Architecture Design

TI2806 Contextproject
Health Informatics

Group A

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

CONTENTS

1

INTRODUCTION

This document describes the architecture of the application that we are going to build during the context project Health Informatics. Primarily it will provide a high level description of the structure of the software.

1.1. DESIGN GOALS

A number of design goals have been established for this application.

1.1.1.

Maintainability

As Steve McConnell, author of Code Complete and Software Estimation: Demystifying the Black Art states/explains and supports with statistical data, the number of errors per 1000 lines does seem to be relatively constant, regardless of the specific language involved, which find itself around a number of 50-100 errors per 1000 locs., so even though you test thoroughly and as exhaustively as you can during a project there still exists a chance that after the initial deployment there will be bugs found/entrenched in your code. So you want your program to be testable, which is explained in a subsection below, changeable, which is also explained in a subsection below, as these two combined derive a code base which is maintainable. There is no progress in having code which is testable, but changeable. So after you found bugs, you have to spend too much time/ or cant even fix the bugs, or the other way around, having code which is immensely good upgrade-able/changeable but is hardly/ next to impossible to test efficiently, is also not very helpful.It is therefore very important to have your code as maintainable as possible

1.1.2.

Changeability

Every software needs to be open for new features and modification.As you do not want to rewrite working programs, in which a large part of the program is still usable and up to date. There is an financial loss, in writing code that could have been reused if only it was more portable for modification or extra features. The other loss could be, most probably will be, robustness, often applications in use are tested endlessly. It has a lot of user testing, integration testing, manual testing and all the software test cases which are written for it. This amount of tests are often not reproduce-able without spending too much time/money, as software for companies have strict deadlines and time schedules. So whilst you could have added some features, and spend time extra testing the already existing application, at code which is changeable, you will have to rewrite from scratch. So you really want to have code which is changeable. That means organizing your code in a fashion that as much as possible future changes or/and features can be implemented as simple as possible, the problem you encounter here is that if you know what modification your code needs you are probably writing code that you should not be writing, as it is incomplete or just plainly wrong. So mostly the future features/modifications are unknown. Which does make it admittedly, but still given for reasons here above very invaluable.

1.1.3.

Testability

Proper testing allows defects to be found and fixed relatively easily. Therefor the system should be designed in such a way that it allows for effective testing. This can be realized through modular design. The system should not only be modular on a high level, but also on unit level. For example, through the use of interfaces in order to decrease coupling between units. So there is a possibility to test code thoroughly with as less effort as possible, as you would like to spend as less effort as possible creating tests, but as much as time possible testing. So that finding bugs is not hard, not meaning that you should write code which has immensely high number of bugs, so that you find bugs easily, but writing you code at such a fashion that finding *entrenched* bugs is as efficient as possible. As this would lead to as much as time possible to write many test cases, which concludes to finding as much as bugs possible, which then can be improved/fixed to upgrade your code again.

1.1.4.

Useability

The application will be designed for skilled analysts and so it must be flexible enough and provide sufficient functionality to allow the analysts to transform and manipulate data so that it can be used for a wide variety of statistical analysis. So we rather have something a little bit more complex than the average personal computer, but when used by a more skilled person far more time-efficient to use.

1.1.5.

Availability

The system will be built in such a way that we will have a demo of the product ready for the client each week. This demo can either be in the shape of an actual working product, or some other way of demonstrating what we have been working on. We work in this way so that the client can experience the changes and additions to the program in as early a stage as possible. So that if the client does not like the new features or would prefer them in a different way, then we can make these changes as soon as possible. So at any given moment the code should be working.

2

SOFTWARE ARCHITECTURE VIEWS

2.1. SUBSYSTEM DECOMPOSITION

The system is divided into five main packages, each with a specific and well defined responsibility. This decoupled design between components allows for easier testing. This section will provide an overview of the different subsystems.

