FINAL REPORT

TEAM THUNDER WOLF SQUADRON

PROGRAMMING LIFE - GROUP 3

1. Introduction 1
2. Overview 1
3. Software Engineering Reflection 2
3.1 SCRUM 2
3.2 Tooling 2
3.3 Design Pattern 2
4. Description of Developed Functionalities 3
4.1 Visualization 3
4.1.1 Visualization Selection Options 3
4.2 Phylogeny 3
4.3 Annotations 3
4.4 Embedded Database 3
5. Interaction Design Section 4
5.1 Research Question 4
5.2 Procedure 4
5.3 Results 6
5.4 Conclusion 7
6. Evaluation of Modules and Failure Analysis 8
6.1 Visualization 8
6.2 Phylogeny 8
6.3 Annotation 8
6.4 Embedded Database 8
7. Outlook 9

1 : Introduction

The product described in this paper was created as part of a project called given by the TU Delft to second year students. Groups of five students are assigned a context which describes a problem that needs to be solved by the student teams and the real world circumstances surrounding the specific problem.

The problem presented in the context was that genetic researchers do not have an intuitive tool that allows them to visualize and compare data extracted through gene sequencing. Such a program would help them identify the genes in various genomes that contribute to problematic traits, such as drug resistance.

Specifically, the genetic researchers at Broad Institute and KwaZulu Natal Research Institute requested a tool that is meets the following requirements:

1. Enable the researchers to interactively explore a sequence graph
2. Provide semantic zooming to enable useful visual interpretation at various zoom levels from whole-genome to individual mutations
3. Put the graph in the context of the evolutionary relationship between bacteria
4. Have visual encodings for different classes of mutations and the ability to filter on mutation class (indel / SNP / large variants)
5. Identify mutations and determine the type of variant uniformly across the samples
6. Put mutations in the graph in the context of gene annotations

They also requested several features that they wished to be included into the product, but were not per se necessary. The first was that the product, provide visual representation of metadata associated with samples, such as drug resistance. Secondly, the product could have indications for convergent evolution of variants. Lastly to integrate with other resources, such as literature databases and mutation databases

2 : Overview

The product, named DNA Lab, is capable of displaying any genetic information given to it from the researcher using it. In addition to gene sequence data as input, it is also capable of loading in lineage coloring data, the newick tree data of a phylogeny, and an excel file for any relevant metadata.

The visualization is represented in the zoomed out levels as ribbons. Ribbons will be explained in further detail in section 4, as a synopsis, they are a way of simplifying visualized data. This helps to obstruct unnecessary information from the user and makes any patterns in the data more apparent. These ribbons have two levels, one where bubbles are collapsed, and one where the bubbles are expanded. The expanded level allows a more detailed visualization. Zooming in further through the expanded view will show the standard graph view with nodes and lines between them.

The product also contains a tab within the program with a phylogeny of the genomes that are present within the visualization. The phylogeny and the visualization are connected by the ability to select and deselect genomes in the phylogeny and have only the selected genomes appear in the visualization. This allows the user to only compare genomes that are pertinent to their interests at that time.

3 : Software Engineering Reflection

There are several aspects to the software engineering aspect of this project. The first is the use of SCRUM throughout the project and using it to allocate our time and regulate tasks. Second are the tools that were used for continuous integration and static analysis. Lastly, the design pattern was chosen for the project: the Model View Controller (MVC) design pattern.

3.1 - SCRUM

By using SCRUM we were able to efficiently manage our time and tasks during each of the week long sprints. During the first couple of weeks of the project we did have some trouble with our time estimations because we had never done many of the tasks that we had to do so we only had a very rough idea of how difficult they were. But as time went on our estimations became better and better as we became able to apply our past experience to the estimation as well as doing a little bit of research to quickly find if anything that could become an obstacle during the sprint. Throughout the project, tasks such as division of labor was fairly straightforward; If someone wanted to do something, they were usually assigned to it, otherwise people were assigned according to their strengths.

