TI2806 Contextproject

Final Report
Group HI4

June 25 2015

Elvan Kula	ekula	4194217
Hans Schouten	hschouten	4314891
Matthijs den Toom	matthijsdentoo	4311248
Remi van der Laan	rvanderlaan	4326156
Sven Boor	sboor	4321057

ABSTRACT

For renal transplant patients it's very important that their health status gets monitored daily to prevent complications such as rejection and side effects. The ADMIRE project introduces a disease management system for patients self-monitoring which includes a home-based medical device for measurements and a website for feedback. Therefore we have created a standalone software application which can be used by researchers to pre-process the data for further statistical analysis and to analyze the behaviour of renal transplant patients.

The end-user's requirements were as follows:

- 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

In ten weeks we built the program using the SCRUM methodology, test-driven development and our design goals of user guidance, flexibility, genericness, good performance and a high quality of product. We were able to fulfill almost all user requirements of the product. The only feature we haven't implemented is the visualization of the Markov Chain graphs due to time constraints.

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

- Input module: This module is able to read in the data from ADMIRE's data sources. Configuration files in XML format can be saved and opened to fill the data in the specified group configuration.
- Select module: For this module we created an SQL-like language and script editor in which the user can specify sequential data operations that need to be carried out on the data.
- Analysis module: This module executes the actual analysis using the eight Cs and additional operations in order to discover any events or use patterns.
- Results module: The results of the analysis can be saved into a format that is ready for further statistical analysis. To support the researcher with the exploration of data the following visualizations are implemented: line charts, frequency bars, box plots, stem and leaf plots, time series, state transition matrixes and histograms.

We are content with the outcome of the project. We achieved high code quality and a test coverage of 85%. The scripting language is executed in a pipelined fashion which resulted in a good performance. The longest script we tried to execute took less than 5 seconds! The program provides user guidance during all processes within the application, which was weekly tested by a potential customer (Wenxin Wang). At the end of the project we did a user acceptance test with five students which revealed that the user-interface was easy to use and consistent.

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 6% since the past two decades. At 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 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-user's 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

In the reflection from a software engineering perspective is explained which requirements our product meets.

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 system's state and the user's activities. This is done with explicit feedback about the user's 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 import different types of files with different delimiters. Raw data from the StatSensor 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. Select
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 create 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 perform sequential data analysis using the eight Cs discussed by Sanderson and Fisher (1994), which are:

- Chunking algorithms: Group records on particular attributes (such as time periods or certain values) to analyze the frequency of these groups in separate blocks.
- Coding algorithms: Automatic coding of behavior patterns and re-coding of event sequences from high frequency events domain.
- Conversion algorithms: Automatic generating behavior of web site response in order to create a new input stream for analysis.
- Connection algorithms: Automatic data synchronization whereby events recorded in different data files are merged into single data file with right event order.
- Constrains algorithms: Automatic filtering data for certain event codes, or time periods.

- Comparison algorithms: Lag Sequential Analysis (Sanderson and Fisher, 1994) for identifying dependency (temporal relationship) between events.
- Computing algorithms: Perform computations such as count, sum, maximum/minimum and average on the data.
- Commenting algorithms: 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:

- Line charts
- Frequency bars
- Box plots
- Stem and leaf plots
- Two-dimensional time series and timelines
- 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 SVG file.

2c Architecture

MVC
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.

Generic data model
The different data files that are imported are linked together to form one sequential data structure. This sequential data structure will be given to the next layer of the program in which the operations that need to be done on the data are specified. The linked data can then be used to perform the data analysis on. To improve the performance of the data analysis a TreeSet is used to store all records of the sequential data structure in chronological order.

The state of the program can be stored to easily load configurations of the files and columns they contain. This is done in an XML format and can be stored in a user specified directory. The display will be extracted from the data and incorporated into a style sheet. Because of this, changes to the display of the output will not require changes to the structure of the data itself.

