Renal Transplant Health Analysis

The analysis of data concerning the health of renal transplant patients

Group C

Faculty Electrical Engineering Mathematics and Computer Science

TU Delft Delft University of Technology

Challenge the future

RENAL TRANSPLANT HEALTH ANALYSIS

THE ANALYSIS OF DATA CONCERNING THE HEALTH OF RENAL TRANSPLANT PATIENTS

by

Group C
Louis Gosschalk, Boudewijn van Groos, Jens Langerak, Chris Langhout & Paul van Wijk
lgosschalk (4214528), bvangroos (4229843), jlangerak (4317327), clanghout (4281705), pvanwijk (4285034)

in partial fulfillment of the requirements for the course of

Context Project
in Computer Science

at the Delft University of Technology,
to be presented publicly on Friday June 26th, 2015.

Project Coordinator:	Prof. Dr. A. Hanjalic
	Dr. A. Bacchelli
Context Coordinator:	Dr. ir. W. P. Brinkman
Software Engineering TA:	T. M. Hegeman

An electronic version of this report is available at
https://github.com/clanghout/Health-Informatics-3/.

TU Delft Delft University of Technology

ABSTRACT

There is a problem. A lot of patients of the ADMIRE project have to be analyzed, but there is no analysis software simple enough to do this easily. So what do you do when you want to do something which should not require a lot of effort but no software exists to easily achieve this? Exactly, you build software for it.

To build this software we worked according to the scrum method and weekly presented a high fidelity prototype. This resulted in a software which, according to previous meetings, has grown to be accurately satisfying the needs of the customer.

But why do we even care about these analyses? We care because these analyses might help us understand the patients better and thus might make it possible to improve the system and help the patients better with, for example, better advice.

At the end of this project our conclusion is that we successfully understood and solved the problem through our implementation.

Contents

1

INTRODUCTION

What have we done this project? In this report we will show you how and why we have done what we have done and, most importantly, what we have done.

We will start with a software overview, which will show the functionalities which have been implemented. A reflection on the software engineering will follow this, to explain how we went to work, what code requirements our group had set, how we handled feedback and more. To fully explain and describe the functionalities of the software, a feature description comes after the reflection, where all functionality aspects will come to discussion. After explaining the features a description will be given about how we managed to design and create our software to meet the requirements which have been taught to us during the interaction design course. This will be followed by a product evaluation chapter to discuss the product in its entirety and last but not least we will give you an outlook on the future of our system.

2

Software Overview

At the start of the project we constructed a product planning in which we described what features we want to implement. In this chapter we will describe to what extend these features are truly implemented.

2.1. Must Have

There were features that were obligatory to have in the software for it to be able to work properly. The next items are the must-haves that we have implemented in our program.

Language We implemented the declarative language SUGAR and it can be used to perform analyses.

Analysis The program can perform chunk, constraint, comparison, computation, code and connection operations on data. All these can be edited and performed in a simple editor. It is also possible to load an existing script from a text file which can be executed by the program.

Load Data We can specify measurement data in an XML file. The files can then be loaded into the program so that analyses can be performed. We also implemented a graphical wizard to create a specification of datafiles which can then be saved to an XML file.

Visualization We implemented features that the user can create barchart, boxplot and state-transition matrix visualizations. It is possible to create a frequency bars plot by making a counting analysis and plotting it into a barchart.

datatypes It is possible to distinguish different datatypes. We have implemented classes to describe integers, floats, booleans, strings, dates, times and datetimes. The types can be distinguished in the specification of the datafile and these types will then be used to perform the analyses on.

Output The application can export the data to csv or textfiles and specify delimiters so further analysis in the same or other software, is possible.

Manual A roadmap is included which describes the basics how the program is to be used. Also a language specification document is included which describes the syntax of SUGAR.

GUI We created a graphical user interface using JavaFX to add some usability to the program. The main interface consists of tabs containing their own subject.

