Renal Transplant Health Analysis

The analysis of data concerning the health of renal transplant patients

Group C

Faculty Electrical Engineering Mathematics and Computer Science

TUDelft 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/.

TUDelft Delft University of Technology

Abstract

abstract here

Contents

1 Introduction

Lezer, Verslag. Verslag, Lezer. Aangenaam. 1 PAGE

2 Software Overview

Introduction to software overview. (what has been made, all must haves implemented, etc). 1 PAGE

2.1. Must Have

Describe all must haves that have been implemented.

2.2. Should Have

Describe the should haves which have been implemented.

2.3. Additional Functionalities

Describer any additional functionalities we have implemented (could & would haves)

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

Write introduction to feature description. 2 PAGES

4.1. Import

4.1.1. XML

4.2. Analysis

4.2.1. The C's

4.2.2. Additional Analysis

4.3. Visualize

4.4. Export

5 Interaction Design

Introduction to this chapter, describe what we are going to show here. 2 PAGES

5.1. Methods

Here we will list and explain the interaction design methods we have used. (emotions and social aspects are irrelevant, etc)

5.2. General Persona

Here we will describe John Doe, an abstraction of our typical user.

5.3. Evaluations

Describe our method of evaluation here (friday evaluations, high fidelity prototype etc)

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
Sprint Plans

B
Sprint Reflections