Final Report

TI2806 Contextproject
Health Informatics

Group A

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

Contents

1

INTRODUCTION

The aim of our project is to develop a stand-alone tool, which will be used for analyzing self-management behavior of kidney transplant patients. The data used for the analysis has been collected over the period of year.

After their kidney transplantation, patients learn to work with a self-management system to help them recover[6]. They have to measure their own creatinine level, which they then enter on a website (admire.mijnnierinzicht.nl) using their account. This website will give them feedback about their current health status.

But the self-management process is not always followed well. Analysts want to be able to analyze the self-management behavior of all these patients. This is where we come in. Our goal is to create a tool which takes the data files from the patients' measurements, website input data and files from the hospital as input. Then it should be able to process the input and perform all sorts of manipulations on the data, in order to create visualizations and output files. This tool should help these analysts draw certain conclusions about the patients' behavior.

1.1. USER REQUIREMENTS

The main end-user's requirement is to be able to answer certain questions about patients' behavior, based on the visualizations and output files. Several modules have played a role in making this possible. The sections below explain which requirements have and which have not been implemented. Later on in this report we will explain the functionalities more in depth.

1.1.1. INPUT MODULE

The user requirements concerning this section have all been implemented. The user can easily specify a new structure of data files (XML) and save it for later use. Also, based on the specified XML, this module is able to read all the necessary raw data files, such as the files from the statSensor and Mijnnierinzicht website.

1.1.2. ANALYSIS SPECIFICATION MODULE

Our tool gives the user the possibility to enter a script that specifies the sequential data operations that need to be carried out on the data. The parser (generated by ANTLR based on our script language specifications) then creates a parse tree. After the script has been parsed, our interpreter is capable of walking the parse tree and determining which actions should be taken.

1.1.3. ANALYSIS MODULE

The requirements that we have implemented are chunking, coding, connection, constrains and comparison and conversion, which covers all the required operations.

1.1.4. VISUALIZATION MODULE

The visualization module gives the user the ability to explore the data visually. This data can take different shapes. For one thing, we have given the user the option to simply save the (raw) data in text or PDF files. But

we have also implemented several visualizations in the shape of charts. The visualizations that we have implemented are the histogram, frequency bar, box plot and state transition matrix. All of which are must haves, following the MoSCoW method[7]. We have not implemented other visualizations than the must haves. Saving these charts was a could have, which we have also implemented.

2

PRODUCT STRUCTURE

2.1. MODULAR DESIGN

For our application we used a Modular design. The main advantage of this design pattern was that it would allow us to separate attention of the group members. That way we could work on and improve each part individually, as described by Steven Bradley [1]. The modular design was implemented with 4 major modules in mind: Input, Operations, Output and Control. To support those a couple of extra modules had to be created. For the output, the visuals module was created to make all visual outputs, for example box-plots and histograms. We also made a GUI module to give the user an easy natural way through the process of input, operations and output. Lastly we made a script module, the script module provides the user with a very dynamic way of using all possible operations of the application.

2.2. SCRIPTING LANGUAGE

We decided to make the user control the operations on the data using a scripting language. This way the user has much more freedom in the use of the application, apposed to a purely GUI based control. The disadvantage of this decision is that a scripting language can be a bit confusing and harder to learn for the end-user. To solve this program we tried to make a scripting language with a naturally feeling grammar. We did this by making more complex queries be like normal sentences, so for example a user could use the command "connect table1 with table2 where column1 = column2".
Our scripting language is parsed using parsers generated by ANTLR, we made this decision to save time in the process of creating this language. We first tried to do all this parsing ourselves, but later found that we were reinventing the wheel and decided to just use the powerful ANTLR libraries.

2.3. PARSE STRUCTURE

The application supports three types of data files: txt, xls and xlsx. These three were chosen because the data produced by the research team were all in one of these formats. A challenge we faced was that all data was a bit differently formatted. So we decided to use XML files to describe the way the data was formatted. To use these XML's we convert them to an object called an InputDescriptor. The InputDescriptor contains all needed information for the parser to successfully parse the given data and create the data structures used in the rest of the program.
We chose for the XML format for describing the data files for a couple of reasons. The first one is that it is a format which is easy to parse using existing libraries in Java. Another reason is that it is easily extendable[2], this was great for us because when we started the project we did not know what kind of things we would find in the data. For example we came across files which had a nonsense last line, because we used the XML format we could easily add a IgnoreLast property.

