Architecture Design

TI2806 Contextproject

Health Informatics

Group A

Contents

1 Introduction

This document describes the architecture of the application that we have built 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

Maintainability is the ease with which a product can be maintained in order to isolate defects or their cause, fix the defects, fix or replace components, make future additions or modifications to the system easier, etc.

In every one of these cases it is very important that the code is well documented and commented. This makes it possible for developers who were not involved in the original creation of the system, to make modifications to the system or add new features to it.

Other software engineering aspects such as the combination of software Testibility and Changeability are also very important to the maintainability of a system. If these aspects are implemented well, the process of maintaining changes, adding new features and testing them will become a much more easy and efficient process.

If maintainability is not implemented well, the product's useful life will be cut very short. A big cause of this will also be that it will hardly be possible to repair or replace faulty or worn-out components of the system without having to replace the parts that still work.

1.1.2. Changeability

All of our software needs to be open to possible future development. There are still many more questions our system could answer in the future, but we do not want to have to rewrite large parts of the program when almost everything is still usable. It should be open to additions and modifications without having to spend a lot of effort recoding and restructuring working parts.

Our system works well, but there are so many possibilities for improvement and expansion, so for us this is a very important design aspect.

1.1.3. Testability

In order for a system to be testable, several aspects of programming should be considered.

It is very important for code to have a very clear and organized structure. This also implies that methods should not be too big. It is very important that the components under test all have a well defined responsibility. If a method contains several functionalities / responsibilities, then it should be split up into seperate smaller methods.

If the previous programming method is followed, it will also automatically make it so much easier to test components in isolation. These two aspects are essential to a quick and effective debugging process. It is then made so much easier to find and isolate the part of the code that is responsible for bugs.

It is clear that, while writing code, one should already consider these aspects. This way a lot of time can be saved while writing tests, and the debugging process will become so much more efficient and effective.

1.1.4. Usability

The appliciation that we have developed is meant for skilled analysts. This implies that our system should provide sufficient and useful functionalities, in order to give the analysts the possibility to apply possibly complex and extensive transformations and manipulations on the input data. The fact that the analysts are skilled does give us the freedom to make the underlying software slightly more complex than average. But when used by these skilled analysts, this should not give them any trouble and will give them the opportunity to perform their complex analysations way more time-efficiently.

1.1.5. Availability

The goal was to build the system in such a way that we would have a demo of the product ready to show to the client at the end of each week. This demo either had the shape of an actual working (part of the) product, or some other way of demonstrating what we had been working on.

The reason for this approach i 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 shape or form, the we could make these changes as soon as possible. So at any given stage in the development process, the code up till that point should be working and should be presentable to the client in some way. This approach was very important for our development process, because this way changes following the feedback can be processed in as early a stage as possible. Otherwise, the longer we would have waited with these changes, the more difficult and time consuming it would have become for us to apply these changes.

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. See figure 2.2 for a visual representation.

- **Graphical User Interface**
  The Graphical User Interface allows the user to select the files to be processed, edit the script to input what they want to have processed and their output or to edit the config XML that is used to transform/read the data. Hence we can view the transformed data in the last panel as a visualisation or just a outputted table.
- **Control Module**
  The Controle Module functions as a bridge. It gets data from the GUI and passes it to the other modules. After the other modules have processed the data, the results are given back to the GUI to show to the user.
- **Input Module**
  The Input Module parses input data provided by the user and models it as a software representation that the other modules can work with. In our case it creates a Table object. Besides input data, the Input Module also asks users for a configuration XML that describes the data set. This enables it to parse almost every format, as long as you have an XML for it.
- **Operations**
  This module functions as an utility package. It owns all our sequential analysis operations. Other modules, especially the Interpreter, can ask this model to perform operations on the data.
- **Parser**
  The parser reads the script that is created by the user, and creates converts it into a data structure. This structure is used by the Interpreter Module.
- **Interpreter**
  The Interpreter executes the instructions that the user has written in the script. It uses the Parser Module to parse it to a format the Interpreter can understand, and it utilizes the Operations Module to perform operations onto the data.
- **Visuals**
  This module can generate several visual representations based on data sets. It helps user get insight into the data. The visuals can be exported as a PDF file for later reference.
- **Output Module**
  The Output Module exports the (manipulated) data sets to several export formats. The exported file can then be used in statistical analysis applications like SPSS. The user can specify any output format that he likes.