Custom scripting language and parser
To specify the transformations that need to be performed on the raw data we created a custom scripting language and a parser. The scripting language is similar to SQL and it consists of the eight Cs and a few basic logical operators. We decided to write our own parser instead of using a library. To parse the language a main parser was implemented, which goes through the script one line at a time. For the parsing of data operations we implemented separate subparsers. The main parser detects typed operations and passes the part of the script containing the operator and arguments to the corresponding subparser for that type of operation. The subparser will make sure that the operation is executed and the results are returned in a sequential data structure. The subparsers make it easy to extend the program with a new data operation by adding a subparser and linking it with the operation's execution code.

Additional data operations
For actually being able to execute the data transformations we needed to add two operations in addition to the eight Cs, namely the condition and pattern operations. Conditions are needed for filtering the data based on a specified condition with the constraining operation. The condition will be parsed into an expression and a corresponding unary or binary operator which will then be evaluated on each record in the selected data. The pattern operation is needed to detect use patterns in the data. This is done by performing a Lag Sequential Analysis (LSA) for identifying dependency between events (e.g. A → B or B → A) followed by building a State Transition Matrix to count how often particular patterns occur.

2d Reflection from a software engineering perspective

In this section we will reflect on the product and process from a software engineering perspective.

2e Process

Continuous integration
Helping us with continuously delivering our product for this project are Git (hosted on GitHub), Maven and Travis-CI. Every time we pushed our code to the server, the latest commit from each branch was built by Travis-CI. Travis-CI checks if the project can be built and runs all tests. This worked really well for us. If somebody forgot to run the tests on the code you could immediately see when this problem was introduced. So we could always be sure that old functionality was still working and that no new problems were introduced.

Pull-based development
During the project we used pull-based development for the first time and we learned a lot from it. It's very useful when you're working in a team because it enables the developers to make their changes independent of each other. We also found it very helpful for maintaining code quality. Quality emerges from attention to detail: code style, documentation, commit formatting and other code guidelines which are explained in the fourth section of the Product Planning. Before merging a branch with the master, all these features were checked including the test coverage of the newly added code.

There were a few challenges we faced with the pull-based development model. First of all, feature isolation: Sometimes contributors submitted pull requests that contained multiple features and affected multiple areas of the project. This can lead to merge conflicts and it can take some time to resolve. At the beginning of the project we didn't have any experience with manually merging to solve the conflicts. Another issue is caused by the distributed nature of pull-based development. As a developer you're dependent on the responsiveness of other team members since you have to wait for others to evaluate your code before you can merge it with the master. During our project some pull request were open for days, where could have been closed a lot faster. Lack of responsiveness hurts the code review process and by extension the project flow. Despite of the challenges we had, we want to work with the pull based development model from now on for future projects because we found it to work helpful when working with multiple persons who work on multiple features at the same time.

Code monitoring tools
We also used a few test-driven, static analysis 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. To keep track of the code quality with these tools, we used a new tool called Octopull. This tool integrates the results of the code monitoring tools within pull requests. It saved us a lot of time because we didn't have to fetch the branch first, build it manually and inspect the report.

Division of tasks
Throughout the whole project we strived for a fair division of tasks. With the documenting tasks we tried to collaborate as much as possible. Every team member had the responsibility to write a subsection and to re-evaluate the subsection of another team member.

Every day we started with a scrum meeting in the morning. These meetings were attended by the whole group. The role of scrum master was assigned to Sven and Matthijs was the one who took notes. The weekly user acceptance tests with Wenxin were also attended by the whole group.

In every sprint plan we tried to assign every team member around 16 hours of programming work. The more complex programming tasks, such as the implementation of the parser and the linking of data, were often done in pairs. Remi was responsible for the design and implementation of the GUI. The others were responsible for their own data transformation module(s) and visualizations.

Problems during the project
In the first half of the project we experienced some coordination problems. In the design phase of the project, especially in the first few weeks, it's very important to discuss extensively on the approach you're going to take. With the design of the scripting language and the parser we only discussed the overall design without going into details. This let everyone go his own way and implement the scripting language according to his own interpretation. At the end of the week the scripts differed too much and didn't follow a structure. Because of this we had to put in extra work to re-write the code. For future projects we will solve these coordination problems by planning more meetings to discuss design details.

Next to this, in the first few weeks we didn't leave enough time to integrate the newly added functionality in the GUI. At the end of every sprint we needed to have a working prototype to show at the demos, but sometimes there was still a lot of work to do in the back-end. Because of this we weren't able to show our full potential at the first few demos. We solved this problem by planning every thursday at least three hours to integrate everything into the GUI and prepare the demo.