- **Graphical User Interface**
  The Graphical User Interface allows the user to select the files to be processed, edit the script/config XML that is used to transform/read the data and view the resulting data and visualizations.
- **Script**
  The script allows the users to input their operations on the data, how they want it to be processed, which ones they want to be processed, and how they want their output to be generated.
- **Control Module**
  The control module is registers the functions used by the script. It parses the data provided by the user with the given config xmls, and returns this in wrappers to the script, so the script can use this data to perform operations on it.
- **Input Module**
  The input module will parse input data provided by the user into a software representation that all modules can work with, which in our case are tables. Input files will require an XML file that describes the layout of the input file, so that if the user would like to use other data, only the XML description file has to be modified, it chooses itself if it is xls, txt or xslx extension, and parses depending on the extension.
- **Operations**
  This is kind of an utility package, which has all our operations which can be performed on the data, which is represented through tables with records and corresponding records.
- **Output Module**
  The output module can generate output files which are support ssp from tables, using any delimiter, format and form that is wanted.
- **Interpreter**
  The interpreter can be seen as an person who translates a language for you, the interpreter translates the language in the script, into functions and calls, which then are traverse through syntactic tree.
- **Visuals**
  Can be called through the script, and generates boxplot/histogram or even an eventmatrix with given data, this is outputted just as the data to the last output panel, and can be chose where to saved to.

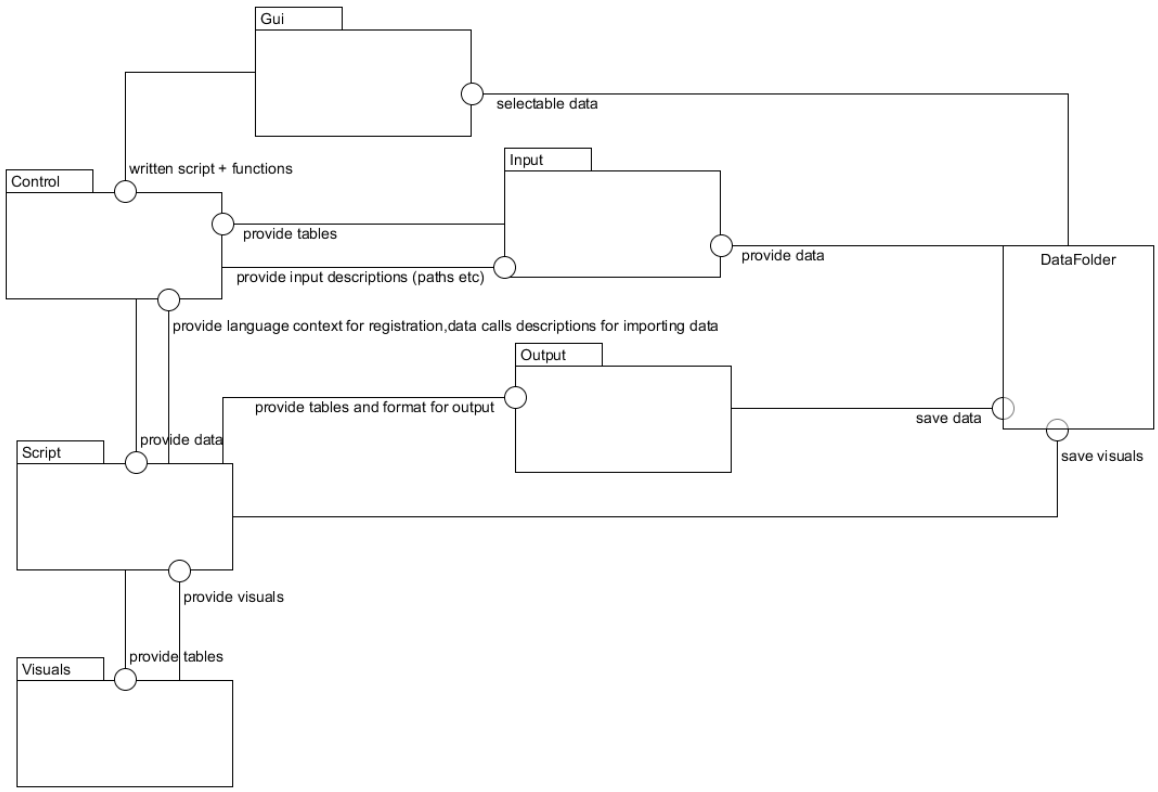

Figure 2.1: Systems workflow.