2.2. Should Have

We also planned some should have features which were not crucial for the program's main functionality but were needed for additional functionality. It is possible to export all the supported visualizations to png files. In the language specification document we provide basic examples of SUGAR scripts.

2.3. ADDITIONAL FUNCTIONALITIES

We implemented a wizard for creating and modifying an XML file, and a wizard for saving the tables to files. We also added a simple editor in which the user can create a new analysis, modify an existing analysis, or execute an analysis.

3

Engineering Reflection

In this section we'll describe how the engineering process went. We discuss how we've used scrum to develop the product. Furthermore we'll discuss the GitHub pull requests and how we ensured that our code was of good quality. Finally we will discuss the feedback we got from SIG.

3.1. Scrum

We used the scrum work flow to develop the product. Friday we created a plan for the coming week. Every week-day we started with a short daily meeting. In that meeting we discussed what we'd done, what we were going to do and which problems we'd encountered. On Friday our sprint ended. We created a sprint reflection and we gave a demo to the client. Next we started on new sprint. At the start of the project we underestimated the time needed for tasks. Therefore a lot of task weren't done during a sprint. Later in the project we got better at estimating the time required and therefore there were less incompleted tasks.

We liked that scrum was focused on features. But we also noticed that because of the focus on features, there is no room for tasks that can not be translated into a feature. Though those tasks could be of importance.

3.2. GitHub Pulls

It wasn't allowed to push changes directly to the master. When a person wanted to merge into the master he had to open a pull request. The pull request had to be approved by at least two team members before it could be merged with the master. The code for the pull request should be of good quality (see section 3.3). We looked very critical at pull request. This meant that the person who'd performed the pull request had to improve some parts of his code. However most comments were related to code style and not directly to the implementation of the code. But still the reviews led to improvement of the code quality.

The size of the pull requests varies a lot. In the beginning of the project they were very large. This made them hard to review and caused them to stay open for a long time. Therefore we tried to create smaller pull requests. This result of this was that pull requests were handled quicker and that we could review them better. However there were still some very large pull requests.

Reviewing the code resulted in good code quality. Therefore we will try use pull based development in future projects. We'll remember to keep the pull requests small.

3.3. Code Quality

During the project we tried to create code of good quality. The code had to meet a few requirements before it was considered to be good quality. First of all the code had to follow the conventions. Furthermore the code should be clear and self-explaining. If the code was not self-explaining it needed comments. The code had to be tested and had to contain JavaDoc. And at last it should work efficiently and it had to be easy to make changes.

To achieve this we used some tools. The first tool we used was Checkstyle. Checkstyle reported the situations where we did not follow the code conventions. For instance if there were unused imports, Checkstyle would report that. Another tool we used was PMD. PMD detects a variety of code smells. For instance it

detects unused variables and methods, duplicated code and a lot of other things. Furthermore we used FindBugs. FindBugs is capable of detecting possible errors.

Before a person opened a pull request he had to fix the issues reported by the tools. Furthermore the code should've been tested. Then other team members would look into the code and make some suggestions to improve the code quality. In general this worked pretty well, and led to good code quality. However the tools were not always run, therefore it happened sometimes that some errors detected by the tools ended up in the master branch.

3.4. SIG FEEDBACK

In week 7 we got feedback from SIG. In general our code quality was good and above average. On duplication and on component balance we scored below average. By the time we got the feedback, we already refactored a part of the code that caused a lot of duplication. Since the teacher of the context project did not found the component balance a big issue. We decided that it was not worth to invest time to improve this. However we have tried to restructure our packages a bit better.

Since our overall feedback was good, we concluded that the way we ensured our code quality worked and there was no need to change that process. Near the end of the project we focused less on quality and more on features. This has done some harm to the code quality.

4

FEATURE DESCRIPTION

Our program offers a variety in features. In this chapter these features are described.

4.1. IMPORT

To specify the data used in the data model we use an XML file. Using a file to specify the data makes our program generic and easy to use since the same XML file can be used more than one time.

