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

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 describe how the engineering process went. We discuss how we use the GitHub pull requests and how we ensured that out code had good quality. As last we will discuss the feedback we got from SIG.

3.1. Scrum

In the development of this project, we used scrum. On Friday we created a plan for the coming week. Every week-day we started with a short daily-meeting. In here we discussed what we had done, what we were going to do and which problems we encountered. At Friday our sprint ended. Than we created a sprint reflection and gave a demo to the client. Next we started a new sprint. So we created a plan for the next week. We liked from scrum that it was cleat what everybody was going to do and that it is focused on features. Because it is focused on features you were constantly working on code that had to be used and therefore did not happen that one created a lot useless functionality. However the focus on features was also a downside of scrum. There were a lot of task that did not belong to a feature bus they sill had to be done. Also a feature had to be split into sub-features, however it was often better to split them in subtask, for instance a task to implement a certain class. So we would like to be able to specify both features as concrete tasks in an sprin planning.

It had happen to often that not all tasks were done. In the beginning of the project this was caused by underestimating the effort needed for the task. But during the project it often happened that it was caused because some team members started to late on their task and therefore they had not enough time to complete it. This led that we could not implement all planned could-haves and that in the last two weeks a lot of work had to be done. However from out experience from previous project we had expected this. Therefor we took this into account with our original planning. The last three weeks had not essential requirements planned. Therefore if got behind our planning we have the possibility to discard the tasks in the last weeks and use them for essential tasks. However besides that we had the possibility to discard those task, we had not the intention to do that. Since we got behind out schedule, we had to use that buffer and discard those tasks.

3.2. GitHub Pulls

It was not allowed to push changes directly to the master. When a person wanted to merge a change 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 contain comments and javadoc. Furthermore it should be tested and the code should have a good quality. And at last, the tools we use to evaluate the quality of the code, must say that the code was good. We look very critical to pull request. So often the one who had requested the pull requested had to improve some parts of his code. This led to good code quality. However most comments were related to code style and not directly to the implementation of the code. The size of the pull requests varies a lot. In the beginning of the project they were very large. But since they were large, reviewing was difficult. And since they were large it was not possible to do the review quick, so people postponed the reviews. 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. The time that a pull request was open varies a lot. Small reviews were most of the time handled within an hour. However large pull request stayed sometimes open for a very long time. For

large pull request it could take sometimes days before it was reviewed and corrected again. So therefore we tried to keep the pull request small. The reviewing of the code, led to good code quality. Near the end of the project, we were looked less critical to the pull request. Those pull requests had to come in the final product, and we had not the time anymore to spend a lot time on improving the code. Therefore we were less critical in that phases. So for future project we will use the pull request again, but we will try to keep the pull requests small.

3.3. CODE QUALITY

During the project we try to create code of good quality. The code had to meet a few requirements before it was considered of good quality. First of all the code had to follow the conventions. Furthermore the public methods had to be explained in javadoc and unclear code had to be explained in comments. Methods had to be short and had a clear descriptive name. Furthermore the code had to be tested by unit tests. Duplication had to be avoided and it should be possible to change the code easy without breaking stuff. And at last is should work efficient. 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 woold report that. Another tool we used was PMD.PMD detects a variety of code smells. It detects dead code. For instance it detects unused variables and methods. Furthermore it detects empty try/catch/switch blocks and empty if/while statements. Furthermore it detects when expression could be done easier and it detects duplicate code. Furthermore we used findbugs. Findbugs is capable of detecting possible errors. When a person wanted to open a pull request, the tool were not allowed to give an error. Furthermore the code should have been tested. Next 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 happens sometimes that some errors detected by the tools ended up in the main 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 re-factored a part of the code that caused a lot of duplication. To improve the component balance we had to create multiple maven modules. But since the teacher of the course did not agree with on this point, we decided that it was not worth to invest time in creating multiple modules. However we have tried to restructure out packages a bit better. Since our overall feedback was good, we concluded that the way we ensured our code quality was good and had no need to change. Near the end of the project we focused less on quality and more on features. This had harm out code quality a bit.

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

Describe the product here in its entirety, its functional modules and the failure analysis. 2 PAGES

6.1. PRODUCT

Evaluate the product: how we thought it would turn out, how it has turned out.

6.2. MODULES

Explain the functional modules (importing analyzing visualizing and exporting (short recap of chapter 4))

6.3. FAILURE ANALYSIS

Talk about the failures in our system. (problems with types (defining abstract methods for float and int) problems with visualisations and importing)

7

OUTLOOK

1 PAGE

This software is part of an educational course. In this course multiple groups create software based on the same customer and requirements. The software most usable to the customer will be used by the customer in his research. If there are any improvements to be made on the software, the improvements will only have to be made if the software is elected amongst the others to be used by the customer.

7.1. IMPROVEMENTS

Describe the possible improvements in case the software will be used

7.2. FUTURE DEVELOPMENT

In case the software will be used, development will probably be continued in the same way, etc etc etc

A

Sprint Plans

B

Sprint Reflections