**Final Report**

*Group Health informatics-2*

*Rick Proost – 4173619*

*Pascal Remeijsen – 4286243*

*Wim Spaargaren – 4178068*

*Arnout vd Knaap – 4223969*

*Jelmer de Boer - 4223152*

Contents

1. Introduction
   1. Preface
   2. Problem description
   3. Requirements
2. Product Overview
3. Reflection from a Software Engineering perspective
   1. Product
   2. Process
4. Functionalities
5. Interaction Design
6. Evaluation
7. Outlook

1. Introduction

1.1 Preface

This document will provide an overview of our project, and give insight into our product as well as the working process used during the project. We will also reflect on the product and process from a software engineering perspective. After that we will describe the developed functionalities, evaluate what works as planned and what does not, and have a section on Interaction Design. In that section we describe how we determined the level of user friendliness, and what we did with the findings. Lastly there will be a outlook on the entire project and discuss what could have gone better, and what the successes of this project were.

1.2 Problem Description

The problem that our group faced in this project was making a program to assist researchers in analyzing and visualizing data. There is a variety of programs to do static analysis on a data set, but data is not always in the right form to use it in other programs. So a solution had to be found, which enables the user to change the form of the data, and also allows the user to do analysis and visualization on that data. The program has to be able to answer a lot of questions, but in case the program can’t handle the question, you can use the tool to transform the data in such a way that you can answer it with another tool. This way, we can empower users to more easily get answers to their questions.

1.3 Requirements

The most important requirement for the end-users is that their questions can be answered by our product. If this is not the case, the data can be formatted so you can use another tool on it. To be able to use multiple forms of input, one of the requirements is to have an editor to specify the structure of the input data. This data specification will be described with a XML file, and users should be able to safe and load the file. There also has to be an editor in which user can specify sequential data operations that need to be carried out on the data set. The data operations that should be supported include : Chunking, Comment, Code, Connect, Compare, Compute and Constraints. Another requirement is that the program has to be able to visually represent the data. Data should be representable with amongst others : Frequency Bar, State Transition Matrix and Box Plots.

2. Product Overview

In the end we have managed to implement all must-have requirement with the exception of specifying the exact format of the output data. However with most tools that users could want to use after our program like SPSS, the form of the data is irrelevant as long as the data is structured, which our program provides. At the start of the project we expected a lot of formatting was necessary to prepare the data for other tools. However, only a small formatting effort was needed, because tools like SPSS can handle loosely structured data.

The final product has a small but powerful graphical user interface which enables the user to load in data, analyze the data and output the data in a file or on screen. We focused on making the graphical user interface as light-weight and simple as possible. The user interface consists of three tabs :

* 1. Input Tab

In the input tab, users can create projects, load files into a project, select which of the loaded files they want to analyze and specify the structure of the input data. This tab also features a help section to guide the user through the process.

* 1. Analyze Tab

In the analyze tab, users can enter queries which are structured according to the language of our product. This language is specified in the help section. This tab will also show the intermediate results of the queries, and show the performed queries in a “used code” section.

* 1. Output Tab

In the output tab, you can see the last resulting data after the execution of the queries. You also have the option to visualize the data with multiple types of graphs, and the user can also export the data.

The product can use .txt files, and .xlsx files as input for the program. The user is provided with a XML-editor, so they can specify the structure of these input files, to allow for as many differently structured files to be usable for our application. With our application they can transform the inputted data sets and through queries they can answer question they have about the data. The user can also utilize multiple kinds of visualization for the data, they gather through the queries. The options include :

* + Box Plot
  + Frequency bar
  + State-Transition matrix
  + Histogram
  + Line-Chart

3. Reflection from a Software Engineering perspective

In reflection on the project of the last few weeks, what stands out the most was the use of libraries. Before we code something ourselves, we should have made sure it hasn’t already been done for us by someone else. The rule of thumb is: am I the only one in the world who has faced this problem? If the answer is no: search for a library that can solve the problem for you for about half an hour, before you start coding. This came to our attention after one of the meetings with our Software Engineering TA. He showed us some of the libraries he would have used for the project instead of what we were doing. When we took a look at those libraries, we all thought: we could have done this so much better if we took the time to look for a library.

Something else that took up a lot of time was the use of the database. The database always keeps the values of previous runs. This makes testing using the database fairly hard. The wrong data stays in the database or tables, especially when tests fail because a previous test already called them. Since our project uses the database in a lot of tests and the error is not thrown where the table is not dropped. Debugging tests is very hard. In future projects using a database, we might want to mock the database where possible, or make a Database unitTestClass that handles the database automatically.

The development process of this project went well. We all had used scrum before, so everybody was clear on what it is, and what to do. The only thing we didn’t do correctly was dividing the problems into small enough tasks in the beginning of the sprint. It is also hard to put small tasks on the planning in the beginning of a project, because of the initial architecture design. Later on in the project this was going a lot better. The only thing that we would do different in the next project is building the language. The language was now built in such a way that we could express everything we had made. However we should have built it first and then changed the code to answer all questions the user wants to ask. We should have built the application top-down, where we built it bottom –up. This would have been a lot less work and lets us show the progress of the application to the user. This project, we could not show working examples in our GUI when we needed to present it.

4. Functionalities

In this section we will give an overview of the implemented functionalities of our application.