2f Product

We are content with the outcome of the project. There were some issues and challenges we had to face to get there, but the final program works quite well and fulfills the end user's needs. We achieved a high code quality by keeping the code clean and writing clear documentation. We found out that the statistic analysis tools for code monitoring really do help for this purpose.

The program also has a high reliability. We put a lot of effort into testing, especially into tests for the language parser. It's very important that the scripting language works as expected since this is the most important part of the program. The test data for the parser was systematically designed to seek out bugs and to trigger error messages. We had some trouble testing the graphical user interface in the beginning, but eventually we found a way to do this. Following test-driven development we achieved a line coverage of 85%.

We gave a high priority to the providence of user guidance during all processes within the application and we managed to make the program user friendly. The user-acceptance tests with Wenxin and students revealed that the user interface was consistent and easy to use. We made sure that the user is guided through the workflow of the program by visualizing the steps of the data analysis on screen. The user always gets status information with error messages and explicit feedback about the user's input and status of the data analysis. We also created a user manual containing example scripts and a Getting started' document explaining how the scripting language works.

One of our design goals was to create a modular program. We also achieved this by implementing the parser, visualization module and the user interface itself in separate components They are connected with the program to make it a part of the workflow but they can also be used outside the program. The parser could easily be adjusted for another language or to use it for another program. The visualization tab can be seen as a separate program. By specifying a function and a dataset you could use it in other disciplines. The user interface was built using the JavaFX framework which provides a structure to separate the interface from the application logic and data. Because of this the user interface could simply be replaced by another XML structure to change the entire layout of the program.

We were able to fulfill almost all user requirements of the product. Those requirements are defined in paragraph 2b of chapter 3. The only feature we haven't implemented is the visualization of the Markov Chain graphs. We left this out from our planning because of time shortage. In the last week of the project we decided to give a higher priority for the last improvements in the graphical user interface and the overall reliability of the program.

3 Description of developed functionalities

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. In the first paragraph we will look at the functionalities related to importing files. The second paragraph focuses on the part of the software in which a subset of the imported data can be selected for further analysis. In the third paragraph we cover all functionalities with regards to the actual analysis and in the final paragraph the ways in which the results of the analysis can be visualized and interpreted.

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 use 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 an input format for 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 should be specified. This is either an integer, double, time, date, date and time, string or a special string containing a comment. When the user specifies a date column, one of the supported date formats can be selected. When a user accidentally fills 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 on which column(s) the chronological ordering of the data should be based on. To specify the sorting, the user can enable the sorting option for a column containing a date and/or time. When all groups, files and columns are specified, the current settings can be exported to an XML file in order to save the user 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 a column as identifier (primary key) of the subset. This can be done by selecting one of the columns from a dropdown box. If instead of a column name the user selects a file name as identifier, 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

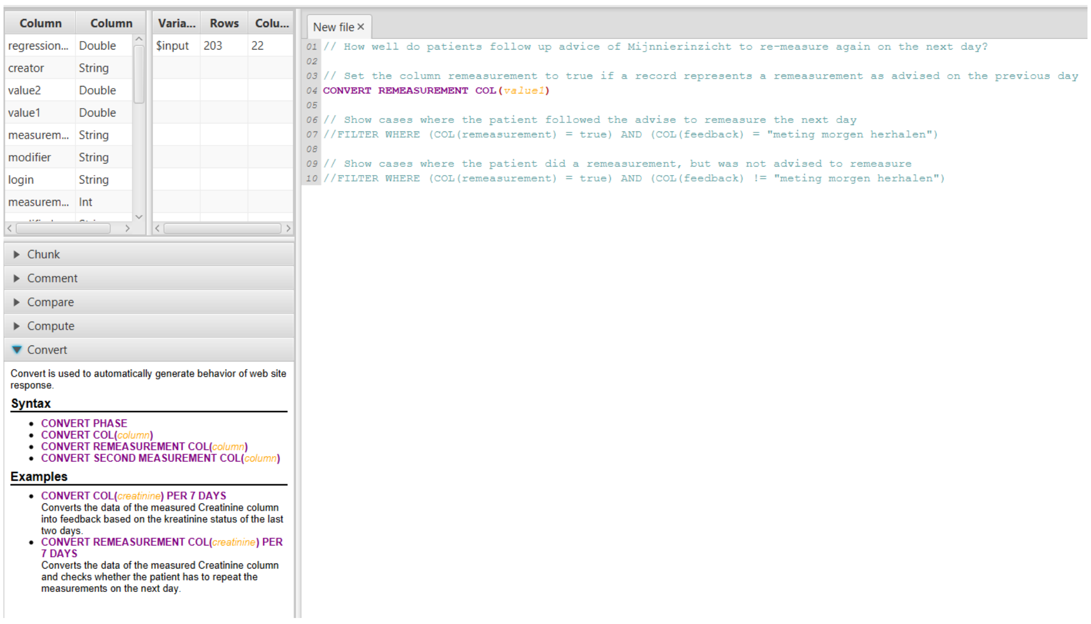
After defining all groups, the software automatically reads through all the data. Each subset - shown as a record on screen - is assigned to the dataset of the corresponding patient.

