TI2806 Contextproject

Final Report
Group HI4

June 18 2015

Contents

1 Introduction

1a Problem description

Chronic kidney disease is seen as a major public health problem that is still growing worldwide (National Kidney Foundation, 2002). At the end of 2013, the number of end-stage renal disease (ESRD) patients getting treatment was around 3.2 million worldwide (European Renal Care Providers Association, 2015). This number has been growing every year around 6At this moment renal transplantation is seen as the best treatment available for patients with ESRD. Multiple studies have proven that the patients survival and quality of life is better with renal transplantation than with dialysis (Weir and Lerma, 2014). However, complications such as rejection and side effects can occur after transplantation (Thomas, Kanso and Sedor, 2008). That is why it is very important that the health status of renal transplant patients gets monitored daily.

The ADMIRE Project - organized by LUMC, TU Delft and TNO - introduces a disease management system for patients self-monitoring. This includes a new home-based medical device for measuring the creatinine and blood pressure levels and a website for feedback on the patient's health status. The data set consists of measurements and website data of a group of 50 patients in the Netherlands. Researchers who are interested in the health status and the behaviour of patients performing self-monitoring need the dataset to be pre-processed for further statistical analysis. The data needs to be analysed sequentially per patient in order to discover any events or use patterns. Therefore we have created a tool which can be used by highly-skilled researchers to analyse the behaviour and possible complications of renal transplant patients. Even though researchers in the field of renal disease are our main target customers, the program is generic enough to be used by any researcher from other disciplines to analyse large data sets and pre-process them for statistical analysis.

1b End-users requirements

The customer wants a standalone software application which is able to do the following tasks:

Read in different kinds of files from three data sources: the portable measurement device StatSensor, the measurements entered on the website and hospital administration.

- Link those files together based on their primary key(s) (e.g. patient ID or timestamp)
- Sequentially analyse the files by performing transformations on their data elements
- These transformations can be specified in a scripting language in an editor
- Visualize the results of the data analysis with graphs
- Export the results in a particular format that is suitable for further use in statistical software applications and save it in a directory specified by the user
- It should be possible to perform all these tasks in a graphical user interface

2 Overview of the developed and implemented product

This chapter gives a high level overview of the product. Paragraph 2.1 gives an overview of the design goals we took in consideration. In paragraph 2.2 the product features are explained per subsection of the program and in the last paragraph the architecture of the system is mentioned.

2a Product attributes

When developing the product we took the following design goals in consideration:

- **Generic:** The program is made as generic as possible, so that the program is also suitable for different forms of sequential data analysis in different disciplines.
- **User guidance:** The interface of the program is graphic, easy to use and consistent. The program provides user guidance during all processes within the application by giving status information for the systems state and the users activities. This is done with explicit feedback about the users input (with error messages) and timely information about the status of the data analysis.
- **Quality of product:** To achieve a high quality we kept the code clean, documented and well tested. We also built the architecture of the program in such a way that the code is easily maintained.
- **Performance:** The data analysis is performed within a reasonable period of time with a limit of half a minute. The program has a responsive design with error management including error prevention and error correction (with error messages) implemented in all layers of the architecture.
- **Scalability and flexibility:** The program is able to process large sets of data with sizes in the gigabyte range. It can handle different types of files and different delimiters.

2b Product features

The program is split up in four different subsystems which represent their own functionality:

1. Import
The program is able to read in different types of files with different delimiters. Raw data from the Stat-Sensor measurement device, Mijnnierinzicht website and hospital appointments data can be imported easily. There is also a possibility to specify the structure and group configuration of the data files by importing an XML file. The user is also able to store this specification file and read in again.

2. Specify
A script editor is implemented in which the user can specify sequential data operations that need to be carried out on the data and to specify codes (labels) that could be linked with observed patterns . Script files created by the user can also be imported and interpreted in the editor.

