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 **1**

2 Software Overview **3**
2.1 Must Have . . . 3
2.2 Should Have . . . 3
2.3 Additional Functionalities . . . 3

3 Engineering Reflection **5**
3.1 Scrum . . . 5
3.2 GitHub Pulls . . . 5
3.3 Code Quality . . . 5
3.4 SIG Feedback . . . 6

4 Feature Description **7**
4.1 Import . . . 7
4.1.1 XML . . . 7
4.2 Analysis . . . 7
4.2.1 The C's . . . 7
4.2.2 Additional Analysis . . . 7
4.3 Visualize . . . 7
4.4 Export . . . 7

5 Interaction Design **9**
5.1 Methods . . . 9
5.2 General Persona . . . 9
5.3 Evaluations . . . 9

6 Product Evaluation **11**
6.1 Product . . . 11
6.2 Modules . . . 11
6.3 Failure Analysis . . . 11

7 Outlook **13**
7.1 Improvements . . . 13
7.2 Future Development . . . 13

A Sprint Plans **15**

B Sprint Reflections **17**

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 dailymeeting. 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 sprintreflection 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 the focused on features there is no room for task that can not be translated into a feature. While those task could be important.

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 have good quality (see section 3.3). We looked very critical to pull request. So often the one who has requested the pull request 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 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 the pull request in future projects. However we will then 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 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 efficient and it had to be easy to change things.

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 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 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

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