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.

3c Analyse

Once a has been patients selected, the user can start the analysis. A script editor is introduced to enable the user to type the script directly into the software. The syntax highlighting makes it easy for the user to 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. Next to this list is a list of all variables, which can be opened to view their content by double clicking on them. 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 import button above enables him to import a script file or to select a script file from the list of recently used scripts.



Column	Column
regression...	Double
creator	String
value2	Double
value1	Double
measurem...	String
modifier	String
login	String
measurem...	Int

Varia...	Rows	Colu...
$input	203	22

Convert is used to automatically generate behavior of web site response.

Syntax

- CONVERT PHASE
- CONVERT COL(*column*)
- CONVERT REMEASUREMENT COL(*column*)
- CONVERT SECOND MEASUREMENT COL(*column*)

Examples

- CONVERT COL(*creatinine*) PER 7 DAYS
  Converts the data of the measured Creatinine column into feedback based on the kreatinine status of the last two days.
- CONVERT REMEASUREMENT COL(*creatinine*) PER 7 DAYS
  Converts the data of the measured Creatinine column and checks whether the patient has to repeat the measurements on the next day.

```
// How well do patients follow up advice of Mijnnierinzicht to re-measure again on the next day?

// Set the column remeasurement to true if a record represents a remeasurement as advised on the previous day
CONVERT REMEASUREMENT COL(value1)

// Show cases where the patient followed the advise to remeasure the next day
//FILTER WHERE (COL(remeasurement) = true) AND (COL(feedback) = "meting morgen herhalen")

// Show cases where the patient did a remeasurement, but was not advised to remeasure
//FILTER WHERE (COL(remeasurement) = true) AND (COL(feedback) != "meting morgen herhalen")
```

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. For example, this can be useful for selecting values that are higher than the average value.

3d Results

After the analysis is completed the software can show the results in many different ways. The basic view is a text editor. This can be used to scroll 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 scroll through the data while having a better overview of the different columns. It is possible to sort the data on any column or on a combination of columns by holding the shift key and clicking on the column header. With a click on the export button the current table view can be exported (while maintaining the current sorting) to the text view or can be be saved to a file in a directory specified by the user.

The software also enables user to explore the results further with the help of graphs. The user can create a boxplot, line chart, bar chart, stem leaf plot, time series, histogram, frequency bar and a state transition matrix. The graphs can be created for the whole dataset or for data of each chunk, if the provided script divides the data into chunks. For each graph the columns that should be used on axes can be specified. Based on the datatype of the defined column 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. For example, this can be useful when the user wants to compare the measured creatinine levels with the values that are entered on the website. The graphs of both columns can be plotted at the same time. This allows the user to instantly see whether there are any differences between the measured and entered values. 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, our application, 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 extra user acceptance tests that could be done to make the program more user friendly.

4a Persona

This paragraph describes 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 introduces a self management system for the daily monitoring of renal transplant patients. This includes self-measuring with a home-based medical device, entering the measured values into a website, getting feedback and reacting. By doing so, it is expected that the patients can know their health status better, be more alert, and visit hospital less (Wang et al., 2012).