3. Analyse
The user is able to transform sequential data analysis using the eight C's discussed by Sanderson and Fisher (1994), which are:

- Chunking algorithm: Automatic breaking up that event sequence for specific time period to analyze frequency specific codes in separate blocks.
- Coding algorithm: Automatic coding of behavior patterns and re-coding of event sequences from high frequency events domain.
- Conversion algorithm: Automatic generating behavior of web site response in order to creating new input stream for analysis.
- Pattern algorithm: Automatic data synchronization whereby events recorded in different data files are merged into single data file with right event order.
- Constrains algorithm: Automatic filtering data for certain event codes, or time periods.

- Comparison algorithm: Lag sequential analysis (LSA) for identifying dependency (temporal relationship) between events.
- Computing algorithm: Perform computations such as count, sum, maximum/minimum and average on the data.
- Commenting algorithm: Add comments to certain data elements, data chunks and conversions.

4. Result

The results of the data analysis can be saved in a specified directory and into a format that is ready for further statistical analysis. To support the researcher with the exploration of data the following visualizations are implemented:

- timeline
- frequency bars
- box plots
- stem and leaf plots
- two-dimensional time series
- state transition matrixes
- histograms

The user is able to select which part of the data needs to be visualized and can also store the graph as a picture and as a PDF file.

2c Architecture

We implemented the application following the Model-View-Controller (MVC) pattern to separate the backend logic from the frontend that represents the program. This enables us to divide domain objects from the GUI elements to keep the code cleaner and the system more maintainable.

2d Reflection from a software engineering perspective

In this section we will reflect the product and process from a software engineering perspective. During the project we have used the scrum methodology. Every sprint started on Monday and ended on Friday with a demo for our stakeholders. The goal was to deliver a working prototype every week. At the end of every sprint we had a working prototype, but sometimes it was not possible to integrate all the functionality in the graphical user interface. This is why we could not always show the features we had implemented during the demos on Friday. Later in the project we planned at least 3 hours every Thursday to integrate newly implemented functions with the interface.

Helping us with continuous delivering our product in this project are git (hosted by GitHub), maven and Travis-CI. Every time you pushed your code to the server the latest commit from each branch is built by Travis-CI. Travis-CI checks if the project can be built and runs all tests. If building is not possible, the build fails, you know that the product is not working as expected. This was quite helpful during the project. If somebody forgot to run the tests on the code you see immediately when this problem was introduced. So we continuously tested the whether the old functionality is still working.

We also used some tools that monitored our code: Cobertura for test coverage, Checkstyle for code formatting and PMD and Findbugs to track potential bugs. These helped us greatly to keep the code clean and readable. Especially Checkstyle is very useful for writing consistent code.

During the project we used the pull-request-based git workflow. This means that when you have implemented a feature and you would like to add it to the product, you create a new branch for it. Before merging it with the master, you have to make sure that the code is sufficiently tested and that it satisfies the code guidelines which are explained in the fourth section of the Product Planning. If you are sure that your changes meet all the described requirements you can do a pull-request. Before merging the code with the master branch, the pull request has to be peer reviewed by at least two other team members. We also used a new tool Octopull developed by a master student, this tool shows the results from the code monitoring tools in the pull-request. This is very useful, if you havent used this tool you have to fetch the branch first, build it manually and inspect the reports, a task that is very time expensive.

For more information about the product and the design choices we made, you can read the Emergent Architecture Design document.

3 Description of developed functionalities

In this chapter the main functionalities of our software application are described. For each of the four subsystems a description is given of what functionalities it brings to the product. The first paragraph will look at the functionalities relating to importing files. The second paragraph focuses on the part of the software in which the subset of the data can be selected that should be analysed. In the third paragraph we cover all functionalities with regards to the actual analysis and in the final paragraph the results of the analysis.