In the XML file all the needed information of a file is found. The path to the file is specified as well as the delimiter used. For each column a data type is specified.

4.1.1. XML WIZARD

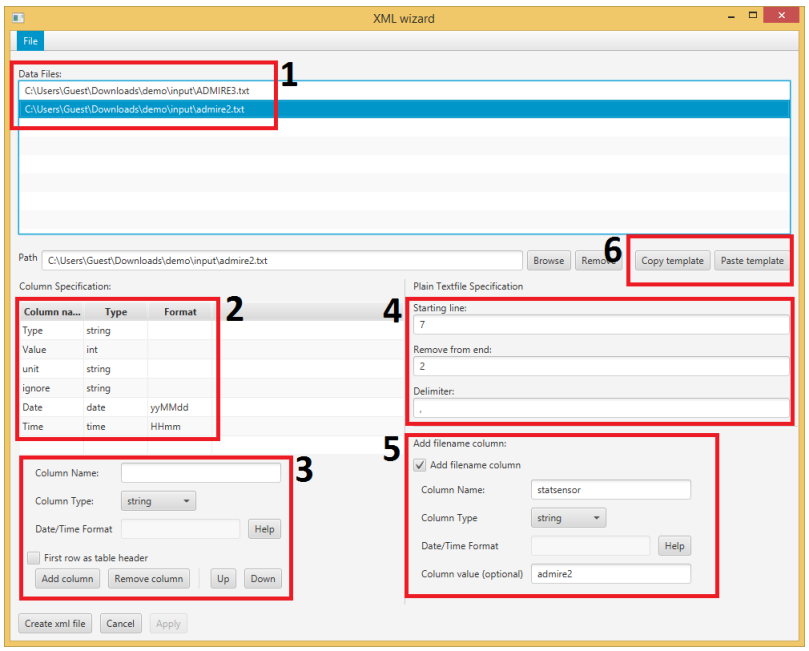

Figure 4.1: The XML Wizard

To make it easier to make an XML file for our program, we created an XML wizard. In this wizard multiple files can be selected for the data [1]. Then per file, columns can be added with the appropriate type and (if needed) format [2, 3]. An option exists to use the first row as the column names. When this option is selected, no name needs to be specified for the columns in the wizard. The lines to skip at the start and the end of the file can be specified [4] and an option is available to add an extra column to the data containing the file-name or text specified by the user [5]. When all the settings are set, this info can be copied and then pasted for a similar file [6]. As in the example, multiple StatSensor files are loaded and use the same settings.

4.2. ANALYSIS

To analyse the data, the user has to switch to the 'analysis' tab. In this tab a text area is present where the user can add code in the program exclusive "SUGAR" language. When a script already exists, this file can simply be loaded into the program. This script can be changed and executed to affect the data. After changes are made, the script can be saved to file again.

4.2.1. THE C'S

Multiple different analyses exist to execute on data. Following the terminology of EDA we implemented 6 out of the Eight C's of the generic ESDA process [1]:

CONSTRAINT

A constraint can be used to select specific values from a column. Numbers can be compared to constants or other columns. Constraints can also be combined with AND, OR and NOT operations. On date values, the extra constraints "before" and "after" are available.

COMPUTATION

Number values can be added, divided, multiplied and subtracted. The modulo operation is available, as well as the square root and the power function. Date values can be added or subtracted and the relative difference can be computed. Separate date and time can be combined to one DateTime value, also the opposite; DateTime values can be split up to either date or time value. All the computations can also be used in the constraints in combination with column values or constants.

COMPARISON

To determine a relation between events, the lag sequential comparison is available. The lag sequential analysis will return a table chronologically sorted on the events.

CODE

Codes can be added to rows when a value of that row satisfies a constraint. A single row can contain zero or more codes. When a row contains a code, it can be selected via a specific constraint that selects only that code.

CHUNKING

Chunks can be created with the group by function. A group by function can be applied on a column or on a constraint combined with a column. This creates a new table with the first column the value of the group, the following columns are created after the functions given by the user, applied to that column.