2.2. HARDWARE/SOFTWARE MAPPING

The application will be a simple desktop application that is ran locally on the user's machine. See figure 2.2.

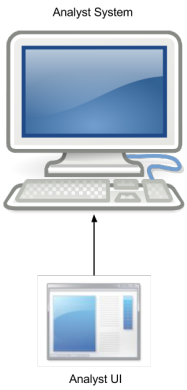

Figure 2.2: Mapping of hardware/software

2.3. PERSISTENT DATA MANAGEMENT

The program will be dealing with three types of persistent data: the input data, output data and visualizations. The input and output data can be either text or binary data. The visualizations will only be binary data. All types of data will be stored and managed by the file system of the user's machine.

2.4. CONCURRENCY

As the application will be executed locally on the user's machine, concurrency between multiple instances of the application is not a concern. However, the software should be designed in such a way that it can be extended to process multiple input files in parallel since it is likely that the user will want to perform the same transformations on an array of input files (e.g. the StatSensor data for each patient).

2.5. TESTING

A simple fact is that whenever you buy something, you want it to work, that is exactly the same with software. So as developers you need to be sure that your program works, and be able to show that, which is achieved by testing.

2.5.1.

Junit /mockito

We chose Junit as our testing frame work, in which we make test cases for methods, and used mockito/powerMockito to stub more difficult reachable code which was needed to test methods.

2.5.2.

Statistical Analyses

Even when you are testing, you can miss sometimes issues. So we used multiple analysis tools that provide us with code quality benchmarks, we chose for 70 - 80 percent test coverage, clean pmd/findbugs log and no checkstyle errors except when they are explained.

- Cobertura/ECLemma
- CheckStyle
- FindBugs
- PMD

2.6. PROGRAMMING

How we tackled our programming environment.

2.6.1. IDE

We chose for eclipse, as eclipse was known to be best integrated with the tools above.

2.6.2. LANGUANGES
An fixed constraint was that the program would be written in java, so there is not much too discuss about the language.

2.6.3. LIBRARIES
We used a fair amount of libraries. We used POI for our xls and xlsx parsers, Swing for our gui and antlr for our script.

2.7. CODE INTEGRATION
2.7.1. TRAVIS
Working in a group, should be as efficient as possible, so you would like to have some automatic code checker/controlling mechanism, whenever someone tries to add new code, you want to know whether it is not broken. A kind of controlling automated server.

MAVEN
When working with your code, you want to have some kind of software, which automatically tests all your code, with any given plug-in and builds it to something you can work with, to test your if your project still functions, to find out about statistics, all together, simply build just with a few clicks, every statistical tool included, it can even zip your whole project or build it into for example an jar, so you can send it and ask people to run it and test it for you.

2.7.2. GITHUB
Working in a group, should be as efficient as possible, so you would like to have a platform at which every member can access the code at any given time.

3

GLOSSARY

- Module = a separate component, which itself performs a defined task and can be linked with other modules to form a larger system. (thefreedictionary.com/module, accessed: 20-May-2015)
- Script = a list of commands that are executed by a certain program or scripting engine. Scripts may be used to automate processes on a local computer. Script files are usually just text documents that contain instructions written in a certain scripting language. (techterms.com/definition/script, accessed: 20-May-2015)
- XML = is used to define documents with a standard format that can be read by any XML-compatible application. XML allows you to create a database of information without having an actual database. (techterms.com/definition/xml, accessed: 20-May-2015)
- Binary data = once a program has been compiled, it contains binary data called "machine code" that can be executed by a computer's CPU. In that case, "binary" is used in contrast to the text-based source code files that were used to build the application. (techterms.com/definition/binary, accessed: 20-May-2015)