3a Import

The software enables the user to import data in the form of comma separated value files. To make the import of different datasets easier, a group can be created for each dataset. Within a group, one or more files can be added that uses the same columns and delimiter. For the selected group the software gives an overview of the imported files and shows a preview of the content of the selected file. While importing the files, the software automatically detects the number of columns and adds for each column the possibility to specify some information about that column. The user can specify a name for each column which could be used later on to access the data of that column. By toggling a checkbox, columns can be included or ignored. Also the data type for each column can be specified. This is either integer, double, time, date, date and time, string or a special string namely a comment. When the user specifies a date column, one of the supported date formats can be selected. If a user accidently filled in the columns in a wrong order, this can be solved by dragging the columns back in the right order. Since the data needs to be analysed sequentially, the user can select which columns should be used for ordering the data. To specify the sorting, the user can select a column containing both date and time or just a date column (with optionally a separate time column). When all groups, files and columns are specified the current settings can be exported to an xml file in order to save the analyst some time the next time he starts the program. In that case the user can browse to the xml file containing the desired configuration or simply select it from the list of recently used configurations. The analysis should be performed on a number of subsets of the whole dataset. In the case of the ADMIRE project the data can be divided in a subset for each patient. The user is able to select for each group of data files how the software can know to which patient this data belongs. This can be done by selecting one of the columns from a dropdown box. If instead of a column name the user selects the option: file name, the software will use the different file names in this group as a way of dividing the data into portions for each patient. Additionally a regular expression can be entered to inform the program how the patient identifier can be extracted from the file name.

3b Select

After defining all groups, the software automatically reads through all the data. Each line of one of the files is assigned to the dataset of the corresponding patient. In the selection component the user can specify for which patients the analysis needs to be performed. A list of all patient identifiers is shown, together with the amount of data that is present for that particular patient. This is expressed in a number of rows and columns. To quickly find some particular patients a search box is present in the software. Only the patient ids that do contain the character sequence given in the search box are shown. To make selecting patients easier a check all, uncheck all, check search result, uncheck search result, check selected patients and uncheck selected patients button is added. Some of these shortcuts can also be accessed by selecting it after right clicking somewhere inside the patient list.

3c Analyse

Once a number of patients are selected, the user can start the analysis. A script editor is introduced to enable the user to type the script directly into the software. By doing this the user can easily see whether he made a typo or syntax error. For example all accepted keywords regarding the operations are displayed in purple, recognised column names in yellow, comments in green and text strings in blue. To help remembering the specified columns, a list is shown containing all column names combined with their data type. Also a dropdown menu is present containing all possible operations. If the user clicks on one of the operations the item expands and a general explanation appears, together with the required syntax and some examples. If the user prefers to use an external editor instead or to open a previously saved script, the software enables him to browse to a script file or to select one file from list of recently used scripts. The script will be executed in a pipelined fashion. The first line will be executed on the input data. The result of that will be passed as an input for executing the next line of code. An exception to this is the usage of variables. A line of code can be assigned to a variable. In that case the next line will be executed with the result of the previous line instead of the line containing the variable assignment. The pipelined input can also be overridden by specifying that a previously defined variable should be used. If a variable contains a single value (for example after calculating an average) the variable cannot be used as an input, but can be used inside a line of code. This can be useful for selecting values that are larger than the average.

3d Results

After the analysis is completed the software can show the results in many different ways. The basic view is a text editor. In this the analyst can go through the raw result data, possibly make some changes and then save the result to a file. The software can also show the result in a table view. In this view the analyst can look through the data having a better overview. It is possible to sort the data on any column or on a combination of columns. With a click on a button the current table view can be exported (while maintaining the current sorting) to the text view or be saved to a file. The software also enables users to explore the results further with the help of graphs. The user can create a boxplot, line chart, bar chart, stem leaf plot, histogram, frequency bar and a state transition matrix. This can be done on the whole dataset or if the provided script divided the data into chunks, the graph can be made for the data of each chunk. For each graph the columns that should be used on the different axis can be specified. Based on the defined column data type the software recognizes that some columns should not be selected, so it prevents the user of selecting wrong data. Besides just plotting a line graph of one numeric column, the software also enables the user to combine different graphs into one. To give an example, both the measured creatinine levels and the values as entered into the website can be plotted at the same time. Now the analyst can instantly see whether there are any differences between those. To make it possible to use the graphs for other purposes, the software can export the graphs as a SVG image.