Back to Paul. The dataset he gathered consists of data from the devices that are used to measure the creatinine levels, from the website where the patients entered their levels and administration 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 patients have followed up the advice that the website has given them and actually filled in the measured values on the website on the same day as they are measured, or if patients only enter measurements that they are satisfied with.

Paul has a lot of questions but he doesn't know how to start. He has a lot of data, but no way of structuring it to analyse it in his expensive statistical programs. He would really like to have a program that is able to group the data for every person. Then he also would like 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 value in the website. It would even be better if the differences in time could be visualized in a graph.

Without AnalyCs Paul would need to do this manually, which costs him a lot of time that he prefers to spend on his research and using his statistical tools which are way cooler to use. He wants to produce the results as soon as possible to publish a paper on it. And that is the part of research that Paul likes most: analysing the data and drawing interesting conclusions from it. That is why Paul requires us to create a tool that transforms the data for him.

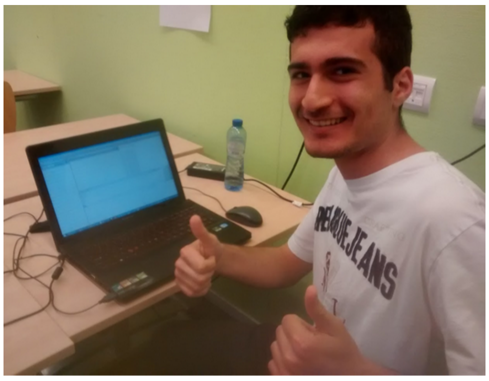
4b Acceptance testing

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

For acceptance testing we need suitable persons that understand how to analyse data. We arranged five students of other project groups, from other context projects, to help us out. For the user acceptance test we asked our testers to solve a simple question: Find cases where there is a difference between device measurement and what is entered in Mijnnierinzicht. To verify the functionality of our program, we told them some where the needed files and documentation could be found. To test the usability, we kept the given information minimal and let them test the program this without any help from us. Only when the tester got stuck we helped them with the next step.

The testers were asked to think out loud while using the program and to tell us what they found confusing or good. We observed that all of the five testers got stuck at some point. Their main problem was that they did not have enough knowledge about the data or what they needed to do to continue. To solve this problem we created a manual, that explains every step needed to do an analysis. This is made clearer by some examples. During testing, we gave the testers some manual instructions so they could continue their analysis.

What the testing students really liked about our program is the workflow. They said that the idea of four tabs visualizing the four steps of data analysis to get to the result is really clear and straightforward. Also the error messages shown are clear in most cases. In some cases however they thought the messages could be more specific. They found the positioning of the error messages intuitive. Lastly they told us the table that shows the results is clear and understandable.



The tests revealed that the user interface was easy to use and suitable for the tasks. The only comment was about the menu bar at the top of the screen which they found to be a little small. This made it hard to find certain functions of our program.

There were a few points that needed some attention. One thing that the testers wanted to be improved was the view of the content in the select screen. It was unclear to them that this screen presented a list of the patient IDs from the data. We solved this problem by explaining it in the user manual. A minor complaint was that the graphs were not automatically updated when another column in selected as input. Another point that needed to be improved was the number of confirmations the user got while using the system. One of the testers commented that he would like a message asking for confirmation to be shown on screen when he switches to another tab. This way the users are sure that the screen is going to change when going to another tab. In the version that the users tested, that was sometimes unclear.

4c Conclusions

The persona helped to let us know who we were programming for and to put the problem in the right context. 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 problems that we as developers never thought of. These tests are really important to get to know whether the program is understandable for the user. This leads to a better understanding of the process workflow by actual users and to a reduction in bug count.

In the future, extra user acceptance testing could be done to have a better view of the user interaction in order to improve it. It would be really helpful to have an interview and acceptance test with a real potential user such as a data analyst, preferably specialized in the ADMIRE project. Also, if time allows, some visits to the workspace of the user could be useful. This helps to get a better look at the environment and situations in which the customer is going to use the program. That is what interaction design is all about; finding out what the user likes most and shaping technology for people's use.