CONNECTIONS

To connect two or more tables, a join function is available. Tables can be joined in multiple ways. The full join is available as well as a left, right and simple join.

4.2.2. ADDITIONAL ANALYSES

Apart from the implemented C's, more functions can be used in the program.

UNION

Two tables can be unified with the union command. This creates a new table with only the rows that have the same values in both tables. Codes of the rows of the two tables are not taken into account when comparing, but when two rows are the same the codes of both rows are combined in the new table.

DIFFERENCE

The opposite of the union is also possible, the difference operation. Two tables are subtracted and only unique rows are added to the resulting table.

4.3. VISUALIZE

No analysis is finished without visualizing the data. A visualization shows the data in an easy and fast interpretable way. In the visualization tab of the program buttons can be pressed to show a pop-up where the data for that visualization can be specified. After a visualization is made, the image can be saved as PNG file.

4.3.1. BOX-PLOT

A single box-plot can be created of one column with the FloatValue or IntValue type. This happens in the graph pop-up where first a table is selected and then the column can be selected. The support of creating multiple box-plots at once is not implemented but it is possible to save box-plots one by one.

4.3.2. BAR-CHART

A bar-chart can also be made via the graph pop-up. For the bar chart's axes, two columns must be selected. An example for the x-axis could be the names of created chunks since these are unique. For the y-axis a column with NumberValues must be picked.

4.3.3. STATE-TRANSITION-MATRIX

The state-transition-matrix has its own pop-up menu. When no codes are present, the pop-up shows an error message and no matrix can be created because codes represent the states in the matrix. When a table is selected, a column can be selected to define the order of the table. Then the user can select the codes to add in the state-transition-matrix. When the create button is pressed, a matrix is created with the amount of transitions between the selected states as values.

5

Interaction Design

In this chapter we will describe the interaction design test we have done with the user. First we will describe the persona of a typical user of the tool and next we will tell how the test. Furthermore we will draw the conclusions from the test.

5.1. Methods

We put the user behind the computer and asked him to perform tasks. We first wanted him to read in files and then perform analyses. If the user can not figure out how to do something, we gave some hints to help him along the way. The more complicated analyses were pre-written so that the user only has to modify the existing analysis to make it work. We did this because the language we use has a high learning curve and we did not have a lot of time at the test to make the user learn the complete language.

5.2. Expectation

In this section we will describe what we expected how the test would go. We expected that the test could be a bit too difficult. This is because the the language we use is not that easy to understand at the first time using it. We provided the users with some example scripts of the language to make this process easier. We expected that the graphical user interface would be easier to understand.

5.3. General Persona

In this section we will describe John Doe who is an abstraction of our typical user.
John is an analyst and he uses analysis tools on a regular basis. His analyses are performed on data that is collected during research. He has a specific goal in mind and he wants to get to an answer to a specific question about the data. John also has minimal experience in programming, but is familiar with scripts from other analysis tools and is eager to learn a new scripting language.

5.4. Evaluations

Every Friday the team had a meeting with the client to show progress and get feedback on the demo. To show our progress to the client we used the high fidelity prototyping principle.

5.4.1. Usability evaluation

To test the usability and functionality of the program, tests are done with our client Miss Wang. A meeting was arranged where the program was used by the client and the functionalities of the program were shown. For the example questions that the client wants to answer pre-made scripts were executed and feedback was given on the overall process.
Little things were changed as a result of the meeting, for example the save button was moved because right below the import button did not make sense. We also discovered during this meeting that the visualizations did not work anymore due to changes in parts the visualizations are dependant on. Overall the client was positive about the product since most of the analyses could be executed.

5.4.2. ADDITIONAL TEST RESULTS

Other tests were done with fellow students. The detailed results are found in appendix A.
Generally the tests resulted that our language is quite difficult to understand, which was already predicted. The user interface was generally clear and only little remarks were made on positioning of buttons. With the provided examples of analysis scripts the testers could reproduce a similar analysis with reasonable effort. Most of the time it was clear which part of the interface belonged to which feature with little exception in the visualization selection. The general impression of the program was good, the interface is clear analysis functionality is good.