4 Interaction Design module

This chapter describes the tools used to make AnalyCs as user friendly as possible. In paragraph one a persona is given. This persona describes a stereotype user of the program. The second paragraph gives a plan for acceptance testing. The third and last paragraph concludes and describes other things that could be done in future projects or when extensions of the product are needed.

4a Persona

This paragraph gives a stereotype user of AnalyCs. Let us introduce Paul. Paul is a data analyst. For months he has collected data from the ADMIRE project. First a short introduction on the ADMIRE project. The ADMIRE project is a project that tries to make life easier for people that have gotten a kidney transplantation. A website has been created where those people can enter their creatinine level. Based on this information advice is given to the patients, on whether they are healthy or whether the should visit a hospital soon.

Back to Paul. A lot of data has been gathered from the devices that are used to measure the creatinine levels, from the website where the patients entered their levels and data from the hospital visits. Now Paul wants to get some useful information from this data that has been gathered. For instance, he wants to know whether users have listened to the advice that the website has given them, or whether the patients tend to fill in data in the website after one week, or if a patient only enters measurements that they are satisfied with themselves after measuring multiple times.

And the questions just keep on coming, but this gives Paul a problem. He has a lot of data, but no way of structuring it in such a way so that he can use it in his expensive analytical programs. He would really like to have a program that is able to group the data for every person. Then he wants to create relations in the data, for example a relation between the time a patient has measured his creatinine level and the time the patient has entered the result in the website. Or he wants to see a graph that shows him the times that a patient performed an action. Now he needs to do those things manually, which costs him a lot of time that he prefers to spend on using his analytical tools which are way cooler to use. And of course this gives him results sooner that he then can write a paper with which he can publish. And that is the part of the research that Paul likes most. Not forming data or transforming it, but looking at it and drawing interesting conclusions from the data. That is why Paul requires us to create a tool that transforms the data for him.

4b Acceptance testing

This paragraph describes a plan for acceptance testing. After the software is finished users should be able to use it. For this an acceptance test is required. Does the user get all the features he needs? Is the user able to figure out how to use the program?

For acceptance testing subjects are required. We arranged some students of other projects groups to help us out. We would really like to have a real analyst take a look at it, but no one of us knows one. So we are going to give our program to 5 students. Those students will also get a task to solve. We will give them the required data, and the documentation that we have. Then they may try to solve the task. We will observe closely what the subjects are doing. We will also ask them to mention what they find strange when performing the task and what they like.

After they have solved the task or have given up we will ask them for more feedback. What went well? What could be improved? Also more specific things are important. For example do you think the help function is visible enough? Were you able to solve the task with the documentation? If not, where were you lost?

4c Conclusion

The persona helped to let us know who we were programming for. It helped to solve a problem for this person. Thanks to the persona, we got a clearer look at what we were going to make. Acceptance testing helps a lot to give you insights in how people use the program. It shows things that programmers never thought of. These tests are really important to get to know whether the program is understandable for the user.

In future projects other things could be done to make the interaction for the user as easy as possible. For example, an interview with a real potential user would be nice. In this case a real analyst would really help. Also, if time allows, some visits of the workspace of the user could be useful. This helps to get a better look at why you are creating something. And that is what interaction design is all about; finding out what the user likes most.

5 Evaluation

In this chapter we will evaluate our product. We start with a section on to what extent its functionality satisfies the user and end with a section on where our product fails to meet the users requirements. Near the start of this project we were given a list of requirements. This list contained questions that our product should be able to answer using data from multiple sources. Furthermore it contained requirements for a few specific modules, namely an input module, an analysis specification module, an analysis module and a visualization module. We have implemented these modules and optimized and enhanced them to the best of our ability, which has caused some of the requirements not to be directly implemented. We will go into further detail of these requirements in the subsections below.