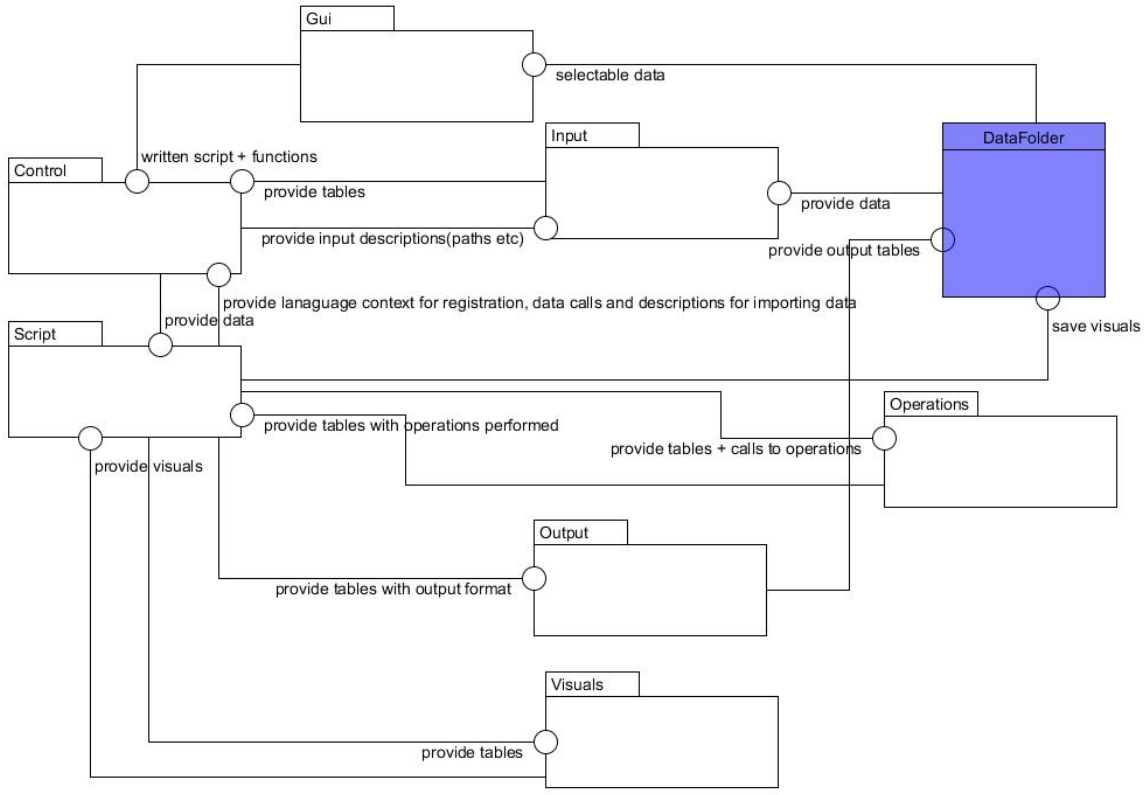
Gui
selectable data
Input
DataFolder
Control
written script + functions
provide tables
provide input descriptions(paths etc)
provide data
provide output tables
provide lanaguage context for registration, data calls and descriptions for importing data
provide data
save visuals
Script
provide tables with operations performed
Operations
provide tables + calls to operations
provide visuals
Output
provide tables with output format
Visuals
provide tables

Figure 2.1: Systems workflow

2.2. HARDWARE AND SOFTWARE MAPPING

The application will be a desktop application that is ran locally on the user's machine. It does not require a connection to a local database or to the Internet. Moreover, it does not require any additional hardware. See figure 2.2.

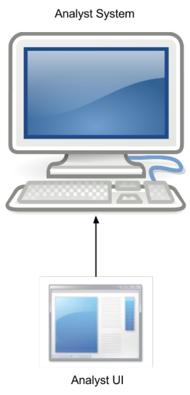

Figure 2.2: Mapping of hardware/software

2.3. PERSISTENT DATA MANAGEMENT

The program will be dealing with data sets, configuration XML files and visualization PDF files. Data sets can be either text or binary data, where visualizations will only be binary data. All data will be stored and managed by the file system of the user's machine, in the data folder of the application.

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 perform multiple analyses in parallel since so that the user does not have to run them sequential.

2.5. QUALITY OF CODE

2.5.1. TESTING

Testing is an essential step in examining the correctness, completeness and quality of the developed software. As our testing framework we chose JUnit, with which we create unit tests for our methods. We chose this library because it is widely adopted adn we are all familiar with it. Next to that, we use Mockito and Powermock to mock objects during the tests. We chose Mockito because we all have experience with it from the course Software Quality and Testing. Finally, we use the EclEmma plugin to examine our coverage. This plugin integrates nicely with our IDE of choice, Eclipse.

2.5.2. STATISTICAL ANALYSIS

Besides testing, it is also important that our code adheres to coding conventions and is checked for bugs. We used CheckStyle, FindBugs and PMD as our tools because, according to the Software Engineering Rubrics, these tools are also used to criticize our code.

3

GLOSSARY

- Module = a separate component, which itself performs a defined task and can be linked with other modules to form a larger system.[1]
- 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.[3]
- 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.[4]
- 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.[2]

BIBLIOGRAPHY

[1] Dictionary, T. F. (2013). Module. thefreedictionary.com/module//. [Online; , accessed: 20-May-2015].

[2] TechTerms (2013a). Binary. techterms.com/definition/binary//. [Online; , accessed: 20-May-2015].

[3] TechTerms (2013b). XML. techterms.com/definition/xml//. [Online; , accessed: 20-May-2015].

[4] TechTerms (2013c). XML. techterms.com/definition/xml//. [Online; , accessed: 20-May-2015].