* Creating or remove projects 🡪 The user is able to create or remove a project. This project is a map in which the user can load files, so the files belong to a certain project.
* Select a project 🡪 A XML file is stored per project, so by selecting a project you change the file in the editor.
* Load in a file 🡪 When users select a project, they can click the “Open File” button to load in a file. These files will be saved, so you can load in multiple files in one session, and the next time the program is used those files can be used again.
* Select a File 🡪 After the files are loaded in you can select them to be analyzed. This way you do not have to remove a file if you don’t want to analyze it, you can just deselect it.
* Specify structure of data in XML editor 🡪 This includes multiple possibilities. The XML editor will show a list of files from the selected project, and per files you can add and remove columns. In a column you can specify name, type, column id and more.
* Ask for help in input page and analyze page 🡪 By clicking the help button, a menu will appear in which all aspects of the respective page will be explained.
* Create Documents 🡪 Under the XML editor there is a button to add a file. A new section in the XML editor will appear where you can specify a path to the file. Then you can immediately specify the other required fields in the XML editor.
* Save XML file 🡪 When users have specified the XML file in the XML editor, they can save the file, and use it in a later session.
* Run queries on the input data 🡪 The user can enter queries into the “code input area” which will be executed when you hit enter or when you use the “analyze” button. These queries will execute operations on the data which was selected in the input page.
* Overview of used code 🡪 When the user executes queries, those queries will be showed in the “used code area” so the user has a nice overview of executed queries.
* Undo queries 🡪 When the user wants to go back to the previous state of the data, you can use the “undo” or “revert” operation, do undo the last query. When users wants to go back to the initial state of the data, they can use “undoall” or “revertall”.
* Show intermediate result 🡪 when a query is executed, the data set will be updated and shown in the intermediate result section.
* Save Script 🡪 When the user clicks the “save script” button, the executed queries that are in the “used code” area will be written to a text file. This way users don’t have to retype it every time they want to use a longer sequence of queries multiple times.
* Load Script 🡪 When the user clicks the “load script” button, they can select a text file which contain a list of queries. This can be a saved script from a previous session.
* Export File 🡪 The last state of the data will also be explained in the output page. Here the user can click the “export file” to write the result to a text file.
* Visualization 🡪 The user can select one or more types of graph to be made of the data.

5. Interaction Design

This will be in the final version.

6. Evaluation

The resulting application of this project, contains all of the must-have requirements. After the must-haves were implemented we also managed to implement several of the should- and could-haves, resulting in a well-rounded product. However, the way in which the user can use these operations, differs from the initial plans. Especially the language has turned out different than initially planned.

In the chapter 4 we discussed which functionalities were correctly implemented. However some of the initially planned functionalities aren’t in the final product. Namely :

* Histograms 🡪 These were not must haves, so due to time constraints this was not implemented in the final product.
* Conversions 🡪 Due to the definitions not being completely clear, in combination with time constraints are the reasons this was not implemented.
* Output Formatting 🡪 The data can be changed by the queries, but it cannot be formatted by the user. This was a feature we would only implement if we had plenty of time. This was however not high priority, and it didn’t make it into the final product.

When implementing features for this project, we tried to make everything as general as we could. We wanted to ensure that researchers outside of the ADMIRE project would find our product useful. However, the initial clients were the researcher for ADMIRE, so there are a few specific features like the algorithm to generate advice according to Creatine values.

In terms of failures as a consequence of the choices we made in the design and implementation process, we have encountered a negative consequence :

* Because we chose to use a combination of database and java objects to represent our data, the project is not suited to handle multiple patients. Looking at multiple patients will produce a longer waiting time for the analysis. Therefore we advise the user to use the pre-filter option to load the data of only one patients.

7. Outlook

After eight weeks of hard work we have finished working on our data-analysis tool. We are proud of what we have accomplished during this project. We have a tool that makes it easy for researcher to analyze data in a simple way.

However, there is no such thing as a perfect project. There are a few aspects which could have been better. Firstly, the graphical user interface could have had a nicer look. At companies where professional software product are made, they often have a big design departments. Sadly, none of us had a lot of experience with visual design, so the end result could have looked better. Although our interface is not the prettiest, we have managed to make it quite simple and easy to use.

A big lesson we have learned is that in our next project we should work in a user-oriented way. The first few weeks work were focused on functionalities that work behind the scenes in our product. However, this has as a consequence that the user can’t see the progress that the team has made. We should instead focus on something we can show to the user, and develop this from start to finish. That way you can show something new to the user at every meeting.

If we were to make another data analysis tool, we would not use a database again. Not because it was not useful, but because we did not realize testing with a database is a nightmare. If you forget to drop a table somewhere, a lot of tests will go wrong, and it is not immediately clear where the problem originates. However we still stand by our opinion that a database-oriented solution did a lot for us, and allowed us to focus on other aspects of the product.

A point of critique on our project, could be the low number of libraries used. Because the group members of this team are so eager to learn new things, we wanted to make as much of the product ourselves. However, when you are working on maximizing the quantity and quality of the product in a short period of time, a library could have saved some work. This way we could have had more time for other projects. This was another important lesson for this team.

Lastly we could have planned a little bit better. In retrospect we did more work in the second half of the project, relatively to the first have. If we had planned better we could have had a better distribution of work over the weeks.

Luckily, we also did a lot of thing right. Especially the cooperation was very pleasant. There was a very democratic system of deciding what we were going to do. There was a productive yet friendly ambiance, and everybody learned from others in the group. The software examination showed that our design of the system was pretty good, and we used multiple design patterns. And most importantly, we delivered a product were happy with.