Product planning

A planning report for the TAGC genome assembly visualizer

Group:

PL5 - Totally Awesome Genome Coders

Members:

Kasper Grabarz	4083245
Thomas Oomens	4422597
Jeffrey Helgers	4318749
Youp Mickers	4246713
Matthijs Rijlaarsdam	4308417

Deliverable Date:

28-4-2016

Abstract

This document describes the development plan for our product *TAGC genome assembly visualizer*. The product will be developed by the TAGC group and delivered by Friday, May 27, 18:55 to our client mr. Thomas Abeel. *TAGC genome assembly visualizer* is a program that parses, processes and visualizes large genome assemblies. These assemblies are stored in files that use the GFA format. This document starts out with an introduction with a brief description of the product and the context it is developed in. It then continues with a high-level product backlog description and a roadmap that contains a major release schedule and release goals. After that it shows a detailed product backlog using user stories and an initial release plan. It ends with a Definition of Done section and a glossary.

Introduction

Since DNA sequencing methods were first developed in the 1970 they found uses in many different fields such as medicine, evolutionary biology, molecular biology, metagenomics and forensics. While DNA sequencing allows us to identify the precise order of nucleotides within DNA molecules it does not help us with the process of deriving useful information based on them. To aid researchers in this process a variety of tools exist that make their life easier. The *TAGC genome assembly visualizer* is one such tool.

The *TAGC genome assembly visualizer* will be able to read genome assemblies stored in files that use the GFA format. It then uses sophisticated graph visualisation algorithms to draw this graph on the screen into a predefined viewport. The user can zoom pan and rotate through the graph and read or add annotations. Our tool will help researchers to find portions of interest and relations between other genomes.

In this report we first start with a high level product backlog and a roadmap. Then we continue with a detailed product backlog using user stories to explain the products features. We end with a *Definition of Done* section and a glossary.

Product

What follows is a high-level product backlog defined in terms of a type of user stories called epics and a roadmap. Epics are statements written from the point of the user of the product describing a particular feature he demands from the product. Epics are larger user stories that capture a general feature can then be broken down into smaller user stories that capture subfeatures in more detail.

Epics

Feature	**Epic**
Navigation	As a user I would like to be able to navigate freely through the whole graph.
Ease of use	As a user I would like to be quickly learn and use this product in an intuitive way.
Information management	As a user I would like to be able to read or add information on the screen and store it for later use.
GFA file handling	As a user I would like to load genome graphs stored in the GFA format.

Roadmap

Week	**Task**
4.1	Develop an initial design plan.
4.2	Browser gui with the ability to zoom and pan around a graph. Graph loading using a REST api implemented as a Java application running on a server.
4.3	Improved graph drawing algorithm that can be used to efficiently process and place a graph encoded in GFA format on to the screen.
4.4	Visual graph annotations and ability for the user to add and store them.
4.5	The ability to move around edges and nodes on the graph and store these changes.

4.6	The ability to seperate individual genomes from a genome graph and visualize them seperately.
4.7	The ability to connect the application to other databases and download or upload graphs and data to them.
4.8	Final product testing and improvement week.
4.9	Friday, May 27, 18:55 the deadline for the final version of the product.
4.10	Deliver a report on the project as a whole.

Product backlog

User stories

Feature	Epic
Navigation	
Zooming	As a user would like to be able to zoom in and out at random locations on the map.
Panning	As a user I would like to be able to pan around the map in any direction.
Rotation	As a user I would like to be able to rotate the graph freely.
Ease of use	
Intuitive use	As a user I would like to be able to learn the application with minimum effort and be able to rely on my intuition to interact with it.
Help facilities	As a user I would like the program to provide me with instructions on how to use the program whenever I am unsure.
Information management	
Annotations	As a user I would like to add annotations to any portion of the graph with information that is relevant to me.
Automatic information inference	As a user I would like the program to infer information from the graph based that can be be automatically derived from the structure of the graph itself
GFA file handling	
File uploading	As a user I would like to be able to load GFA files through the gui so that the program stores the graph for later use.
File management	As a user I would like to be able to use the gui to manage previously uploaded graphs and store any changes to the gui .

MoSCoW

We used the MoSCoW principle to define the requirements of our product.

Must have

- The program must have a minimap that tells the user where in the genome graph he/she is looking.
- The minimap must have to tell the user where in the graph interesting changes are.
- The user must be able to semantic zoom both on the genome graph and the minimap.
- The user must be able to jump from one place in the graph to a totally different place.
- The user must have to be able to see how two genomes are related to each other in matter of time.
- The program must have less than two hours of processing time at its first initialisation.
- The program must have to read the graph from a graph file containing nodes and edges.
- The graph must have to identify different mutations and the type of variant (insertion, deletion, SNP).
- The program must have the ability to sort by different classes of mutations.

Should have

- The program should have to create a database to store the graph.
- The program should have to calculate how many genomes go through a edge.
- The GUI should have to be user friendly.

Could have

- The program could provide visual representation and encoding of metadata associated with samples, such as drug resistance, location of isolation.
- The program could have indications for convergent evolution of variants.
- The program could be integrated with other tools, like literature and mutation databases.

Wouldn’t have

- The program wouldn’t have no processing time when the program is started for the first time.

Definition of done

Features

- All the user stories of a feature are met.
- Each feature is correctly tested by the programmer.
- A finished feature is checked by someone else in the group.
- There are no errors in the build. If so they are fixed immediately.

Sprints

- Each week a demo is given to the customer.
- The demo contains an executable file that executes the program.
- All the user stories in the sprint plan are met.
- The demo is tested for at least 75%.
- All the written tests do pass.
- The written code is correctly documented.

Release

- The product and its documentation are given to the customer.
- The GUI is user friendly.