3

FUNCTIONALITIES

3.1. GUI

The Graphical User Interface allows the user to select the files to be processed, edit the script to input what they want to have processed and their output or to edit the configuration XML that is used to transform/read the data. Hence we can view the transformed data in the last panel as a visualization or just an outputted table.

3.2. XML-WIZARD

Creating an XML file to describe the data set can be cumbersome. The XML Wizard provides a more user-friendly way to create and edit XML files from the GUI.

3.3. INPUT FILES AND XML-CONFIGURATION FILES

The application supports three different types of input data: text files, XLS files and XLSX files. For all formats an accompanying XML file is required that defines columns for each row of data, as well as specify a few format-specific properties. An input file can contain three different types of data: numbers, strings and dates.

For text files, the XML-configuration file can specify the: delimiting character between columns; string indicating the start of the data; string indicating the end of the data; the number of trailing lines to ignore.

For XLS and XLSX files, the XML-configuration file specifies the: row and column at which the data starts; number of trailing rows to ignore.

3.4. SCRIPTS

The script allows a user to transform data, output data and generate visualizations of data. The supported transformations are:

- *chunking* - Breaking up the data into chunks based on a given period of time or base on equality of a column between records, and then performing calculations within in a chunk such as: the number of records; the average value of a column; the sum of values in a column; the minimum or maximum value within a column.
- *coding* - Abstracting the data to a list of codes based on the records, and finding occurrences of given sequences of codes.
- *connecting* - Merge the data of different input files and sorting the records by date.
- *constraining* - Filtering the data base on a given condition.
- *comparison* - Calculating temporal relationship between events and generation of state transition matrices of the transitions between events.

As described in the article by Sanderson[5].

3.5. OUTPUT

The application can generate two different kinds of output: data and visualizations. Data can only be outputted as a text file. The user is able to simply use a default format, in which each record is printed on a new line and each column separated by a comma, but it is also possible to have full control over the formatting by specifying a format string.

The application supports four different types of visualizations:

- Box plot
- Frequency bars
- Histogram
- State transition matrix

The visualizations can be exported to a PDF file, but they can also be inspected in the output panel of the user interface.

4

REFLECTION ON PROCESS AND PRODUCT

4.1. OUR TEAM

Most of us already knew each other before we started the project or had already worked together on other occasions, which made communication and discussions about the project easier. Everyone in our team has varying skills.

Some of us are relatively more experienced in programming, others are better in visualizing what it is the client wants exactly and some are better at things like languages. Everyone has been involved in each aspect of the project. We divided the roles so that everyone would have the opportunity to be involved in things he/she is good at and other things that were a doable challenge and would be a useful learning experience.

Using git inspector we were able to conclude (what we actually already thought) who was mostly responsible for which big aspect of the project. The results showed that Lizzy was mostly responsible for the visualizations, Bjorn for the XML wizard and GUI, Adam and Daan for the operations and parsing input files and Martijn for the script interpreter and table structure.

4.2. PROCESS

The process of the development of our product has not always gone according to plan. We had a bit of a rough time starting up. At the beginning it was not always clear what the requirements were exactly, which made it hard to define our goals quickly and to determine where we should start.

This is why our task division was not very good at the beginning. Some of us were already programming from the start while others were writing reports. This division was not our smartest move. We should have divided it in such a way that everyone was doing at least some programming right from the start of our project. This was not a very efficient and effective approach and it unfortunately led to some process loss on our end

Also we may have sometimes made things more complicated for ourselves than would have been necessary. One example is our lack of usage of libraries. Especially in the case of our script parser. We searched for a way in which we could implement this, and once we found a solution, we immediately decided to use that strategy. Only after our midterm assessment did we realize that this was not one of our most tactical choices. We simply went ahead with one of the first viable ideas we came across. What we should have been doing is search for a better, simpler and more efficient solution. After the midterm assessment we decided to continue our project using ANTLR, instead of writing our own parser.

We have learned our lesson. Essentially we wasted valuable time reinventing something that already existed. This does not mean that we were doing it wrong, we were simply not making use of work that had already been done for us in libraries. We should have used this from the start, but better late than never.