6

Product Evaluation

6.1. Product

Even though our product has many features, it can be reduced to five simple modules:

- The Data Importation Module
- The Data Viewing Module
- The Data Exportation Module
- The Analysis Module
- The Visualization Module

These are the same modules that'd been mentioned in the first context specific lecture, so in general we managed to implement the most required features. For our features we focused mostly on the must haves. These were roughly specified in the first lectures and after a couple of weeks we got the actual product requirements.

We'd hoped to finish the must haves early and perhaps tackle a couple of could haves. However due to some issues we encountered we had to adjust our planning to finish all the must haves. This meant that some features we were hoping to get in didn't make the cut.

Overall we must say that it is really hard to anticipate exactly how the final product will turn out. You have an idea of which features you should provide, but the way all those features work together is something else entirely. In the end we're satisfied with the result and how everything turned out. We missed the time to really polish it up and get all the bugs out of there, but it is a program which does what it is supposed to and it can do a whole lot more.

6.2. Modules

In this section we'll discuss the precise operation of the various modules.

6.2.1. Data Importation

The data importation module consists of two parts: the data reading part and the data describing part. The data reading part takes an XML file which can be generated using the data describing part. We've noticed the describing process isn't completely intuitive and it is something we should have spend more time on. That being said, the process in itself is rather complex and a UI can only get you so far.

6.2.2. Data Viewing

The data viewing module is rather simple. You have a list of the left of the various tables present in the program and a table on the right showing the contents of the table. There is also a button to save a table which opens a wizard, more on this in the next section. This module is rather simple and we're satisfied with its simplicity.

6.2.3. DATA EXPORTATION
This module consists of a dialog which can be used to save a table to a file. It's got a couple of simple options to select how exactly the file is to be saved and it's all very user-friendly.

6.2.4. ANALYSIS
The analysis module consists of a textbox and a couple of buttons to run, save and load scripts. This is by far the largest module as the entire point of our program was performing those analyses. We support a great number of analyses and some other processes which can do some rather complicated stuff with the input data. We wish we'd implemented a couple more analyses, but there was only so much we could do in the given time. In the end we're content with the rather complicated, yet subtle workings of our language. We do realize that the complex workings of our language present the user with a bit of a challenge to learn our language.

6.2.5. VISUALIZATION
In the visualization module we support generating a box plot, frequency graph and state transition matrix. All of this is done through a simple dialog, which is simple and easily understandable. We do think that a little more time could have polished it up a bit and removed a couple of bugs. However the graphs created are pretty and useful. Also we don't think that the visualization is the core part of our program, there exist other better tools to do such things.

6.3. FAILURE ANALYSIS
In this section we'll discuss the various issues that are still present in the program.

One of the major issues we encountered in the realisation process was that Java generics aren't available in runtime. This is something we encountered along the way and didn't anticipate beforehand. Currently it means that we're not able to do type checking in our language. So multiplying a date by a boolean is okay for the parser and it will throw a non descriptive runtime exception. In the future I think it is something we'll take into account when dealing with generics.

One of the other major issues is that our program isn't very user friendly. This isn't a major concern as the program is very specialistic and the main target audience is very limited. However it is something we should have paid more attention to.

Another issue is one of performance. The data that is entered in our program is quite large and it hasn't been optimized one bit. This means the user has to wait a while for the larger analyses. However it is perfectly possible to reduce the data by performing a simple analysis. This should speed things up for the user.

Last of all there is a bug in the visualization module: an error message doesn't go away when it should, this could have been solved by testing more thoroughly.

7

Outlook

Our product is finished for this course. It satisfies all the crucial needs of the customer and even covers some additional non-crucial desires. Though, if our software were to be used by a more broad range of customers in the future, a couple of changes should be made to make the software usable for more than just the ADMIRE analysts.