3.2 - Tooling

The tools that were used during the project were the tools that were recommended by it is coordinating professors. The use of these tools helped with keeping the code readable and, for the most part, free of convention errors. There were times when we disagreed with the warnings that some of the tools were presenting us, but every time that happened we were able to rewrite the code in such a way that both us and the tools were happy with the outcome. During the start of the project we had a hard time keeping the code up to date with the warnings that the tools were giving. But just like with the time estimations, as time went on we improved to the point where there were no longer any warnings or errors.

Travis : This tool allowed for a continuous build status to appear both on the travis website and the github page, letting anyone who saw how well tooling warnings were handled and if tests were passing.

Checkstyle / PMD / FindBugs : These two tools focus on the same area, finding defects in the code. PMD and Checkstyle are used more for finding defects with convention and formatting, whereas FindBugs finds defects with how the code will actually run. Both of these tools were used for static analysis of the code to keep it clean.

Coveralls / Cobertura : Both of these are code coverage tools that show how fully the code is tested. Originally only Cobertura was in use, but later when the SIG requirement came into play, Coveralls had to be implemented as well.

3.3 - Design Pattern

The design pattern that was chosen was the Model View Controller design pattern. Using this design pattern, a well-designed and maintainable code base was created. This design pattern was chosen mainly because it was the design pattern was found most useful during the Software Engineering course. This is because it is designed in a way that makes adding additional functionality to the program

follow a basic workflow. It also helps separate out the different aspects of the project, which is useful for avoiding incorrect coding practices.

4 : Description of Developed Functionalities

4.1 : Visualization

Of all the functionalities, the obvious frontrunner would be the actual visualization itself. This functionality has several aspects to it. The first is the collapsed view where all low level bubbles are collapsed before the visualization is displayed on the screen. The second is the expanded ribbon view. In this view all bubbles are expanded and the data is represented as ribbons. The third is the normal graph view that consists of nodes containing sequence data connected together in a graph structure. Transitioning between these levels is possible by zooming in and out in the visualization, this represents our semantic zooming functionality. At the same time navigating through the visualization is possible by both zooming and dragging the screen to move through the data visually. Each separate level of the visualization has lineage coloring as well. This is calculated by the majority of genomes that pass through either a ribbon or a node.

The vertical spacing of both ribbons and the nodes on the graph represents how 'popular' that path is, which is also represented by how thick the line is. The further away from the norm, in other words how long the ribbon is or how far away the node is vertically is an indicator as to how many genomes pass through that specific ribbon or node. The higher or lower vertically it is, the less popular it is and the thinner the line will be.

4.1.1 : Visualization Selection Options

The visualization also has two selection options. The user has either they choice to select and highlight all of the SNP's in the visualized data, or select and highlight all of the indels in the data. Either selection will turn everything that does not fall into the selected category grey, while the selected items will maintain their lineage colorings.

4.2 : Phylogeny

Any patterns behind the colorings become more apparent with the use of another one of the programs functionalities: the phylogeny. This functionality shows every genome present in the provided dataset from the given .nwk file. Just like the visualization, lineage colorings are present here as well. By using the phylogeny the user is able to select and deselect the genomes that appear in the visualization. This allows for a certain degree of customization within the visualization. Additionally, hovering the mouse over a genome name will show all metadata about that genome in a tooltip.

4.3 : Annotations

The program also implements an annotation viewer, which allows the user to see the annotations inside the representative genome from the dataset. By hovering the mouse cursor above a box in the annotation viewer a tooltip will appear with the name of the annotation. Then, by using the lowest level view, it is possible to see where the annotation is inside the visualization directly above the annotation viewer.

4.4 : Embedded Database

These functionalities were able to come to realization thanks to one of the other functionalities, the embedded database. The base for the embedded database was created by the H2 Database Engine,

which was then used and applied to the product. Thanks to this database the program can take in much more information than would be possible with other alternatives. This is because it is all parsed into the database for storage instead of using the memory that is provided to the JVM. This also means that once data is parsed into the database, it will remain there even if the program is closed; allowing preprocessing to only occur once for every dataset.

