Emergent Architecture Design

Boudewijn van Groos[1], Chris Langhout[2], Jens Langerak[3], Paul van Wijk[4], and Louis Gosschalk[5]

[1]bvangroos , 4229843
[2]clanghout , 4281705
[3]jlangerak , 4317327
[4]pjvanwijk , 4285034
[5]lgosschalk , 4214528

May 22, 2015

Contents

1 Introduction

This document describes the design of the architecture and it is continuously updated to the current state of the software. It also states the design goals we want to achieve during the project. Furthermore the subsystem decomposition is laid out.

2 Design goals

2.1 Modularity

The program can be split into a number of modules. These modules should be implemented independently of the other modules. Interaction between modules is only possible by making use of defined interfaces.

2.2 Continuous Integration

There should be always a working product. Each sprint should add one or more features.

2.3 Understandability

We must also make the software user-friendly. Even though the users of the product are already familiar with analysis software, our application should be easy to learn. It is essential that the user understands the usage of the scripting language in our application before he/she is able to use it for analysis.

2.4 Code quality

In the project we strive to have good code readability and the code must be easy to understand. This is also an important part of the static analysis. We handle the pull based system on Github. For each new feature that will be implemented the author of that feature must create a new branch and do all his work in that branch so that the master branch always consists of a working version of the software. If someone wants to commit he has to make a pull request first. His code will then be reviewed by at least 2 teammembers and eventually fixed before it can be merged with the working version in the master branch.
We make use of static analysis tools that provide information about the quality of the software:

Corbertura We use this tool to find out how much test coverage we have in our system. Our goal is to keep the test coverage at at least 75%.

Checkstyle To improve code quality in terms of styling we have code conventions which the writer should follow before his code is accepted. This is done by the Checkstyle plugin in the IDE that makes sure these conventions are followed.

FindBugs This tool will help us find where our program may contain some faults. This also makes code reviewing for teammembers easier since the

process is automated. If there are faults, the writer should fix them to make the code acceptable.

PMD To find some other potential mistakes in the code we use PMD to detect these mistakes.

It is up to the writer to fix the warnings produced by these tools, or provide a clear explanation whether the warning will not be fixed.

3 Software architecture views

3.1 Subsystem decomposition

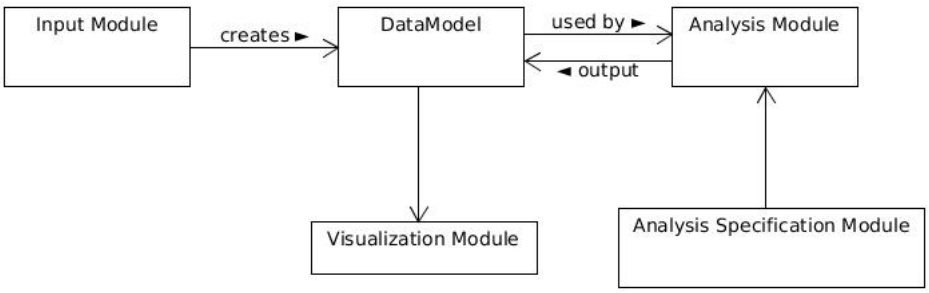

Figure 1: Modules of the system

The input module should be able to read the data. With that data a dataModel can be constructed.
The Analysis Specification module processes the script that defines how the data should be analyzed. The user provides this script. This module translate the script in operations that can be performed by the Analysis Module.
The Analysis Module perform the analyses over the dataModel. The output of this module is the result of the analysis, this is a dataModel.
The Visualization module creates a certain visual representation of a dataModel. For example it can create a box-plot of the creatine levels.

3.2 State diagram

Below is the state diagram which describes the flow of the program.

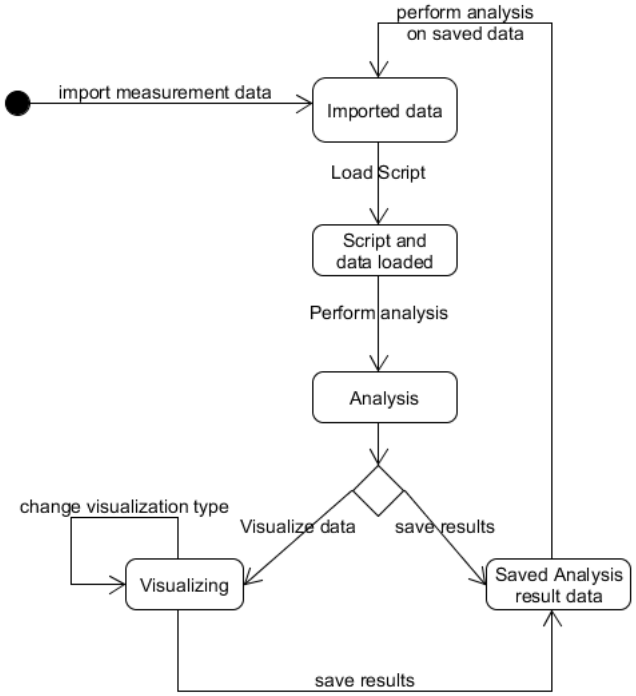

Figure 2: State diagram of the program

4 Software architecture

4.1 Programming languages and auxiliary programs

The software will mainly be written in Java since this was obligatory in the project. For the graphical user interface we make use of the JavaFX framework and is written using fxml. We use Eclipse and IntelliJ IDEA as IDE to write the Java code with. For continuous integration and version control we use Git and Github. Locally we use Maven to build the project. On GitHub we use Travis CI to verify the builds. We also use the JavaFX Scene Builder as a tool to design parts of the GUI because this provides a better preview of what the interface will look like while creating it.

4.2 Used libraries

JavaFX Creating the graphical user interface. We chose JavaFX since this is Java's latest framework to create these interfaces and provides some improvements on the previous frameworks. One of these is the use of fxml to construct the static part (view) of the GUI. Some of us already gained some experience with it during the Software Engineering Methods course.

Mockito Mocking objects to test behavior. We did not specifically choose for this framework but we have used it before in previous courses and it did

not seem like a big necessity to change to a different mocking framework as this will only require more effort to learn it.

Apache POI Reading data analysis files stored in the Microsoft Excel format. This is one of the main excel parsers and easy to use. We chose this library because it applies very good solution for reading the data that was provided to us in excel formats. Another reason is that it is developed by the Apache Software Foundation which also created Maven so it was easy to include in the build.

Java SAX Reading the xml file that specify the datafiles. This choice is made because it is easy to use and well documented. The library is also included in the JDK so it did not require any additional installation effort.

Parboiled

5 Testing

5.1 Unit testing

To make sure that all the code works as it should, the author must always write unit tests for the code he has written. We make use of the JUnit framework to write the tests. To verify behaviour of methods we use the Mockito framework to stub certain object needed by the method.

6 Glossarium

Analysis An analysis that can be done on measuring data to come to an answer to a specific question.

Git & GitHub Git is the tool to control the versions in a repository. GitHub is the web-tool to make the versions available for group use.

GUI Graphical User Interface.

IDE Integrated Development Environment

JDK Java development kit

SUGAR The scripting language used to create analyses.

Pull Request Are made on GitHub to get your commits to the code verified and merged with the master branch.