Product Planning Context Project
Desoxyribonucleïnezuur

Toine Hartman - tjbhartman - 4305655
Martijn van Meerten - mvanmeerten - 4387902
Iwan Hoogenboom - ihoogenboom - 4396634
Yannick Haveman - yhaveman - 4299078
Ivo Wilms - iwilms - 4488466

17 May 2017

Abstract

This document contains the planning for the development of our multi genome visualizer. A high-level product backlog is given, which contains the prioritization of features in the form of user stories. Next, a roadmap of milestones is given, to guide us in our development. Furthermore, a detailed product backlog is defined, and finally, the definition of done for certain aspects of the project is given.

Table of contents:

1. Introduction

This document is the Product Planning of the Programming Life context. In this context we will build a multi genome visualizer. The visualizer will be useful to compare multiple genomes. The purpose of this document is to create a overview of essential features for the application. The document also gives a planning of when we want to include which features. We will, where possible, follow the planning and guidelines as described.

2. Product

2.1 High level product backlog

Here we will discuss features we want to have in our program using the MoSCoW model. The MoSCoW method is a prioritization technique where you divide features in groups depending on priority. The MoSCoW model makes use of 4 groups:

Must haves: These are the features that the program can not go without.

Should haves: These are essential features, but the program will not fail if these features are not part of the application.

Could haves: These are features that are very nice to have but are more quality of life features. They will only be implemented if we have time after implementing all should haves and must haves.

Won't haves: These are features we will not implement because they will take too much time, or add too little.

- Must haves:

The application must be able to load in and parse files that are in the GFA format.
The application must display the multiple genomes and their differences in the form of a graph.
The application must be independent from the platform using it.
The application must be able to zoom in and out on different parts of the graph to give a better overview of the data.
The application must be able to load files containing a large amount of genomes.
The application must allow the user to select a node and view up to 10,000 nodes around the selected node
The application must allow the user to navigate over the graph by panning.
The application must dynamically load new nodes so the user does not have to redraw.

- Should haves:

The application should allow the user to copy the base pairs in a segment to the clipboard.
The application should have a search function to search on the annotations.
The application should visually encode segments to display more detail about segments. This can be done by using color codings.
The graph should use glyphs to simplify bubbles depending on the height of the view.

- Could haves:

The edges could change of thickness to visualize the amount of genomes that use that edge.
The application could include different types of GUI layouts. One examples is a dark mode.
The application could have different color schemes for nodes (change on what criteria the color of the nodes is determined).
The application could have different different color scheme for the visually impaired (colorblindness).
The application could have a way of exporting GFA files.
The application could have a way of editing GFA files.
The application could have a way of searching for a specific sequence across all nodes.

- Won't haves:

The application won't be portable to mobile devices.
The application won't support plugins.
The application won't have cloud support.

2.2 Roadmap

Milestones are to be completed each week starting from the second week. They will be accompanied with a release at the end of each week. The milestones set to be completed are described below.

Milestone 1: Basics
For the first milestone we want to have a most basic version of the application. We must be able to load smaller files like the 10TB file. Have some basic graph visualization and a simple UI.

Milestone 2: Data scaling
As second milestone we want to increase the amount of data our application can handle. We want to be able to load all three datasets: 328 TB,Tomato,Human chr19. We also want to change the way we draw to a center point and a radius and make it possible to draw around 500 nodes around the center point. We also want to add the possibility to create bookmarks which allow you to go back to a certain view you have bookmarked.

Milestone 3: Layout and Navigation
In this milestone we want to be able to layout the graph as clear as possible with no overlapping nodes and minimal overlapping edges. We also want to be able to dynamically pan and zoom.

Milestone 4: Visual encodings
This milestone is all about visual encodings. We want to be able to search and filter on genomes. We want to be able to color the nodes and edges based on the genomes that it is part of. We want edges that get used by most of the genomes to be thicker than rarely used edges and we want to be able to select a genome and highlight its path.

Milestone 5: Interval annotation
This milestone extends on the bookmark functionality. Aside from being able to jump to previous searches, we also want the user to be able to annotate and highlight selections of the graph. We also want to be able to search on extra fields such as annotations. This functionality further improves the user-friendliness of our application.

Milestone 6: Semantic zoom
In this milestone we want to implement semantic zooming. Semantic zooming means that SNP bubbles and other mutations get collapsed into a smaller replacement. This makes the graph more clear when zoomed out. When you zoom all the details should be shown.

Milestone 7: Eye-candy
This milestone will mostly focus in making the application better looking and more user friendly. The things we want to do here are not defined yet.

3. Product Backlog

3.1 User stories of features

As a user
When I launch the application
I want to be able to load a Graphical Fragment Assembly (GFA) file.

As a user
When I have loaded a GFA file,
And have not yet done anything
A random subgraph must be displayed.

As a user
When I have displayed the graph
I want to be able to zoom in and pan around

As a user
When I zoom in I want the graph to semantically load the parts on which I zoom in.
When I zoom out I want small details to be merged together.

As a user
When I am looking at a subgraph
And it contains a pattern queried by the user.
I want them to be displayed.

As a user
I want to be able to put annotations on nodes and groups of nodes.

Before I close the program I want to be able to save these annotations.

3.2 User stories of defects.
As a developer
When a defect is found, I will make an issue and add it to the backlog.
Then I will determine the importance of the bug.
To decide whether I want to fix the bug or develop new features.

3.3 User stories of technical improvements.
As a developer,
When reading code,
I want to be able to easily understand the functionality of methods and fields.
This can be achieved by proper documentation and clear code.

As a developer,
I want the code to be stored on github.
So my whole team can access all the code in parallel.

As a developer,
When I push a commit to git,
I want Travis to run a maven clean test build
So that I know the project runs on a clean machine

3.4 User story of know-how acquisition
As a development team,
When we decide on features in a field we are inexperienced in,
We will read appropriate literature to gain a better understanding of that field

3.5 Initial Release Plan
In our initial release all the functionality of the must haves will be included. Also all the should haves will be included unless they are deemed not worth the effort for the result. The could haves will be implemented if all must haves and should haves are implemented far before the release date.

4. Definition of Done

Here we will give our definition of when a backlog item, sprint and a release is done.

A backlog item is considered done when:
The functionality it describes is implemented.
The code is (where possible) tested to a branch coverage of at least 75%.
The code is peer reviewed and Travis CI does not fail.
The feature is merged with the develop branch without any problems.

Sprints are considered done when:
All the backlog items for that sprint are done or not needed to be implemented anymore.
A sprint reflection has been made.

A release is considered done when a working version of the software is released containing all must-have features described in section 2.1.

5. Glossary TODO

Github: A web-based Git or version control repository.
GFA: A file extension for graphical fragment assembly files.
MoSCoW: A model for designing requirements documents. Based on prioritizing features in 4 different categories.
TB: Acronym for tuberculosis bacteria.
UI: Acronym for User Interface.
MRF: Minimum required features.
SNP: Single nucleotide polymorphism