5 : Interaction Design Section

5.1 : Research question

For the interaction design part of the Context Project, we set out research the usability of our application. We want our application to be easy to use and not force users into a complex learning curve before they can properly make use of it. Furthermore, we want it to work fast, so that users do not waste time waiting for application to finish processing data, and we want there to be enough functionality to satisfy the user's needs.
We set up the following research question: "Does the usability of our application satisfy the average user's needs?". To answer the research question, we followed the guidelines of the Empirical method "Experiment" as discussed in the lectures.

5.2 : Procedure

Test goals and test plans
We decided to conduct a summative evaluation, since the product was almost done. Also, we chose to test the whole system. For this it was made sure to choose tasks that would cover all functionality of the system. Use case scenarios were created to help setting up said tasks. To observe the behaviour of the users during the experiment, it was decided one person would sit with them. This person would hold a think aloud interview with each user during the experiment (but not during the execution of a task, so as not to influence the time measurement). It was also decided to simply use the real application for testing, since this would provide us with the most reliable results.

A pilot test was performed by our members to see if the tasks were feasible enough for new users to complete. A debriefing was planned after performing the test tasks with the users.
The time planned for one cycle through all test tasks was 10 minutes. All equipment used was one of our standard TU Delft laptops. As for the budget, none was available, though none was required.

Getting test users
The best users to test an application are those that work in the field associated with, or corresponding to, the field of the application itself. We contacted 5 Biology Master graduates from Leiden University. Additional background information (fake initials were used to not harm their privacy):

- A female Biology teacher (N)
- A male DNA researcher with also a master in Computer Science (B)
- A male DNA researcher (R)
- 2 male Biology teachers (K, J)

B and the male teachers (K, J) are also fervent online gamers with quite some experience with computers. The female teacher (N) and the other DNA researcher (R) have less encompassing knowledge about computers.

Use case scenarios

- As a user, I want to be able to load new database files into the application.
- As a user, I want to know the controls of the graph and the meaning of the symbols and colors I see on screen.
- As a user, I want to filter on specific genomes.
- As a user, I want to have specific metadata on a genome.
- As a user, I want to zoom to a specific node and acquire its DNA sequence.
- As a user, I want to be able to identify mutations.

Test tasks

Task ID	Task description
1	Load a new database file.
2	Find the help menus of graph and control.
3	Have only the first genome of the phylogenetic tree displayed in the graph.
4	Have all LIN2 genomes of the phylogenetic tree displayed in the graph.
5	Name the drug resistance of the first LIN4 genome to Streptomycin.
6	From the top level view, zoom in on the first graph node.
7	Copy this graph node's nucleotide sequence.
8	Highlight and identify a SNP.
9	Highlight and identify an Insertion.

Task ID	When task considered completed
1	When application starts loading new database file.
2	When cursor is hovering over the graph and control help menu items.
3	When only first genome is loaded into the graph.
4	When only LIN2 genomes are loaded into the graph.
5	When user has named the resistance (S or R).
6	When lowest level zoom on first node is achieved.
7	When Nucleotide sequence is copied into the clipboard.
8	When cursor hovers over a highlighted SNP.
9	When cursor hovers over a highlighted Insertion.

Test procedure

First a pilot test was performed to test the feasibility of these tasks. It was concluded that all tasks were divided up enough in sub-tasks and likely to be completed within at most 30 seconds.