But on a more positive note, we have made the most of the useful feedback that we were given during the midterm assessment. Ever since the assessment we have been working in overdrive, in order to catch up as

well as possible. Working all day every day trying to finish all the requirements in time. After all our hard work we are proud to say that our tool is working very well and that we have finished all the requirements in time.

Besides that, we did quite a good job implementing Scrum. Every week we decided on which tasks to tackle in the coming sprint, writing it down in a Sprint Plan. Every morning at 10am we had a meeting, updating each other on what was already done and what was still at hand. At the end of every sprint we reflect upon the week, creating a Sprint Reflection.

4.3. PRODUCT

4.3.1. MODULE DEPENDENCIES

There are a couple of things in our program that could have been better implemented. The first thing is about our modular design. If you would look at our architecture design, you would see a lot of dependencies, more than needed. This could later create problems, when trying to reuse different module.

4.3.2. UTILITY CLASSES

Another thing is that we have a lot of static utility classes. We use those mostly for operations on the data and visualizations. It would have been better to create factories for those modules. That way it would be easier to extend with extra functionality.

4.3.3. MULTI-THREADING

Lastly our program is not multi-threaded. If we would use multi-threading, we could speed up the parsing a bit, because multiple files could be parsed at the same time. But the biggest advantage would be that the program would not freeze if a long analysis is running, and a user could already make a script for the next run.

5

HUMAN-COMPUTER INTERACTION

For the human-computer interaction aspect of our development process we decided to conduct a usability test. To prepare and conduct the test we consulted a book on usability testing by Jeffrey Rubin and Chisnell.[4] In this chapter we describe the preparation and the execution of the usability test. We also take a look at the results and what we have done with the feedback.

5.1. OUR GOAL

When preparing the usability test we set ourselves two concrete goals. Firstly, we wanted to find out what users liked most about our application. Secondly, we wanted to find out what users found least intuitive.

5.2. TEST GROUP

According to Jakob Nielsen, five participants is sufficient to find 85% of the usability issues when you have comparable users who will be using the product in fairly similar ways.[3] Because our users will be highly educated researchers, we decided that our test group should consist of the researchers of the future: students. Because we wanted a diverse group, we recruited both Bachelor and Master students.

5.3. TEST MATERIAL

In preparation we created a test packet containing, besides the application, a data set, an XML, some premade scripts and a manual describing the product and the test. Their task was to set up and complete an analysis.

5.4. CONDUCTING THE TEST

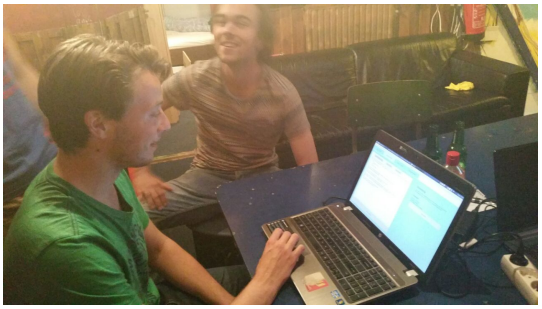
Before the test group arrived we made sure everything was set up. We asked the group to think out loud during the test. This way we could capture all their considerations. At any time during the test, at least one of us was with the test person, answering questions and assisting where needed, and noting their thoughts if they spoke them out.

5.5. COLLECTING DATA AND FEEDBACK

While one was busy guiding the test, the others talked with the students who have completed the test already. We prepared a set of three questions:

1. On a scale of 1 to 10, how would you rate our product?
2. What part of the interface felt most intuitive to you?
3. What part of the interface felt least intuitive to you?

With these three main questions we tried to start a conversation with the test group. The questions are closely related to our goal.



5.6. RESULTS

It turns out that the participants found our GUI quite intuitive, rating it 7.6 out of 10 on average. The students were especially positive about the simple and clear design of the application. The three big tabs at the top gave a good representation of the steps the user has to take in order to perform an analysis. On the other hand, we also found out that most users could not initially find the 'Run analysis' button. It turned out that when users entered the script section, their focus initially went to the button above the script editor. Then when they wanted to run the analysis, they expected the 'Run analysis' button to be in that area as well.