5 Evaluation

In this chapter we will evaluate our product. The section will start with a part on to what extent the product's functionality satisfies the user's needs and it will end with a subsection on cases in which our product fails to fulfil the user's 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 with the data analysis. 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 the Markov Chain graphs not to be implemented as a visualization form due to a lack of time. In the last week of the project we decided to focus on the last improvements of the user interface and reliability of the program. We will go into further detail of these requirements in the subsections below.

5a Functionality

Input module
The input module is working as intended. The program is able to read in the data from the four data sources: the StatSensor, the handheld device for blood pressure and heart rate measurements, data from the Mijnnierinzicht website and hospital appointment data. Configuration files in XML format can be saved and opened to fill the data in the specified group configuration.

Extra functionality: When the user imports a data file, the columns will automatically be detected and shown on screen with their corresponding data types. The user can also specify multiple date columns on which the data should be sorted.

Analysis specification module
The analysis specification module meets the user's needs. For the specification module we created an SQL-like language and script editor in which the user can specify sequential data operations that need to be carried out on the data. In the editor the user can also specify codes (labels) that could be linked with observed patterns. The script will be interpreted to execute the transformations on the data. Script files containing the data operations can also be imported in the editor. We are very content with the way the script editor worked out.

Extra functionality: We also added syntax highlighting and variables to the script editor. By using variables you can execute the next line of your script on the result of the previous line.

Analysis module
The analysis module meets the required functionality: the module consists of parser functions the user can use on data or processed data and functions to store the results in that can be read by statistical applications. We also implemented the option to sort the results on a specified column. The data operations include the eight C's and conditions and pattern operations.

Extra functionality: We implemented the analysis module in a pipelined fashion which really improved the performance of the data analysis. The execution of the longest script we tried took less than 5 seconds!

Example questions
The questions our product is supposed to answer can all be answered using our program. Most of them were evaluated and approved by Wenxin Wang and others were shown at demonstrations during meetings. The files containing the scripts to answer these questions were added to the program.

Visualization module
The visualization module was also implemented as specified in the requirements: the user can specify what data need to be visualized and besides showing the visualization, the graph can also be stored as a picture or a file.

Extra functionality: Besides just plotting a line graph of one numeric column, the software also enables the user to combine different graphs into one. We also made the visualization module reactive: the graph shown on screen is automatically updated when another column in selected as input for the graph.

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 parser, which took a lot more time than using a parser generator. This decision caused some time shortage problems later on in the project. This could have been 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 scripting language using the so-called eight Cs. These are operations which help with analysing large amounts of sequential data. The problem was that we were unable to meet the requirements using just these operations. The user would not be able to answer the example questions that are specific for the context of renal patients. Luckily we noticed this on time so we could implement a few operations in addition to the eight Cs to enable the user to answer the questions.

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 on the usability or are the most important to implement.

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. Likewise our lack of knowledge on parser libraries caused us to make the mistake of building our own parser. Our parser for the script language is lacking in some areas as we described earlier in this report and it caused time shortage later on in the project. For future projects we will gain more background knowledge and brainstorm more often before implementing something to avoid these kind of problems.

As we described in the reflection from a software engineering perspective, we also faced some coordination and code integration problems. The coordination problems were caused by too little discussion on design details in the first weeks of the project. For future project we will solve this by meeting more often and extensively discuss our approach in the design of the product. At the beginning of the project there were also some integration issues because we didn't leave enough time to integrate newly added functionality into the GUI. Because of this we couldn't always show our full potential at demos. We solved this problem by planning every thursday at least three hours to integrate everything into the GUI and prepare the demo.

6b Functionality improvements

Because of time constraints and the misconception that we only had to implement the eight Cs we had to create some operators specifically for the example questions of this context, which counteracts our goal of keeping our product generic. It would be nice if 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 it's not possible at the moment.

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] Sanderson, P. M., & Fisher, C. (1994). *Exploratory Sequential Data Analysis: Foundations*. Human-Computer Interaction, 9, 251 -317.

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

[6] Wang W, Brinkman W. P., Rvekamp T. J. M., van der Boog P. J. M., Alpay L., & Neerincx M. A. (2012). *Feedback to Renal Transplant Patients in a Self-management Support System*. European Conference on Cognitive Ergonomics 2012. :147-150.

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