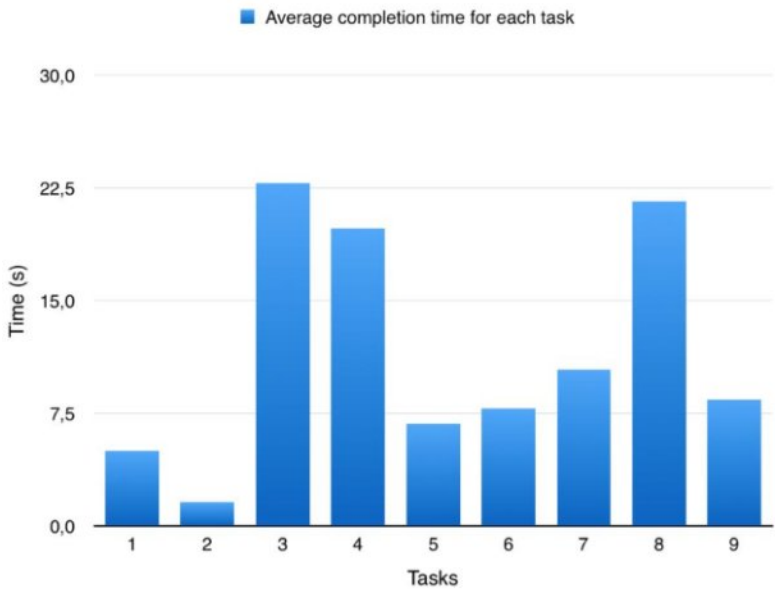
All participants were given a small briefing beforehand. These included some technical terms (like SNPs, Insertions) and the purpose of our application. We were careful not to spill too much information, so as not to ruin the experiment's test for intuitiveness.

The participants performed the test tasks separately; only one of our members was allowed to sit with them. A debriefing was held afterwards.

5.3 : Results

Test task completion times

	Participants' time taken (seconds)					
Task ID	**User "N"**	**User "B"**	**User "R"**	**User "K"**	**User "J"**	**Average**
1	5	4	5	7	4	5,0
2	2	1	2	2	1	1,6
3	28	20	25	17	24	22,8
4	22	16	19	19	23	19,8
5	6	6	8	6	8	6,8
6	8	7	8	7	9	7,8
7	12	8	7	10	15	10,4
8	21	16	27	20	24	21,6
9	9	5	8	13	7	8,4



Think aloud interviews feedback

We received the following feedback and ideas from the think aloud interviews with all 5 participants:

- The names of the menu items to open new databases are not logical.
- In the phylogenetic tree view, it is cumbersome to search for the lineage of a genome in a huge list of tooltip text.
- It is in fact cumbersome to search anything in this tooltip. The tooltip itself is handy, but there should be other ways to easily access the metadata.
- The nodes in the phylogenetic trees are hard to click, because they are so small. Also sometimes one click does not seem to be enough (might be bugged).

- Semantic zooming is confusing. When zooming in, the participants lost track of orientation.
- It is cumbersome to click on every node separately to see only a portion of the nucleotide sequence that is stored inside. It would be easier to see the entire strain when hovering over with the mouse.
- It is unclear how many nucleotides are in each node, when all nodes are exactly the same size.
- It would be nice to find certain sequences of nucleotides and repetitions thereof.
- Sometimes one wants to view both insertions and snps. Alternative symbols for these mutations would also be handy.

5.4 : Conclusion

The test task execution times did not exceed our initial expectations, which was that no task would take longer than 30 seconds. Nevertheless, there were 3 tasks that approached this upper limit, namely those with Task ID 3, 4 and 8. From the feedback, we can gather that 3 and 4 took relatively long because people had trouble clicking the nodes and accessing the metadata. 8 took long because people had trouble understanding what represented a SNP and where to find the option for highlighting. Once found, task 9 obviously went faster. As for all the other tasks, they were completed relatively quickly. There were some minor issues, which are all discussed in the feedback of the think aloud reviews.
Generally speaking, the users found the application to be rather intuitive. It was not hard to figure out how to do the tasks, thus they do not think there is a steep learning curve to overcome to make proper use of our application's functionalities. It is just that the application's functionalities could use some refining.

There are some limitations to this experiment. For instance, we only tested this on 5 people, only two of which are actual DNA researchers. Every person has different tastes and different requirements and our 5 participants do not necessarily represent the average user. Also, "DNA research" is an incredibly broad term. DNA researchers specialising in different fields could have radically different requirements and preferences for a DNA analysing application. This experiment should thus only be taken as a guideline for some general improvements to usability and clarity, not as the holy grail to create the perfect DNA analysing application.