We used this feedback to redesign the script section. Among other things, we gave the 'Run analysis' button a more prominent place on the top, next to the other buttons. They also found our script editor a bit dull, being only a plain white text area. To make the editor more attractive, we changed the font to a more bold one. In addition, we also added line numbers.

NOTE: Bjorn van der Laan (4151348) has passed course TI2600 Interaction Design.

6

Evaluation of Functionalities

In general each of the functional modules is performing its functionalities as required by the user. We have only been able to determine two cases where a functionality of an individual module has failed.

- The graphical user interface allows the user to conveniently select input files via drag 'n drop. In addition to this, the input files can also be grouped by their format XML file by dragging and dropping an input file into a group. However, once an input file is grouped, it is not possible to remove that file from the group. The only option is to remove the input file and then add it again.
- The script allows the user to perform all necessary transformations on data, but it is not very consistent in its syntax. For some operations, such as coding, connecting and constraining, the script allows the user to write special expressions. However, there are certain operations which are performed via a more traditional function call, such as *tableWithDays*. In addition to this, there is another function, *addTimeToDate* which modifies its arguments and does not return a value, whereas every other operation in the script is purely functional.

Furthermore, when evaluating the product as a whole, there are several aspects where the program's functionalities fall short.

- We could have transferred more responsibility from the script to the GUI, for example in exporting and generating the visualizations. An advantage of our approach is that because we kept these operations in the script, we were able to keep the GUI simple.
- Syntax errors that occur within the user provided script generate only a very obscure error message. We should have put more focus on generating descriptive error messages. In addition, for both syntactical and semantic errors, we should have integrated error handling better with the GUI, as currently there is no way for a user to see which line or lines are the cause of the error.

7

OUTLOOK

We are quite pleased with the result of our end product. However, we do see possibilities for improvement and extension of certain features.

7.1. IMPROVEMENT

Firstly, we would make the data analysis threaded. So that it is possible to multitask instead of having to wait for the results of the analysis, which might take a while. We would also like to add some sort of panel with a list of the analyses that are in progress.

Secondly, there should be a better way to report syntax errors in the input of the user. Currently certain syntax errors are reported by showing a pop-up with the text: "An unhandled exception occurred while executing the script: null". Clearly this is not exactly the cleanest way to handle syntax errors. In the ideal situation such errors would be highlighted in the input text and the error message would be descriptive. This way it is immediately clear which lines of code cause these errors and it will be easier to resolve them.

Thirdly, we would like to make our scripting language more consistent. Take for example the operations "addTimeToDate" (1) and "tableWithHoursOfDay" (2). Number 1 performs the operations on the existing table. Number 2 takes the table as input, performs the operations and returns a new table, which can then be used for further usage. Instead, they should either both return a new table or perform the operations on the existing table.

7.2. EXTENSION

To start with, for extending our project, we would like to implement more features concerning the visualizations. For example, the ability to create a Stem-and-Leaf plot.

Subsequently, we would like to improve our ability to deal with user questions related to the routines of the patients. We are now able to show the information which gives the user the possibility to conclude a certain routine himself. Our goal would be that if the user gives us a certain routine as input, that we would be able to detect situations where the behavior of the patient differs from this routine. Then the user can draw certain conclusions from our output.

BIBLIOGRAPHY

[1] Bradley, S. (2013). The benefits of modular design. http://www.vanseodesign.com/web-design/modular-benefits/.

[2] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (1998). Extensible markup language (xml). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, 16.

[3] Nielsen, J. (2000). Why you only need to test with 5 users. http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

[4] Rubin, J. and Chisnell, D. (2008). Handbook of usability testing. In *Handbook of Usability Testing*. Wiley.

[5] Sanderson, P. M. and Fisher, C. (1994). Exploratory sequential data analysis: Foundations. *Human–Computer Interaction*, 9(3-4):251–317.

[6] van Lint, C., van der Boog, P., Schenk, S., van Dijk, S., Romijn, F., and Cobbaert, C. (2014). Zelfmonitoring van nierfunctie na niertransplantatie: de patiënt als regisseur. *Voorjaarscongres Nederlandse Vereniging voor Klinische Chemie en Laboratoriumgeneeskunde (NVKC)*.

[7] Van Vliet, H., Van Vliet, H., and Van Vliet, J. (1993). *Software engineering: principles and practice*, volume 3. Wiley.