the user.
- The application should be able to show multiple views on the data to better contribute to the understanding of the data.

Could haves:

- It could be possible to extend the application using extensions which can perform actions by calling functions of the API which is provided by the application.
- It could be possible to swap modules of the application, such as the parser, with other modules which implement the same functionality (i.e. the interface of the module)
- It could be possible to change the layout of the user interface of the application while running the program, so the user can organize everything according to his own preferences.
- It could be possible to do easy tutorials while using the application (tutorials which are build into the application), so new users can learn while using the application without having to use external resources to guide them through the application.

Won't haves:

- The application won't be able to run in a distributed fashion by using for example hadoop.
- The application won't be ported to work on mobile devices such as tablets and phones.

Product backlog

In this section we will discuss the elements of the application that should be implemented and provided over the course of the project.

User stories of features

As a user,
When I start the application,
And have not yet done anything,
I must be able to select and load the genome files in the application.

As a user,
When I have loaded the genome files,
And have not yet done anything,
I should be able to browse through the phylogenetic tree to select the genomes to compare.

As a user,
When I have loaded the genome files,
And have selected a set of genomes to compare,
The subgraph of the selected genomes must be shown, using circles to visualize the nodes and lines between the nodes as edges.

As a user,
When I am comparing a set of genomes,
I must be able to get an overview of the complete graph and zoom in on the parts which are interesting by semantically zooming in (similarly to google maps[1]).

As a user,
When I looking at the phylogenetic tree or DNA graph,
And I click on any part of the graph or tree,
Then I must get relevant extra information about the part of the tree or graph which I clicked on.

As a user,
When I am looking at the DNA graph,
And the part of the graph which I am looking at contains a known pattern (such as a well known mutation),
Then I must be informed about this by the application with a visual clue and be able to view detailed information about this pattern.

As a user,
When I am viewing any part of the DNA graph,

[1] "Google Maps." 2005. 4 May. 2016 <https://maps.google.com/>

I want to get information that is relevant for the current semantically zoomed level. If I am zoomed out I want to see more general information and when I am zoomed in I want to see more detailed information.

As a user,
When I am interaction with the user interface of the application,
I want to feel familiar with the way the user interface elements interact. The user interface should act in much the same way as other modern applications.

User stories of technical improvements

As a programmer,
When reading the code,
I want to be able to easily identify the functionality of methods and the meaning of fields by reading the comments. Whenever possible the names of the methods and fields alone should already give a clear indication of the functionality.

As a programmer,
When interacting with someone else's code,
I want to be able to rely on this code. To make this possible, all code which is testable should be tested and well written, using appropriate design patterns where possible.

As a programmer,
When I want to contribute to the project,
I should be able to pull the project off of github without having to install any extra dependencies. This is done by using maven to handle the downloading of dependencies.

As a programmer,
When I want to contribute to the project,
And I pull the project off of github,
The project should be able to build and run without errors and all tests should succeed. This is done by using travis as continuous integration server.

User stories of know-how acquisition

As a student team,
When we want to succeed,
We must attend all obligatory lectures.

As a development team,
When we make an important decision in a field about which we don’t know much,
We will read appropriate literature about it to get more knowledgeable in this field.

As a designer,
When designing a part of the user interface,
I have to keep in mind the target audience and make it understandable for them.

Initial release plan

29-04-2016 Milestone 1: 10 Genomes on screen

- Loading 10 genomes works
- Display phylogenetic tree with some zooming on 10 genomes.
- Display sequence graph

06-05-2016 Milestone 2: 340 genomes

- Loading 340 genomes works
- Basic interface development
- First pre-processing steps on data

13-05-2016 Milestone 3: 1000 genomes, basic interface

- Loading 3000 genomes works
- Working bubbling algorithm
- Complete interface with two graphs and phylogenetic tree selection
- Mature data pre-processing and semantic layer preparation

20-05-2016 Milestone 4: 3000 genomes

- Loading 3000 genomes
- Data pre-processing completed.
- Fine tuned interface
- Solid, clean codebase with fundamental elements

27-05-2016 Milestone 5: 6000 genomes, statistics & annotations

- Loading all genomes achieved
- Data pre-processing finetuned and optimized.
- Added genome annotations for statistical algorithms
- Added statistical algorithms to interface

03-06-2016 Milestone 6: Advanced interface views

- Added more views to display specific parts of the data
- Added UX elements, such as hotkeys and user options

10-06-2016 Milestone 7: Finalizing interface

- Polished interface
- Remove or fix any undesired elements

17-06-2016 Milestone 8: Final Release

- All bugs are fixed
- Codebase is clean, modifiable and generally well-designed
- Codebase is ready for potential future work

Definition of done

This section describes the definition of done: this specifies when something is done. This way it's ensured that all group members know when something can be considered done. It focuses on the following aspects:

- Features: when is a feature done?
- Sprints: when is a sprint done?
- Releases: when is a release done?

First, features are discussed. A feature is considered done when:

- The feature has been appropriately tested through unit tests and functional tests. The test coverage should be at least 80%.
- The code is of good quality. First of all, this means that it should be properly commented. Second, static analysis tools are used (e.g. Checkstyle) to ensure high quality and consistent code.
- The feature meets all the requirements and user acceptance criteria that were set.
- The feature needs to be checked by at least two group members who haven't been working on it to be accepted.

A sprint is considered done when:

- All features that were in the sprint plan have been implemented successfully, or it has been clearly stated why a feature couldn't be implemented.
- A sprint reflection has been made.

A release is considered done when:

- Feedback has been given on the release (after a demo) by the customer.
- The release builds properly in Travis.

Glossary

- **MoSCoW**
  Prioritization technique used in management.
- **Source code**
  The human readable version of the computer instructions.
- **Sprint**
  The basic unit of development, in our case a single sprint was one week.
- **Github**
  A web-based version control system hosting service.
- **Genome**
  The genetic material of an organism.
- **Phylogenetic tree**
  Also evolutionary tree, is a branching diagram showing the evolutionary relationships among species.
- **(DNA) sequence graph**
  A graph containing a set of DNA samples, where nodes represent specific base sequences and each DNA sample corresponds to a specific path through the graph. Points in the graph where the paths of multiple DNA samples overlap indicate common base sequences.
- **GFA format**
  A text format for describing a set of sequences and their overlap.[2]
- **Newick format**
  A text format for describing a tree.
- **Semantic zooming**
  Zooming that also reveals more detail that is relevant for the current zoom level.
- **API**
  Acronym of Application programming interface. Any interface that allows inter-program-communication.
- **UX**
  Acronym of user experience. Usually refers to design behind optimizing the user experience, technical and graphical.
- **Hotkeys**
  Also keyboard shortcut, are keys that can be assigned to execute a set action in the application.
- **Codebase**
  See *source code*

[2] "(GFA) Format Specification - GitHub." 4 May. 2016 <https://github.com/pmelsted/GFA-spec>