From the test results and feedback we acquired, we can definitely derive some design changes. We thought of the following;

- The "Open Existing" and "Open Recent" should be under a separate tab named "Database", so it is clear that these options are about databases.
- Give a separate legend for the lineage colour in the phylogenetic tree.
- Make a separate tab for the genome metadata; the tooltip alone gives for too much hassle to track down specific metadata.
- Create a search function for genomes in the phylogenetic tree and for metadata.
- Allow for filtering on specific metadata in the phylogenetic tree view.
- Make the phylogenetic nodes larger, so they are easier to click.
- Make semantic zooming to be determined by the cursor's position, not by the scrollbar.
- In the graph, add tooltips to graph nodes that are longer than 5 sequences.
- Make the graph nodes' size correspond to the amount of nucleotides in them.
- Add a search function to be able to search for a certain sequence of nucleotides.
- Add a minimap, so semantic zooming does not confuse your sense of orientation.
- Give mutations dedicated symbols to more clearly represent them.

6 : Evaluation of Modules and Failure Analysis

6.1 : Visualization

First, the visualization module is among our most polished due to its prominence in the program. Both the user interface aspect and code structure of the visualization were where our team spent the most time by far. The ribbons are well done and serve their purpose in simplifying the visualization. The fact that they also have the lineage coloring makes them both more visually appealing and more meaningful to the researchers using the program. The spacing and varying thickness of the ribbons is something that is very nice to have present because it makes it easy to find even more meaning from the visualization.The graph is not anything spectacular because it is the standard. It does have different visual indicator for things such as path popularity and node sequence length, but that is the extent of it.

For the graph, we would have liked to make each node interactive and give more information about itself, but failed with this wish due to time constraints created by other more important implementations. This problem is the main cause for our failures to implement several of the functionalities that we had originally wanted to have.

6.2 : Phylogeny

The phylogeny is, much like the graphical level of the visualization, fairly standard. Something that is different than some other groups is that it only shows the genomes that are present in the visualized data. This is nice because it hides superfluous information from the user and simplifies the program. The phylogeny has interactivity with the visualization in that you can use the tree to select and deselect genomes to be visualized. We chose this instead of a list or checkbox style because it simplifies the process for the user and is more visually appealing than either of those alternatives.

There was a failure here however because in the beginning, it was envisioned that the user would be able to search for a genome or gene from the phylogeny. This was also something that was requested from the client, but due to time constraints, as well as having to fix functionalities that had bugs in them, it was not able to be implemented.

6.3 : Annotation

The annotation view is capable of showing the user where the annotations from the representative genome appear in contrast to the other genomes that are being visualized. Originally, the annotation view was only based on the percentage of the genome that could be viewed on screen. So if the user was 50% into the visualization at the lowest level, the annotation view would also be at 50%. But the client explained that this was not satisfactory, so it was changed to instead be representative of the position of the start and end coordinate from the annotation. This means that annotations line up with the nodes where they stop and end.

6.4 : Embedded Database

The embedded database is well implemented and works almost better than expected. The embedded database was chosen as opposed to say, saving data into a text document or using xml parsing to save data, because a database can be queried. The querying functionality is very useful because we have to do a fair amount of specific lookups during both the preprocessing and visualization sections of the project. The reasoning behind choosing an embedded database is because it is easier to meet the requirements to run the program as minimal as possible. By using an embedded database, the program can be run on any capable computer, anytime, anywhere, without setup being needed.

7 : Outlook

For future implementations, developers (or ourselves) would be able to use the model view controller design pattern to implement any additional major functionality. The code itself is well documented, so even if there is no formal explanation of it, even outside developers should easily be able to understand what does what and why.

As for functionalities that would most benefit the product, having more interactivity with the graph level of the visualization and the user would be number one; perhaps the ability to search for specific nodes in the graph and then be able to select them and view more detailed information about that specific node. The implementation of this would not be terribly hard in theory. Each node would need a unique id that the search function can use to locate it, and then each node would have to include a listener that would either listen for mouse hovering or clicking and then display the information accordingly.