7.1. Improvements

Certain improvements would be advisable to make the software usable to a broad crowd of analysts. For example, when rescaling the main window of the program, the text fields do not adjust accordingly. This is an improvement which is not at all necessary but is a great improvement to the usability of the product. Another important improvement would be to do more extensive testing and thus solving more bugs, since a broader crowd means a more varying set of customers and thus requires a more robust, solid software.

Our software has been created with the aim to assist the ADMIRE analysts, but in the process we decided to keep the language rather generic. This means our analysis language is already quite suitable for a larger audience than just the ADMIRE analysts. A possible addition to the language would be a way to generate visualizations without having to generate the table and to create the visualization in the separate tab for this. This would improve the usability because it reduces the required steps for a user to do visualizations when they do not explicitly want to see the result data in a table.

Another feature which would come in handy is to be able to export into excel format. A lot of analysts appear to be working with excel and excel files can of course also be loaded into SPSS and other analysis tools, so adding this feature would surely not be a waste of effort for the product quality.

7.2. Future Development

Assuming we win an award and continuation of development becomes interesting, we will most likely continue in the same way we are working now. Though, probably without the daily get togethers. We will most likely have multiple virtual meetings weekly to discuss the scrum progress and set some weekly goals to accomplish. This way we can combine the development with the daily lives of all members.

A

Results of Interaction Design Tests

- **Mark** is a student in the same year as us. He is also a participant of the context project but he works on the Programming Life context.

  The first thing that is noticed is that in the help button in the menu bar the about button does not function. Mark is linked to the external manual for our program. Then he got the exercise to change an existing XML file using our wizard. It took some time for him to find the load function. It is actually located in the menu bar in the wizard. We asked to copy the settings of one file to another and then Mark wondered why the "copy template" and "paste template" buttons are next to the path of the file. Then an analysis was loaded into the internal editor. The analysis was kind of hard to understand but he recognised the form of the definitions.
  Next we went to the visualization part. We started by letting him create a bar chart. It turned out to be not so clear where the columns selected stand for. While creating a box plot he discovered a bug, the error message when selecting a non number column does not disappear when a correct column is selected and the pop-up has to be closed and reopened to create the box plot. After this nothing worth mentioning happened and we finished the test.

- **Robin** is also a student in the same study year. He is on the same context as us, so he knows well what the purpose of our program is.

  Robin also got the task to change an already existing XML file. He quickly found the load button in the menu bar, but it was not clear that files can be selected to change its contents. When adding files he wondered why the buttons for the files are "browse" and "remove" instead of "add" and "remove". When he edited the path of the file, it was not clear that the apply button had to be pressed to confirm this changes. Because he started with loading an XML file, he wondered why he had to specify a location when the "create XML" button was pressed, because he expected to automatically override the existing file. When copy pasting templates in the wizard, he sometimes had to press the copy button twice before the template was actually copied. He wondered if he could get feedback when copying the template but than stated that normal the normal computer copy also gives no visible feedback.
  At the analysis a pre made analysis was loaded. He had quite a hard time understanding the language. Robin wondered where the vertical lines in our pipe system were for. To help him we showed the language specification, where our pipe system is explained. After that a simple constraint analysis was made by him. While saving the analysis Robin noted that the file selectors always go to the home folder where as their program saves the previous selected location for user friendliness. After some scripts were loaded and slightly adjusted, codes were set on the demo table.
  With the codes set we gave the exercise to create a state-transition-matrix. When in the pop-up for creating the matrix he asked the reason why he had to select a column so we pointed to the label above the selector which he missed. He told us that he liked the option to select the codes to use for the matrix. After the matrix was created and saved, one last remark was made by Robin. He asked why the default value in the selectors for columns for the visualizations are not already columns. After this the test was over.

BIBLIOGRAPHY

[1] Penelope M Sanderson and Carolanne Fisher. Exploratory sequential data analysis: Foundations. pages 266 – 267, 1994.