5a Functionality

The questions our product is supposed to answer can all be answered using our program. Most of them were evaluated and approved by a stakeholder and others were shown at demonstrations during meetings. The input module is also working as intended. Specification files in XML format can be saved and opened to store how to import data files. The analysis specification module, for specifying operations and codes and the interpretation for them was also successfully implemented. The analysis module, for using the eight Cs and exporting the result that can be opened by other applications was successfully implemented as well. The visualization module, which describes how data can be explored visually using graphs, was also implemented as was specified. The detailed descriptions of our implementation of these modules can be found earlier in this document.

5b Failure analysis

The biggest mistake we have made was the choice to write our own parser for the scripting language. We had no idea there were libraries that could fulfill the task of what we had in mind so we decided to create our own, which caused some problems later on in the project. This could have avoided by doing more research on the Internet or asking for advice from our SA.

The second biggest mistake was more of a misunderstanding. At the beginning of the project we were told to implement the language using the so-called eight Cs. These are operations which help with analysing large amounts of sequential data. The problem was that we realized we were unable to meet the requirements using just these operations. Luckily we noticed this on time so we could create some operations in addition to the eight Cs.

6 Outlook

This chapter contains an outlook regarding possible improvements for the future and how to achieve them. We begin with the improvements that, in our opinion, will have the most effect or are the most important to implement. There are also many smaller changes that we would like to make, but for we will not be able to describe some for the sake of brevity.

6a Software engineering improvements

Due to our inexperience with the JavaFX platform we made some rookie mistakes and inefficient implementations at the start of the project. Some of these were at the core of the application, which made it difficult to refactor them later on.

Another improvement could be the implementation of a parser from an external library. Our own implementation of the parser for the script language is lacking in some areas as we described earlier in this report.

6b Functionality improvements

Because of time constraints and ease of use we had to create some operators specifically for this context, which counteracts our goal of keeping our product generic. It would be nice of these operations could be refactored into more generic or already existing operations.

A convenient improvement would be the ability to import and export Excel files. Many of the files in the dataset we were given were of this format and had to be converted to comma separated files to import them in our program. This would be a tedious and time consuming task if this has to be done for large amounts of files. It also introduces the risk of importing errors when commas besides the delimiters are entered in these files.

For the language there are a number of improvements to be made. Variables are currently limited to a set of data and numbers. It would be useful to use dates and strings to answer some questions, but its not impossible at the moment. Simple operations such as multiplying numbers or subtracting some days of a date are also missing, but we havent had much need for them as of yet.

An idea that we came up with at the very start of the project was to interchangeably switch between the script as text and a graphical representation of the script. This solves the issue of having to learn the language and can be more user friendly.

7 References

[1] ADMIRE Project (2013). *Assessment of a Disease management system with Medical devices in Renal disease*. Retrieved May 2, 2015, from `http://ii.tudelft.nl/admire/`

[2] European Renal Care Providers Association (2015). *Facts and Figures*. Retrieved May 5, 2015, from `http://ercpa.eu/facts-figures/`

[3] National Kidney Foundation (2002). *KDOQI Clinical Practice Guidelines for Chronic Kidney Disease: Evaluation, Classification, and Stratification*. American journal of kidney diseases: the official journal of the National Kidney Foundation, 39(2), S1-266.

[4] Thomas, R., Kanso, A., & Sedor, J. R. (2008). *Chronic Kidney Disease and Its Complications*. Journal of Primary Care and Community Health, 2009 Jun 1.

[5] Weir, M. R., & Lerma, E. V. (2014). *Kidney Transplantation: Practical Guide to Management* (1st ed.). New York, Springer-Verlag.